Sergiu CORLAT **Anatol Gremalschi**

GRAFURI

metodologia predării în cadrul instruirii de performanţă la disciplinele Matematică & Informatică

Chişinău, 2013

Aprobată pentru editare de Senatul Universităţii de Stat Tiraspol

Lucrarea este destinată cadrelor didactice, activitatea cărora ţine de instruirea de performanţă la disciplinile Informatica şi Matematica, dar şi studenţilor facultăţilor de matematică şi informatică a instituţiilor de învăţământ superior din ţară, care studiază cursul de teorie a grafurilor. Este de asemenea utilă elevilor pentru pregătirea către concursurile naţionale şi internaţionale de programare.

Autori:

Sergiu Corlat, lector superior universitar, UST
Anatol Gremalschi, dr. hab., profesor universitar, UTM

Recenzenţi:

Ilie Lupu, dr. hab., profesor universitar, UST
Andrei Braicov, dr. conferenţiar universitar, UST

Corlat, Sergiu.
Grafuri : Metodologia predării în cadrul instruirii de performanta la disciplinile Matematică & Informatică : [pentru uzul studenţilor] /
Sergiu Corlat, Anatol Gremalschi;
Acad. de Ştiinte a Moldovei, Univ. de Stat din Tiraspol. – Chişinău.
Universitatea de Stat din Tiraspol, 2014. - 158 p. - (informatica)
100 ex.
ISBN 978-9975-76-122-2.
519.17+004.421.2(075.8)
C71

Cuprins

Introducere

Apărute din necesitatea de a modela diverse situații, relații sau fenomene în formă grafică, grafurile şi-au găsit o multitudine de aplicații în cele mai diverse sfere ale activității umane: construcții şi sociologie, electrotehnică şi politologie, chimie şi geografie … acest şir poate fi continuat la nesfârşit.

Teoria grafurilor a luat naştere de la *problema podurilor din Konigsberg*, cercetată de Euler şi s-a dezvoltat ca un compartiment al matematicii clasice până la momentul apariției sistemelor electronice de calcul şi a teoriei algoritmilor. În contextul rezolvării problemelor de calcul automat, grafurile s-au dovedit a fi un instrument universal şi extrem de flexibil, devenind un compartiment al matematicii aplicate.

O adevărată revoluție a cunoscut-o teoria grafurilor în anii 60 – 80 ai secolului trecut, când a fost stabilită posibilitatea de utilizare a lor pentru rezolvarea problemelor de optimizare. Algoritmii pentru determinarea drumului minim, punctelor mediane, centrelor, de maximizare a fluxurilor, dar şi multe altele au devenit componente vitale ale cercetărilor operaționale şi a metodelor de optimizare.

În aspect informatic grafurile apar şi în calitate de structuri eficiente de date, în special *arborii*, care permit realizarea optimă a algoritmilor de sortare şi căutare.

Numărul de lucrări, care studiază aspectele algoritmice ale teoriei grafurilor este unul impunător. Printre primele apariții se numără *Applied graph theory* de Wai-Kai Chen; *Graph Theory. An Algorithmic approach* de Nicos Christofides; urmate de *Algorithmic ghaph theory* de Alan Gibbons, *Algorithms in C* de Thomas Sedgewick şi multe alte ediții. Totuşi, majoritatea edițiilor se axează doar pe descrierea matematică a algoritmilor, fără a le suplini prin implementări într-un limbaj de programare sau altul, or, tocmai implementarea algoritmică este pentru programatorii practicieni o componentă de importanță maximă.

În prezenta lucrare se încearcă abordarea „verticală" a algoritmilor clasici ai teoriei grafurilor: de la noțiuni teoretice, definiții

şi teoreme, către descrieri matematice a algoritmilor, urmate de implementări integrale sau parțiale (fără prezentarea subprogramelor auxiliare) în limbajele de programare C sau Pascal, însoțite de prezentarea şi analiza rezultatelor.

Un motiv al prezentării parțiale a implementărilor algoritmilor este concepția de *manual* a lucrării: restabilirea părților lipsă a programelor devine un exercițiu practic pentru toți cei care studiază cursul de „Teorie a grafurilor" în instituțiile de învățământ superior din țară.

Ediția este destinată nu doar studenților facultăților de profil, dar şi elevilor pasionați de informatică, precum şi profesorilor, care au în grija lor pregătirea de performanță în domeniul Informaticii – teoria grafurilor este inclusă în calitate de compartiment obligatoriu în curriculumul internațional de performanță pentru Computer Science. Exercițiile, cu care se finalizează fiecare capitol, pot fi folosite în calitate de lucrări de laborator – rezolvarea lor presupune prezența cunoştințelor teoretice dar şi a competențelor practice de programare.

Venim şi cu o recomandare pentru instrumentele informatice, care pot fi utilizate pentru rezolvarea exercițiilor propuse la finele fiecărui capitol: cele mai „prietenoase" medii de programare s-au dovedit a fi compilatoarele Dev C++, MinGW Developer Studio, Free pascal – toate produsele fiind în distribuție liberă.

Sperăm că ediția va deveni nu doar un suport teoretic eficient, dar şi unul aplicativ pentru toți cei care studiază elementele de programare şi cursurile de teorie a grafurilor.

Autorii

Capitolul 1. Noțiuni generale

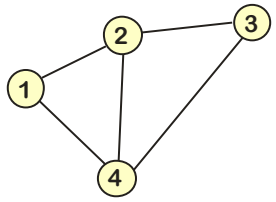
În acest capitol:

- Grafuri. Vârfuri, muchii.
- Grad al vârfului, vecini
- Căi, cicluri
- Subgrafuri
- Grafuri orientate, planare, conexe
- Metode de reprezentare a grafurilor: matricea de adiacenţă, matricea de incidenţă, lista de muchii, lista de vecini

1.1 Definiţii

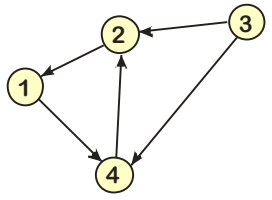
Def. Graf neorientat (graf) – o pereche arbitrară $G=(V,E)$ $E \subseteq \{\{u,v\} : u,v \in V \,\&\, u \neq v\}$

Def. Graf orientat (graf) – o pereche arbitrară $G=(V,E)$, în care $E \subseteq V \times V$



Des 1.1. Graf neorientat

V formează mulțimea *vârfurilor* grafului, ***E*** – mulțimea *muchiilor*. De obicei vârfurile grafului sunt reprezentate în plan prin puncte sau cercuri iar muchiile – prin segmente care unesc vârfurile.



Des 1.2 Graf orientat

Pentru graful din desenul 1.1 $V=\{1,2,3,4\}$, $E=\{\{1,2\},\{2,3\},\{3,4\},\{1,4\},\{2,4\}\}$

Într-un graf orientat muchiile[1] se reprezintă prin săgeți, care indică direcția de deplasare pe muchie. Pentru graful orientat din desenul 1.2 $V=\{1,2,3,4\}$, $E=\{(1,4),(2,1),(3,2),(3,4),(4,2\}$.

Muchia (u,v) este *incidentă* vârfurilor u,v, iar acestea sunt *adiacente* muchiei.

Grade

Def. *Gradul* vârfului v, $d(v)$ este numărul de muchii, incidente acestuia[2]. Un vârf este *izolat*, dacă gradul lui este 0.

Mulțimea de *vecini* ai vârfului v_i, $\Gamma(v_i)$ este formată din vârfurile adiacente la v_i. Într-un graf orientat mulțimea vecinilor este formată din două componente distincte: $\Gamma(v_i)=\Gamma^+(v_i)\cup$.

$\Gamma^+(v_i)$ este formată din vârfurile arcelor cu originea în v_i. $\Gamma^-(v_i)$ este formată din vârfurile-origine ale arcelor care se termină în v_i. Pentru vârful 4 din graful reprezentat pe desenul 1.2 $\Gamma^+(4)=\{2\}$, $\Gamma^-(4)=\{1,3\}$.

Căi şi cicluri

Def. *Cale* în graf este o consecutivitate de vârfuri $v_1,v_2,...v_k$ astfel încât $\forall i=1,...,k-1$ vârfurile v_i,v_{i+1} sunt adiacente[3]. Dacă toate vârfurile $v_1,v_2,...v_k$ sunt distincte, calea se numeşte *elementară*. Dacă $v_1=v_k$ atunci $v_1,v_2,...v_k$ formează un *ciclu* în G. Ciclul este *elementar*, dacă $v_1,v_2,...v_{k-1}$ sunt distincte.

[1] Muchia în graful orientat se numeşte şi *arc*.
[2] Pentru vârfurile din grafuri orientate se definesc *semigrade de intrare* (numărul de muchii, care intră în vârf) şi *semigrade de ieşire* (numărul de muchii cu originea în vârful dat)
[3] Există muchia (v_i,v_{i+1})

Pe desenul 1.1 secvența de vârfuri 1,2,4 formează o cale; secvența 1,2,4,1 – un ciclu elementar.

În grafurile orientate noțiunea de cale este substituită prin *lanț*.

Def. Lanțul este o consecutivitate de muchii (arce) luate astfel, încât vârful arcului (i) coincide cu originea arcului (i+1).

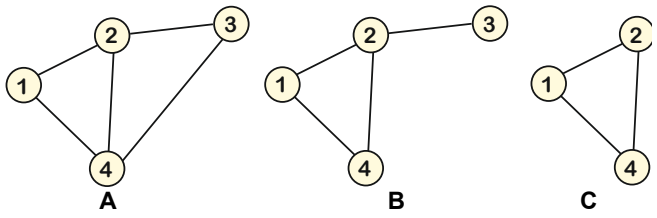
Pe desenul 1.2 secvența de arce (3,4)(4,2)(2,1) formează un lanț; secvența (1,4)(4,2)(2,1) – un ciclu elementar.

Subgrafuri

Def. *Subgraf* al grafului $G=(V,E)$ se numeşte orice graf $G'=(V',E')$ astfel încât $V'\subseteq V, E'\subseteq E$.

Subgrafurile se obțin din graful inițial fie prin excluderea muchiilor, fie prin excluderea muchiilor şi vârfurilor din graful inițial. În cazul când din graful inițial se exclude vârful v_i, odată cu el se exclud şi toate muchiile incidente acestuia. Excluderea muchiei (u,v) nu presupune şi excluderea din graf a vârfurilor u,v.

Dacă în subgraful $G'=(V',E')$ are loc relația $V'=V$, subgraful este unul *bazic* (desenul 1.3 B). Subgraful $G_S^{'}(V_S,E_S)$ în care $V_S\subset V, E_S\subset E$ se numeşte *subgraf generat* dacă pentru orice $v_i\in V_S, \Gamma(v_i)=\Gamma(v_i)\bigcap$ (desenul 1.3 C). Cu alte cuvinte, $G_S^{'}$ este format dintr-o submulțime V_S a vârfurilor din G împreună cu muchiile, care au ambele extremități în V_S.

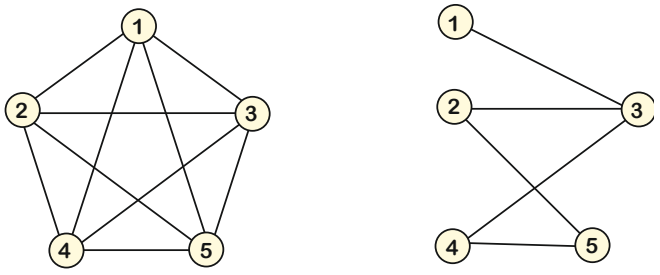


Des 1.3 Graful inițial (A), subgraf bazic (B), subgraf generat (C).

Tipologia grafurilor

Def. Graful $G=(V,E)$ se numeşte *complet* dacă între oricare pereche de vârfuri $v_i, v_j \in V$ există cel puţin o muchie, care le uneşte. Un graf complet cu n vârfuri se notează K_n. (desenul 1.4 A)

Def. Graful neorientat $G=(V,E)$ este *bipartit*, dacă mulţimea V a vârfurilor lui poate fi divizată în două submulţimi distincte V^A şi V^B, astfel încât orice muchie din E are începutul în V^A, iar sfârşitul în V^B. (desenul 1.4 B) Graful orientat $G=(V,E)$ este *bipartit*, dacă mulţimea V a vârfurilor lui poate fi divizată în două submulţimi distincte V^A şi V^B, astfel încât orice arc din E are originea în V^A, iar sfârşitul în V^B.



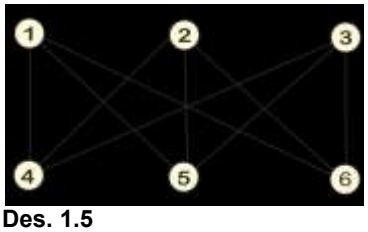
Des 1.4
Graf complet K_5 (A), graf bipartit (B).

Def. Graful bipartit este numit *complet* dacă
$\forall v' \in V^A, \forall v'' \in V^B \; \exists (v', v'') \in E$

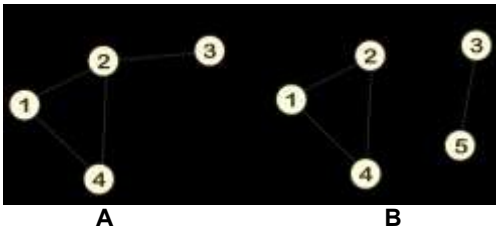
Graful bipartit complet cu părţi formate din n şi m vârfuri se notează $K_{n,m}$ (desenul 1.5)

Def. Graful $G=(V,E)$ este *conex*, dacă pentru orice două vârfuri $v_i, v_j \in V$ în G există cel puţin o cale, care le uneşte (desenul 1.6

A). În cazul în care graful este format din câteva subgrafuri conexe separate el este *neconex* (desenul 1.6 B).



Des. 1.5
Graful bipartit complet $K_{3,3}$



A **B**

Des. 1.6 Graf conex (A), graf neconex (B)

Def. Graful $G=(V,E)$ este numit *arbore*, dacă este conex şi nu conține cicluri.

Def. Graful $G=(V,E)$ este numit *graf planar*, dacă poate fi reprezentat în plan fără intersecții ale muchiilor.

Def. O *față* a grafului este o regiune a planului, delimitată de muchiile acestuia, care nu conține în interior muchii sau vârfuri ale grafului.

Pentru un graf amplasat pe o suprafață există relație între numărul de vârfuri, muchii şi fețe. Relația se numeşte *caracteristică Euler a suprafeței*.

Teorema 1. (formula Euler) Într-un graf planar conex are loc următoarea relație:

$$n-m+r=2$$

unde: n – numărul de vârfuri ale grafului, m – numărul de muchii, r – numărul de fețe.

□

Demonstrație. Prin inducție după numărul de muchii m.

$m=0. \quad \Rightarrow \quad n=1 \,\&\, r=1. \quad 1-0+1=2.$

Fie teorema este adevărată pentru $m=k. \qquad n-k+r=2.$

Se adaugă încă o muchie $\qquad m=k+1.$

Dacă muchia uneşte două vârfuri existente, atunci $r' = r + 1$.
$n - (k+1) + (r+1) = n - k + r - 1 + 1 = n - k + r = 2.$
Dacă muchia uneşte un vârf existent cu unul nou, atunci $n' = n + 1$.
$(n+1) - (k+1) + r = n - k + r + 1 - 1 = n - k + r = 2.$
■

Corolar. Într-un graf planar ($n > 3$), conex, $m \le 3n - 6$.
□
Fiecare față este delimitată de cel puțin 3 muchii, iar fiecare muchie delimitează cel mult două fețe. Prin urmare $3r \le 2m$. Înlocuind în formula precedentă, se obține:

$$2 = n - m + r \le n - m + \frac{2m}{3} \quad \Rightarrow \quad 6 \le 3n - 3m + 2m \quad \Rightarrow \quad m \le 3n - 6$$

■

Corolar. Un graf este planar atunci şi numai atunci când nu conține subgrafurile K_5 şi $K_{3,3}$

□ Demonstrație: [4, pag. 284] ■

Ponderi

În unele cazuri muchiile grafului posedă caracteristici numerice suplimentare, numite ponderi. Muchiei (arcului) (v_i, v_j) i se pune în corespondență valoarea $c_{i,j}$ - *ponderea* (costul, lungimea etc.). Graful, muchiilor căruia î-i sunt asociate ponderi se numeşte *graf cu muchii ponderate*. Pentru rezolvarea unor probleme este necesară şi aplicarea ponderilor la vârfurile grafului. Vârfului v_i i se pune în corespondență caracteristica numerică c_i - *ponderea* (costul). Graful, vârfurilor căruia î-i sunt asociate ponderi se numeşte *graf cu vârfuri ponderate*.

Dacă graful $G = (V, E)$ este unul ponderat, atunci pentru căile din graf se introduce caracteristica numerică l - *cost* (lungime) egală cu suma ponderilor muchiilor din care este formată o cale C.

$$l(C) = \sum_{(v_i, v_j) \in C} c_{i,j}$$

1.2 Structuri de date pentru reprezentarea unui graf

Pentru rezolvarea problemelor pe grafuri cu ajutorul calculatorului, reprezentarea lor „naturală" în formă de puncte (pentru noduri) şi linii care le unesc (muchii) nu este încă cea mai optimă. Selectarea şi utilizarea corectă a structurilor de date care modelează un graf poate influența ordinul complexității algoritmului, prin urmare şi eficiența lui.

Structurile de date, prezentate în paragraful dat sunt cel mai des utilizate în rezolvarea problemelor standard pe grafuri. Ele pot fi folosite atât pentru grafurile neorientate, cât şi pentru cele orientate.

Structura de date clasică pentru reprezentarea unui graf $G=(V,E)$ este considerată **matricea de incidență**. Este realizată prin un tablou bidimensional cu N linii (N – numărul de vârfuri în graf, $N=|V|$) şi M coloane (M – numărul de muchii, $M=|E|$). Fiecare muchie (u,v) este descrisă într-o coloană a tabloului. Elementele coloanei sunt egale cu 1 pentru liniile care corespund vârfurilor u şi v, 0 – pentru celelalte. În cazul grafului orientat vârful din care începe arcul este marcat cu -1, vârful final – cu +1.

Pentru grafurile din des. 1.1,1.2 matricele de incidență vor fi

Des.1.1

	(1,2)	(1,4)	(□,3)	(2,4)	(3,4)
1	1	1	0	0	0
2	1	0	1	1	0
3	0	0	1	0	1
4	0	1	0	1	1

Des. 1.2

	(2,1)	(1,4)	(3,2)	(2,4)	(3,4)
1	+1	-1	0	0	0
2	-1	0	+1	+1	0
3	0	0	-1	0	-1
4	0	+1	0	-1	+1

Matricea de incidență pentru un graf cu N noduri va avea N linii, numărul de coloane fiind proporțional cu N^2. Numărul total de elemente în structura de date este proporțional cu N^3. Utilizarea memoriei este în

acest caz ineficientă, deoarece doar câte două elemente din fiecare linie vor conține date real utilizabile.

O altă reprezentare matriceală a grafului $G=(V,E)$ este **matricea de adiacență**. Matricea de adiacență este şi ea realizată prin un tablou bidimensional, cu N linii şi N coloane, în care elementul cu indicii (i,j) este egal cu 1 dacă există muchia care uneşte vârful v_i cu vârful v_j şi 0 – în caz contrar. Datele despre muchia (v_i,v_j) se dublează în elementele tabloului cu indicii (i,j) şi (j,i) În grafurile orientate, pentru arcul (v_i,v_j) primeşte valoarea 1 doar elementul (i,j) al tabloului.

Pentru grafurile din des. 1.1,1.2 matricele de adiacență vor fi

Des. 1.1

	1	2	3	4
1	**0**	**1**	**0**	**1**
2	1	**0**	**1**	**1**
3	**0**	**1**	**0**	**1**
4	**1**	**1**	**1**	**0**

Des. 1.2

	1	2	3	4
1	**0**	**0**	**0**	**1**
2	**1**	**0**	**0**	**1**
3	**0**	**1**	**0**	**1**
4	**0**	**0**	**0**	**0**

Matricea de adiacență pentru un graf cu N vârfuri are N linii şi N coloane. Numărul total de elemente în structura de date este N^2.

O altă categorie de reprezentări ale grafului o formează reprezentările prin liste. Cea mai simplă pentru implementare listă este **lista de muchii**. Lista de muchii conține M perechi de forma (v_i,v_j), fiecare pereche reprezentând o muchie din graf, descrisă prin vârfurile care o formează. Într-un graf orientat perechea descrie un arc, începutul lui fiind determinat de primul indice din pereche.

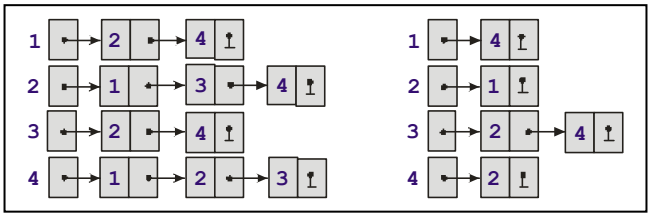
Pentru grafurile din des. 1.1,1.2 listele de muchii vor fi

Des.1.1	Des. 1.2
(1,2)(1,4)(2,3)(2,4)(3,4)	(1,4)(2,1)(2,4)(3,2)(3,4)

Lista de muchii este formată din M perechi de elemente. M – numărul de muchii. Cu toate că structura este mai compactă decât matricea de incidență sau matricea de adiacență, majoritatea operațiilor standard pe graful reprezentat în acest mod necesită parcurgerea întregii liste, ceea ce scade din eficiența structurii. În grafurile orientate, primul element al perechii care descrie arcul va fi vârful sursă, al doilea – vârful destinație. În cazul în care muchiile (arcele) au ponderi asociate, fiecare muchie (arc) va fi descrisă de un triplet: indicii vârfurilor care o formează şi ponderea acesteia.

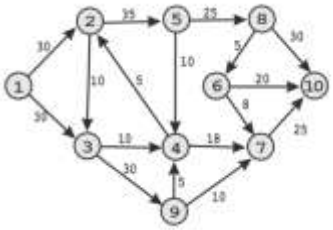
Încă o structură eficientă este **lista de incidență.** Pentru fiecare nod $v \in V$ ea va conține o listă unidirecțională alocată dinamic, cu toate vârfurile $u : \exists (v,u) \in E$. Indicatorii către începutul fiecărei liste pot fi păstrați într-un tablou unidimensional. Elementul cu indicele i din tablou va conține indicatorul către lista de vârfuri incidente vârfului v_i din graf. Pentru grafurile neorientate descrierea fiecărei muchii se dublează, iar operațiile de adăugare (lichidare) a muchiilor presupun prelucrarea a două liste.

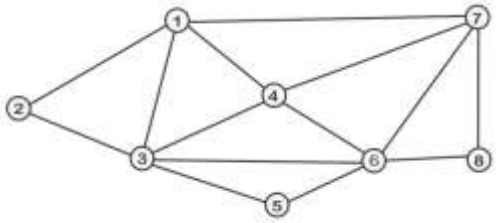
Pentru grafurile din des. 1.1,1.2 listele de vârfuri vor fi:



Exerciţii:

1. Pentru grafurile din imagini construiți:





   a) matricea de incidență;
   b) matricea de adiacență;
   c) lista de muchii;
   d) lista de vecini.

2. Elaborați un program pentru citirea dintr-un fişier text a matricei de adiacență a grafului ($|V| \le 20$) şi afişarea ei pe ecran. Prima linie a fişierului de intrare va conține un număr întreg n – dimensiunea matricei. Următoarele n linii vor conține câte n numere întregi, separate prin spațiu – elementele matricei de adiacență a grafului.

Capitolul 2. Parcurgeri. Conexitate

În acest capitol:

- Parcurgerea grafului în lăţime
- Parcurgerea grafului în adâncime
- Grafuri tare conexe
- Determinarea componentelor tare conexe

2.1 Parcurgerea grafului

Cele mai multe din problemele formulate pe grafuri necesită o cercetare a legăturii între vârfurile acestora. Evident, un algoritm eficient va accesa muchia (vârful) o singură dată, sau de un număr constant de ori. De aici rezultă necesitatea unor metode eficiente pentru parcurgerea vârfurilor unui graf.

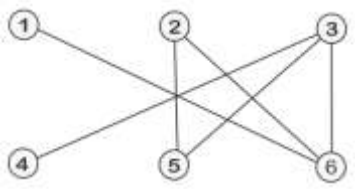
În caz general problema parcurgerii se formulează în felul următor: Fie dat graful $G=(V,E)$. Pentru un vârf dat $v \in V$ să se determine mulțimea $U \subseteq V$: $\forall u \in U$ există cel puțin o cale între v şi u.

Una dintre cele mai eficiente metode de parcurgere în graf este **parcurgerea în adâncime**[4]. La baza metodei stă principiul de selectare recursivă a vârfurilor şi etichetare a lor. Inițial toate vârfurile se consideră *neatinse*. Fie v_0 vârful de la care începe parcurgerea. v_0 se etichetează ca fiind *atins*. Se alege un vârf u, adiacent v_0 şi se repetă procesul, pornind de la u. În general, fie v vârful curent. Dacă există un vârf u, nou (neatins), adiacent v $(\exists (v,u) \in E)$, atunci procesul se repetă pornind de la u. Dacă pentru vârful curent v nu mai există vârfuri vecine neatinse, el se etichetează ca fiind *cercetat*, iar procesul de parcurgere revine în vârful precedent (din care s-a ajuns în v). Dacă

[4] Depth first search (eng.)

$v = v_0$ parcurgerea a luat sfârşit. Pentru realizarea algoritmului se vor folosi marcaje aplicate vârfurilor grafului (0 – nod *nou*, 1 – *atins*, 2 – *cercetat*).

Exemplu: pentru graful din imaginea alăturată se va simula parcurgerea în adâncime din vârful 1, în conformitate cu algoritmul descris anterior:



Pas 1	vârf	**1**	2	3	4	5	6	Pas 7	vârf	1	**2**	3	4	5	6
	stare	1	0	0	0	0	0		stare	1	1	1	2	1	1
Pas 2	vârf	1	2	3	4	5	**6**	Pas 8	vârf	1	2	3	4	**5**	6
	stare	1	0	0	0	0	1		stare	1	2	1	2	1	1
Pas 3	vârf	1	2	**3**	4	5	6	Pas 9	vârf	1	2	**3**	4	5	6
	stare	1	0	1	0	0	1		stare	1	2	1	2	2	1
Pas 4	vârf	1	2	3	**4**	5	6	Pas 10	vârf	1	2	3	4	5	**6**
	stare	1	0	1	1	0	1		stare	1	2	2	2	2	1
Pas 5	vârf	1	2	**3**	4	5	6	Pas 11	vârf	**1**	2	3	4	5	6
	stare	1	0	1	2	0	1		stare	1	2	2	2	2	2
Pas 6	vârf	1	2	3	4	**5**	6	Pas 12	vârf	1	2	3	4	5	6
	stare	1	0	1	2	1	1		stare	2	2	2	2	2	2

Un exemplu simplu de realizare a procedurii de parcurgere în adâncime pentru un graf cu N vârfuri, descris prin matricea de adiacență, îl constituie funcția DFS de mai jos. Matricea de adiacență a grafului este stocată în tabloul **A**. Marcajele vârfurilor se păstrează în tabloul liniar **B** (`B[i]` – starea vârfului *i*). Inițial toate marcajele vârfurilor sunt nule. Vârful din care este lansată parcurgerea – s.

```
int DFS (int s)
{
 int i;
  b[s]=1;
   for(i=1;i<=n;i++)
        if(a[s][i] !=0 && b[i]==0) DFS(i);
   printf("%d ", s);
   return 0;
}
```

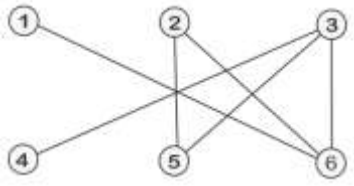
Complexitatea funcției DFS în acest caz este $O(N^2)$. Odată marcat, nodul v nu mai permite relansarea parcurgerii DFS(v), iar numărul maxim de apeluri ale funcției este N. Numărul de operații în corpul funcției este de asemenea proporțional cu N.

Funcția propusă lucrează corect atât pe grafuri neorientate, cât şi pe grafuri orientate.

În procesul de parcurgere, cu cât mai târziu este atins un vârf, cu atât mai repede el va fi cercetat (modelarea prin structuri tip LIFO). Există însă şi probleme în care este important ca vârfurile să fie cercetate în ordinea atingerii (modelarea procesului prin structuri de date FIFO). Pentru rezolvarea lor se poate utiliza o altă metodă de cercetare a grafului – **parcurgerea în lățime**[5].

La fel ca metoda precedentă, parcurgerea în lățime începe de la un nod dat v_0, plasat în într-o structură tip coada (inițial vidă). Se foloseşte principiul de etichetare a vârfurilor, identic celui folosit în parcurgerea în adâncime. În caz general, se extrage din coadă nodul v (la prima iterație $v = v_0$), se determină *toate* vârfurile u noi (care încă nu au fost plasate în coadă, *neatinse*), adiacente v ($\exists (v,u) \in E$), şi se adaugă consecutiv în coadă. La adăugarea în coadă vârfurile se etichetează ca fiind *atinse*. După adăugarea vârfurilor adiacente în coadă, nodul v este marcat *cercetat*. Dacă coada devine vidă - parcurgerea a luat sfârşit. Pentru realizarea algoritmului se vor folosi aceleaşi marcaje aplicate vârfurilor ca şi în cazul parcurgerii în adâncime.

Exemplu: pentru graful din imaginea alăturată se va simula parcurgerea în lățime din vârful 1, în conformitate cu algoritmul descris anterior:



[5] Breadth first search (eng.)

```
Pas 1   atins     1
        cercetat
Pas 2   atins     6
        cercetat  1
Pas 3   atins     2 3
        cercetat  1 6
Pas 4   atins     3 5
        cercetat  1 6 2
Pas 5   atins     5 4
        cercetat  1 6 2 3
Pas 6   atins     4
        cercetat  1 6 2 3 5
Pas 7   atins
        cercetat  1 6 2 3 5 4
```

În funcția BFS de mai jos coada este implementată printr-un tablou unidimensional **B**, începutul ei fiind elementul cu indicele **st**, iar sfârşitul – elementul cu indicele **dr**. Elementele cu indicii **1,...,st-1** formează mulțimea nodurilor cercetate la moment. Structurile de date **A, B, n** au aceleaşi semnificație ca şi în funcția **DFS**. Tabloul C modelează stările curente ale vârfurilor. Funcția BFS realizează parcurgerea în lățime, începând de la vârful cu indicele s.

```
int BFS (int s)
{
  int i, st, dr;
  c[s]=1; b[1]=s; st=1;dr=1;
  while (st<=dr)
  {  for(i=1;i<=n;i++)
      if(a[b[st]][i] !=0 && c[i]==0)
      { dr++; b[dr]=i; c[i]=1; }
   printf("%d ", b[st]);
   st++;
  }
  return 0;
}
```

Funcția BFS este implementată nerecursiv cu o complexitate $O(N^2)$. Numărul de operații în corpul funcției este determinat de două instrucțiuni ciclice incluse, ambele având maxim N repetări.

O realizare practică a parcurgerii în lățime este metoda undei numerice, care se dovedeşte a fi foarte eficientă în cazul problemelor de optimizare pe tablouri bi- şi tridimensionale.

2.2 Grafuri tare conexe

Def. Un graf orientat $G=(V,E)$ se numeşte *tare conex* dacă pentru orice două vârfuri $v_i, v_j \in V$ există cel puțin câte un lanț care uneşte v_i cu v_j ($v_i \mapsto$) şi v_j cu v_i ($v_j \mapsto$). Într-un graf tare conex orice două vârfuri sunt reciproc accesibile.

Def. *Componentă tare conexă* a unui graf orientat $G=(V,E)$ se numeşte mulțimea maximală de vârfuri $V' \subseteq V$: $\forall v_i, v_j \in V'$ există cel puțin câte un lanț $v_i \mapsto$ şi $v_j \mapsto$.

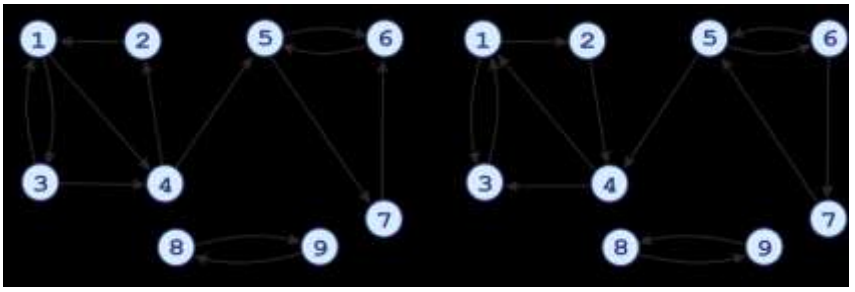
Determinarea componentelor tare conexe

Pentru determinarea componentelor tare conexe va fi folosit *graful transpus* lui *G*. Pentru un graf orientat $G=(V,E)$, graful transpus este $G^T=(V,E^T)$ unde $E^T=\{(v_i,v_j):(v_j,v_i)\in E\}$. Graful transpus are aceleaşi componente tare conexe ca şi graful inițial.

Obținerea grafului transpus $G^T=(V,E^T)$ cere un timp liniar după numărul de arce în graf (sau pătratic față de numărul de vârfuri). Procesul se realizează în mod diferit în dependență de modul de reprezentare a grafului. Fie graful $G=(V,E)$ reprezentat prin matricea de adiacență $A(n\times n)$. Matricea de adiacență a grafului $G^T=(V,E^T)$ se obține conform următoarei formule:

$$A^T_{i,j}=\begin{cases}1 & dacă \quad A_{j,i}=1\\ 0 & în\ caz\ contrar\end{cases} \quad i,j=1,...,n$$

Exemplu: desenul 2.1: Graful inițial $G=(V,E)$ şi graful transpus $G^T=(V,E^T)$, reprezentate grafic.



A **B**

Des. 2.1 Graful iniţial (A) şi graful transpus (B).

Algoritmul pentru determinarea componentelor tare conexe are la bază observația, că componentele tare conexe rămân aceleaşi atât în graful inițial cât şi în cel transpus. Se vor folosi două parcurgeri în adâncime pe grafurile G^T şi G.

Algoritmul Kosaraju

Pseudocod

Pas 1. Se construieşte graful $G^T = (V, E^T)$.

Pas 2. Se lansează căutarea în adâncime pornind de la fiecare vârf necercetat din G^T. Pentru fiecare parcurgere se memorează cumulativ ordinea de cercetare a vârfurilor în vectorul f.

Pas 3. Se lansează căutarea în adâncime pe graful inițial G, consecutiv, pornind de la ultimul vârf inclus în f către primul, după vârfurile necercetate.

Pas 4. La fiecare căutare în adâncime realizată în pasul 3, se afişează vârfurile cercetate – acestea formează o componentă tare conexă.

Exemplu de implementare:

Se foloseşte matricea de adiacenţă **a** pentru reprezentarea grafului iniţial, **at** pentru reprezentarea grafului transpus, vectorul **b** – pentru descrierea stării vârfurilor, vectorul **black** – pentru ordinea de cercetare.

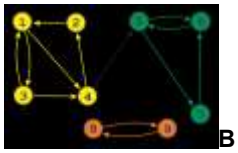
Input: Graful $G=(V,E)$.

Output: componentele tare conexe ale grafului, separate pe linii.

```
int DFS_DIR (int s) { // . . . descrisa anterior - DFS }

int DFS_TRANS (int s)
{  int i;
   b[s]=1;
   for(i=1;i<=n;i++)
      if( at[s][i] !=0 && b[i]==0) DFS_TRANS(i);
   //amplasarea in stiva ordinii de cercetare
   k++; black[k]=s;
  return 0;
}

int main()
{ // . . .citirea datelor initiale
  // transpunerea grafului
 for (i=1;i<=n;i++)
      for (j=1; j<=n; j++)
             if (a[i][j]==1) at[j][i]=1;
 // Cautarea in adancime pe graful transpus
 for(i=1;i<=n; i++) if (b[i]==0) DFS_TRANS(i);
 // resetarea starii varfurilor
 for(i=1;i<=n;i++) b[i]=0; printf("\n");
 // parcurgerea in adancime pe graful initial
 for(i=n;i>=1;i--) //Afisarea componentelor tare conexe
    if (b[black[i]]==0) {DFS_DIR(black[i]); printf("\n");}
 return 0;
}
```

Reprezentarea grafică a rezultatelor:


A

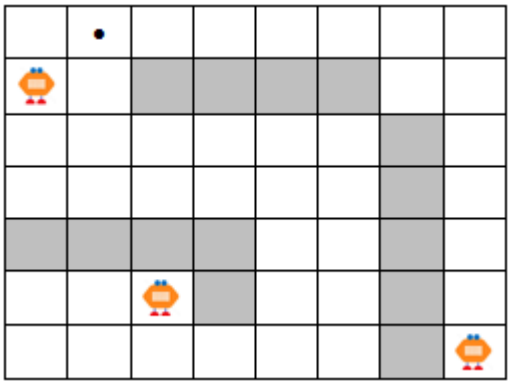

B

Des. 2.2. Graful iniţial (A), Componentele tare conexe ale grafului (fiecare componentă e colorată aparte) (B)

Algoritmul are o complexitate pătratică față de numărul de vârfuri în graf. Pasul 1 al algoritmului necesită un număr de operații proporțional cu $n \times n$. Paşii 2 şi 3 repetă căutări în adâncime, fiecare cu o complexitate $O(N^2)$. Prin urmare, complexitatea totală a algoritmului este $O(N^2)$.

2.3 Probleme rezolvate

Roboţii



Se consideră un câmp dreptunghiular de lucru, divizat în pătrățele (vezi desenul). Pătrățelele libere sunt de culoare albă, iar cele cu obstacole – de culoare închisă. Pe câmpul de lucru, în pătrățele libere, se află mai mulți roboți. Fiecare robot se poate deplasa din pătrățelul curent în pătrățelul vecin doar prin latura comună a acestora. Evident, roboții se pot deplasa doar prin pătrățelele libere.

Inițial, toți roboții sunt în starea de repaus. După sosirea comenzii START, roboții încep să se deplaseze, viteza de deplasare fiind de un pătrățel în fiecare unitate de timp. Fiecare robot se poate opri ori de câte ori el consideră că acest lucru este necesar. Numărul de roboți, care concomitent se pot afla într-un pătrățel liber, nu este limitat.

Sarcină. Scrieți un program, care calculează timpul minim, necesar pentru ca toți roboții se adune în unul din pătrățelele libere ale câmpului de lucru.

Date de intrare. În memoria calculatorului câmpul de lucru este reprezentat printr-un tablou format din numere întregi, cu n linii şi m coloane. Fiecare din elementele tabloului poate lua una din următoarele valori: 0 – pătrățel liber; 1 – pătrățel ce conține un obstacol; 2 – pătrățel liber în care se află un robot.

Fişierul text ROBOT.IN conține pe prima linie numerele întregi n, m separate prin spațiu. Fiecare din următoarele n linii ale fişierului de intrare conține câte m numere întregi separate prin spațiu, care reprezintă pătrățelele respective ale câmpului de lucru.

Date de ieşire. Fişierul text ROBOT.OUT va conține pe o singură linie un număr întreg – timpul minim, necesar pentru ca toți roboții se adune în unul din pătrățelele libere ale câmpului de lucru.

Restricții. $1 \leq n,m \leq 40$. Pe câmpul de lucru se pot afla cel mult 10 roboți. Se garantează că toți roboții se pot aduna în unul din pătrățelele libere ale câmpului de lucru. Timpul de execuție nu va depăşi o secundă. Programul va folosi cel mult 32 Megaocteți de memorie operativă. Fişierul sursă va avea denumirea ROBOT.PAS, ROBOT.C sau ROBOT.CPP.

Exemplu. Pentru desenul din enunțul problemei, fişierele de intrare şi ieşire sunt:

ROBOT.IN

```
7 8
0 0 0 0 0 0 0 0
2 0 1 1 1 1 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
1 1 1 1 0 0 1 0
0 0 2 1 0 0 1 0
0 0 0 0 0 0 1 2
```

ROBOT.OUT

```
12
```

Timpul minim $t = 12$ se obține în cazul în care toți roboții se adună în pătrățelul aflat la intersecția rândului 1 şi a coloanei 2. Pe desenul de mai sus acest pătrățel este marcat printr-un punct.

Rezolvare

Pentru a rezolva problema, vom simula deplasarea fiecărui robot cu ajutorul unei unde numerice. Pentru a nu confunda undele numerice ale fiecăruia din roboți, vom crea câte un câmp de lucru pentru fiecare din ei.

Presupunem că se doreşte simularea undei numerice a unuia din roboți. Vom nota prin k valoarea undei numerice. Evident, în cazul pătrățelului în care se află robotul, $k = 2$. Pentru a propaga unda numerică, înscriem în toate pătrățelele libere, care sunt vecine cu

pătrățelul în care se află robotul, valoarea $k = k + 1 = 3$. În continuare, parcurgem câmpul de lucru şi înscriem în toate pătrățelele libere, vecine cu cele ce conțin valoarea curentă k, valoarea $k + 1$ Repetăm acest proces pentru $k = 3, 4, 5$ ş.a.m.d. până când unda numerică nu mai avansează. Pentru exemplificare, pe desenele de mai jos sunt reprezentate rezultatele propagării undelor numerice pentru fiecare din roboții din enunțul problemei. Din modul de propagare a undei numerice rezultă că valorile înscrise în pătrățelele câmpului de lucru reprezintă lungimea celui mai scurt drum, mai exact numărul de pătrățele minus doi, pe care trebuie să le parcurgă robotul pentru a ajunge din poziția inițială în fiecare din pătrățelele respective.

3	4	5	6	7	8	9	10
2	3					10	11
3	4	5	6	7	8		12
4	5	6	7	8	9		13
				9	10		14
16	15	14		10	11		15
15	14	13	12	11	12		16

De exemplu, pentru a ajunge în pătrățelul cu coordonatele $(1, 1)$, adică în pătrățelul de la intersecția rândului 1 cu coloana 1, primul robot trebuie să parcurgă $3 - 2 = 1$ pătrățel, robotul al doilea trebuie să parcurgă $15 - 2 = 13$ pătrățele, iar robotul al treilea $15 - 2 = 13$ pătrățele. Prin urmare, dacă toți cei trei roboți s-ar aduna în pătrățelul cu coordonatele $(1, 1)$, timpul necesar ar fi egal cu 13. Într-un mod similar, ne convingem, că dacă roboții s-ar aduna în pătrățelul $(1, 8)$, timpul necesar ar fi egal cu 18; în pătrățelul $(7, 1)$ – cu 25 ş.a.m.d.

15	14	13	12	11	10	9	8
16	15					8	7
17	16	17	18	19	20		6
18	17	18	19	20	21		5
				21	22		4
28	27	26		22	23		3
27	26	25	24	23	24		2

15	14	15	16	17	18	19	20
14	13					20	21
13	12	11	10	9	10		22
12	11	10	9	8	9		23
				7	8		24
4	3	2		6	7		25
5	4	3	4	5	6		26

Evident, timpul cerut în enunțul problemei poate fi găsit prin parcurgerea câmpurilor în care sunt înscrise valorile undelor numerice ale fiecăruia din roboți, selectarea pentru fiecare din pătrățelele omonime a valorilor maximale şi alegerea din valorile astfel calculate a valorii minimale.

În cazul exemplului de mai sus, valoarea minimală este obținută în cazul pătrățelului (1,2), pentru care valorile undelor numerice sunt 4, 14 şi 14, timpul respectiv fiind egal cu 12.

În programul ce urmează, pentru a evita verificările de la margini, câmpul de lucru al fiecărui robot este încadrat într-un chenar format din obstacole.

```
Program Robotii;
{ Clasele 10-12 }
const nmax=40; mmax=40; rmax=10;
type CampDeLucru = array[0..nmax+1, 0..mmax+1] of integer;
var C : array[1..rmax] of CampDeLucru;
    n, m, r : integer;
    t : longint;

procedure Citeste;
var i, j : integer;
    Intrare : text;
begin
  assign(Intrare, 'ROBOT.IN');
  reset(Intrare);
  readln(Intrare, n, m);
  for i:=1 to n do
    begin
      for j:=1 to m-1 do read(Intrare, C[1][i, j]);
      readln(Intrare, C[1][i, m]);
    end;
  close(Intrare);
  { incadram campul intr-un chenar din obstacole }
  for j:=0 to m+1 do C[1][0, j]:=1;
  for j:=0 to m+1 do C[1][n+1, j]:=1;
  for i:=0 to n+1 do C[1][i, 0]:=1;
  for i:=0 to n+1 do C[1][i, m+1]:=1;
end; { Citeste }

procedure Scrie;
var Iesire : text;
begin
  assign(Iesire, 'ROBOT.OUT');
  rewrite(Iesire);
  writeln(Iesire, t);
  close(Iesire);
end; { Scrie }
```

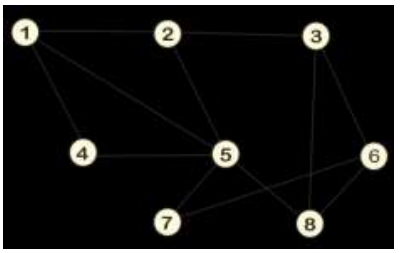
```
procedure NumaraRobotii;
{ Inscrie in r numarul de roboti }
var i, j : integer;
begin
  r:=0;
  for i:=1 to n do
    for j:=1 to m do
      if C[1][i, j]=2 then r:=r+1;
end; { NumaraRobotii }

procedure InitializeazaCampurile;
var i, j, p, q : integer;
begin
  { cream r exemplare ale campului de lucru }
  for p:=2 to r do C[p]:=C[1];
  { lasam pe fiecare camp cate un singur robot }
  for p:=1 to r do
    begin
      q:=0;
      for i:=1 to n do
        for j:=1 to m do
          if (C[p][i, j]=2) then
            begin
              q:=q+1;
              if (p<>q) then C[p][i, j]:=0;
            end;
    end;
end; { InitializeazaCampurile }

procedure PropagaUndaNumerica(var A : CampDeLucru);
{ Propaga unda numerica in campul A }
var i, j : integer;
    k : integer;   { valoarea curenta a undei numerice }
    Ind : boolean; { indicator }
begin
  k:=1;
  repeat
    k:=k+1;
    Ind:=true;
    for i:=1 to n do
      for j:=1 to m do
        if A[i, j]=k then
              begin
                if A[i-1, j]=0 then
                  begin A[i-1, j]:=k+1; Ind:=false; end;
                if A[i+1, j]=0 then
                  begin A[i+1, j]:=k+1; Ind:=false; end;
```

```
                if A[i, j-1]=0 then
                  begin A[i, j-1]:=k+1; Ind:=false; end;
                if A[i, j+1]=0 then
                  begin A[i, j+1]:=k+1; Ind:=false; end;
              end;
      until Ind;
end; { PropagaUndaNumerica }

procedure CalculeazaTimpulMinim;
{ Calculeaza timpul minim }
var i, j, p, k : integer;
    Ind : boolean; { indicator }
begin
  for p:=1 to r do
    PropagaUndaNumerica(C[p]);
  t:=MaxInt;
  for i:=1 to n do
    for j:=1 to m do
      if C[1][i, j]>1 then
        begin
          k:=2;
          for p:=1 to r do
            if C[p][i, j]>k then k:=C[p][i, j];
          if k<t then t:=k;
        end;
   t:=t-2;
end; { CalculeazaTimpulMinim }

begin
  Citeste;
  NumaraRobotii;
  InitializeazaCampurile;
  CalculeazaTimpulMinim;
  Scrie;
end.
```

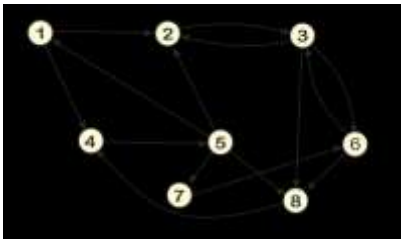
Din analiza textelor procedurilor din programul de mai sus rezultă că cel mai mare număr de operații se efectuează în cazul propagării undei numerice. Astfel, procedura PropagaUndaNumerica conține trei cicluri imbricate: **repeat** ... **for** ... **for** ... . Instrucțiunile **if** din componența acestor cicluri vor fi efectuate de cel mult knm ori, unde k este lungimea celui mai lung drum de pe câmpul de lucru. Întrucât $k < nm$, timpul cerut de această procedură va fi proporțional cu $(nm)^2$. În procedura CalculeazaTimpulMinim se efectuează r apeluri ale

procedurii PropagaUndaNumerica. Prin urmare, timpul cerut de program va fi proporțional cu $r(nm)^2$. Conform restricțiilor problemei, $r \leq 10$, iar $n, m \leq 40$. Evident, timpul cerut de programul Robotii va fi proporțional cu $10 \cdot (40 \cdot 40)^2 = 2{,}56 \cdot 10^7$, mărime mai mică decât capacitatea de prelucrare a calculatoarelor personale.

Exerciţii:



1. Simulați, pe paşi, pentru graful din imagine, parcurgerea în lățime, pornind de la vârful 1.



2. Simulați, pe paşi, pentru graful din imagine, parcurgerea în adâncime, pornind de la vârful 1.

3. Elaborați un program pentru parcurgerea în adâncime a unui graf ($|V| \leq 20$) (abordare recursivă).

4. Elaborați un program pentru parcurgerea în adâncime a unui graf ($|V| \leq 20$) (abordare iterativă).

5. Elaborați un program pentru parcurgerea în lățime a unui graf ($|V| \leq 20$).

6. Elaborați un program pentru determinarea componentelor tare conexe ale grafului ($|V| \leq 20$).

Capitolul 3. Mulțimi independente

În acest capitol

- Noţiunea de mulţime independentă
- Mulţimi maximal independente.
- Generarea mulţimilor maximal independente

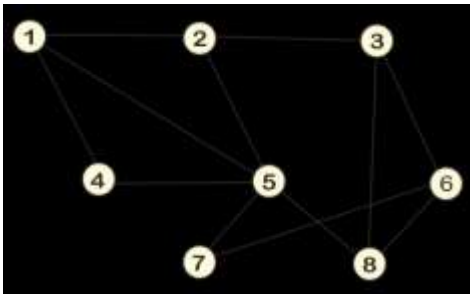
3.1 Mulţimi independente

Fie dat un graf neorientat $G=(V,E)$.

Def. Mulțime *independentă* se numeşte o mulțime de vârfuri ale grafului, astfel încât oricare două vârfuri din ea nu sunt unite direct prin muchie.

Formulată matematic, mulțimea independentă S este o mulțime, care satisface relația: $S \subseteq V : S \bigcap \quad \varnothing$.

Def. Mulțimea S se numeşte maximal independentă, dacă nu există o altă mulțime independentă S', pentru care se îndeplineşte condiția: $S \subseteq S'$.



Des. 3.1. {1, 3,7} {2,8,7,4} {4,6} – mulţimi independente. Mulţimile {2,8,7,4} , {2, 4, 6} sunt maximal independente, iar {1,3}, {2,4} – nu.

Nu toate mulțimile maximal independente au acelaşi număr de vârfuri. Prin urmare modul de selecție a celei *mai bune* mulțimi maximal independente va depinde de condițiile inițiale ale problemei concrete: în unele cazuri aceasta va fi mulțimea maximal independentă cu un număr maxim de vârfuri, în altele - mulțimea maximal independentă cu un număr minim de vârfuri, etc.

Def. Fie Q mulțimea tuturor mulțimilor independente a grafului $G=(V,E)$. Numărul $\alpha[G]=\max_{S\in Q}|Q|$ se va numi *număr de independență* a grafului G, iar mulțimea S^*, pentru care el se obține – *mulțime maximă independentă*.

Exemplu: pentru graful din desenul 3.1 setul de mulțimi maximale independente este {1, 3, 7}, {1, 6}, {1, 7, 8}, {2, 4, 6}, {2, 4, 7, 8}, {3, 4, 7}, {3, 5}, {5, 6}. Cea mai mare putere a mulțimilor este 4, deci α [G] = 4. Mulțimea maximă independentă este {2, 4, 7, 8}

3.2 Generarea tuturor mulţimilor maximal independente

Fie $G=(V,E)$. Prin *graf complementar* se înțelege graful $\tilde{G}$ unde $\tilde{I}$ $: u,v\in V;(u,v)\notin E\}$

Problema mulțimilor maximal independente se reduce direct la problema generării subgrafurilor complete, rezolvată pe graful complementar $\tilde{G}$. Aceasta din urmă este o problemă de complexitate exponențială. De aici rezultă şi complexitatea exponențială a algoritmului pentru determinarea tuturor mulțimilor maximal independente. Tehnica generală se bazează pe metoda reluării, care poate fi parțial optimizată prin ordonarea vârfurilor după micşorarea puterii acestora. Algoritmul de parcurgere sistematică a fost propus de Bron şi Carbosh. În acest algoritm generarea repetată a mulțimilor este evitată prin îmbunătățirea mulțimilor existente.

Motivarea algoritmului

Algoritmul se bazează pe arborele de căutare. Prin urmare, este eficientă realizarea lui recursivă.

În general, la pasul k mulțimea independentă S_k se extinde prin adăugarea unui vârf nou, pentru a obține la pasul $k+1$ mulțimea S_{k+1}. Procesul se repetă atât timp, cât este posibilă adăugarea vârfurilor noi. La momentul, în care nu mai putem adăuga vârfuri, avem obținută o mulțime maximal independentă.

Fie Q_k - mulțimea maximală de vârfuri, pentru care la pasul k avem $\Gamma(S_k)\bigcap$. Prin adăugarea unui vârf din Q_k în S_k se obține S_{k+1}. În general, Q_k este formată din două componente: Q_k^- – vârfurile deja folosite în procesul de căutare pentru extinderea S_k şi Q_k^+ – vârfurile care încă nu au fost folosite în căutare. Pentru adăugarea în S_k se vor folosi doar vârfurile din Q_k^+. Astfel, procedura de adăugare a vârfului nou e următoarea:

a) selectarea unui nod $x_{i_k} \in Q_k^+$ (*)

b) construirea mulțimii $S_{k+1} = S_k \bigcup$

c) formarea
$$Q_{k+1}^- = Q_k^- - \Gamma(x_{i_k})$$
$$Q_{k+1}^+ = Q_k^+ - \{\Gamma(x_{i_k}) \bigcup$$

Pasul de întoarcere presupune eliminarea vârfului x_{i_k} din S_{k+1} pentru revenirea la mulțimea S_k cu mutarea lui x_{i_k} din Q_k^+ în Q_k^-.

Soluția (mulțimea maximal independentă) se va obține în cazul în care $Q_k^+ = \varnothing$ şi $Q_k^- = \varnothing$. Dacă $Q_k^- \neq \varnothing$, rezultă că mulțimea S_k a fost extinsă la o etapă precedentă din contul adăugării unui vârf din Q_k^-, de aceea nu este o mulțime maximal independentă nouă.

Prezența în Q_k^- a unui vârf $x \in Q_k^-$: $\Gamma(x) \bigcap$ (**)
este un indicator pentru a efectua pasul de întoarcere, deoarece în acest caz Q_k^- nu va deveni vidă, indiferent de modul de selecție a vârfurilor.

Optimizarea algoritmului

Optimizarea poate fi obținută din contul apariției cât mai rapide a pasului de întoarcere. Prin urmare, se va încerca îndeplinirea cât mai grabnică a condiției (**). O metodă sigură (în special pentru grafuri mari) este de a alege mai întâi în Q_k^- un vârf x', pentru care valoarea $\Delta(x') = \left|\Gamma(x') \bigcap \right.$ va fi minimală, apoi, la fiecare pas următor în alegerea unui x_{i_k} din Q_k^+ : $x_{i_k} \in \Gamma(x')$. Aceasta asigură apropierea cu o unitate de situația care generează pasul de întoarcere.

Pseudocod

Pas 1. $S_0 \leftarrow Q_0^- \leftarrow \varnothing, \quad Q_0^+ \leftarrow V, \quad k \leftarrow 0.$

Adăugarea vârfurilor

Pas 2. Este selectat un vârf $x_{i_k} \in Q_k^+$, după principiul formulat anterior. Se formează $S_{k+1}, Q_{k+1}^+, Q_{k+1}^-$ fără a modifica Q_k^+, Q_k^-. $k \uparrow$

Verificarea posibilității de continuare

Pas 3. Dacă există $x \in Q_k^-$: $\Gamma(x) \bigcap$ se trece la pasul 5, altfel - la pasul 4.

Pas 4.

a) Dacă $Q_k^+ = \varnothing$ şi $Q_k^- = \varnothing$ se afişează (memorează) mulțimea maximal independentă S_k şi se trece la pasul 5.

b) Dacă $Q_k^+ = \varnothing$, dar $Q_k^- \neq \varnothing$ se trece direct la pasul 5

c) Dacă $Q_k^+ \neq \varnothing$, şi $Q_k^- \neq \varnothing$ se revine la pasul 2

Mişcarea înapoi

Pas 5. $k \downarrow$

a) x_{i_k} se exclude din S_{k+1}, pentru a reveni la S_k.

b) Se reconstruiesc Q_k^+, Q_k^-: $Q_k^+ \leftarrow Q_k^+ - \{x_{i_k}\}$ $Q_k^- \leftarrow Q_k^- + \{x_{i_k}\}$

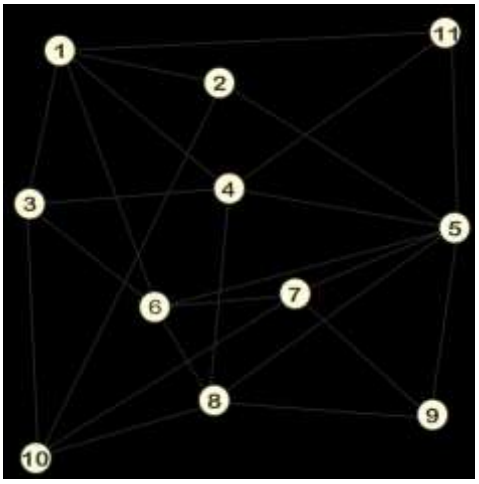
c) Dacă $k = 0$ şi $Q_k^+ = \varnothing$, atunci SFÂRŞIT (au fost afişate [memorate] toate mulțimile independente maximale). În caz contrar se revine la pasul 3.

Implementare. În următorul exemplu este realizată o implementare simplificată a algoritmului, fără optimizarea prin reordonarea vârfurilor. Generarea repetată a mulțimilor maximal independente este evitată prin selectarea vârfurilor pentru includere în ordine lexicografică. Mulțimile Q_k^+, Q_k^-, S_k se formează şi se gestionează prin intermediul apelurilor recursive. Restricțiile de dimensiune $|V| \le$ **num_el**. Structurile de date: **a** – matricea de adiacență a grafului, **s** – tabloul etichetelor vârfurilor, incluse în soluția curentă (**s[i]=1**), **q** – tabloul etichetelor vârfurilor care pot fi adăugate la soluție (**q[i]=1**), **rest** – tabloul etichetelor pentru restabilirea Q_k^+ .

```
int fillc(int *x, int z, int num)
{ … // elementele cu tabloului x primesc valoarea z}

int readdata()
{ … // citirea matricei de adiacenţă a grafului A}

int print()
{ … // funcţia pentru afişarea mulţimii maximal independente
curente}

int mind(int *s, int *q)
{ int i,j,k=0,r, rest[num_el];
   for (i=1;i<=n;i++) if(q[i]!=0) k=1;
   if (k==0) {print();} // afişarea soluţiei
   else { j=n;
          while (s[j]==0 && j>=1) j--;
```

```
            r=j+1;
            for (i=r;i<=n;i++)
             if (q[i]==1)
                { fillc(rest,0,n);
                  s[i]=1; q[i]=0;
                  for (j=1;j<=n;j++)
                   if (a[i][j] != 0 && q[j]==1)
                       {q[j]=0; rest[j]=1;}
                  mind (s,q);    // mişcarea inainte
                  s[i]=0;q[i]=1; //mişcarea înapoi
                  for (j=1;j<=n;j++)
                      if(rest[j]==1) q[j]=1;
         }    }
    return 0;
}
int main()
{   readdata();
    fillc(s,0,n); fillc(q,1,n);
    mind(s,q); return 0;
}
```

Pentru graful reprezentat pe desenul 3.2, programul determină următoarele mulțimi maximal independente

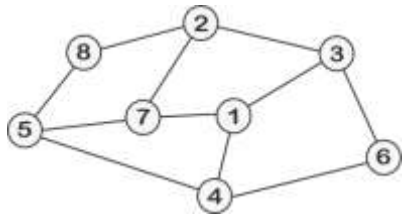



```
1 5 10
1 7 8
1 8 10
1 9 10
2 3 7 8 11
2 3 9 11
2 4 6 9
2 4 7
2 6 9 11
3 5
4 6 9 10
6 9 10 11
8 10 11
```

Des. 3.2 Graful inițial şi mulţimile independente identificate.

Exerciţii

1. Pentru graful din imagine, determinați:
   a) mulțimea maxim independentă de putere minimă;
   b) mulțimea maxim independentă de putere maximă.



2. Completați implementarea prezentată mai sus cu subprogramele lipsă şi structurile de date necesare pentru a avea un program funcțional, ce va realiza algoritmul pentru grafuri cu $|V| < 30$.

3. Realizați o modificare a programului elaborat în exercițiul 2, care va verifica şi toate optimizările indicate în descrierea algoritmului (3.2).

4. Elaborați o funcție pentru generarea aleatorie de grafuri şi estimați statistic timpul mediu de lucru al algoritmilor elaborați ăn exercițiile de mai sus în funcție de numărul de vârfuri şi muchii ale grafului.

Capitolul 4. Colorări

În acest capitol:

- Numărul cromatic al grafului
- Algoritmi exacţi de colorare a grafurilor
- Algoritmi euristici de colorare

4.1 Numărul cromatic

Colorarea grafului – procesul de atribuire unor caracteristici coloristice vârfurilor acestuia.

Graful se numeşte *r*-cromatic, dacă vârfurile lui pot fi colorate, folosind *r* culori diferite, astfel încât să nu existe două vârfuri adiacente, colorate la fel. Valoarea minimă a lui *r*, pentru care o asemenea colorare este posibilă se numeşte *număr cromatic* al grafului $G=(V,E)$ şi se notează $\gamma(G)$.

Nu există o formulă generală pentru determinarea $\gamma(G)$ reieşind din numărul de vârfuri (n) şi muchii (m) ale grafului. Este evidentă posibilitatea de a colora graful în k culori ($\forall\ \gamma(G)\le k\le n$). Totuşi, reieşind din faptul că vârfurile colorate formează mulțimi independente, pot fi stabilite anumite numere cromatice:

$\gamma(\overline{K}_n)=1,\ \gamma(K_n)=n,\ \gamma(K_{n_1,n_2})=2,\ \gamma(T)=2$, etc.

Într-un graf colorat, mulțimea vârfurilor, cărora le este atribuită una şi aceeaşi culoare se numeşte *clasă monocromă*.

Teorema 1. $\gamma(G)\le 1+\Delta(G)$.[6]

□ Demonstrație: [12, pag. 39] ■

Teorema 2. Fie $\gamma=\gamma(G),\quad \overline{\gamma}=\gamma(\overline{G})$. Atunci sunt adevărate relațiile:

[6] $\Delta(G)$ - puterea maximă a vârfurilor din G.

$$2\sqrt{n} \le \gamma + \bar{\gamma} \le n+1$$
$$n \le \gamma\bar{\gamma} \le \left(\frac{n+1}{2}\right)^2$$

□ Fără demonstrație ■

Teorema 3. În orice graf planar există un vârf de grad nu mai mare decât cinci. (corolar din formula Euler)
□ [12, pag. 40] ■

Teorema 4. (celor 5 culori). Pentru orice graf planar $\gamma(G) \le 5$.
□ [4, pag. 285] ■
Ipoteza (celor 4 culori). Orice graf planar poate fi colorat cu patru culori.

4.2. Algoritmul exact de colorare

Abordare recursivă, pseudocod.
Input: Graful G.
Output: Toate colorările posibile ale grafului G.

Pas 0. $k \leftarrow 0$ (indicele culorii curente).
Pas 1. În graful G se alege o careva mulțime maximal independentă S.
Pas 2. $k \uparrow$. Mulțimea S se colorează în culoarea k.
Pas 3. $G \leftarrow G - S$. Dacă $G \neq \varnothing$ se revine la pasul 1.

Algoritmul generează toate posibilitățile de formare a mulțimilor independente disjuncte pe vârfurile grafului G. Prin urmare, va fi generată şi o colorare optimă cu număr minim de culori. Complexitatea algoritmului este una exponențială – pasul 1 are o asemenea complexitate, deoarece generează mulțimi maximal independente. Suplimentar, se formează şi un arbore de soluții, numărul de noduri în care, în general, este proporțional cu 2^n.

Cele expuse denotă ineficiența algoritmului exact pentru grafurile cu un număr de vârfuri mai mare decât 20.

4.3. Algoritmi euristici de colorare

Algoritmul exact, descris anterior nu poate fi utilizat pentru a obține o colorare eficientă într-un timp rezonabil. Problema poate fi rezolvată cu ajutorul unor algoritmi euristici, care vor furniza soluții ce pot diferi de cea optimă, fiind totuşi, destul de apropiate de ea.

Algoritmul de colorare consecutivă

Input: Graful G.
Output: O colorare posibilă a grafului G.

Pas 1. Se sortează vârfurile din G în ordinea descreşterii gradelor d.

Pas 2. Inițializare $V \leftarrow \{1,...,n\}$ $C[\,] \leftarrow 0$, $k \leftarrow 1$. Vectorul culorilor se inițializează cu 0. Culoarea activă – 1.

Pas 3. Cât $V \neq \varnothing$ se repetă

Pentru fiecare $v \in V$

  Pentru fiecare $u \in \Gamma^+(v)$

    Dacă $C[u] = k$, se trece la următorul v

  $C[v] \leftarrow k$; $V \leftarrow V - \{v\}$

$k \uparrow$

Implementare

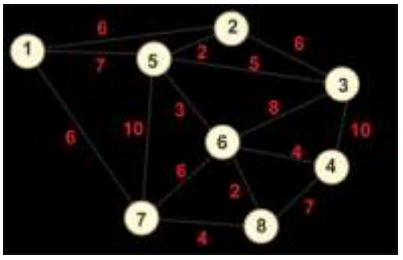
Input: Graful G: matricea de adiacență **a**, structura vârfurilor **v**.
Output: O colorare posibilă a grafului G, în vectorul culorilor **c**.

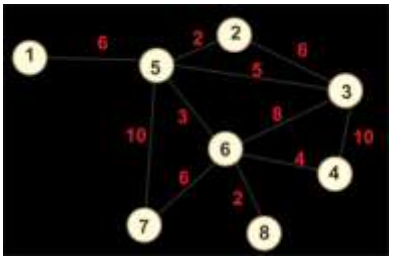
```
int sort ()
{ … // sortează vârfurile în descreşterea gradelor }

int readdata()
{ … // citeşte graful G. – matricea de adiacenţă }
```

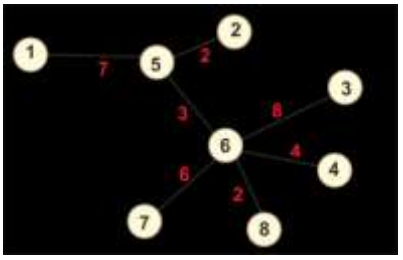
```
int printcc()
{ … // afişează o colorare a grafului G }

int nocolor()
{ verifică prezenţa vârfurilor necolorate }

int main(int argc, char *argv[])
{ int j,i,r;
  readdata();
// initializare
  for (i=1;i<=n;i++)
   {v[i].ind=i;
    for (j=1; j<=n;j++) v[i].grad+=a[i][j];
   }
// sortare
   sort();
// colorare
k=1;
fillc(c,0,n);
while (nocolor())
{  for( i=1;i<=n;i++)
  { if (c[v[i].ind]==0)
      { r=0;
        for (j=1;j<=n;j++)
            if (a[v[i].ind][j] != 0 && c[j]==k) r++;
        if (r==0)  c[v[i].ind]=k;
      }
   }
   k++;
}
printcc();
return 0;
}
```

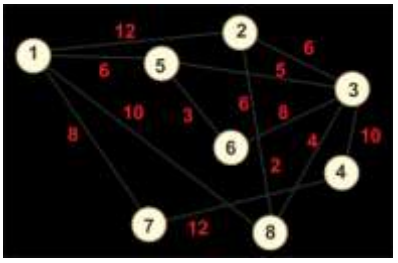
Exerciţii:

1. Pentru grafurile din imaginile de mai jos determinați $\gamma(G)$:









2. Elaborați un program care va determina colorarea cu un număr minim de culori pentru grafuri cu $|V| \le 10$. Implementați algoritmul exact de colorare.

3. Elaborați un program care va determina o colorarea aproximativă pentru grafuri cu $|V| \le 100$, folosind algoritmul euristic de colorare.

4. Identificați grafuri, pentru care algoritmul euristic, descris anterior, va construi soluții ce diferă de cele optime.

Capitolul 5. Drumuri minime în graf

În acest capitol:

- Drumul minim între două vârfuri ale grafului
- Drumul minim de la un vârf la toate vârfurile grafului (algoritmul Dijkstra)
- Drumul minim între toate perechile de vârfuri (algoritmul Floyd)

Preliminarii

Ponderea unei muchii (u,v) în graf este o caracteristică numerică a relației între vârfurile u şi v. Ea poate specifica distanța dintre vârfuri, sau capacitatea unui canal de transmitere a datelor, a unei conducte, magistrale auto etc. Atribuirea de ponderi muchiilor unui graf permite formularea unei serii noi de probleme, inclusiv a problemelor de optimizare. Una din astfel de probleme este problema drumurilor minime.

Problema drumurilor minime într-un graf arbitrar $G=(V,E)$, muchiile căruia au ponderi nenegative, descrise de matricea costurilor $C=C[i,j]$, constă în găsirea unui drum de lungime minimă ce leagă vârfurile date *s* şi *t* ale grafului, cu condiția că un astfel de drum există. Pentru problema dată există mai multe formulări. Iată doar câteva din ele:

- să se determine distanța minimă între două vârfuri ale grafului (dacă între ele există un drum);
- să se determine distanța minimă de la un vârf al grafului la toate celelalte vârfuri;
- să se determine distanțele minimale între toate perechile de vârfuri.

5.1 Distanţa minimă între două vârfuri. Algoritmul Dijkstra

Algoritmul Dijkstra se bazează pe aplicarea marcajelor pentru vârfurile în care se poate ajunge dintr-un vârf dat. Inițial marcajele au valori temporare suficient de mari, care pe parcursul repetării iterațiilor se micşorează, până la atingerea valorilor minime. La fiecare iterație a algoritmului, exact un dintre marcaje devine permanent, adică indică drumul minim până la unul dintre vârfuri. Prin urmare algoritmul va conține cel mult n iterații.

Fie s vârful de la care se calculează distanțele minime, t – vârful până la care se calculează distanța. Marcajul vârfului v_i va fi notat prin $l(v_i)$. Se va folosi un marcaj cu două componente (spre deosebire de algoritmul clasic Dijkstra). Componentele vor conține a) valoarea distanței curente de la s la v_i şi b) indicele vârfului v_j din care se ajunge în v_i.

Pseudocod

Pas 1. Se consideră $l(s) \leftarrow (0, s)$ – marcaj temporar pentru vârful s. Pentru toate vârfurile $v_i \in V, v_i \neq s$ se consideră marcajele nedefinite. Vârful activ $p \leftarrow s$.

Pas 2. (*Recalcularea marcajelor*)
Pentru toate vârfurile $v_i \in \Gamma(p)$ care nu au marcaje permanente, se recalculează componentele distanță ale marcajelor în corespundere cu formula:
$l(v_i).dist \leftarrow \min\{l(v_i).dist, l(p).dist + c[p][v_i]\}$.

Dacă $l(v_i).dist > l(p).dist + c[p][v_i]$ atunci se modifică şi componenta marcajului, care indică sursa $l(v_i).sursa \leftarrow p$.

Vârfului p i se atribuie marcaj permanent.

Pas 3. (*Determinarea vârfului pentru cercetare*)

Între toate vârfurile cu marcaje temporare se determină cel cu marcaj minim pentru componenta distanță:

$l(v_i^*).dist = \min l(v_i).dist$

Se consideră $p \leftarrow v_i^*$

Pas 4. (*Repetarea iterației*)

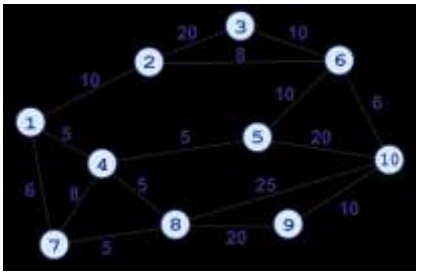
Dacă $p = t$ atunci marcajul $l(t).dist$ indică costul minim a drumului di *s* în *t*. Pentru restabilirea drumului minim se trece la pasul 5. În caz contrar $p \neq t$ se revine la pasul 2.

Pas 5. Pentru restabilirea traiectoriei se foloseşte a doua componentă a marcajului: se consideră $x = t$.

Se include în traiectorie muchia $(x, l(x).sursa)$. Se consideră $x \leftarrow l(x).sursa$ Procesul se repetă cât timp $x \neq s$.

Exemplu:

Se consideră graful $G = (V, E)$ de pe desenul 5.2.



Des. 5.2

Se cere să se determine distanța minimă de la vârful 1 la vârful 10.

Inițializarea

Vârf	1	2	3	4	5	6	7	8	9	10
distanţa	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
sursa	1									
marcaj	p									

Iterația 1

Vârf	1	2	3	4	5	6	7	8	9	10
distanţa	0	10	∞	5	∞	∞	6	∞	∞	∞
sursa	1	1		1			1			
marcaj	p	t		t*			t			

Iterația 2

Vârf	1	2	3	4	5	6	7	8	9	10
distanţa	0	10	∞	5	10	∞	6	10	∞	∞
sursa	1	1		1	4		1	4		
marcaj	p	t		p	t		t*	t		

Iterația 3

Vârf	1	2	3	4	5	6	7	8	9	10
distanţa	0	10	∞	5	10	∞	6	10	∞	∞
sursa	1	1		1	4		1	4		
marcaj	p	t*		p	t		p	t		

Iterația 4

Vârf	1	2	3	4	5	6	7	8	9	10
distanţa	0	10	30	5	10	18	6	10	∞	∞
sursa	1	1	2	1	4	2	1	4		
marcaj	p	p	t	p	t*	t	p	t		

Iterația 5

Vârf	1	2	3	4	5	6	7	8	9	10
distanţa	0	10	30	5	10	18	6	10	∞	30
sursa	1	1	2	1	4	2	1	4		5
marcaj	p	p	t	p	p	t	p	t*		t

Iterația 6

Vârf	1	2	3	4	5	6	7	8	9	10
distanţa	0	10	30	5	10	18	6	10	30	30
sursa	1	1	2	1	4	2	1	4	8	5
marcaj	p	p	t	p	p	t*	p	p	t	t

Iterația 7

Vârf	1	2	3	4	5	6	7	8	9	10
distanţa	0	10	30	5	10	18	6	10	30	24
sursa	1	1	2	1	4	2	1	4	8	6
marcaj	p	p	t	p	p	p	p	p	t	t*

Lungimea drumului minim din vârful 1 în 10 este 24. Traseul va fi determinat de calea: 10, 6, 2, 1

Pentru a determina distanțele minime de la un vârf dat până la toate celelalte vârfuri ale grafului (a componentei conexe) algoritmul va fi oprit în cazul când toate vârfurile au marcaje permanente.

Implementare

Graful este descris prin matricea de adiacență **a**. Marcajele se păstrează în tabloul liniar **vert** cu elemente tip articol, având componentele *distanța, sursa, stare*.

```
int find()
{ … // fuctia returneaza indicele varfului cu marcaj temporar
    //de valoare minima. Daca nu exista, returneaza 0.
}

int tipar()
{ … // functia afiseaza tabloul de marcaje
}

int main()
{… // citire date
 … // formare marcaje
 for (i=1;i<=n;i++)
        for (j=i+1; j<=n; j++) k+=a[i][j];
 k++;
 for (i=1;i<=n;i++)
      {vert[i].dist=k; vert[i].sursa=i; vert[i].stare=0;}
 vert[s].stare=1; vert[s].dist=0;
// repetare iteratii
 while (find ())
 { p=find();
```

```
   for(i=1;i<=n;i++)
    if (a[p][i] !=0 && vert[i].stare !=2 )
     { dc= vert[p].dist+a[p][i];
       if(dc<vert[i].dist){vert[i].dist=dc; vert[i].sursa=p;}
       vert[i].stare=1;
     }
   vert[p].stare=2;
 }
// afisare rezultate
tipar();
return 0;
}
```

5.2 Distanţa minimă între toate vârfurile. Algoritmul Floyd

Din paragraful precedent rezultă o soluție evidentă a problemei determinării tuturor drumurilor minime între vârfurile grafului: algoritmul Dijkstra se lansează având în calitate de sursă, consecutiv, toate vârfurile grafului. Eistă şi un algoritm mai eficient, propus de Floyd [7, p. 480]. La baza algoritmului este ideea unei secvențe din n transformări consecutive ale matricei distanțelor C. La transformarea cu indicele k matricea conține lungimea drumului minim între orice pereche de vârfuri, cu restricția că pentru orice două vârfuri v_i, v_j drumul minim dintre ele trece doar prin vârfurile mulțimii $\{v_1, v_2, ..., v_k\}$

Pseudocod

Pas 0. (Preprocesare). Fie dată matricea distanțelor C, în care

$$C[i][j] = \begin{cases} 0, & i = j \\ d_{i,j}, & \exists (i,j) \in E \\ \infty, & \overline{\exists}(i,j) \in E \end{cases}$$

Pas 1. (Inițializare) $k \leftarrow 0$

Pas 2 $k \uparrow$

Pas 3 Pentru toți $i \neq k : C[i][k] \neq \infty$, şi pentru toți $j \neq k : C[k][j] \neq \infty$ se calculează $C[i][j] = \min\left[C[i][j], C[i][k] + C[k][j]\right]$

Pas 4 dacă $k = n$ – sfârşit, în caz contrar se revene la pasul 2.

Dacă problema rezolvată necesită şi restabilirea traseelor care formează drumurile minime, este formată o matrice auxiliară *T*, care inițial are elementele liniei *i* egale cu *i*.

Mai apoi, la transformarea elementului $C[i][j]$ din matricea distanțelor se transformă şi elementul respectiv din matricea *T*:

$T[i][j] = T[k][j]$ dacă $C[i][j] > C[i][k] + C[k][j]$.

Traseul ce corespunde drumului minim între vârfurile v_i, v_j va fi format din secvența $v_i, v_1, v_2, ..., v_r, v_{r+1}, v_j$, unde

$v_{r+1} = T[i][j], \quad v_r = T[i][v_{r+1}], \quad v_{r-1} = T[i][v_r], ...$

Implementare

Graful este descris prin matricea de adiacență **a**, care ulterior este transformată în matricea distanțelor.

```
int main()
{
… // citire date inițiale
// modelare infinit
 for (i=1;i<=n;i++)
        for (j=1; j<=n; j++)
              p+=a[i][j];
 p++;

// creare matrice costuri
 for (i=1;i<=n;i++)
        for (j=1; j<=n; j++)
         if (a[i][j]==0 && i != j) a[i][j]=p;
// calculare drumuri minime
for (k=1;k<=n;k++)
 for (i=1;i<=n;i++)
  if (i!=k && a[i][k]!=p)
```

```
      for (j=1;j<=n;j++)
          if (j!=k && a[k][j] !=p)
             if (a[i][j]>a[i][k]+a[k][j])
                 a[i][j]=a[i][k]+a[k][j];
… // afisare rezultate
return 0;
}
```

5.3 Probleme rezolvate

Reţele de calculatoare[7]

Enunț
:

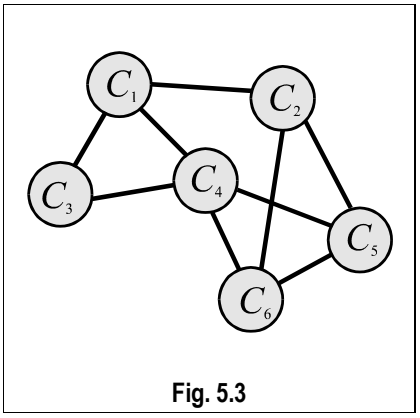


Fig. 5.3

Se consideră n calculatoare C_1, C_2, ,...., C_n, reunite într-o rețea de topologie distribuită (*Fig. 5.3*).

Structura de comunicație a rețelei este formată din linii bidirecționale de transmisie a informației. Fiecare linie asigură comunicarea directă între perechile de calculatoare adiacente.

În absența unei linii directe între calculatoarele C_a, C_b $(a = 1, 2, ..., n;\ b = 1, 2, ..., n;\ a \neq b)$, schimbul de informații între ele se realizează prin intermediul altor calculatoare, cu ajutorul cărora se formează canale de comunicare. Mai exact, un canal de comunicare între calculatoarele C_a, C_b reprezintă o succesiune de calculatoare distincte (C_a, C_k, C_l, ..., C_m, C_b), cu proprietatea că între calculatoarele vecine (C_a, C_k), (C_k, C_l), ..., (C_m, C_b) există cîte o linie directă de transmisie a informației.

Prin definiție, lungimea canalului de comunicare (C_a, C_k, C_l, ..., C_m, C_b) este egală cu numărul liniilor directe de transmisie a informației (C_a, C_k), (C_k, C_l), ..., (C_m, C_b) ce formează canalul respectiv.

În cazul topologiilor distribuite, între calculatoarele C_a, C_b pot exista mai multe canale de comunicare. Elaborați un program care

[7] Olimpiada Republicană la Informatică, 2006

determină lungimea celui mai scurt canal de comunicare între calculatoarele C_a, C_b.

Input:
Fişierul text RETELE.IN conține pe prima linie numărul întreg n. Fiecare din următoarele n linii ale fişierului de intrare conține numere întregi separate prin spațiu. Linia i+1 a fişierului de intrare conține numerele întregi ce reprezintă calculatoarele care sînt legate prin linii directe cu calculatorul C_i. Ultima linie a fişierului de intrare conține numerele naturale C_a, C_b separate prin spațiu.

Output:
Fişierul text RETELE.OUT va conține pe o singură linie numărul întreg q – lungimea celui mai scurt canal de comunicare între calculatoarele C_a, C_b.

Exemplu:

RETELE.IN

```
6
3 4 2
1 5 6
1 4
1 3 5 6
2 4 6
4 2 5
1 6
```

RETELE.OUT

```
2
```

Restricții:
$2 \le n \le 100$; $1 \le C_a, C_b \le n$; $C_a \ne C_b$. Timpul de execuție nu va depăşi 0,5 secunde. Fişierul sursă va avea denumirea RETELE.PAS, RETELE.C sau RETELE.CPP.

Rezolvare

Pentru a determina lungimea celui mai scurt canal de comunicare între calculatoarele C_a, C_b vom folosi metoda propagării undei numerice, cunoscută în literatura de specialitate ca algoritmul lui *Lee*. Introducem în studiu tabloul $Z = |z_i|$, unde z_i reprezintă lungimea celui mai scurt canal de comunicare de la calculatorul C_a la calculatorul C_i. Inițial, calculatoarele la care încă nu a ajuns unda numerică, vor fi

marcate în tabloul Z printr-o valoare negativă, de exemplu, „–1”. Evident, $z_a = 0$.

În continuare propagăm prin liniile de transmisie a informației o undă numerică L, unde L ia consecutiv valorile 0, 1, 2, 3 ş.a.m.d. Pentru aceasta parcurgem iterativ tabloul Z şi atribuim valoarea L+1 elementelor z_i ce corespund calculatoarelor, vecine cu calculatorul în care a ajuns unda numerică L. Procesul iterativ se termină în momentul cînd unda numerică ajunge la calculatorul C_b. Pentru exemplificare, în *Fig. 2* sînt prezentate valorile tabloului Z pentru rețeaua din enunțul problemei.

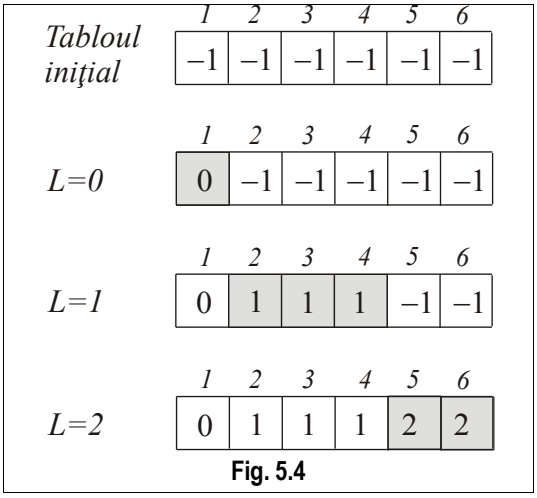


Fig. 5.4

În programul ce urmează rețeaua de calculatoare este reprezentată prin tabloul $R = \left| r_{ij} \right|_{n \times n}$. Elementul r_{ij} al acestui tablou are valoarea 1 dacă între calculatoarele C_i, C_j există o linie directă de transmisie a informației şi 0 în caz contrar.

```
Program Retele;
const nmax = 100;
var n, a, b, q : integer;
    R : array[1..nmax, 1..nmax] of integer;
    Z : array[1..nmax] of integer;
    Intrare, Iesire : text;
```

```
procedure Citeste;
var I, j : integer;
begin
  assign(Intrare, 'RETELE.IN');
  reset(Intrare);
  readln(Intrare, n);
  for i:=1 to n do
    for j:=1 to n do R[i, j]:=0;
  for i:=1 to n do
    begin
      while not eoln(Intrare) do
        begin read(Intrare, j); R[i, j]:=1; end;
      readln(Intrare);
    end; { for }
  readln(Intrare, a, b);
end; { Citeste }

procedure Scrie;
 { Scrierea datelor in fisierul de iesire }
begin
  assign(Iesire, 'RETELE.OUT');
  rewrite(Iesire);
  writeln(Iesire, Z[b]);
  close(Iesire);
end; { Scrie }

procedure UndaNumerica;
 { Propagarea undei numerice }
var i, j, L : integer;
begin
  for i:=1 to n do Z[i]:=-1;
  Z[a]:=0;
  L:=0;
  while Z[b]=-1 do
    begin
      for i:=1 to n do
        if Z[i]=L then
          for j:=1 to n do
            if (R[i, j]=1) and (Z[j]=-1) then
Z[j]:=L+1;
      L:=L+1;
    end; { while }
end; { UndaNumerica }

begin
  Citeste;
  UndaNumerica;
  Scrie;
end.
```

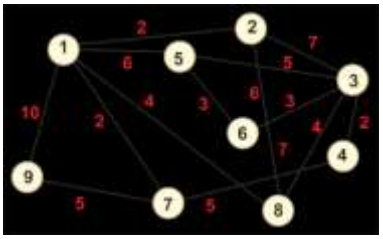
Din analiza textului procedurii UndaNumerica se observă că instrucțiunea **if** din componența ciclurilor imbricate **for** va fi executată de cel mult n^2 ori. Corpul ciclului **while** va fi executat de cel mult q ori, unde q este lungimea celui mai scurt canal de comunicare de la calculatorul C_a la calculatorul C_b. Evident, lungimea acestui canal de comunicare nu poate depăşi valoarea $n-1$, deci $q < n$. Prin urmare, în cel mai nefavorabil caz, numărul de execuții a instrucțiunii **if** va fi de ordinul n^3.

Conform restricțiilor din enunțul problemei, $n \leq 100$. Prin urmare, numărul necesar de operații este de ordinul 10^6, mărime cu mult mai mică decît capacitatea de prelucrare a calculatoarelor personale.

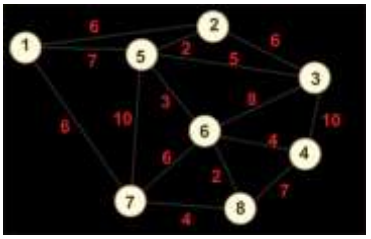
Exerciţii

1. Determinați distanța minimă între vârfurile indicate pe grafurile de mai jos:

3 – 9



1 – 4



2. Pentru implementarea algoritmului Dijkstra, propusă în paragraful 5.1, elaborați o funcție pentru restabilirea lanțului ce corespunde drumului minim de la vârful dat *s* către destinația *t*.

3. Realizați o implementare a algoritmului Floyd (paragraful 5.2) fără matricea auxiliară pentru restabilirea lanțurilor care corespund drumurilor minime.

4. Extindeți implementarea din exercițiul 2, adăugând opțiunea de restabilire a lanțurilor ce corespund drumurilor minime.

5. Estimați complexitatea algoritmului Dijkstra.

6. Estimați complexitatea algoritmului Floyd.

Capitolul 6. Centre în graf

În acest capitol:

- Centre în graf
- Centre interioare şi exterioare
- Raza grafului
- Algoritmi exacţi pentru determinarea centrului

În aplicațiile practice foarte des se întâlnesc problemele legate de amplasare optimă a unor puncte de deservire (stații, puncte de control, utilaje) într-o rețea de localități, încăperi etc. Punctele pot fi unul sau câteva, în dependență de condițiile problemei. Formulată în termeni uzuali, problema este de a găsi un punct, care ar minimiza suma distanțelor de la oricare alt punct al rețelei până la el, sau în termenii teoriei grafurilor - de a determina vârful grafului sau punctul geometric, care aparține unei muchii, astfel încât acesta va minimiza suma distanțelor până la toate celelalte vârfuri ale grafului. Se va considera suplimentar, că drumurile pentru localitățile (vârfurile) situate pe acelaşi lanț sunt distincte.

6.1 Divizări

Pentru orice vârf v_i al grafului $G=(V,E)$ prin $R_\lambda^0(v_i)$ se va nota mulțimea de vârfuri v_j ale grafului G care pot fi atinse din v_i prin căi, ce nu depăşesc mărimea λ. Prin $R_\lambda^t(v_i)$ se va nota mulțimea de vârfuri v_j ale grafului G din care v_i poate fi atins prin căi ce nu depăşesc mărimea λ. Astfel, pentru orice vârf v_i al grafului G se notează:

$$R_\lambda^0(v_i)=\left\{v_j : d\left(v_i,v_j\right)\le\lambda,\ v_j\in V\right\}$$

$$R_\lambda^t(v_i)=\left\{v_j : d\left(v_j,v_i\right)\le\lambda,\ v_j\in V\right\}$$

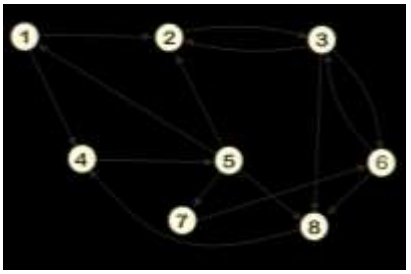
Pentru fiecare din vârfurile v_i vor fi definite 2 mărimi:

$$s_0(v_i) = \max_{v_j \in V} \left[d\left(v_i, v_j\right) \right]$$

$$s_t(v_i) = \max_{v_j \in V} \left[d\left(v_j, v_i\right) \right]$$

Valorile $s_0(v_i)$ şi $s_t(v_i)$ se numesc *indici de separare exterior* şi, respectiv, *interior* ai vârfului v_i.

Exemplu:



Des. 6.1 Graf tare conex şi matricea distanţelor lui.

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	s_0
v_1	0	1	2	1	2	3	3	3	3
v_2	5	0	1	3	4	2	5	2	5
v_3	4	1	0	2	3	1	4	1	4
v_4	2	2	3	0	1	3	2	2	3
v_5	1	1	2	2	0	2	1	1	2 *
v_6	4	4	1	2	3	0	3	1	4
v_7	6	3	2	3	4	1	0	2	6
v_8	3	3	4	1	2	4	3	0	4
s_t	6	4	4	3 *	4	4	5	3 *	

Este evident, că pentru vârful v_i, $s_0(v_i)$ va fi determinată de valoarea maximă din rândul i al matricei distanțelor, iar $s_t(v_i)$ - de valoarea maximă din coloana i. Tot de aici rezultă că valorile $s_0(v_i)$ şi $s_t(v_i)$ au valori finite pentru orice i numai dacă graful este tare conex.

6.2 Centrul şi raza grafului

Def. Vârful v_0^* pentru care are loc relația $s_0(v_0^*) = \min_{v_i \in V}\left[s_0(v_i)\right]$ se numeşte *centru exterior* al grafului *G*. Vârful v_t^* pentru care are loc relația $s_t(v_t^*) = \min_{v_i \in V}\left[s_t(v_i)\right]$ se numeşte *centru interior* al grafului *G*.

Centrul exterior este vârful (sau vârfurile) care minimizează cea mai lungă distanță *de la el* spre oricare alt vârf al grafului. Din matricea distanțelor el poate fi determinat ca minimul valorilor maxime de pe fiecare linie. Pentru graful de pe desenul 6.1 centrul exterior este vârful 5. Centrul interior este vârful (sau vârfurile) care minimizează cea mai lungă distanță *spre el* de la oricare alt vârf al grafului. Din matricea distanțelor el poate fi determinat ca minimul valorilor maxime de pe fiecare coloană. Pentru graful de pe desenul 6.1 centre exterioare sunt vârfurile 4 şi 8. Într-un graf neorientat centrul interior şi cel exterior coincid.

Def. Valoarea $s_0(v_0^*) = \min_{v_i \in V}\left[s_0(v_i)\right]$ se numeşte *raza exterioară* a grafului *G*. Valoarea $s_t(v_t^*) = \min_{v_i \in V}\left[s_t(v_i)\right]$ se numeşte *raza interioară* a grafului *G*.

Pentru graful de pe desenul 6.1 raza exterioară are valoarea 2, în timp ce raza interioară are valoarea 3.

Pentru un graf neorientat raza interioară şi raza exterioară sunt egale, iar centrele exterioare coincid cu cele interioare.

Def. Valoarea $s(v_0^*) = \min_{v_i \in V}\left[s(v_i)\right]$ se numeşte *raza* grafului neorientat *G*. Valoarea $s_t(v_t^*) = \min_{v_i \in V}\left[s_t(v_i)\right]$ se numeşte *raza interioară* a grafului *G*.

Def. Vârful v_0^* pentru care are loc relația $s(v_0^*) = \min_{v_i \in V}\left[s(v_i)\right]$ se numeşte *centru* al grafului neorientat *G*.

6.3 P-centre

Vom porni de la o situație frecvent întâlnită. Fie că într-un oraş sunt amplasate mai multe (P) centre de asistență medicală urgentă, cu echipe mobile de medici. În cazul recepționării unei cereri de asistență la centrul comun de apel, către solicitant se deplasează echipajul de medici de la cel mai apropiat centru de asistență.

În acest caz amplasarea inițială a celor P centre de asistență medicală urgentă este organizată într-un mod, care asigură minimul de aşteptare a pacientului, indiferent de locația acestuia.

Fie V_p o submulțime din V. $|V_p| = p$. Prin $d(V_p, v_i)$ se va nota distanța de la cel mai apropiat vârf din V_p până la vârful v_i:

$$d(V_p, v_i) = \min_{v_j \in V_p}\left[d(v_j, v_i)\right].$$

La fel, prin $d(v_i, V_p)$ se va nota distanța de la vârful v_i până la cel mai apropiat vârf din V_p: $d(v_i, V_p) = \min_{v_j \in V_p}\left[d(v_i, v_j)\right]$.

Indicii de separare pentru mulțimea V_p se calculează la fel ca şi pentru vârfurile solitare:

$$s_0(V_p) = \max_{v_j \in V}\left[d(V_p, v_j)\right]$$

$$s_t(V_p) = \max_{v_j \in V}\left[d(v_j, V_p)\right]$$

Def. Mulțimea de vârfuri $V^*_{p,0}$ pentru care are loc relația $s_0(V^*_{p,0}) = \min_{V_p \in G}\left[s_0(V_p)\right]$ se numeşte *p-centru exterior* al grafului *G*. Mulțimea de vârfuri $V^*_{p,t}$ pentru care are loc relația $s_t(V^*_{p,t}) = \min_{V_p \in G}\left[s_t(V_p)\right]$ se numeşte *p-centru interior* al grafului *G*.

Def. Valoarea $s_0(V^*_{p,0}) = \min_{V_p \in G}\left[s_0(V_p)\right]$ se numeşte *p-raza exterioară* a grafului *G*. Valoarea $s_t(V^*_{p,t}) = \min_{V_p \in G}\left[s_t(V_p)\right]$ se numeşte *p-raza interioară* a grafului *G*.

Algoritmul euristic pentru determinarea p-centrelor

Algoritmul face parte din clasa algoritmilor de optimizare locală, care se bazează pe ideea λ optimizării pe grafuri.

Fie graful neorientat $G = (V, E)$. O mulțime de vârfuri $S \subseteq V$, $|S| = p$ se numeşte λ optimă $(\lambda \le p)$ pentru problema Q, dacă înlocuirea oricăror λ vârfuri din S cu λ vârfuri din $V - S$ nu va îmbunătăți soluția problemei Q, obținută pe mulțimea S. Algoritmul descris este unul general şi poate fi folosit pentru diverse probleme formulate pe grafuri.

Inițial în calitate de soluție este selectată o mulțime arbitrară de vârfuri C, care aproximează *p* centrul. Apoi se verifică, dacă un careva vârf $v_j \in V - C$ poate înlocui vârful $v_i \in C$. Pentru aceasta se formează mulțimea $C' = \left\{C \cup \{v_j\} - \{v_i\}\right\}$ după care se compară $s(C)$ şi $s(C')$. Ulterior C' este cercetată în acelaşi mod , pentru a obține C''. Procesul se repetă până la obținerea unei mulțimi $\overline{C}$, pentru care nu mai pot fi efectuate îmbunătățiri prin procedeul descris. Mulțimea de vârfuri $\overline{C}$ este considerată *p*-centrul căutat.

Pseudocod

Pas 0. Se formează matricea distanțelor minime (algoritmul Floyd).

Pas 1. Se alege în calitate de aproximare inițială a p-centrului o mulțime arbitrară de vârfuri C. Vârfurile $v_j \in V - C$ vor fi considerate având marcajul stării egal cu 0 (neverificate).

Pas 2. Se alege un vârf arbitrar de stare 0 $v_j \in V - C$ şi pentru fiecare vârf $v_i \in C$ se calculează valorile $\Delta_{i,j} = s(C) - s(C - \{v_i\} \cup \{v_j\})$.

Pas 3. Se determină $\Delta_{i_0,j} = \max_{v_i \in C} \left[\Delta_{i,j} \right]$.

i. Dacă $\Delta_{i_0,j} < 0$ vârful v_j este marcat „verificat" (i se aplică marcajul de stare - 1) şi se revine la pasul 2.

ii. Dacă $\Delta_{i_0,j} > 0$, $C \leftarrow \left(C - \{v_i\} \cup \{v_j\}\right)$ vârful v_j este marcat „verificat" (i se aplică marcajul de stare - 1) şi se revine la pasul 2.

Pas 4. Paşii 2 – 3 se repetă atât timp cât există vârfuri cu marcaj de stare 0. Dacă la ultima repetare a paşilor 2 – 3 nu au fost efectuate înlocuiri (3.ii) se trece la pasul 5. În caz contrar tuturor vârfurilor din $V - C$ li se atribuie marcajul de stare 0, apoi se revine la pasul 2.

Pas 5. STOP. Mulțimea de vârfuri curentă C este o aproximare a *p*-centrului căutat.

6.4 Probleme rezolvate

Megaşcoli[8]

Enunț Pe insula Tortuga sunt N localități(numerotate de la 1 la N). Guvernatorul insulei, fostul corsar Ion Vrabie, a hotărât să construiască două megaşcoli moderne, la care vor merge copii din toate localitățile. Copiii din fiecare localitate urmează să meargă la cea mai apropiată dintre megaşcoli.

Guvernatorul doreşte să selecteze localitățile în care vor fi construite megaşcolile astfel, încât timpul necesar pentru a ajunge la megaşcoală din cea mai îndepărtată localitate să fie cât mai mic posibil.

Funcționarii au întocmit o hartă a insulei, pe care sunt indicate localitățile şi segmentele de drumuri care unesc unele perechi de localități. Pentru fiecare segment de drum care uneşte două localități este cunoscut timpul necesar pentru ca copiii să ajungă dintr-o localitate în cealaltă. Pentru oricare două localități de pe insulă există cel puțin o secvență de segmente de drum, care le uneşte.

Note:

a. un segment de drum începe la ieşirea dintr-o localitate şi se termină la intrarea în alta. El nu trece prin careva localități intermediare.
b. Pentru a asigura securitatea copiilor, intersecțiile segmentelor de drum sunt denivelate – nu poți trece de pe un segment de drum pe altul în afara localităților.
c. Timpul de deplasare în interiorul localității se neglijează (se consideră 0)
d. Timpul de deplasare pe o secvență de segmente de drum este egal cu suma timpilor de deplasare pe fiecare segment de drum, inclus în secvență.

[8] Concursul: Punct Campion, 2011

Fiind dată harta drumurilor între localitățile insulei, şi lungimea (în timp) a fiecărui segment de drum, să se scrie un program care să determine indicii localităților, în care se vor construi megaşcolile.

Input: Fişierul de intrare **schl.in** conține pe prima linie numărul N. Pe fiecare dintre următoarele N linii se află câte N numere naturale separate prin spații, reprezentând elementele unui tablou A de dimensiune N x N.
Elementul A[i,j] al tabloului specifică:

- A[i,j]≠0 : există segmentul de drum, între localitățile cu indicii i şi j. Timpul de deplasare între localități este A[i,j].
- A[i,j]=0, i≠j : între localitățile i şi j nu există un segment de drum, care să le unească direct. Drumul din i spre j trece prin careva localități intermediare.
- A[i,j]=0, i=j: conform notei b., în interiorul localității timpul de deplasare e 0.

Output: fişierul de ieşire schl.out va conține în prima linie trei numere, separate prin spațiu – indicii localităților în care vor fi amplasate megaşcolile ordonați lexicografic şi timpul maxim necesar pentru a ajunge la şcoală. Dacă există mai multe posibilități de alegere a localităților, se va indica perechea lexicografic minimă.

Restricții 2≤N≤100, 0≤A[i,j]≤32000.

Exemplu:

school.in	school.out	Explicaţii
4 0 3 4 2 3 0 2 5 4 2 0 3 2 5 3 0	1 2 2	(1,2) – din localitatea 3 pleaca in 2 (timpul 2), din localitatea 4 – în 1 (timpul 2). Maxim din (2,2)=2. Acelaşi rezultat se obţine pentru (1,3) şi (3,4), dar soluţiile sunt lexicografic mai mari. Pentru perechile (1,4) (2,3) (2,4) se obţine timpul minim 3.

Rezolvare

Problema se rezolvă în două etape.

1. Determinarea timpului minim, necesar pentru a ajunge din localitatea cu indicele i în localitatea cu indicele j (pentru toți i,j de la 1 la N.)

Se rezolvă prin aplicarea algoritmului Floyd asupra matricei de adiacență a grafului, vârfurile căruia reprezintă localitățile, iar muchiile – drumurile între localități.

2. Este dată matricea cu timpii minimi necesari pentru deplasarea între orice două localități. Se cere să se determine 2 loalități, care minimizează cel mai mare timp de deplasare până la ele.

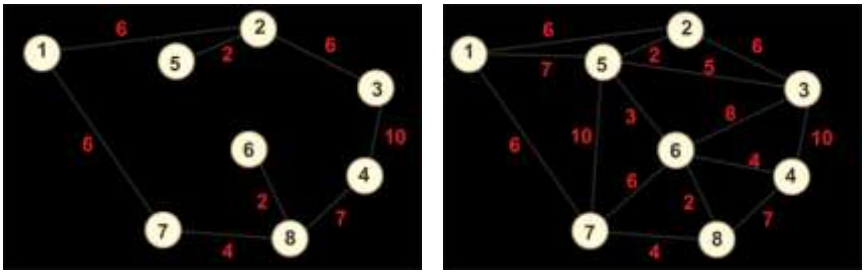
Este problema clasică a 2-centrului pe un graf neorientat. 2-centrul se determină prin selectarea consecutivă a celei mai bune soluții din mulțimea tuturor perechilor de vârfuri (i,j). i de la 1 la N-1, ,j de la i+1 la N.

Implementare (C)

```
#include <stdio.h>
#include <stdlib.h>

int n,i,j,l,k,p=0,a[101][101],b[101],c[101],bestsol,
currentsol;
struct pereche{int v1;int v2;} best, current;
FILE *f, *g;

int tipar()
{ int i,j;
  for (i=1; i<=n; i++)
  {   for (j=1;j<=n; j++)
        printf("%5d", a[i][j]);
      printf("\n"); }
return 0; }

int readdata()
{ int i,j;
      f=fopen("00-schl.in", "r");
      fscanf(f, "%d", &n);
      for (i=1;i<=n;i++)
        for (j=1;j<=n; j++)
              fscanf(f, "%d", &a[i][j]);
      fclose(f);
  return 0; }

int floyd()
{ int i,j,k;
// modelare infinit
 for (i=1;i<=n;i++)
        for (j=i+1; j<=n; j++) p+=a[i][j];
 p++;
 // matrice cost
```

```
 for (i=1;i<=n;i++)
        for (j=1; j<=n; j++)
         if (a[i][j]==0 && i != j) a[i][j]=p;
// floyd
for (k=1;k<=n;k++)
      for (i=1;i<=n;i++)
              if (i!=k && a[i][k]!=p)
                      for (j=1;j<=n;j++)
if (j!=k && a[k][j] !=p)
      if (a[i][j]>a[i][k]+a[k][j]) a[i][j]=a[i][k]+a[k][j];
return 0;
}

int min(int x, int y)
{if (x<y) return x; else return y; }

int main()
{ readdata();
  floyd();
  tipar();
  g=fopen("00-schl.cor", "w+");
  bestsol=p;
  for (i=1;i<n;i++)
   for (j=i+1;j<=n;j++)
    { current.v1=i; current.v2=j;
     for (k=1;k<=n;k++)
      {
       for (l=1;l<=n;l++)
        b[l]=min(a[l][i],a[l][j]);
        c[k]=0;
        for (l=1;l<=n;l++) if (c[k]<b[l]) c[k]=b[l];
      }
      currentsol=c[1];
      for (l=1;l<=n;l++)
       if (c[l]<currentsol) currentsol=c[l];
       if (bestsol>currentsol)
          {bestsol=currentsol; best=current;}
    }
      fprintf(g, "%d %d %d\n", best.v1, best.v2, bestsol);
       fclose(g);
      return 0;
}
```

Exerciţii:



Des. 6.2. A B

1. Determinați vârfurile centru pentru grafurile de pe desenul 6.2.

2. Elaborați un program pentru determinarea centrelor relative exterioare şi interioare ale unui graf orientat, descris prin matricea sa de adiacență($|V| < 50$).

3. Elaborați un program pentru determinarea centrelor relative ale unui graf neorientat ($|V| < 50$).

4. Determinați 2-centrul pentru grafurile de pe desenul 6.2.

Capitolul 7. Mediane

În acest capitol:

- Mediane în graf
- Mediane interioare şi exterioare
- Algoritmi exacţi pentru determinarea medianei

7.1 Mediane

Mai multe probleme de amplasare a punctelor de deservire presupun minimizarea sumei distanțelor de la o serie de puncte terminale până la un punct central (de colectare, comutare, depozite etc.)

Punctele care corespund soluției optime ale problemei se numesc puncte *mediane* ale grafului.

Fie graful $G=(V,E)$. Pentru fiecare vârf v_i se definesc două valori, care se numesc *indici de transmitere*:

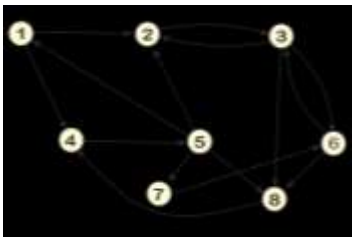
$$\sigma_0(v_i)=\sum_{v_j\in V}\left[d\left(v_i,v_j\right)\right]$$

$$\sigma_t(v_i)=\sum_{v_j\in V}\left[d\left(v_j,v_i\right)\right]$$

Aici $d(v_i,v_j)$ – distanța minimă dintre vârfurile v_j şi v_i. Valorile $\sigma_0\left(v_i\right)$ şi $\sigma_t\left(v_i\right)$ se numesc *indice interior şi exterior de transmitere* a vârfului v_i. Indicii de transmitere pot fi calculați din matricea distanțelor $D(G)$: $\sigma_0\left(v_i\right)$ ca suma elementelor din linia i a matricei D, iar $\sigma_t\left(v_i\right)$ ca suma elementelor din coloana i.

Def. Vârful $\overline{v_0}$ pentru care $\sigma_0\left(\overline{v_0}\right)=\min_{v_i\in V}\left[\sigma_0\left(v_i\right)\right]$ se numeşte *mediană exterioară* a grafului G. Vârful $\overline{v_t}$ pentru care $\sigma_t\left(\overline{v_t}\right)=\min_{v_i\in V}\left[\sigma_t\left(v_i\right)\right]$ se numeşte *mediană interioară*.

Exemplu: Fie graful din desenul ce urmează:



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	σ_0
v_1	0	1	2	1	2	3	3	3	15
v_2	5	0	1	3	4	2	5	2	22
v_3	4	1	0	2	3	1	4	1	
v_4	2	2	3	0	1	3	2	2	15
v_5	1	1	2	2	0	2	1	1	10
v_6	4	4	1	2	3	0	3		18
v_7	6	3	2	3	4	1	0	2	21
v_8	3	3	4	1	2	4	3	0	20
σ_t	25	15	15	14	19	16	21	12	

Pentru exemplul prezentat mediana exterioară este vârful v_5 cu indicele de transmitere 10, mediana interioară – vârful v_8 cu indicele de transmitere 12.

Algoritmul general pentru determinarea medianelor presupune calculul distanțelor minime între vârfurile grafului, ulterior calcularea sumelor elementelor matricei pe linii (coloane) şi selectarea minimului din sumele calculate (minimul sumelor pe linii pentru mediana exterioară, pe coloane – pentru mediana interioară).

7.3 P-mediane

Noțiunea de *p*-mediană se introduce prin analogie cu noțiunea de *p*-centru.

Fie V_p o submulțime din V. $|V_p| = p$. Prin $d(V_p, v_i)$ se va nota distanța de la cel mai apropiat vârf din V_p până la vârful v_i:

$$d\left(V_p, v_i\right) = \min_{v_j \in V_p}\left[d\left(v_j, v_i\right)\right] \qquad (*)$$

La fel, prin $d\left(v_i, V_p\right)$ se va nota distanța de la vârful v_i până la cel mai apropiat vârf din V_p: $d\left(v_i, V_p\right) = \min_{v_j \in V_p}\left[d\left(v_i, v_j\right)\right]$ (**)

Dacă v_j' este vârful din V_p, pentru care se obține valoarea minimă a (*) sau (**), se spune că v_i este arondat la v_j'.

Indicii de transmitere pentru mulțimea V_p se calculează la fel ca şi pentru vârfurile solitare:

$$\sigma_0(V_p) = \sum_{v_j \in V}\left[d\left(V_p, v_j\right)\right], \qquad \sigma_t(V_p) = \sum_{v_j \in V}\left[d\left(v_j, V_p\right)\right]$$

Def. Mulțimea de vârfuri $\overline{V}_{p,0}$ pentru care are loc relația $\sigma_0(\overline{V}_{p,0}) = \min_{V_p \subseteq V}\left[\sigma_0(V_p)\right]$ se numeşte *p-mediană exterioară* a grafului *G*. Mulțimea de vârfuri $\overline{V}_{p,t}$ pentru care are loc relația $\sigma_t(\overline{V}_{p,t}) = \min_{V_p \subseteq V}\left[\sigma_t(V_p)\right]$ se numeşte *p-mediană interioară* a grafului *G*.

Pentru un graf neorientat *p*-mediana exterioară şi cea interioară coincid, prin urmare se cercetează doar noțiunea de *p*-mediană, ca mulțime de vârfuri, care minimizează suma distanțelor până la toate celelalte vârfuri ale grafului.

7.4 Algoritmi pentru determinarea *p*-medianei

Algoritmul direct

Algoritmul direct presupune generarea tuturor submulțimilor din *p* vârfuri ale grafului *G* şi selectarea mulțimii $\overline{V}_p$ pentru care $\sigma(\overline{V}_p) = \min_{V_p \subseteq V}\left[\sigma(V_p)\right]$. O asemenea abordare presupune cercetarea a C_n^p mulțimi, ceea ce necesită resurse exponențiale de timp pentru valori

mari ale n sau valori ale p apropiate de $n/2$. În particular, pentru valori ale lui n, p, ce nu depăşesc 20, poate fi folosit un algoritm clasic de generare a submulțimilor unei mulțimi, descris în [9, p. 32]. O modificare elementară va permite generarea doar a mulțimilor din p elemente, ceea ca va reduce considerabil timpul de lucru al algoritmului. O altă componentă a soluției este calcularea matricei D a distanțelor minime, care va servi în calitate de sursă de date pentru determinarea p-medianei.

Algoritmul euristic

Pentru valori mari ale lui n şi p poate fi folosit un algoritm euristic, similar algoritmului pentru determinarea p-centrului.

Pas 0. Se formează matricea distanțelor minime (algoritmul Floyd).

Pas 1. Se alege în calitate de aproximare inițială a p-medianei o mulțime arbitrară de vârfuri C. Vârfurile $v_j \in V - C$ vor fi considerate având marcajul stării egal cu 0 (neverificate).

Pas 2. Se alege un vârf arbitrar de stare 0 $v_j \in V - C$ şi pentru fiecare vârf $v_i \in C$ se calculează valorile $\Delta_{i,j} = \sigma(C) - \sigma(C - \{v_i\} \cup \{v_j\})$

Pas 3. Se determină $\Delta_{i_0,j} = \max_{v_i \in C} \left[\Delta_{i,j} \right]$.

i. Dacă $\Delta_{i_0,j} < 0$ vârful v_j este marcat „verificat” (i se aplică marcajul de stare - 1) şi se revine la pasul 2.

ii. Dacă $\Delta_{i_0,j} > 0$, $C \leftarrow \left(C - \{v_i\} \cup \{v_j\} \right)$ vârful v_j este marcat „verificat” (i se aplică marcajul de stare - 1) şi se revine la pasul 2.

Pas 4. Paşii 2 – 3 se repetă atât timp cât există vârfuri cu marcaj de stare 0. Dacă la ultima repetare a paşilor 2 – 3 nu au fost efectuate înlocuiri (3.ii) se trece la pasul 5. În caz contrar tuturor vârfurilor din $V - C$ li se atribuie marcajul de stare 0, apoi se revine la pasul 2.

Pas 5. STOP. Mulțimea de vârfuri curentă C este o aproximare a p-medianei căutate.

Implementare

Input: Graful G: matricea de adiacență **d**, ulterior matricea distanțelor; vectorul p-medianei **pmed**, vectorul stare **st**.
Output: O p-medianăposibilă a G, în vectorul **pmed.**

```
int tipar()
{…// afişează mediana curentă }

int calcmed()
{ … // calculează valoarea p-medianei curente }

int main()
{ …// citire date
  …// modelare valoare „infinit" - p
  …// algoritmul Floyd. D – matricea distanţelor
// initializare p-mediana
   for(i=1;i<=n; i++) st[i]=0;
   for(i=1;i<=pmed; i++) {st[i]=2; med[i]=i;}
do {
   for (i=1;i<=n; i++) if (st[i]==1) st[i]=0;
   // reinitializare stare varfuri
   for (i=1;i<=n;i++)
       if (st[i]==0)
        {delta=0; sumtot=calcmed();
          for (j=1;j<=pmed; j++)
           { tmp=med[j]; med[j]=i;
             st[i]=2; st[tmp]=0; // i se introduce in poz j
             medcurent=calcmed();
             if (delta<sumtot - medcurent)
                {delta=sumtot-medcurent; sw1=j;sw2=i;}
             // micsorare indice de transmtere
             med[j]=tmp; st[tmp]=2; // restabilire j
           }
            if (delta>0 ) // a fost detectata o îmbunătăţire
            { tmp=med[sw1]; med[sw1]=sw2;
              st[tmp]=1; st[sw2]=2;}       // inlocuire
            else st[i]=1;
```

```
        }
} while (delta>0);
tipar(); printf(" %d ", calcmed());
return 0; }
```

7.5 Probleme rezolvate

Fabrica[9]

Enunț: Pe insula Tortuga sunt N localități (numerotate de la 1 la N). Guvernatorul insulei, fostul corsar Ion Vrabie, a hotărât să construiască o fabrică modernă de prelucrare a laptelui, care va prelucra materia primă colectată din toate localitățile. Colectarea este efectuată cu o autospecială, care face câte o cursă separată pentru fiecare localitate. Localitatea în care va fi construită fabrica se va selecta astfel, încât distanța parcursă de autospecială pentru a colecta laptele din toate localitățile şă fie cât mai mic posibilă.

Fiind date distanțele între toate localitățile insulei, să se scrie un program care să determine indiciele localității, în care se va construi fabrica de prelucrare a laptelui.

Input Fişierul de intrare fabrik.in conține pe prima linie numărul N. Pe fiecare dintre următoarele n linii se află câte n numere naturale separate prin câte un spațiu, reprezentând elementele unui tablou A de dimensiune N x N. Elementul A[i,j] indică distanța dintre localitățile cu indicii i şi j.

Output fişierul de ieşire fabrik.out va conține în prima linie un număr natural – indiciele localității în care va fi amplasată fabrica. Dacă există mai multe posibilități de alegere a localității, se va indica cea cu indicele minim.

Restricții $2 < N < 50$, Elementele matricei – numere naturale ≤ 32000.

[9] Concursul zonal între colegii, zona Centru-Sud, 2012

Exemplu:

x.in	X.out	Explicaţii
4 0 3 4 2 3 0 2 5 4 2 0 3 2 5 3 0	1	Pentru vârful 1 distanţa totală este 18 Pentru vârful 2 distanţa totală este 20 Pentru vârful 3 distanţa totală este 18 Pentru vârful 4 distanţa totală este 20 Distanţa totală minimă este 18, se obţine pentru localităţile 1 şi 3. Indicele minim – 1.

Rezolvare:

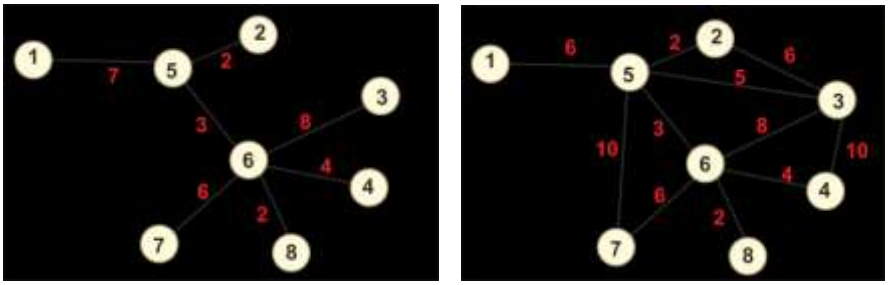
Este o problemă-tip, unde se cere să se determine mediana într-un graf neorientat. Datele inițiale corespund matricei distanțelor în graful, care descrie schema de comunicații între toate localitățile. Astfel, rămâne doar de calculat sumele elementelor din fiecare linie a tabelului, după care să se determine suma minimă şi indicele primei linii, în care aceasta se obține.

Implementare:

```
#include <stdio.h>
#include <stdlib.h>

int n,i,j,a[101][101],b[101], bestsol;
FILE *f, *g;

int readdata()
{ int i,j;
      f=fopen("fabrik.in", "r");
    fscanf(f, "%d", &n);
 for (i=1;i<=n;i++)
       for (j=1;j<=n; j++)
              fscanf(f, "%d", &a[i][j]);
      fclose(f);
return 0;
}

int main()
{
readdata();
```

```
g=fopen("fabrik.out", "w+");

for (i=1;i<=n;i++)
  { b[i]=a[i][1];
       for (j=2;j<=n;j++)
         b[i]+=a[i][j];
  }
  bestsol=1;
  for (i=2;i<=n;i++) if (b[i]<b[bestsol]) bestsol=i;
  fprintf(g, "%d %d", bestsol, b[bestsol]*2);
  fclose(g);
}
```

Exerciţii:



A B

Des 7.1

1. Determinați medianele pentru grafurile de pe desenul 7.1.
2. Elaborați un program pentru determinarea medianelor interioare şi exterioare ale unui graf $G=(V,E),\ \ |V|\le 200$.
3. Determinați 2-medianele pentru grafurile de pe desenul 7.2.
4. Elaborați un program pentru determinarea exactă a 2-medianelor interioare şi exterioare ale unui graf $G=(V,E),\ \ |V|\le 20$.
5. Elaborați un program pentru determinarea exactă a p-medianelor ale unui graf $G=(V,E),\ \ |V|\le 20$.

6. Elaborați un program pentru determinarea aproximativă, folosind algoritmul euristic $(\lambda = 1)$, a p-medianelor ale unui graf $G = (V, E), \quad |V| \leq 100$.

7. Elaborați un program pentru determinarea aproximativă, folosind algoritmul euristic $(\lambda = 2)$, a p-medianelor ale unui graf $G = (V, E), \quad |V| \leq 100$.

Capitolul 8. Arbori

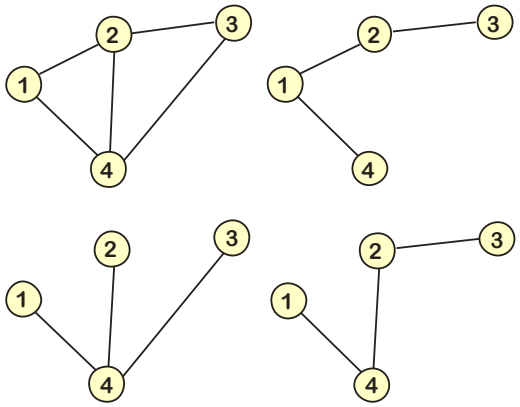
În acest capitol:

- Definiţii ale arborilor
- Generarea tuturor arborilor parţiali ai unui graf
- Arbori parţiali de cost minim
- Algoritmul Kruskal pentru determinarea arborelui de cost minim.
- Algoritmul Prim pentru determinarea arborelui de cost minim

8.1 Arbori de acoperire

Carcase

Un graf neorientat este numit *arbore* dacă este conex şi nu conține cicluri. Pentru graful $G=(V,E)$ fiecare arbore $G^*=(V,T)$ $T \subseteq E$ este numit *carcasă* sau *arbore parțial de acoperire* a grafului G. Muchiile din G, care apar în G^* sunt numite *ramuri*, cele din E/T - *corzi*. Numărul carcaselor unui graf este determinat de structura acestuia.



Des. 8.1. Graful (stânga sus) şi câteva dintre carcasele sale

Algoritmii de parcurgere a unui graf conex permit şi construirea concomitentă a unei carcase. Într-adevăr, proprietatea de conexitate asigură atingerea fiecărui nod u a grafului pe parcursul lucrului algoritmului. Pentru nodul u, atins din v, algoritmii de parcurgere nu

mai permit atingerea repetată din alt nod. Prin urmare, apariția ciclurilor în procesul parcurgerii este imposibilă.

Următoarea modificare a funcției DFS, studiată în capitolele precedente, afişează lista muchiilor pentru carcasa ce corespunde parcurgerii în adâncime a grafului:

```
int DFS (int s)
{ int i;
  b[s]=1;
  for(i=1;i<=n;i++)
       if(a[s][i] !=0 && b[i]==0)
          {printf("\n %d - %d", s, i); DFS(i);}
 return 0;
}
```

Generarea tuturor carcaselor unui graf

Mai multe probleme impun necesitatea de a genera nu doar o carcasă a grafului dat, ci toate carcasele posibile. De exemplu, trebuie să fie selectată „cea mai bună carcasă" după un criteriu complex, care nu permite aplicarea unor optimizări directe. În alte situații, generarea tuturor subarborilor în calitate de subproblemă permite reducerea complexității la rezolvarea unor probleme de calcul economic.

Numărul posibil de carcase ale unui graf variază în dependență de numărul de muchii în graful inițial, dar şi de structura conexiunilor. Pentru un graf complet cu n vârfuri, de exemplu, numărul calculat al carcaselor este egal cu n^{n-2}. Prin urmare, problema generării tuturor carcaselor este una de complexitate exponențială (în caz general) . Respectiv, algoritmul pentru generarea acestora va avea la origine tehnica reluării. O posibilă structură a algoritmului a fost dată de Kristofides. Totuşi, algoritmul propus în [6, p. 154] este unul iterativ, ceea ce face mai complicată implementarea lui. În cele ce urmează se propune un algoritm recursiv, care formează carcasele în baza listei de muchii ale grafului *G*.

Preprocesare: fie $G=(V,E)$. Se formează lista de muchii $e_1,e_2,...,e_m$ ale grafului *G*.

Funcția recursivă **CARCASE(k,c)**

Pas A.

(i) $i \leftarrow k$.

(ii) Muchia curentă $e_i=(v',v'')$. Pe graful T se lansează funcția modificată de căutare în adâncime **DFS**(v',v''), pentru a determina existența unei căi între v' şi v''.

(iii) Dacă funcția returnează 1 (în T există o cale între v' şi v'') STOP .

În caz contrar funcția returnează 0 (nu există o cale între v' şi v'')

$T=T\bigcup$, $c \leftarrow c+1$

Dacă $c=n-1$,se afişează carcasa construită *T*, apoi se trece la pasul 4.

În caz contrar pentru toți i de la $k+1$ la m **CARCASE(i,c).**

Pas B.

Excludere muchie $T=T-e_i$ $c \leftarrow c-1$

Sfârşit funcție **carcase**.

În funcție se generează toate carcasele, care încep de la muchia e_k .

Algoritm

Pas 1. $k \leftarrow 0$.

Pas 2 Indicele muchiei de la care continuă construcția carcasei $k \leftarrow k+1$.

Se formează un graf T din vârfurile izolate ale lui *G:* $T=(V,\varnothing)$

Numărul de muchii în carcasă $c \leftarrow 0$.

Pas 3. **CARCASE(k,0).**

Pas 4 dacă $k < m-n+1$ se revine la pasul 2, altfel STOP – toate carcasele au fost generate.

Implementare

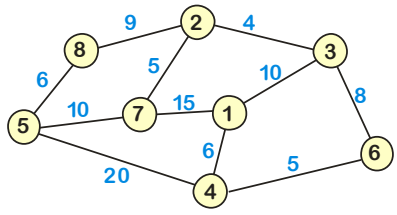
Input: Graful G : matricea de adiacență **a**; lista de muchii **v**.
Output: Arborii parțiali a grafului G, generați consecutiv în tabloul **arb.**

```
int tipar()
{ .. //afişează soluţia curentă – matricea de adiacenţă a
arborelui parţial}

int scoatemuchie(int ind)
{ …// exclude muchia cu indicele ind din arborele în
construcţie}

int punemuchie(int ind)
{ …// include muchia cu indicele ind în arborele în
construcţie}

int DFS (int s,int tt)
{ …// verifică prezenţa unui drum între vârfurile s şi tt}

int ciclu(int im)
{ …// verifică apariţia unui ciclu la adăugarea muchiei cu
indicele im }

int back(int indicemuchie, int numarmuchii)
{ int i;
      if (ciclu(indicemuchie)) return 0;
      else
      { punemuchie(indicemuchie);// se adaugă muchia
        numarmuchii++;
        if (numarmuchii==n-1) // e detectată o soluţie
         {tipar(); scoatemuchie(indicemuchie); numarmuchii--;
          return 0;}
        else
        {for (i=indicemuchie+1;i<=k;i++) back(i,numarmuchii);
```

```
            // se adaugă următoarele muchii
            scoatemuchie(indicemuchie);numarmuchii--; return 0;
            // se exclude muchia la pasul de întoarcere
          }
        }
}
int main()
{ …// citire date
   // formarea lista muchii
   for (i=1;i<=n;i++)
        for (j=i+1; j<=n; j++)
              if (a[i][j]==1) {k++; ed[k].vi=i; ed[k].vj=j;}
   // căutare carcase
   for (i=1;i<=k-n+1;i++)
        back(i,0);
   return 0;  }
```

8.2 Arbori de acoperire de cost minim

După cum s-a menționat şi în capitolele precedente, mai multe probleme formulate pe grafuri presupun existența unei caracteristici suplimentare, atribuite muchiilor grafului – *ponderea*. Pentru muchia (v,u) ponderea se notează $c_{v,u}$. De obicei, ponderea are o expresie numerică, care indică distanța dintre noduri, capacitatea sau timpul de transfer de-a lungul muchiei etc.



$$\begin{bmatrix} 0 & 0 & 10 & 6 & 0 & 0 & 15 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 5 & 9 \\ 10 & 4 & 0 & 0 & 0 & 8 & 0 & 0 \\ 6 & 0 & 0 & 0 & 20 & 5 & 0 & 0 \\ 0 & 0 & 0 & 20 & 0 & 0 & 10 & 6 \\ 0 & 0 & 8 & 5 & 0 & 0 & 0 & 0 \\ 15 & 5 & 0 & 0 & 10 & 0 & 0 & 0 \\ 0 & 9 & 0 & 0 & 6 & 0 & 0 & 0 \end{bmatrix}$$

Des. 8.2. Graf ponderat şi matricea lui de adiacenţă

Pentru stocarea ponderilor muchiilor nu sunt necesare modificări esențiale în structurile de date utilizate pentru descrierea grafurilor: în matricele de incidență sau adiacență valorile unitare ce corespund muchiilor sunt înlocuite prin valorile ponderilor (desenul 8.2); în listele de muchii şi de incidență fiecare

pereche (element) este suplinită de o componentă ce conține valoarea ponderii.

Prin *cost* al arborelui $G^* = (V, T)$ se va înțelege suma ponderilor muchiilor, care îl formează.

$$S(G^*) = \sum_{(v,u)\in T} c_{v,u}$$

Pentru grafurile neponderate toți arborii de acoperire au acelaşi cost. În cazul prezenței ponderilor în graf apare şi problema selectării unui arbore de acoperire de cost minim. Există mai mulți algoritmi eficienți pentru determi-narea arborelui de acoperire de cost minim, cei mai cunoscuți fiind:

Algoritmul Kruskal

Fie $G = (V, E)$ un graf ponderat cu n vârfuri. Algoritmul Kruskal foloseşte tehnica greedy şi presupune crearea unui graf $G^* = (V, T)$, în care inițial $T = \varnothing$. Ulterior, muchiile din G se sortează după creşterea ponderilor. La următoarea etapă se efectuează adăugarea consecutivă în G^* a muchiilor din şirul sortat, cu condiția că la adăugarea muchiei în G^* nu se formează un ciclu. Lucrul algoritmului se sfârşeşte când numărul de muchii adăugate devine n-1.

Pseudocod:

Pas 1. Se construieşte $G^* = (V, T)$, $T = \varnothing$.

Pas 2. Muchiile din $G = (V, E)$ se sortează în ordinea creşterii ponderilor. Se obține şirul $m_1, ..., m_{|E|}$.

Pas 3. $k \leftarrow 0, \quad i \leftarrow 1$.

Pas 4. While $k \neq |V| - 1$ do

a) if $T \cup m_i$ *nu formează un ciclu în* G^*, then

$T \leftarrow T \cup m_i, k\uparrow$;

b) $i\uparrow$.

Implementare:

Input: Graful G: matricea de adiacență **a**; liata de muchii **v**; arborele parțial de cost minim **arb**, tabloul componentelor conexe **c**.
Output: Un arbore parțial de cost minim a G, în tabloul **arb.**

```
int sort ()
{ …// functia de sortare a listei de muchii }
int readdata()
{ …// functa de citire a datelor initiale }

int printv()
{ … // functia de tipar a rezultatelor}

int makeedgelist()
{ …// functia pentru crearea listei de muchii }

int main(int argc, char *argv[])
{ int j,i,r;
  readdata();
  makeedgelist();
  sort();
for (i=1;i<=n;i++) c[i]=i; // initializare componente conexe
i=0;
while(k<n-1)
{ i++;
if (c[v[i].vi] != c[v[i].vj])
// verificare posibilitate adaugare muchie
  {k++; arb[k]=v[i];
   for(j=1;j<=n;j++)
        if (c[j] == c[v[i].vj] ) c[j]=c[v[i].vi];
  }
 }
printv(); // afisare rezultate
return 0;
}
```

Algoritmul Prim

Spre deosebire de algoritmul Kruskal, algoritmul Prim generează arborele parțial de cost minim prin extinderea unui singur

subgraf T_s, care inițial este format dintr-un vârf. Subarborele T_s se extinde prin adăugarea consecutivă a muchiilor (v_i, v_j), astfel încât $v_i \in T_s, \quad v_j \notin T_s)$, iar ponderea muchiei adăugate să fie minim posibilă. Procesul continuă până la obținerea unui număr de $n-1$ muchii în T_s.

Pseudocod

Pas 1. Se construieşte $G^* = (V, T)$, $T = \varnothing$.

Pas 2. Muchiile din $G = (V, E)$ se sortează în ordinea creşterii ponderilor. Se obține şirul $m_1, ..., m_{|E|}$. Toate muchiile se consideră *nefolosite*.

Pas 3. $k \leftarrow 0$.

Pas 4. Cât timp $k < |V| - 1$:

a) $i \leftarrow 1$.

b) Dacă muchia $m_i = (v', v'')$ este *nefolosită* şi $v' \in T_s \ \& \ v'' \notin T_s)$ sau $v' \notin T_s \ \& \ v'' \in T_s)$, atunci m_i se adaugă la T_s, m_i se consideră *folosită* şi $k \uparrow$, după care se revine la începutul pasului 4. În caz contrar $i \uparrow$ şi se repetă pas 4.b.

Implementare

Input: Graful G: matricea de adiacență **a**; lista de muchii **v**; arborele parțial de cost minim **arb**, tabloul **c** de stare a nodurilor (se foloseşte pentru a modela starea muchiilor).
Output: Un arbore parțial de cost minim a G, în tabloul **arb.**

```
int sort ()
{ …// functia de sortare a listei de muchii }

int readdata()
{ …// functa de citire a datelor initiale }

int printv()
```

```
{ … // functia de tipar a reyultatelor}

int makeedgelist()
{ …// functia pentru crearea listei de muchii }

int main(int argc, char *argv[])
{ int j,i;
  readdata();
  makeedgelist();
  sort();
  for (i=1;i<=n;i++) c[i]=0;
  c[v[1].vi]=c[v[1].vj]=1; k=1;
  arb[1]=v[1];
while(k<n-1) // cat arboreal nu e construit
{ i=1;
 while (c[v[i].vi]+ c[v[i].vj] != 1 ) i++;
 // se gaseste cea mai scurta muchie care poate fi adaugata
 k++; arb[k]=v[i]; // se adauga muchia
 c[v[i].vj]=c[v[i].vi]=1; // se modifica starea nodurilor
muchiei adaugate
}
printv();
return 0;
}
```

8.3 Probleme rezolvate

Reţele

Se consideră n calculatoare C_1, C_2, ,..., C_n ce trebuie reunite într-o rețea. Într-o astfel de rețea oricare două calculatoare distincte C_i, C_j pot comunica fie printr-o conexiune directă, dacă există, fie prin intermediul altor calculatoare, legate între ele (Fig. 1). Conexiunile rețelei se vor realiza prin cablu. Din considerente tehnologice, cablul ce leagă oricare două calculatoare nu poate avea ramificații.

Elaborați un program care determină lungimea minimală Lmin a cablului, necesar pentru realizarea rețelei. Poziția fiecărui calculator C_i, i=1,2,…,n, este definită cu ajutorul coordonatelor carteziene x_i, y_i.

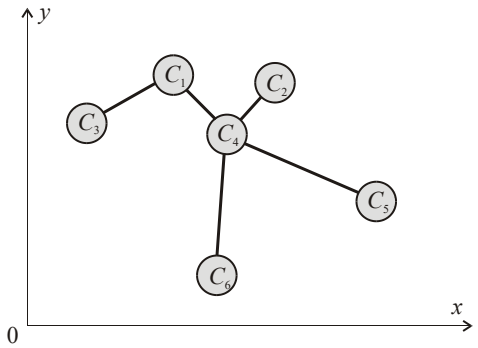


Fig. 1

Input: Fişierul text RETELE.IN conține pe prima linie numărul natural n. Fiecare din următoarele n linii ale fişierului de intrare conține cîte două numere reale separate prin spațiu. Linia i+1 a fişierului de intrare conține coordonatele xi, yi ale calculatorului Ci.

Output: Fişierul text RETELE.OUT va conține pe o singură linie numărul real Lmin scris fără specificatori de format.

Exemplu.

RETELE.IN

```
6
3 8
6 7.8
1.5 7
5 5
8 3
4.5 2
```

RETELE.OUT

```
1.442958120348338E+001
```

Restricții. $2 \le n \le 200$; $-10000 \le x_i, y_i \le 10000$. Timpul de execuție nu va depăşi o secundă. Fişierul sursă va avea denumirea RETELE.PAS, RETELE.C sau RETELE.CPP.

Rezolvare

Rețeaua ce necesită o cantitate minimală de cablu poate fi construită cu ajutorul algoritmului lui Prim:

1. Inițial mulțimea B a calculatoarelor reunite în rețea conține doar calculatorul C_1.

2. În continuare, la fiecare pas al algoritmului:
   a) calculăm distanțele dintre calculatoarele din mulțimea B şi calculatoarele rămase;
   b) includem în mulțimea B calculatorul care se află la cea mai mică distanță.
3. Punctul 2 se repetă pînă ce toate calculatoarele vor fi incluse în mulțimea B.

Amintim că distanța D_{ij} dintre calculatoarele C_i, C_j se calculează conform relației:

$$D_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} .$$

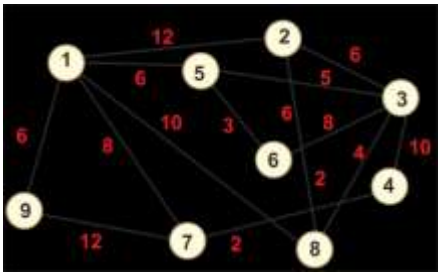
```
Program Retele;
 { Clasele 7-9 }
var X, Y : array[1..200] of Real;
    Total : Real;
    n : Integer;

procedure Citire;
var f : Text;
var i : Integer;
begin
  Assign(f,'RETELE.IN');
Reset(f);
  readln(f, n);
  for i:=1 to n do Readln(f, X[i], Y[i]);
  Close(f);
end;

Procedure Scriere;
var f : Text;
begin
  Assign(f, 'RETELE.OUT');
  rewrite(f);
  Writeln(f, Total);
  close(f);
end;

function Dist(i, j : Integer) : Real;
begin
  Dist:=sqrt(sqr(X[i]-X[j])+sqr(Y[i]-Y[j]));
end;
```

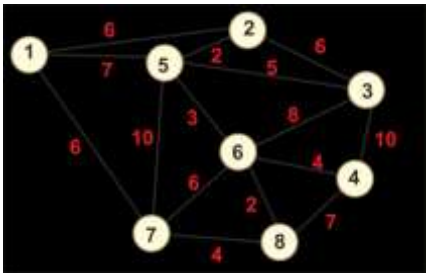
```
function Min(a, b : Real) : Real;
begin
  if a<b then Min:=a else Min:=b;
end;

procedure Prim;
var i, j : Integer;
    B : array[1..200] of Boolean;
    { B[k] indica daca calculatorul k a fost inclus deja in
retea }
    D : array[1..200] of Real;
    { D[k] reprezinta distanta de la calculatorul k la retea
}
    pmin : Integer;
    k : Integer;
begin
  B[1]:=true;
  for i:=2 to n do B[i]:=false;
  for i:=2 to n do
    begin D[i]:=Dist(1, i); end;
  for k:=2 to n do
    begin
      pmin:=0;
      { gaseste cel calculatorul cel mai apropiat de retea, }
      { dare care inca nu a fost inclus in ea }
      for i:=1 to n do
        begin
          if B[i] then continue;
          if pmin=0 then pmin:=i
                    else if D[pmin]>D[i] then pmin:=i;
        end;
      { adauga calculatorul cu indicele pmin la retea }
      Total := Total + D[pmin];
      B[pmin]:=true;
      { actualizarea distantele de la restul calculatoarelor
la retea }
      for i:=1 to n do
        begin
          if B[i] then continue;
          D[i]:=min(D[i], Dist(i, pmin));
        end;
    end;
end; { Prim }
```

```
begin
  Citire;
  Prim;
  Scriere;
end.
```

Din analiza textului programului prezentat mai sus se observă, că complexitatea temporară a algoritmului respectiv este de ordinul $O(n^2)$. Prin urmare, în cazul restricțiilor din enunțul problemei, timpul de calcul nu va depăşi o secundă.

Exerciţii





Des. 8.3

1. Determinați arborii parțiali de cost minim pentru grafurile de pe desenul 8.3.

2. Elaborați un program pentru determinarea tuturor carcaselor unui graf $G = (V, E), \quad |V| \leq 20$.

3. Folosind algoritmul Kruskal, elaborați un program pentru determinarea arborelui parțial de cost minim a unui graf $G = (V, E), \quad |V| \leq 20.()$.

4. Folosind algoritmul Prim, elaborați un program pentru determinarea arborelui parțial de cost minim a unui graf $G = (V, E), \quad |V| \leq 20$.

5. Estimați complexitatea temporală a algoritmului Kruskal.

6. Estimați complexitatea temporală a algoritmului Prim.

7. Estimați complexitatea temporală a algoritmului de generare a tuturor carcaselor unui graf $G = (V, E)$.

Capitolul 9. Cicluri

În acest capitol:

- Numărul ciclomatic al grafului
- Mulţimea fundamentală de cicluri
- Problema Euler, Ciclul eulerian
- Teorema de existenţă a ciclului (lanţului) eulerian
- Algoritmi pentru construirea ciclului (lanţului) eulerian
- Ciclul şi lanţul hamiltonian
- Algoritmi exacţi pentru determinarea ciclului (lanţului) hamiltonian

9.1 Numărul ciclomatic şi mulţimea fundamentală de cicluri

Fie dat graful $G=(V,E)$ cu n vârfuri, m muchii şi p componente conexe.

Valoarea $\rho(G)=n-p$ stabileşte numărul total de muchii, în fiecare din arborii parțiali ai grafului G pe toate componentele de conexitate ale acestuia. În particular, dacă graful este conex, numărul de muchii în oricare arbore parțial va fi egal cu $n-1$.

Valoarea $v(G)=m-n+p=m-\rho(G)$ se numeşte *numărul ciclomatic* al grafului G. Valoarea $\rho(G)$ se numeşte număr *cociclomatic*. Caracteristica ciclomatică a grafului stabileşte numărul maximal de cicluri independente[10], care pot fi construite concomitent pe graf.

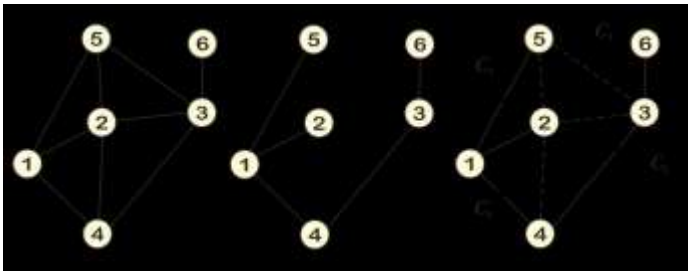
Astfel, dacă se construieşte un arbore parțial T al grafului, apoi se formează cicluri prin adăugarea a câte o muchie a grafului, care nu aparține arborelui T, în final se va obține o mulțime de cicluri $C_1, C_2, ..., C_{v(G)}$, independente între ele. De remarcat că ciclul C_i se

[10] Cicluri independente – dacă conțin cel puțin câte o muchie, care aparține doar unuia din ele

formează prin adăugarea unei muchii (v_j, v_k) la lanțul care unește vârfurile v_j, v_k în arborele T.

Fie dat un arbore parțial T. Mulțimea de cicluri ale grafului G, fiecare dintre care este format prin adăugarea la T a unei muchii din G/T formează mulțimea ciclurilor fundamentale, asociate arborelui T. Oricare două dintre ciclurile fundamentale sunt independente între ele şi orice alt ciclu, care nu face parte din mulțimea ciclurilor fundamentale poate fi reprezentat ca o combinație liniară a acestora.

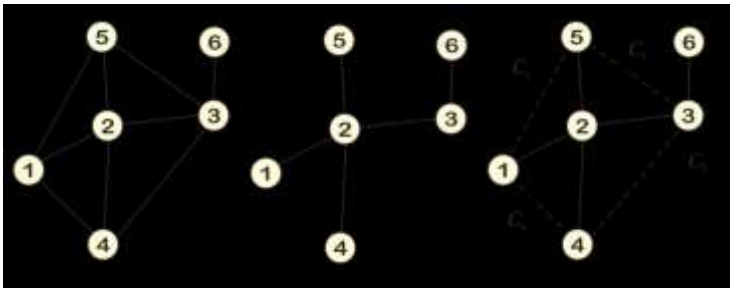
Exemplu. Fie dat graful $G = (V, E)$ şi unul din arborii lui parțiali – T. (desenul 9.1).



Desenul 9.1 Graful *G* (a), un arbore parţial *T* (b), ciclurile fundamentale (c)

Pentru graful G: $n = 6, m = 9, p = 1$. Numărul ciclomatic $v(G) = 9 - 6 + 1 = 4$. Ciclurile fundamentale sunt: C_1, C_2, C_3, C_4

De menționat că ciclurile fundamentale se modifică în dependență de arborele parțial construit (desenul 9.2)

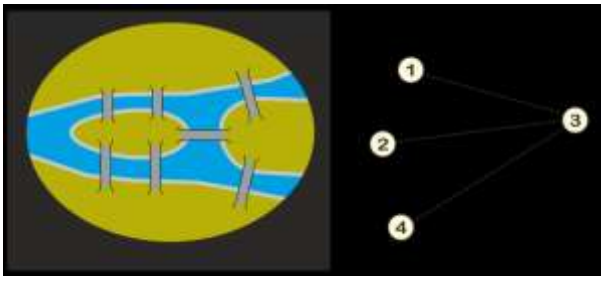


Desenul 9.2 modificarea mulţimii de cicluri fundamentale a grafului din exemplul precedent la selecţia unui alt arbore parţial.

9.2 Cicluri Euler

Problema podurilor din Königsberg

Oraşul Konigsberg (în prezent, Kaliningrad) este traversat de râul Pregel. Zonele oraşului, situate pe ambele maluri ale râului, dar şi pe două insule, erau unite prin şapte poduri, după schema din desenul 9.3. În anul 1736 locuitorii oraşului i-au trimis celebrului matematician Euler o scrisoare, rugându-l să rezolve următoarea problemă: poate fi găsit un traseu, care pornind dintr-un punct dat, să parcurgă toate cele şapte poduri câte o singură dată şi să revină în punctul de pornire?

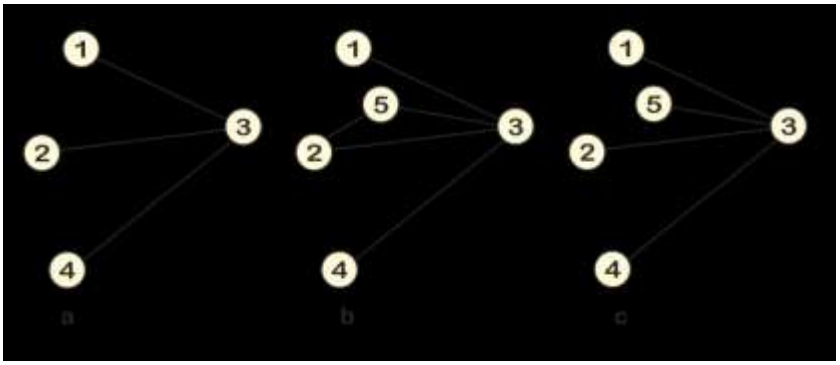


Des. 9.3 Schema amplasării podurilor în Konigsberg şi graful asociat.

Euler a demonstrat imposibilitatea existenței unui asemenea traseu, iar demonstrația respectivă a pus începutul teoriei grafurilor.

Cicluri Euler

Def. Fie $G=(V,E)$ un graf neorientat. *Ciclu eulerian* se numeşte ciclul care trece pe fiecare muchie a grafului G o singură dată. *Lanț eulerian* se numeşte lanțul, care trece exact câte o dată pe fiecare muchie a grafului.



Des. 9.4 Graf fără ciclu şi lanţ eulerian (a), cu ciclu eulerian (b), cu lanţ eulerian (c).

În procesul cercetării problemei despre podurile din Konigsberg a fost formulată următoarea teoremă:

Teorema 1. Un graf conex, neorientat G conține un ciclu (lanț) eulerian atunci şi numai atunci când numărul vârfurilor de grad impar este 0 (0 sau 2 – pentru lanț).

□ ■

Teorema 2. Un graf conex, orientat G conține un ciclu (lanț cu începutul în p şi sfârşitul în q) eulerian atunci şi numai atunci când:

(i) $\forall v_i \in V, \left|\Gamma^+(v_i)\right| = \left|\Gamma^-(v_i)\right|$ - pentru ciclul eulerian

(ii) $\forall v_i \in V, v_i \neq p,q \left|\Gamma^+(v_i)\right| = \left|\Gamma^-(v_i)\right|; \left|\Gamma^+(p)\right| = \left|\Gamma^-(p)\right|+1; \left|\Gamma^+(q)\right| = \left|\Gamma^-(q)\right|-1$

- pentru lanțul eulerian.

□■

9.3 Algoritmul de construcţie a ciclului eulerian

Input. Graful $G=(V,E)$ dat prin matricea de adiacență **a**.

Output: Lista ordonată de parcurgere a vârfurilor L pentru obținerea lanțului eulerian.

Pas 0. $L = \varnothing$.

Pas 1. Este selectat un vârf arbitrar v_0. $L \leftarrow v_0$. vârful curent $v \leftarrow v_0$.

Pas 2. Din vârful v se parcurge o muchie, care nu divizează graful în componente conexe separate. Fie aceasta (v,v'). Muchia (v,v') se exclude din G. $L \leftarrow v'$ $v \leftarrow v'$.

Pas 3. Cât timp în G mai există muchii se revine la pasul 2.

Pas 4. Se afişează lista L.

Implementare (cazul lanţului Euler)

Input. Graful $G = (V,E)$ dat prin matricea de adiacență **a**.

Output: Lista ordonată de parcurgere a vârfurilor S pentru obținerea lanțului eulerian

Structuri de date auxiliare: `transa` - matricea de adiacență a grafului curent `lant` - ciclul de creştere curent, `d` - marcajele vârfurilor la crearea ciiclului

```
int readdata()
{ … // subprogramul de citire a datelor initiale din fisier }
int print( int m)
{ … // subprogramul de afisare a lantului curent }
int verif()
// subprogramul de verificare dacă graful este unul eulerian
// se stabilesc şi vârfurile între care se va construi lanţul
{ int i,j, k;
for (i=1;i<=n;i++)
{   k=0;
    for (j=1;j<=n; j++)
      if (a[i][j] !=0 ) k++;
      if (k%2==1 && s1==0) s1=i; else
      if (k%2==1 && s1 !=0 && t==0) t=i; else
      if (k%2==1 && s1 !=0 && t!=0) return 0;
}
      if (s1==0 && t==0 ) {s1=1; t=2; return 1;}
      if (s1 != 0 && t==0 ) {return 0;}
      return 1;
}
```

```
int lan () // subprogramul de construcţie a primului lanţ
{ int  i, x;
  for (i=1;i<=n;i++) d[i].st=0;
  d[s1].st=1; x=s1;
do
{ for(i=1;i<=n;i++)
      if(a[x][i] !=0 && d[i].st==0)
        {   d[i].st=1; d[i].sursa=x;}
 d[x].st=2;
 k=1; while (d[k].st !=1) k++;
 x=k;
} while (x!=t);
 l=1; k=1;
 s[l]=lant[k]=t;
 while (x !=s1)
 {      x=d[x].sursa;
        l++; k++;
        s[l]=lant[k]=x;
 }
return 1;
}
```

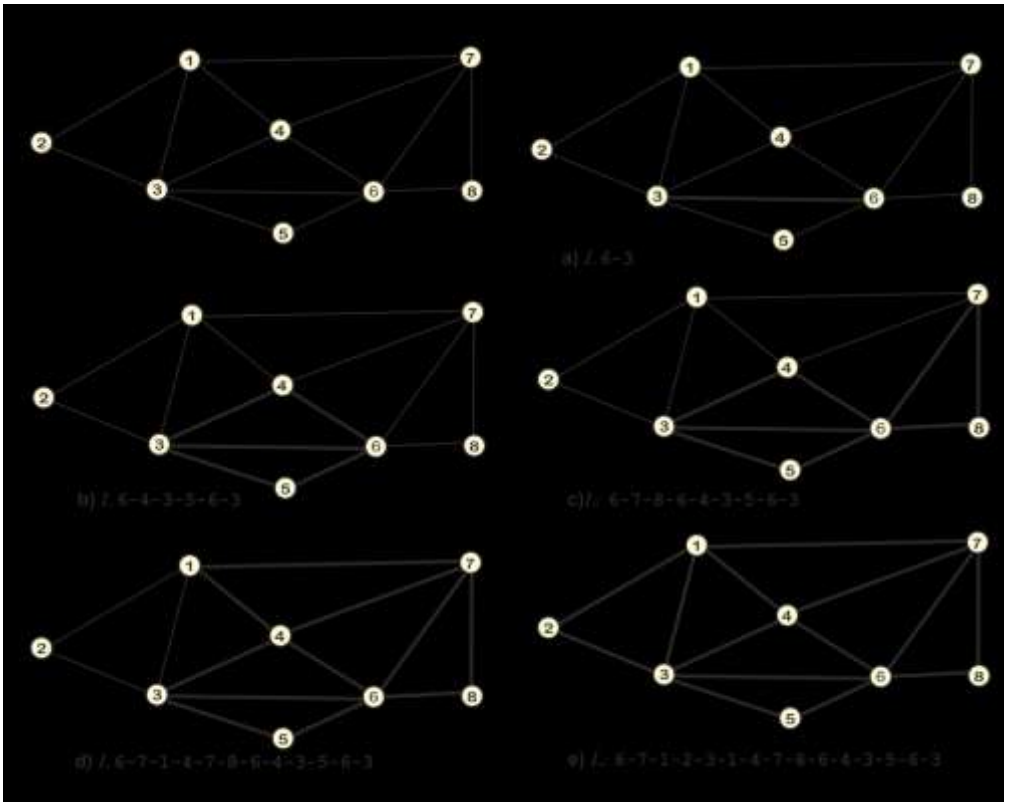
```
int grad(int x)
// subprogram pentru determinarea gradului vârfului în graf
{ int s=0,i;
for (i=1;i<=n; i++)
       if (transa[x][i]!=0) s++;
return s;
}
```

```
int exclude_cic()
// subprogram pentru excluderea ciclului curent din graf
{ int i;
  transa[s[1]][s[l]]=transa[s[l]][s[1]]=0;
  for (i=1;i<l;i++)
      transa[s[i]][s[i+1]]= transa[s[i+1]][s[i]]=0;
  return 1;
}
```

```
int exist()
// verificare a existenţei vârfurilor pentru adăugare
{ int i;
  for (i=1;i<=n;i++) if (grad(i)>0) return i;
  return 0;
}
```

```
int insert(int x) // subprogram pentru inserarea ciclului
curent în lanţ
{  int i,j, news[???];
   for (i=1;i<=x; i++) news[i]=s[i]; i--;
   for (j=1;j<=k; j++) news[i+j]=lant[j];j--;
   for (i=x+1;i<=l;i++) news[i+j]=s[i];
   l+=k;
   for (i=1;i<=l;i++) s[i]=news[i];
   return 1;
}

int primul()
// determinarea primului vârf de creştere a lanţului
{ int i;
      for (i=1;i<=l;i++)
            if (grad(s[i])>0) {pr=i; return s[i];}
      return 0;
}

int ciclu () // determinarea ciclului de creştere a lanţului
{ int  i, x;
  s1=primul();
  for (i=1;i<=n;i++) if (transa[s1][i] > 0) {t=i; break;}
  for (i=1;i<=n;i++) d[i].st=0;
  d[s1].st=1; x=s1;
  transa[s1][t]=transa[t][s1]=0;
do
{ for(i=1;i<=n;i++)
      if(transa[x][i] !=0 && d[i].st==0)
        {   d[i].st=1; d[i].sursa=x;}
  d[x].st=2;
  k=1; while (d[k].st !=1) k++;
  x=k;
} while (x!=t);
  k=1; lant[k]=t;
  while (x !=s1)
  {
        x=d[x].sursa;
        k++; lant[k]=x;
  }
 return 1;
}
```

```
int main()
{ readdata();
  if (verif()==0) {printf ("/n Graful nu este eulerian /n ");
      return 0;}
  lan();
  do {   exclude_cic();
         if (exist()!=0)
          {  ciclu();
             insert(pr);
             print(l);
          }
     } while (exist()>0);
  return 0;
}
```



Des. 9.5 Ilustrarea rezultatelor generate de program. Lanţul iniţial **6-3** (**a**). Cicluri de creştere **6-4-3-5-6 (b)**, **6-7-8-6 (c)**, **7-1-4-7 (d)**, **1-2-3-1 (e).**

9.4 Cicluri şi lanţuri hamiltoniene

Să examinăm mai întâi o situație frecvent întâlnită în viața reală. Mai multe procese industriale presupun prelucrarea unui număr de piese cu ajutorul unei singure instalații. Timpul total şi cheltuielile depind doar de numărul operațiilor de calibrare (ajustare, curățare etc.) a instalației între prelucrarea a oricare două piese p_i, p_j. Mai mult, există perechi de piese, prelucrarea consecutivă a cărora permite funcționarea continuă a instalației, fără calibrare. Prin urmare, atât timpul total de prelucrare, cât şi cheltuielile pot fi minimizate prin selectarea unei consecutivități de prelucrare a pieselor cre va necesita un număr minim (sau nul) de operații de calibrare.

Dacă reprezentăm piesele prelucrate $p_1, p_2, ..., p_n$ prin vârfurile unui graf, iar perechile de piese „compatibile" (p_i, p_j) – prin muchiile lui, atunci determinarea unui ciclu, care trece exact câte o singură dată prin fiecare vârf al grafului este echivalentă cu determinarea unei consecutivități de prelucrare a pieselor care nu necesită nici o operație de calibrare. Problema rămâne aceeaşi, indiferent de faptul dacă graful este unul orientat sau nu.

Ciclul, care trece exact câte o dată prin fiecare vârf al grafului se numeşte *ciclu hamiltonian.*

Există câteva formulări ale problemei, în dependență de tipurile de grafuri şi restricții. În continuare vor fi studiate două probleme distincte:

a. Este dat un graf (ne)orientat . Se cere să se determine unul sau toate ciclurile hamiltoniene, dacă acestea există.
b. Este dat un graf complet cu muchii ponderate (ponderea muchiei $(i, j) \rightarrow c_{i,j}$). Se cere să se determine un ciclu (lanț) hamiltonian de cost minim.

Spre deosebire de cazul ciclului eulerian, pentru ciclul hamiltonian nu există un criteriu exact de determinare a existenței acestuia într-un graf arbitrar.

9.5 Algoritmul pentru determinarea ciclurilor (lanţurilor) hamiltoniene

Pentru problema determinării ciclurilor (lanțurilor) hamiltoniene nu există algoritmi eficienți de complexitate polinomială. Problema este una din clasa NP [11, p. 378]. Prin urmare, un algoritm recursiv, bazat pe tehnica reluării reprezintă o abordare rezonabilă. Inițial, algoritmul propus de Roberts şi Flores [6, p. 249] presupunea abordarea iterativă:

Pseudocod (Roberts-Flores)

Pas 1. Se formează tabloul $M=[m_{i,j}]$ de dimensiuni $(k\times n)$, în care elementul $m_{i,j}$ este cel de al i–ulea vârf (fie v_q), pentru care în $G=(V,E)$ există arcul (v_j,v_q). Numărul de linii în matrice va fi determinat de cel mai mare semigrad de ieşire a vârfurilor din G. S – mulțimea vârfurilor incluse în soluție, în ordinea parcurgerii lor. $S\leftarrow\{\varnothing\}$.

Pas 2. Se alege un vârf arbitrar (fie v_i), de la care începe construcția ciclului (lanțului). Se include în $S\leftarrow S\bigcup \cdot v_i$.

Pas 3. Se alege primul vârf nefolosit din coloana v . Fie acesta v_j Se încearcă adăugarea acestuia la S . Există două cazuri, când un vârf nu poate fi adăugat la mul-țimea S:

a. În coloana v nu există vârfuri nefolosite.
b. Numărul de elemente în S este n. Secvența de vârfuri din S formează un lanț hamiltonian. În acest caz, dacă există un arc (muchie) de la ultimul element din S la primul – a fost

identificat un ciclu hamiltonian. În caz contrar există doar lanțul hamiltonian.

În cazurile (a),(b) se trece la pasul 5, altfel $S \leftarrow S \bigcup$ şi $v \leftarrow v_j$.

Pas 4. Se repetă pasul 3 până la apariția uneia din situațiile descrise în 3(a) sau 3(b).

Pas 5. Fie $S = (v_1, v_2, v_{k-1}, v_k)$. $S \leftarrow S - \{v_k\}$. Dacă $S \neq \{\varnothing\}$, $v \leftarrow v_{k-1}$ apoi se trece la pasul 3, altfel STOP – toate secvențele posibile au fost cercetate.

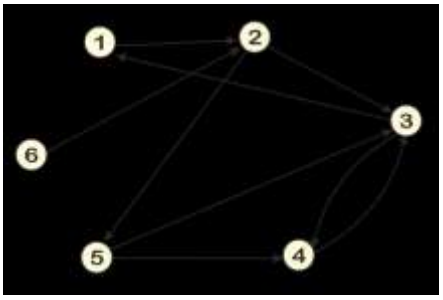
Implementare (recursiv)

Următorul program determină lanțuriler hamiltoniene. Matricea de adiacență este stocată în tabloul a, soluțiile se formează în tabloul s. Programul permite urmărirea dinamicii construcției lanțurilor hamiltoniene. Tehnica de implementare – reluarea.

```
#include <conio.h>
#include <stdio.h>

int a[30][30], m[30][30], s[30], n,i,j,k,ne;
FILE *f;

int readdata()
{ int i,j;
  f=fopen("dataham.in", "r");
  fscanf(f, "%d", &n);
  for (i=1;i<=n;i++)
   for (j=1; j<=n; j++)
    fscanf(f, "%d", &a[i][j] );
  fclose(f);
  return 0;
}

int make_m()
{ int i,j,k;
for (i=1;i<=n;i++)
{ k=0;
  for (j=1;j<=n;j++)
        if ( a[i][j] != 0)
        { k++;
```

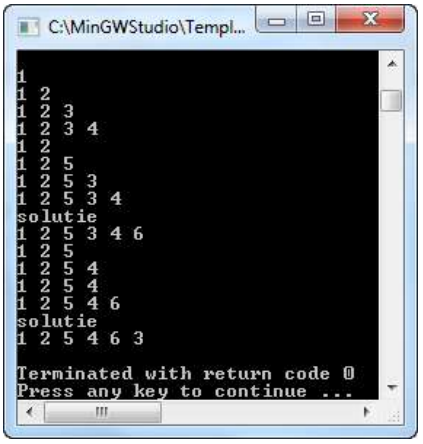
```
          m[k][i]=j;
        }
}
}

int print_s(int q)
{ int i;
  printf("\n");
  for (i=1; i<=q; i++) printf("%d ", s[i]);
  getch();
  return 0;
}
int exist (int a,int pos)
{ int i,k=0;
for (i=1;i<=pos;i++)
      if (s[i] == a) k=1;
return k;
}
int hamilton(int element, int index)
{ int i,j;
  if (exist (element, index-1)) return 0;
  s[index]=element;
  if  (index==n)  {  printf("\n  solutie  ");  print_s(index);
index--; return 0;}
  else { index++;
         j=1;
         do
         {  hamilton(m[j][element], index); j++; }
        while (m[j][element] !=0);
  index--;
  }
  return 0;
}

int main()
{ readdata();
  make_m();
  hamilton(1,1);
  return 0;
}
```

Exemplu



Matricea M pentru graful din imagine este

2	3	1	3	3	1
0	5	4	6	4	2
0	0	0	0	0	3



Exerciţii

1. Elaborați un program pentru verificarea existenței unui ciclu Euler într-un graf $G = (V, E), \quad |V| \leq 20$.

2. Elaborați un program pentru verificarea existenței unui lanț Euler într-un graf $G = (V, E), \quad |V| \leq 20$.

3. Elaborați un program pentru verificarea existenței unui ciclu hamiltonian într-un graf $G = (V, E), \quad |V| \leq 20$.

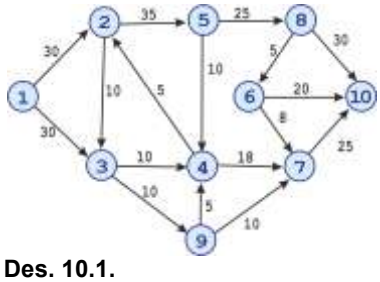
Capitolul 10. Fluxuri în rețea

În acest capitol

- Noţiune de reţea pe graf. Elemente ale fluxului
- Teorema despre tăietura minimă
- Algoritmul Ford-Fulkerson
- Fluxul maxim pentru surse şi destinaţii multiple
- Fluxul maxim pe grafuri bipartite
- Aplicaţii ale problemei de flux maxim

10.1 Preliminarii

Problema fluxului maxim, la fel ca multe alte probleme formulate pe grafuri, îşi are rădăcinile în economia modernă, mai exact în optimizarea economică. Mai recente sunt aplicațiile în domeniul rețelelor şi fluxurilor informaționale. Dacă este cercetată o rețea de transportare a unui material din un punct în care acesta este produs (sursă) într-un punct de depozitare sau prelucrare (stoc) prin canale de transport cu anumite capacități, inevitabil apare problema determinării capacității de transportare a materialului de la sursă către stoc pentru toată rețeaua. Materialul şi canalele de transport pot avea cea mai diversă natură: produse petroliere şi conducte; piese şi transportoare; pachete de date şi canale informaționale, etc.



Des. 10.1.
Reţea de transport cu sursa în vârful `1` şi stocul – în vârful `10`.

Pe grafuri problema se formulează astfel: Este dat un graf orientat $G=(V,E)$, cu ponderi anexate arcelor: pentru arcul (i,j) ponderea este $c_{i,j}$. Pentru două vârfuri date $s,t\in V$ se cere să se

determine fluxul maxim care poate fi deplasat din sursa s în stocul t (Des. 10.1).

10.2.Algoritm

Descriere generală

Algoritmul se bazează pe determinarea iterativă a unor drumuri de creştere a fluxului şi acumularea acestora într-un flux total, până la apariția în rețea a unei tăieturi[11], care separă sursa de stoc.

Se cercetează graful $G=(V,E)$ cu capacitățile arcurilor $c_{i,j}$, sursa s şi stocul t ($s,t\in V$). Pe arcuri se definesc marcajele $e_{i,j}$, care vor indica valoarea curentă a fluxului de-a lungul arcului (i,j). Pentru marcajele arcurilor se va respecta următoarea condiție:

$$\sum_{v_j\in\Gamma(v_i)} e_{i,j} - \sum_{v_k\in\Gamma^{-1}(v_i)} e_{k,i} = \begin{cases} w & v_i = s \\ -w & v_i = t \\ 0 & v_i \neq s,t \end{cases}$$

Prin alte alte cuvinte, sursa s produce un flux de mărime w. Pentru orice vârf intermediar fluxul de intrare este egal cu fluxul de ieşire. În stocul t intră un flux de mărime w.

Problema fluxului maxim se reduce la determinarea unui w:

$$w = \sum_{v_j\in\Gamma^{+}(s)} e_{i,s} = \sum_{v_k\in\Gamma^{-}(t)} e_{k,t} \to \max .$$

Legătura între fluxul maxim şi tăietura minimă în graf este dată de următoarea teoremă:

Teoremă: Într-un graf orientat $G=(V,E)$ valoarea w a fluxului maxim din s în t este egală cu mărimea tăieturii minime $(V_m \to V_m)$, care separă s de t.

[11] Tăietura în graf constă dintr-un set de muchii (arcuri), lichidarea cărora divide graful în două componente conexe.

Tăietura $(V_0 \to V_0)$ separă s de t dacă $s \in V_0, t \notin V_0$. Mărimea tăieturii este suma ponderilor arcurilor din $G=(V,E)$, cu începutul în V_0 şi sfârşitul în V_0, sau $w\left(V_0 \to V_0\right) = \sum_{(v_i,v_j)\in(V_0 \to V_0)} c_{i,j}$.

Tăietura minimă $(V_m \to V_m)$ este tăietura pentru care $w\left(V_m \to V_m\right) = \min_{(V_0 \to V_0)} \left(\sum_{(v_i,v_j)\in(V_0 \to V_0)} c_{i,j} \right)$ □ ■

Notații:

Pentru implementarea algoritmului vor fi folosite atât *marcaje pentru arcuri* cât şi *marcaje pentru vârfurile grafului*. Marcajul unui vârf v este format din trei componente: *precedent*, *flux*, *stare*, care au semnificația:

precedent – vârful care îl precede pe v în drumul de creştere curent

flux – mărimea fluxului, care ajunge în vârful v pe drumul de creştere curent

stare – starea curentă a vârfului v (vârful poate fi în una din trei stări: *necercetat* [marcaj nul, valoarea - 0], *atins*[vecin al unui vârf cercetat, valoarea - 1], *cercetat*[toți vecinii – atinşi, valoarea - 2]).

Vecinii vârfului x

- $\Gamma^+(x)$: $V^+ \subset V : \forall z \in V^+, \exists (x,z) \in E$ (mulțimea vârfurilor în care se poate ajunge direct din vârful x).
- $\Gamma^-(x)$: $V^- \subset V : \forall z \in V^-, \exists (z,x) \in E$ (mulțimea vârfurilor din care se poate ajunge direct în vârful x).

Pseudocod

Pas 0. Marcajele arcurilor se inițializează cu 0.

Pas 1. Marcajele vârfurilor se inițializează: *precedent* ← 0, *stare* ← 0, *flux* ← ∞. Marcajele muchiilor ***se păstrează***.

Pas 2. Inițializarea sursei *s*.

Marcajul sursei *s* : $(+s, +\infty, 1)$[12] Se consideră $v_i \leftarrow s$.

Pas 3. Cercetarea vârfului atins v_i.

a. Pentru toate vârfurile necercetate $v_j \in \Gamma^+(v_i) : e_{i,j} < c_{i,j}$ se aplică marcajul $(+v_i, \Delta, 1)$, unde $\Delta = \min(\Delta_{v_i}, c_{i,j} - e_{i,j})$.

b. Pentru toate vârfurile necercetate $v_j \in \Gamma^-(v_i) : e_{j,i} > 0$ se aplică marcajul $(-v_i, \Delta, 1)$, unde $\Delta = \min(\Delta_{v_i}, e_{j,i})$.

c. Vârful v_i este marcat cercetat. (Componenta *stare* primeşte valoarea 2).

Pas 4.

a. Dacă vârful *t* este *atins*, se trece la pasul 5;(drumul curent de creştere a fost construit), altfel:

b. Dacă există vârfuri *atinse*, dar *t* încă nu este *atins*, este selectat un vârf *atins* v_i şi se revine la pasul 3, altfel:

c. Fluxul maxim a fost obținut. SFÂRŞIT[13].

Pas 5. *Creşterea fluxului. Modificarea marcajelor pe arcuri.*

a. Se consideră $v \leftarrow t$; $\Delta_c = \Delta_t$.

b. Dacă vârful v are marcajul de forma $(+z, \Delta, *)$ valoarea fluxului de-a lungul arcului (*z, v*) este majorată cu Δ_c :

$e_{z,v} \leftarrow e_{z,v} + \Delta_c$,

altfel, dacă vârful v are marcajul de tip $(-z, \Delta, *)$ valoarea fluxului de-a lungul arcului (*v, z*) este micşorată cu Δ_c :

$e_{v,z} \leftarrow e_{v,z} - \Delta_c$;

c. $v \leftarrow z$. Dacă $v \neq s$, se revine la pasul 5.b, altfel la pas 1.

---

[12] *Precedent* – se consideră tot nodul sursă *s*, valoarea *fluxului* în *s* se consideră infinită, *s* se consideră *atins*.

[13] Nu mai există o creştere a fluxului, care ar ajunge la destinaţia (stocul) *t*. Creşterea precedentă a fluxului a determinat tăietura minimă, care separă *s* de *t*.

Exemplu

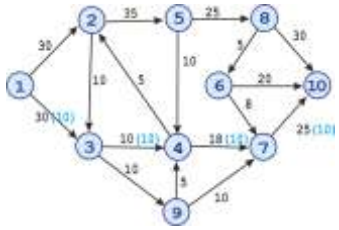
Graful 10.1. Sursa - vârful **1**, stocul – vârful **10**.

ITERAŢIA 1

iniţializare sursa 1: 1-(1, ∞,1)

```
cercetare 1 : 2-(1,30,1); 3-(1,30,1);   1-(1,∞,2 )
cercetare 2 : 5-(2,30,1);               2-(1,30,2)
cercetare 3 : 4-(3,10,1); 9(3,10,1)     3-(1,30,2)
cercetare 4 : 7-(4,10,1);               4-(3,10,2)
cercetare 5 : 8-(5,25,1);               5-(2,30,2)
cercetare 7 : 10 -(7,10,1);             7-(4,10,2)
```

nod	1	2	3	4	5	6	7	8	9	10
precedent	1	1	1	3	2	0	4	5	3	7
flux	∞	30	30	10	30	0	10	25	10	10
stare	2	2	2	2	2	0	2	1	1	1



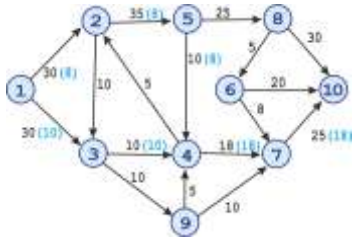
În stoc se ajunge cu un flux de valoare 10. Marcajele arcurilor, care formează drumul de creştere a fluxului (7,10) (4,7) (3,4) (1,3) se modifică.

ITERAŢIA 2

iniţializare sursa 1: 1-(1, ∞,1)

```
cercetare 1 : 2-(1,30,1); 3-(1,20,1);   1-(1,∞,2 )
cercetare 2 : 5-(2,30,1);               2-(1,30,2)
cercetare 3 : 9(3,10,1)                 3-(1,20,2)
cercetare 5 : 4-(5,10,1); 8-(5,25,1);   5-(2,30,2)
cercetare 4 : 7-(4,8,1);                4-(5,10,2)
cercetare 7 : 10 -(7,8,1);              7-(4,8,2)
```

nod	1	2	3	4	5	6	7	8	9	10
precedent	1	1	1	5	2	0	4	5	3	7
flux	∞	30	20	10	30	0	8	25	10	8
stare	2	2	2	2	2	0	2	1	1	1



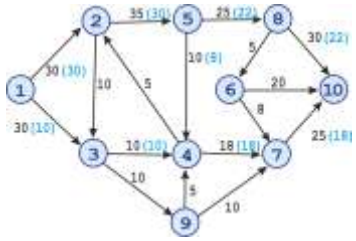
În stoc se ajunge cu o creştere a fluxului de valoare 8. Marcajele arcurilor, care formează drumul de creştere (7,10) (4,7) (5,4) (2,5) (1,2) se modifică.

ITERAŢIA 3

iniţializare sursa 1: 1-(1, ∞,1)

```
cercetare 1 : 2-(1,22,1); 3-(1,20,1);   1-(1,∞,2 )
cercetare 2 : 5-(2,22,1);               2-(1,22,2)
cercetare 3 : 9(3,10,1)                 3-(1,20,2)
cercetare 5 : 4-(5,2,1); 8-(5,22,1);    5-(2,22,2)
cercetare 4 :                           4-(5,2,2)
cercetare 8 : 6-(8,5,1);10 -(8,22,1);   8-(5,22,2)
```

nod	1	2	3	4	5	6	7	8	9	10
precedent	1	1	1	5	2	8	0	5	3	8
flux	∞	22	20	2	22	5	0	22	10	22
stare	2	2	2	2	2	1	0	2	1	1



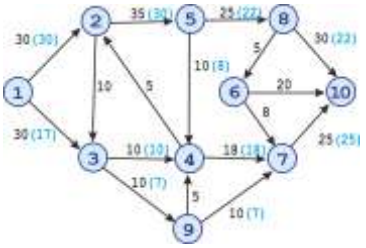
Creşterea fluxului: 22. Marcajele arcurilor, care formează drumul de creştere (8,10) (5,8) (2,5) (1,2) se modifică.

ITERAŢIA 4

iniţializare sursa 1: 1-(1, ∞,1)

```
cercetare 1 : 3-(1,20,1);                1-(1,∞,2 )
cercetare 3 : 9(3,10,1)                  3-(1,20,2)
cercetare 9 : 4-(9,5,1); 7-(9,10,1);     9-(3,10,2)
cercetare 4 : 2-(4,5,1); 5-(-4,5,1)      4-(9,5,2)
cercetare 2 :                            2-(4,5,2)
cercetare 5 : 8-(5,3,1)                  5-(-4,5,2)
cercetare 7 : 10-(7,7,1)                 7-(9,10,2)
```

nod	1	2	3	4	5	6	7	8	9	10
precedent	1	4	1	9	-4	0	9	5	3	7
flux	∞	5	20	5	5	0	10	3	10	7
stare	2	2	2	2	2	0	2	1	2	1



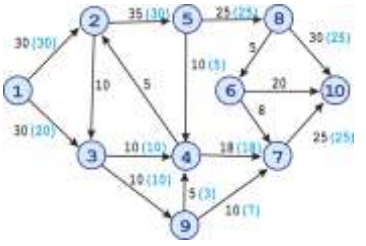
Creşterea fluxului: 7. Marcajele arcurilor, care formează drumul de creştere (7,10) (9,7) (3,9) (1,3) se modifică.

ITERAŢIA 5

iniţializare sursa 1: 1-(1, ∞,1)

```
cercetare 1 : 3-(1,13,1);                1-(1,∞,2 )
cercetare 3 : 9(3,3,1)                   3-(1,13,2)
cercetare 9 : 4-(9,3,1); 7-(9,3,1);      9-(3,3,2)
cercetare 4 : 2-(4,3,1); 5-(-4,3,1)      4-(9,3,2)
cercetare 2 :                            2-(4,3,2)
cercetare 5 : 8-(5,3,1)                  5-(-4,3,2)
cercetare 7 :                            7-(9,3,2)
cercetare 8 : 10-(8,3,1)                 8-(5,3,2)
```

nod	1	2	3	4	5	6	7	8	9	10
precedent	1	4	1	9	-4	8	9	5	3	8
flux	∞	3	13	3	3	3	3	3	3	3
stare	2	2	2	2	2	1	2	2	2	1



Creşterea fluxului: 3. Marcajele arcurilor, care formează drumul de creştere (8,10) (5,8) (5,4) (9,4) (3,9) (1,3) se modifică. Se observă micşorarea fluxului pe arcul (5,4) cu compensarea pe arcurile (3,9)(9,4).
Tot odată poate fi observată şi tăietura, formată de flux(5,8)(7,10). Prin urmare fluxul maxim între nodurile 1 şi 10 are valoarea 50. Următoarea iteraţie nu va mai realiza o creştere a fluxului.

Implementarea algoritmului

Următorul fragment de cod realizează o variantă didactică a algoritmului pentru determinarea fluxului maxim în rețea. Rețeaua din **N** vârfuri este descrisă prin matricea de adiacență **A[N×N]**, marcajele arcurilor (fluxul generat) se păstrează în tabloul **E[N×N]**, marcajele vârfurilor – în tabloul **VERT[N]**.

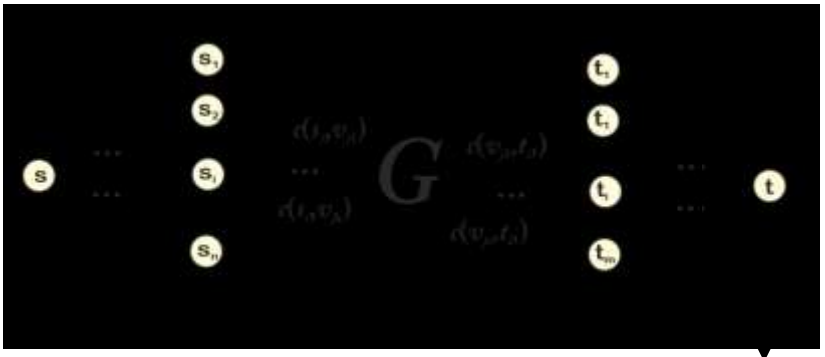
```
#include <conio.h>
#include <stdio.h>

int a[30][30], e[30][30],i,j,n,s,t, delta, big=0;
struct vertex{int flux; int sursa; int stare;} vert[30];
FILE *f;

int readdata()
 { int i,j;
   f=fopen("flux.in", "r");
   fscanf(f, "%d", &n);
   fscanf(f, "%d %d", &s, &t);
    for(i=1;i<=n;i++)
          for(j=1;j<=n;j++)
                  {fscanf(f, "%d", &a[i][j]); big+=a[i][j];}
  fclose(f);
  return 0;
 }

int init_vert()
 { int i;
   for (i=1;i<=n;i++)
     {vert[i].flux=big; vert[i].sursa=vert[i].stare=0;}
   vert[s].stare=1; vert[s].sursa=+s;
   return 0;
 }

int activ()
 {    int i;
      for  (i=1;i<=n;i++)
       if (vert[i].stare ==1) return i;
  return 0;
 }
```

```
int fluxtotal()
 { int i,ft=0;
   for (i=1;i<=n;i++) ft+=e[s][i];
  return ft;
 }

int abs(int x)
{ if (x<0) return x*-1; else return x;
}

int flux()
{ int x,i,d,q;
  // miscarea inainte, constructie lant
  do
    { x=activ();
      //dupa G+
       for (i=1;i<=n;i++)
        if (vert[i].stare==0 && a[x][i]>0 && e[x][i]<a[x][i])
                { d=a[x][i]-e[x][i];
                  if ( d<vert[x].flux) vert[i].flux=d;
                  else vert[i].flux=vert[x].flux;
                  vert[i].stare=1; vert[i].sursa=+x;
                };
      // dupa G-
       for (i=1;i<=n;i++)
        if (vert[i].stare==0 && e[i][x]>0)
                { d=e[i][x];
                  if (d<vert[x].flux) vert[i].flux=d;
                  else vert[i].flux=vert[x].flux;
                  vert[i].stare=1; vert[i].sursa=-x;
                }
     vert[x].stare=2;
    }
 while (vert[t].stare !=1 && activ() !=0 );
   // miscarea inapoi, extinderea fluxului
   delta=0;
 if (vert[t].stare==1 )
      { x=t;
        delta=vert[t].flux;
        do
          { q=abs(vert[x].sursa);
            if (vert[x].sursa>0) e[q][x]=e[q][x]+delta;
            if (vert[x].sursa<0) e[x][q]=e[x][q]-delta;
            x=q;
          }
```

```
        while (x!=s);
 }

int main()
{ readdata();
  do
  { init_vert();
    flux();
  } while (delta !=0);
 printf("\n%d", fluxtotal());
 return 0;
}
```

10.3 Flux maxim cu surse şi stocuri multiple

Fie dat un graf $G=(V,E)$ cu surse (n_s) şi stocuri (n_t)multiple. Se presupune că fluxul poate fi direcționat de la orice sursă la orice stoc. Problema determinării unui flux de mărime maximă de la toate sursele la toate stocurile se reduce la problema clasică a fluxului maxim prin adăugarea unei surse virtuale şi a unui stoc virtual, conectate prin arce la celelalte surse, respectiv, stocuri (desenul 10.2).



$$c(s,s_i)=\sum_{j=1}^{k}c\left(s_i,v_j\right) \qquad c(t_i,t)=\sum_{j=1}^{r}c\left(v_j,t_i\right)$$

Des. 10.2 Adăugarea sursei şi stocului virtual

Pentru fiecare arc (s,s_i) ponderea acestuia, $c\left(s,s_i\right)=\sum_{j=1}^{k}c(s_i,v_j)$ se calculează ca fiind suma ponderilor arcelor cu originea în s_i.

Pentru fiecare arc (t_i,t) ponderea acestuia, $c(t_i,t)=\sum_{j=1}^{r}c\left(v_j,t_i\right)$ se calculează ca fiind suma ponderilor arcelor cu vârful în t_i.

10.4 Flux maxim pe grafuri bipartite

Algoritmul pentru determinarea fluxului maxim poate fi utilizat eficient şi pentru rezolvarea altor probleme pe anumite clase de grafuri. Astfel, problema *cuplajului maxim* pe grafuri bipartite se rezolvă eficient prin adăugarea unei surse virtuale la o parte a grafului şi a unui stoc virtual la cealaltă parte.

Fie $G=(V,E)$ bipartit: $V=V'\bigcup \quad \bigcap$ şi $\forall e=(u,v)\in E \quad u\in V',v\in V''$ sau $v\in V',u\in V''$, neorientat, neponderat.

Transformarea grauluif

Pas 1. Se adaugă două vârfuri *s, t* – inițial izolate. $V=V\bigcup \quad$.

Pas 2. Toate muchiile se transform în arce, orientate de la V' către V''

Pas 3.

a) Pentru fiecare vârf $v\in V'$ se formează arcul c, ponderea căruia este $c_{s,v}=\left|\Gamma^{+}\left(v\right)\right|$.

b) Pentru fiecare vârf $u\in V''$ se formează arcul (u,t), ponderea căruia este $c_{u,t}=\left|\Gamma^{-}\left(u\right)\right|$.

Pas 4. Pe graful astfel format se lansează algoritmul determinării fluxului maxim din s în t.

Pas 5 Rezultat:
Arcele de forma şi $e=(u,v)\in E \quad u\in V',v\in V''$ sau $v\in V',u\in V''$ incluse în fluxul maxim vor forma cuplajul maxim pe muchiile grafului inițial G.

10.5 Flux maxim pe grafuri cu capacităţi restricţionate ale vârfurilor şi muchiilor

Mai multe probleme, rezolvarea cărora implică utilizarea algoritmilor de flux maxim conţin restricţii (caracteristici) asociate nu doar arcurilor grafului ($c_{i,j}$), dar şi vârfurilor acestuia (z_j).

Fie graful $G=(V,E)$ cu ponderile arcelor $c_{i,j}$ $(i=1,...,n; j=1,...,n)$ şi capacităţile vârfurilor z_j $(j=1,...,n)$. Restricţia impusă de capacităţile vârfurilor determină valoarea fluxului total $e_{i,j}$ $(i=1,...,n)$, la intrare în vârful v_j ca $\sum_{i=1}^{n} e_{i,j} \le z_j \quad (j=1,...,n)$.

Se cere determinarea pe graful $G=(V,E)$ a unui flux maxim de la vârful *s* la vârful *t*.

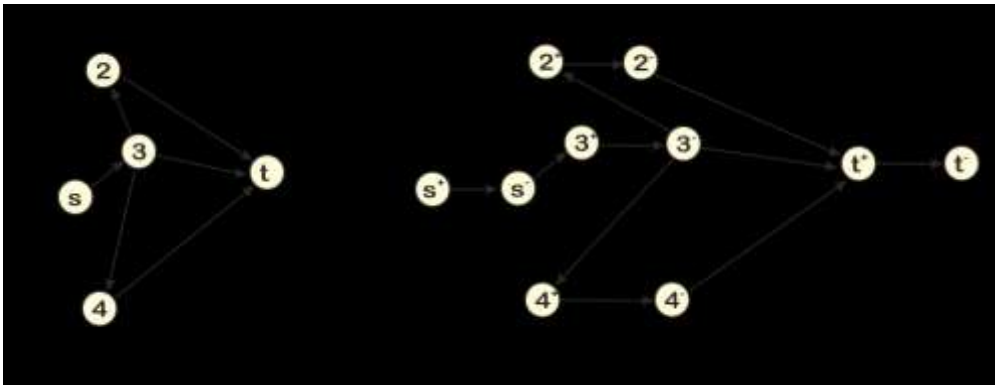
Rezolvarea presupune transformarea grafului iniţial $G=(V,E)$ într-un alt graf $G_0=(V_0,E_0)$, care va conţine doar restricţii la capacităţile arcelor. Pe graful $G_0=(V_0,E_0)$ se va aplica un algoritm clasic pentru determinarea fluxului maxim, care va genera şi soluţia problemei iniţiale.

Modelul de transformare este următorul: fiecare vârf v_j al grafului iniţial este înlocuit cu o pereche de vârfuri v_j^+, v_j^- şi un arc $\left(v_j^+, v_j^-\right)$ care le uneşte. Capacitatea arcului este egală cu capacitatea vârfului $v_j : c\left(v_j^+, v_j^-\right) = z_j$ (desenul 11.3).

Des. 10.3 Transformarea vârfurilor grafului *G*.

Arcele $\left(v_i, v_j\right)$ din graful iniţial se transformă în G_0 în arce $\left(v_i, v_j^+\right)$ cu aceeaşi capacitate. La fel se transformă arcele care îşi au

originea în vârful v_j: (v_j, v_i) trec în (v_j^-, v_i). Un exemplu de transformare este prezentat pe desenul 11.4.



Des. 10.4 Transformarea grafului *G*

După transformările efectuate, fluxul maxim pe graful G_0 va corespunde fluxului maxim pe graful G . Este important de menționat, că în cazul în care tăietura determinată de fluxul maxim pe G_0 nu conține nici unul din arcele formate la transformarea grafului G, restricțiile de capacitate ale vârfurilor nu sunt semnificative.

10.5 Probleme rezolvate

Universitate[14]

Enunț

Pe parcursul anilor în universitatea T&P au fost procurate mai multe echipamente periferice externe, care se conectează la calculator prin diferite porturi, identificate după o valoare numerică (paralel - 1, serial - 2, USB - 3, SCSI - 4, IEEE1394 - 5). Recent rectorul a decis înlocuirea tuturor calculatoarelor din universitate. Calculatoarele noi nu au toate porturile indicate mai sus, întrucât unele din ele au fost aleator excluse pentru a micşora costurile. În consecință, apare problema de reconectare a echipamentelor. Şeful de gospodărie vrea să repartizeze

[14] Concurs .campion, 2009

calculatoarele astfel, încât numărul echipamentelor care rămân neconectate să fie cât mai mic (sau chiar să fie egal cu 0). El are lista calculatoarelor primite, în care pentru fiecare calculator sunt indicate porturile disponibile, şi lista echipamentelor, în care pentru fiecare dispozitiv sunt indicate porturile prin care se poate face conectarea la calculator. Fiecare dispozitiv se conectează la un calculator distinct şi fiecare calculator are conectat nu mai mult de un singur dispozitiv extern.

Scrieți un program, care va ajuta şeful de gospodărie să repartizeze calculatoarele astfel, încât numărul echipamentelor rămase neconectate să fie minim.

Restricții:
$1< M < N < 100$, $0<K\leq 5$, unde M este numărul de dispozitive, N – numărul de calculatoare, iar K – numărul de porturi distincte.

Input
Fişierul text univ.in va avea în prima linie trei numere întregi, separate prin spațiu: N, M, K. Următoarele N linii vor conține de la 0 la K numere, separate prin spațiu: în linia cu indicele i+1 se vor regăsi numerele porturilor, existente în calculatorul cu numărul i. Urmează M linii, care vor conține de la 1 la K numere întregi, separate prin spațiu: linia cu indicele N+1+i va descrie numerele porturilor prin care echipamentul i poate fi conectat la calculator.

Output
Fişierul text univ.out va conține un singur număr întreg – numărul de echipamente ce nu pot fi conectate la calculatoarele noi.

Exemple

Univ.in	Univ.out	Explicaţie
3 3 5 1 3 5 2 4 1 2 3 4 5 1 2 3 4	0	Vom nota C – calculator, E - echipament Unul dintre modurile posibile de conectare ar fi C1 E1, C2 E3, C3 E2. Toate echipamentele sunt conectate.

Univ.in	Univ.out	Explicaţie
4 3 2 1 1 2 1 2 2 1	1	E1 şi E2 pot fi conectate numai la C3. Prin urmare, unul dintre ele rămâne neconectat. E3 poate fi conectat la oricare dintre C1, C2, C4.

Rezolvare

Problema este de tip de flux maxim, care urmează să fie determinat pe un graf „tripartit". Prima componentă a grafului simbolizează calculatoarele, a doua tipurile de conexiuni (porturi), iar a treia – echipamentele periferice. Calculatorul i se uneşte printr-un arc cu portul j, dacă dispune de un asemenea port. Portul j se uneşte printr-un arc cu echipamentul k, dacă echipamentul dispune de un asemenea port. La următoarea etapă o sursa virtuală se conectează la grupul de calculatoare, iar grupul de echipamente este conectat la o destinație virtuală. Pe graful astfel obținut se lansează algoritmul standard de flux maxim.

```
type t2=array[1..207, 1..207] of shortint;
     nod=record
       pr,del,st:integer;
     end;
     t3=array[1..207] of nod;
var e,a:^t2;
    b:t3;
    fmv,fm,nr,delta,s,t,k,n,m:integer;
procedure readdata;
   var i,j: integer;
begin
   assign(input,'universitate.in'); reset(input);
   assign (output, 'universitate.out'); rewrite(output);
   readln(n,m,k);  nr:=n+m+k+2;
   for i:=2 to n+1 do
    begin
     while not seekeoln do
      begin
        read(j);
        a^[i,n+j+1]:=1; a^[1,i]:=1;
```

```
      end;
      readln;
    end;
    for i:=1 to m do
    begin
     while not eoln do
      begin
        read(j);
        a^[n+1+j,n+k+1+i]:=1; a^[i+n+k+1,nr]:=1;
      end;
      readln;
    end;
  close(input);
end;

function sum(t: integer): integer;
var i,s : integer;
 begin
  s:=0;
  for i:=1 to nr do  s:=s+e^[i,t];
  sum:=s;
 end;

procedure init_b;
var i : integer;
begin
  fillchar(b, sizeof(b),0);
  for i:=1 to nr do
  begin
   b[i].del:= 100;
   b[i].pr :=0;
  end;
  b[s].st:=1; b[s].pr:=+s;
end;

function activ : integer;
var i : integer;
 begin
  activ:=0;
  for i:=nr downto 1 do if b[i].st=1 then  activ:=i;
 end;

procedure ford;
 var x,i,d1,q: integer;
begin {ford}
 {miscarea inainte, construim lantul}
  repeat
```

```
   x:=activ;
 {dupa G+}
   for i:=1 to nr do
   if (b[i].st=0) and (a^[x,i]>0) and (e^[x,i]<a^[x,i]) then
   begin
    d1:=a^[x,i]-e^[x,i];
    if d1<b[x].del then b[i].del:=d1 else b[i].del:=b[x].del;
    b[i].st:=1; b[i].pr:=+x;
   end;
{dupa G-}
   for i:=1 to nr do
    if (b[i].st=0) and (e^[i,x]>0) then
   begin
    d1:=e^[i,x];
    if d1<b[x].del then b[i].del:=d1 else b[i].del:=b[x].del;
    b[i].st:=1; b[i].pr:=-x;
   end;
   b[x].st:=2;
  until (b[t].st=1) or (activ=0);
 {miscarea inapoi, extinderea fluxului}
 delta:=0;
 if b[t].st=1 then
    begin   x:=t;
            delta:=b[t].del;
            repeat
             q:=abs(b[x].pr);
             if b[x].pr>0 then e^[q,x]:=e^[q,x]+delta;
             if b[x].pr<0 then e^[x,q]:=e^[x,q]-delta;
             x:=q;
            until x=s;
   end;
end; {ford}

begin
 new(a); fillchar(a^, sizeof(a^), 0);
 new(e); fillchar(e^, sizeof(e^), 0);
 readdata;
 s:=1; t:=nr; fm:=0;
repeat
  init_b; ford; fmv:=fm; fm:=sum(t);
until (delta=0) or (fm-fmv=0);
writeln(m-fm);
close(output);
end.
```

Exerciţii

1. Elaborați un program pentru separarea mulțimii de vârfuri a unei rețele de transport în două componente distincte: mulțimea de surse şi mulțimea de stocuri.

2. Elaborați un program pentru determinarea fluxului maxim pe grafuri cu surse şi destinații multiple.

3. Elaborați un program pentru transformarea grafului cu restricții aplicate pe vârfuri într-un graf cu restricții aplicate pe muchii.

4. Elaborați un program pentru determinarea fluxului maxim pe grafuri cu restricții aplicate pe vârfuri.

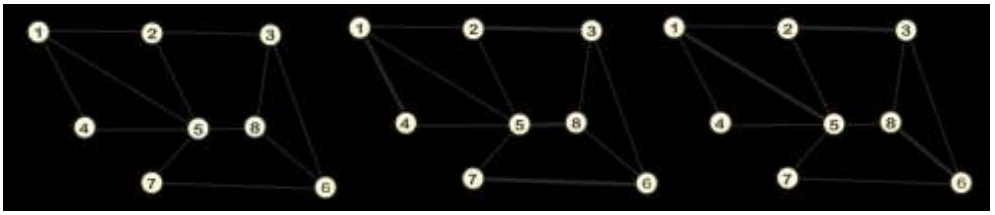
Capitolul 11. Cuplaje

În acest capitol

- Noţiune de cuplaj, cuplaj maxim
- Grafuri asociate
- Legătura între problema cuplajului maxim şi a mulţimii maximal independente
- Algoritmi pentru determinarea tuturor cuplajelor maxime
- Complexitatea algoritmului pentru determinarea tuturor cuplajelor maxime

11.1 Cuplaje

Într-un graf neorientat $G=(V,E)$ mulțimea de muchii M se numeşte *cuplaj* dacă oricare două muchii din M nu an vârfuri comune. Cuplajul M se numeşte *maxim*, dacă $\forall e' \in E-M$, în mulțimea $M \cup \{e'\}$ există muchia $e'': e' \cap e'' = v^*; v^* \in V$. La fel ca şi mulțimile independente maxime, cuplajele maxime pot fi formate din mulțimi cu număr diferit de elemente (vârfuri – pentru mulțimile maximal independente, muchii – pentru cuplaje). Un astfel de exemplu este prezentat pe desenul 11.1.



a **b** **c**

Des. 11.1 Cuplaje maxime în graful **(a)**. Cuplaj maxim pe patru muchii (1,4) (2,3) (5,8) (6,7) **(b).** Cuplaj maxim pe trei muchii (1,5) (2,3) (6,8) **(c)**

Pe un graf arbitrar pot fi formulate mai multe probleme, rezolvarea cărora necesită determinarea unui cuplaj cu proprietăți prestabilite. Pentru unele categorii de grafuri, roblema cuplajului maxim poate fi rezolvată eficient, de exemplu, în cazul grafurilor bipartite, în care ea se reduce la problema fluxului maxim.. În cazul grafurilor arbitrare se aplică aşa nuimitul algoritmul maghiar [6, p. 396].

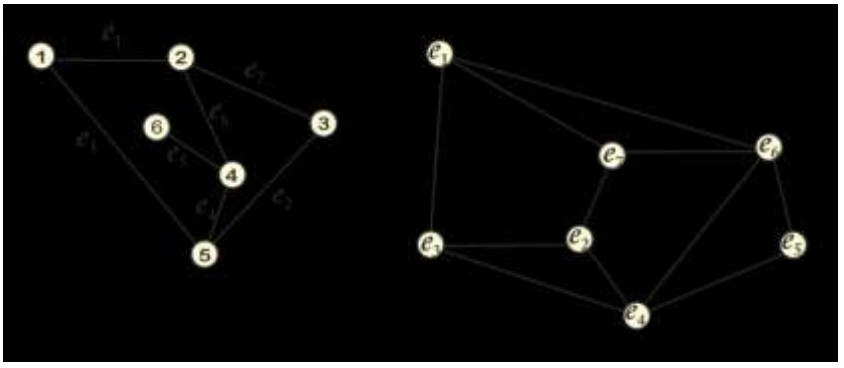
În continuare va fi studiată problema comună de generare a tuturor cuplajelor unui graf neorientat arbitrar $G=(V,E)$.

Problema va fi rezolvată prin reducerea în timp pătratic la o altă problemă pe grafuri, rezolvarea căreia este deja cunoscută – problema mulțimii maximal independente, mai exact – *problema generării tuturor mulțimilor maximal independente*.

11.2 Graful asociat

Fie dat graful neorientat $G=(V,E)$ $E=\{e_1,...,e_M\}$. Se va construi graful $G_A=(V_A,E_A)$, unde $V_A=\{e_1,...,e_M\}$, iar $E_A=\{(e_i,e_j):\exists u\in V, u\in e_i \,\&\, u\in e_j\}$.

Exemplu. Se consideră graful $G=(V,E)$ din figura 11.2.a. Fiecare muchie din graful inițial se transformă într-un vârf al grafului asociat $G_A=(V_A,E_A)$, prezentat pe desenul 11.2.b.



a **b**

Des. 11.2 Graful iniţial (a) şi asociat (b).

Oricare două vârfuri (e_i, e_j) din $G_A = (V_A, E_A)$ sunt conectate prin muchie, dacă în graful inițial există un vârf comun u al muchiilor e_i, e_j.

Teoremă. Fiecărui cuplaj maxim în graful $G = (V, E)$ îi corespunde o mulțime maximal independentă în $G_A = (V_A, E_A)$ şi invers, fiecărei mulțimi maximal independente în $G_A = (V_A, E_A)$ îi corespunde un cuplaj maxim în $G = (V, E)$.

□ ■

11.3 Funcţia de generare a grafului asociat

Din cele expuse mai sus rezultă că problema determinării tuturor cuplajelor maxime pe $G = (V, E)$ este echivalentă cu problema determinării tuturor mulțimilor maxim independente pe $G_A = (V_A, E_A)$.

Pentru a rezolva problema este necesar, mai întâi, să se construiască graful asociat $G_A = (V_A, E_A)$.

Algoritm:

Pas 1. Se creează lista muchiilor din $G = (V, E)$. $L = \{e_1, ...e_m\}$.

Pas 2. Se creează tabloul bidimensional E' – matricea de adiacență a grafului G_A.

Pas 3. Pentru toți i de la 1 la m-1.

Pentru toți j de la i+1 la m

Dacă în $G = (V, E)$ $e_i \bigcap (e_i, e_j \in E)$,

atunci $E'_{i,j} \leftarrow E'_{j,i} \leftarrow 1$ altfel $E'_{i,j} \leftarrow E'_{j,i} \leftarrow 0$.

Implementare

Intrare: graful $G = (V, E)$, descris în tabloul bidimensional **a**.

Ieşire: matricea de adiacență a grafului $G_A = (V_A, E_A)$. Matricea de adiacență este localizată în tabloul bidimensional `b`.

```
int asociat ()
{ int i,j;
// modelare lista muchii
 for (i=1;i<=n;i++)
        for (j=1+i; j<=n; j++)
            if (a[i][j]!=0 ){m++; list[m].v1=i;list[m].v2=j;}
// modelare matrice adiacenta
 for (i=1;i<=m;i++)
        for (j=1+i; j<=m; j++)
            if (list[i].v1==list[j].v1 ||
            list[i].v1==list[j].v2 || list[i].v2==list[j].v1
            || list[i].v2==list[j].v2  ) b[i][j]=b[j][i]=1;
}
```

11.4 Generarea tuturor cuplajelor maxime

Problema se rezolvată direct prin aplicarea consecutivă a doi algoritmi descrişi anterior. Mai întâi se formează graful asociat $G_A = (V_A, E_A)$, apoi pentru acesta este aplicat algoritmul de generare a mulțimilor maxim independente. La generarea fiecărei soluții, vârfurile din $G_A = (V_A, E_A)$ sunt transformate în muchiile corespunzătoare din $G = (V, E)$.

Exemplu: Program-prototip pentru generarea tuturor cuplajelor maxime.

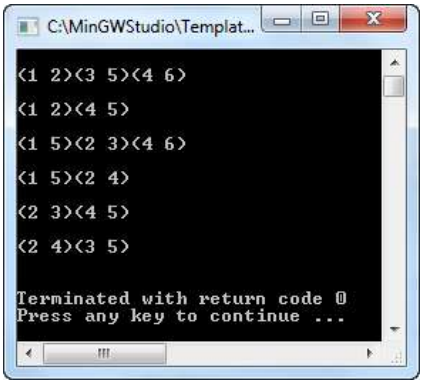
Intrare: graful $G = (V, E)$.
Ieşire: toate cuplajele maxime ale grafului $G = (V, E)$.

```
#include <conio.h>
#include <stdio.h>
struct edge{int v1; int v2;} list[400];
int a[20][20],b[400][400], q[20], s[20], m=0,i,j,n,k;
FILE *f;
```

```
int asociat ()
{ int i,j;
  for (i=1;i<=n;i++)
        for (j=1+i; j<=n; j++)
        if (a[i][j]!=0 ){m++; list[m].v1=i;list[m].v2=j;}
  for (i=1;i<=m;i++)
        for (j=1+i; j<=m; j++)
         if (list[i].v1==list[j].v1 || list[i].v1==list[j].v2
         || list[i].v2==list[j].v1 || list[i].v2==list[j].v2
         ) b[i][j]=b[j][i]=1;
}

int fillc(int *x, int z, int num)
{ int i;
  for (i=1;i<=num;i++) x[i]=z;
  return 0;
}

int print()
{ int i;
printf("\n");
   for (i=1;i<=m;i++)
   if (s[i]==1) printf("(%d %d)", list[i].v1, list[i].v2);
  printf("\n");
}

int mind(int *s, int *q)
{ int i,j,k=0,r, rest[20];
   for (i=1;i<=m;i++) if(q[i]!=0) k=1;
   if (k==0) {print();}
   else { j=m;
          while (s[j]==0 && j>=1) j--;
          r=j+1;
          for (i=r;i<=m;i++)
           if (q[i]==1){ fillc(rest,0,m);
                         s[i]=1; q[i]=0;
                         for (j=1;j<=m;j++)
                            if (b[i][j] != 0 && q[j]==1)
                              {q[j]=0; rest[j]=1;}
                         mind (s,q);
                         s[i]=0;q[i]=1;
                         for (j=1;j<=m;j++)
                           if(rest[j]==1) q[j]=1;
                        }
```

```
        }
    return 0;
   }

int main()
{
f=fopen("data.in", "r");
fscanf(f, "%d", &n);
 for (i=1;i<=n;i++)
       for (j=1;j<=n;j++)
              fscanf(f, "%d", &a[i][j]);
 fclose(f);
 asociat();
 fillc(s,0,m);
 fillc(q,1,m);
 mind(s,q);
return 0;
}
```

Rezultate:
Pentru graful din desenul 11.2 (a) se obțin cuplajele:



11.5 Probleme rezolvate

Poliţiştii

Enunț. Străzile din New Chişinău sunt deosebite – ele sunt rectilinii şi se intersectează doar în punctele extreme. Astfel, o stradă se extinde de la o intersecție la alta. Aceasta tentează mai mulți şoferi să organizeze curse la viteză pe străzi.
Recent Domnul Ghiță a devenit comisar al poliției municipale şi a decis să facă ordine în oraş. El planifică să amplaseze polițişti doar pe unele străzi, astfel încât ei să poată monitoriza toate străzile din oraş. Fiecare polițist se poate deplasa pe strada „sa" de la intersecție la intersecție. Aflându-se în intersecție, polițistul poate controla şi străzile adiacente, fixând încălcările de pe aceste străzi cu ajutorul unui iPad. Dacă doi

polițiști se întâlnesc într-o intersecție, ei încep să discute şi, cu regret, cam uită de obligațiunile de serviciu. Comisarul Ghiță doreşte să excludă posibilitatea apariției unor asemenea situații, dar, totodată, să minimizeze numărul total de polițiști.

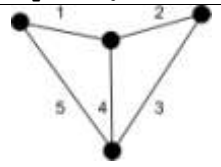
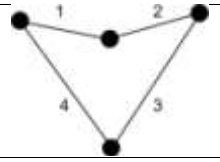
Cerință. Scrieți un program care determine numărul minim de polițiști, necesari pentru a controla toate străzile din oraş, dar, totodată, oricare doi polițiști să nu se poată întâlni.

Input. Fişierul de intrare *police.in* conține pe prima linie numărul N de străzi din oraş. Urmează N linii, care conțin de la 1 la N-1 numere naturale separate prin spații, descriind străzile adiacente. Linia i+1 a fişierului de intrare conține lista străzilor, care sunt adiacente străzii cu numărul i.

Output. Fişierul de ieşire *police.out* va conține un singur număr – numărul minim de polițiști, necesari pentru a monitoriza toate străzile din oraş.

Restricții 2<N<40.

Exemplu:

police.in	police.out	Explicații
5 2 4 5 1 3 4 2 4 5 1 2 3 5 1 3 4	1	
4 2 4 1 3 2 4 1 3	2	

Rezolvare

Rețeaua de drumuri şi intersecții reprezintă un graf planar. Soluția problemei reprezintă un cuplaj maxim de putere minimă (format dintr-un număr minim de muchii). Prin urmare, este suficient să se construiască graful asociat rețelei de străzi, apoi, pe graful construit să se determine mulțimea maximal independentă de putere minimă.

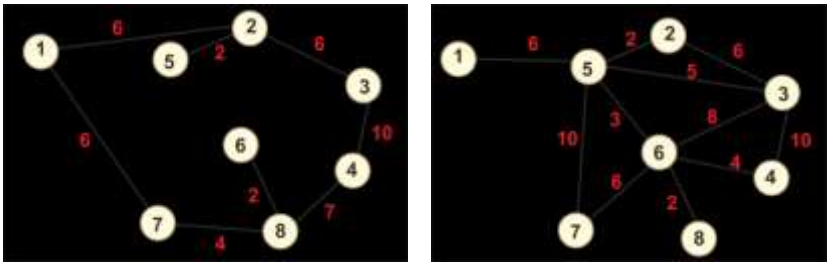
Modul inițial de prezentare a datelor permite construcția directă a matricei de adiacență a grafului asociat.

Implementare

```
var a:array[1..51,1..51] of shortint;
    q,s: array[1..51] of shortint;
    n:integer;
procedure readdata;
var f:text;
    i,k: integer;
begin
assign(f,'police00.in'); reset(f);
readln(f,n);
for i:=1 to n do
begin
 while not eoln(f) do
  begin
   read(f,k);
   a[i,k]:=1;
  end;
readln(f);
end;
end;

procedure print;
var i:integer;
begin
 for i:=1 to n do
  if s[i]=1 then write(i,' ');
 writeln;
end;

procedure mind;
var i,j,k,r :integer;
    rest: array[1..50] of shortint;
begin
k:=0;
for i:=1 to n do if q[i]<>0 then k:=1;
if k=0 then begin write('solutie :'); print; end
       else begin
            j:=n;
            while (s[j]=0) and (j>=1) do dec(j);
            r:=j+1;
            for i:=r to n do
              if (q[i]=1) then
```

```
                        begin
                        fillchar(rest, sizeof(rest),0);
                        s[i]:=1; q[i]:=0;
                        for j:=1 to n do
                         if (a[i,j]<>0) and (q[j]=1) then
                          begin
                           q[j]:=0; rest[j]:=1;
                          end;
                        mind;
                        s[i]:=0; q[i]:=1;
                        for j:=1 to n do
                        if rest[j]=1 then q[j]:=1;
                        end;
            end;
end;

begin
readdata;
fillchar(s, sizeof(s),0);
fillchar(q, sizeof(q),1);
mind;
readln;
end.
```

Exerciţii



A **B**

Des. 11.3

1. Determinați cuplajele maxime de putere maximală în grafurile de pe desenul 11.3.

2. Determinați cuplajele maxime de putere minimală în grafurile de pe desenul 11.3 .

3. Elaborați un program pentru generarea tuturor cuplajelor unui graf arbitrar $G = (V, E), |V| \le 20$, fără a construi grafulasociat.

4. Elaborați un program pentru determinarea cuplajului cu un număr minim de muchii al unui graf arbitrar $G = (V, E), |V| \le 20$.

5. Elaborați un program pentru determinarea cuplajului cu un număr maxim de muchii al unui graf arbitrar $G = (V, E), |V| \le 20$.

Capitolul 12. Probleme propuse pentru rezolvare

14.1 Cratere pe Lună[15]

Enunț

Ciocnirile asteroizilor cu Luna sunt destul de frecvente. În urma ciocnirilor se formează cratere circulare.

Astronomul Tudor a început studierea hărții suprafeței selenare. Observând că unele cratere sunt incluse unul în altul, el a hotărât să determine cea mai lungă consecutivitate de cratere incluse. Pentru a rezolva această problemă el are nevoie de un program de calculator.

Intrare

Prima linie a fişierului de intrare conține numărul întreg N — numărul de cratere de pe hartă ($1 \le N \le 500$). Următoarele N linii conțin descrierile craterelor de la 1 la N. Fiecare crater e descris în o linie aparte, descrierea fiind formată din trei numere din intervalul [–32768, 32767], separate prin spațiu. Primele două numere sunt coordonatele carteziene ale centrului, al treilea – raza.

Ieşire

Prima linie a fişierului de ieşire va conține lungimea celui mai lung şir de cratere incluse, cea de a doua – indicii craterelor din serie, începând cu cel mai mic, până la cel mai mare (după diametru). Numerele craterelor se separă prin spațiu. Dacă există mai multe soluții, se afişează oricare din ele.

Exemplu

Crater.in	Crater.out
4 0 0 30 -15 15 20 15 10 5 10 10 10	3 3 4 1

[15] Rusia, Şcoala de vară la Informatică, 1999

12.2 Translatori[16]

Enunț

Pentru organizarea unui simpozion internațional o companie de Relații Publice are nevoie să angajeze translatori din limbile native ale participanților în limba engleză. Baza de candidați este formată din N persoane, pentru fiecare din ei fiind cunoscute limbile în care el poate efectua traducerea. Toți translatorii beneficiază de salarii egale. Fiind cunoscut setul S de limbi în care urmează să fie efectuate traduceri, să se determine un număr minim de translatori, care vor asigura toate traducerile.

Intrare

Prima linie a fişierului de intrare conține numărul întreg N — numărul de translatori (1≤N≤100). Următoarele N linii ale fi;ierului de intrare conțin de la 1 la 200 numere întregi pozitive – indicii limbilor în care translatorul poate efectua traducerea. Linia i+1 a fi;ierului d eintrare con'ine infoația despre limbile în care poate efectua traducerea translatorul i. Urmează o linie cu cel mult 300 numere întregi pozitive, separate prin spațiu – indicii limbilor utilizate la simpozion.

Ieşire

Fişierul de ieşire va conține în prima linie numărul întreg K – numărul de translatori necesari. Cea de a doua linie va conține indicii translatorilor selectați, în ordine lexicografică, separați prin spațiu. Dacă există mai multe soluții, se afişează oricare din ele.

Exemplu

Translate.in	Translate.out
5 1 2 4 2 3 6 1 5 6 1 4 7 6 7 1 2 3 4 5 6 7	3 2 3 4

[16] [6, p 63.]

12.3 Problema celor cinci dame[17]

Enunț
Pe o tablă de şah cu dimensiune standard să se amplaseze cinci dame astfel încât ele să țină sub lovitură toată suprafața tablei de şah.
Scrieți un program care va determina toate posibilitățile de amplasare a damelor.

Ieşire
Fişierul de ieşire va conține câte cinci linii pentru fiecare soluție identificată. Fiecare linie va conține câte două numere separate prin spațiu: indicele liniei şi coloanei la intersecția cărora se amplasează dama.

12.4 Împărţirea administrativ-teritorială[18]

Enunț
Trecerea de la județe la raioane şi invers impune necesitatea de a tipări harta republicii la fiecare împărțire teritorială nouă. Pentru a micşora prețul hărților, la colorarea lor se utilizează un număr cât mai mic de culori. Evident, la colorarea oricăror două unități teritoriale învecinate (care au frontieră comună) pot fi utilizate doar culori ce diferă. Fiind cunoscută împărțirea teritorial-administrativă, să se determine numărul minim de culori, necesare pentru a colora harta.

Intrare
Fişierul de intrare conține pe prima linie numărul întreg n – numărul de unități administrativ-teritoriale ($n < 101$). Fiecare din următoarele n linii ale fişierului de intrare conține câte n numere întregi (0 sau 1), separate prin spațiu. Liniile resopective formează matricea de adiacență a unităților teritorial-administrativ.

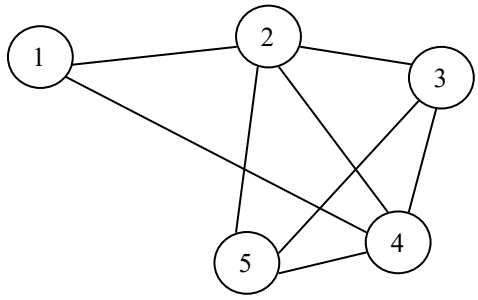
[17] [6, p 70.]
[18] Republica Moldova, Şcoala de vară la Informatică, 2002

Ieşire

Fişierul de ieşire va conține pe o singură linie numărul minim de culori, necesare pentru colorara hărții.

Exemplu:

```
color.in        color.out
5               4
0 1 0 1 0
1 0 1 1 1
0 1 0 1 1
1 1 1 0 1
```



12.5 Safeuri[19]

Enunț

Încăperea securizată ce conține safeurile personale ale Băncii "Gringotts Ltd" are forma unui dreptunghi cu laturile n×m şi este divizată în n × m platforme pătrate de dimensiunea 1 × 1 unități de lungime. Unica platformă liberă este cea din colțul stânga sus. Ea marchează ieşirea din încăpere. Unele platforme sunt ocupate cu echipamente de supraveghere, montate inamovibil, iar pe celelalte se află câte un safeu, dimensiunile căruia coincid cu cele ale platformei. În unul din safeuri se păstrează Piatra Filozofală. La cererea lui Dumbledore, spiriduşii trebuie să scoată din incăpere safeul cu Piatra Filozofală. Unicul tip de operațiii pe care le pot efectua spiriduşii constă în deplasarea oricărui safeu de pe platforma pe care el se află pe o platformă învecinată, cu condiția că aceasta este liberă. Sunt considerate ca fiind învecinate doar platformele ce au o latură comună.

[19] F11, Ediţia I, 2011, Iaşi, România.

Cerință
Scrieți un program, care determină numărul minim de operații necesare pentru a scoate safeul cu Piatra Filozofală din încăperea securizată.

Intrare
Prima linie a fişierului de intrare conține numerele întregi n şi m — dimensiunile încăperii ($1 \leq n, m \leq 50$). Urmează n linii a câte m simboluri fiecare. Simbolul "." marchează poziția liberă. Unica poziție liberă este intrarea în depozit. Simboul "#" marchează o platformă cu echipament de supraveghere. Platformele cu echipamente de supraveghere nu pot fi deplasate şi pe ele nu se plasează safeuri. Simbolul "c" semnifică o platformă cu safeu. Simbolul "X" — platforma pe care se află safeul cu Piatra Filozofală. Safeul se consideră scos din încăpere, dacă el este adus pe platforma ce reprezintă intrarea în depozit. Se garantează, că cel puțin unul dintre numerele n, m este mai mare decât 1, iar fiecare dintre simbolurile "." şi "X" apar doar câte o singură dată. Simbolul "." este întotdeauna amplasat în colțul stâng sus al încăperii securizate.

Ieşire

Dacă safeul nu poate fi scos din încăperea securizată, în fişierul de ieşire se va înscrie cuvântul „Impossible".
În caz contrar unica linie a fişierului va conține un singur număr – numărul minim de operații, necesar pentru scoaterea safeului cu Piatra Filozofală din încăperea securizată.

Exemple

Safeu.in	Safeu.out
3 3 .#X ccc c#c	Impossible
2 3 .cX ccc	7

14.7 Plicuri şi felicitări[20]

Enunț

Administrația unei instituții doreşte să felicite conducerea țării cu Ziua Independenței. Pentru aceasta au fost cumpărate N felicitări şi M plicuri. Din păcate, atât plicurile cât şi felicitările au dimensiuni diferite şi unele felicitări nu pot fi puse în oricare plic.

Scrieți un program care determină o astfel de repartiție a felicitărilor în plicuri, încât numărul de plicuri ce vor conține felicitări să fie maximal. În fiecare plic se poate pune doar o singură felicitare.

Intrare

Prima linie a fişierului de intrare conține numerele întregi M şi N, separate prin spațiu. M reprezintă numărul de plicuri, iar N număruul de felicitări, 0≤M, N≤100). Următoarea linie a fişierului de intrare conține M perechi de numere întregi, separate prin spațiu, ce reprezintă dimensiunile plicurilor (înălțimea şi lățimeaa). Urmează o linie ce conține N perechi de numere întregi separate prin spațiu, ce reprezintă în acelaşi format dimensiunile felicitărilor. Toate dimensiunile sunt numere naturale ce nu depăşesc valoarea 32767.

Ieşire

Prima linie a fişierului va conține numărul întreg K — numărul maxim de felicitări, care pot fi trimise prin poştă. Cea de a doua linie va conține K perechi de numere, cu indicii felicitării şi a plicului în care felicitarea respectiva va fi pusă.

Exemplu

Input	Output
4 4 3 3 141 282 282 141 201 100 3 1 140 280 141 282 201 1	4 **1 1 2 3 3 2 4 4**

[20] Rusia, Şcoala de vară la Informatică, 1999

12.7 Agenţii[21]

Enunț

Din cauza arestării de către eventualul inamic a unui număr mare de agenți ce activau în Mioritic Land, Very Intelligense Agency a hotărât sa îmbunătățească regulile de activitate a acestora. Cea mai mare problemă este asigurarea întâlnirilor sigure ale agenților. Siguranța urmează să fie asigurată pe o rețea prestabilită de drumuri şi pentru anumite poziții inițiale ale agenților. Pentru a organiza o întâlnire sigură, agenții vor respecta următoarele reguli:

- agenții se deplasează ziua, iar întâlnirile au loc seara;
- agentul trebuie să-şi schimbe locul de aflare în fiecare zi;
- agenții circulă doar dea lungul drumurilor indicate (unele din aceste drumuri au o singură direcție de deplasare);
- un agent nu poate să se deplaseze prin mai multe oraşe într-o singură zi, (doar într-un oraş vecin cu cel în care se află dimineața);
- distanța între orice două oraşe legate printr-un drum poate fi parcursă timp de o zi (până la venirea serii);
- întâlnirea are loc doar atunci când ambii agenți se află în acelaşi oraş în aceeaşi seară.

Scrieți un program, care:

- citeşte numărul de oraşe şi descrierea rețelei de drumuri din fişierul AGE.IN;
- verifică dacă există posibilitatea unei întâlniri sigure şi dacă da – de câte zile este nevoie pentru a o organiza;
- scrie rezultatul în fişierul AGE.OUT.

Intrare

Prima linie a fişierului de intrare AGE.IN, conține numerele întregi n şi m, separate prin spațiu,l $1 \le n \le 250$, $0 \le m \le n(n-1)$.

Linia a doua a fişierului de intrare conține numerele întregi a_1 şi a_2, separate prin spațiu, $1 \le a_1, a_2 \le n$ şi $a_1 \ne a_2$. Numerile respective reprezintă pozițiile agenților Nr. 001 şi Nr. 002.

[21] Polonia, Olimpiada națională la Informatică, 2000

Fiecare din următoarele m linii ale fişierului de intrare conține câte două numere *a* şi *b* separate prin spațiu $1 \le a,b \le n$ şi $a \ne b$, numere ce indică existența unui drum de la *a* la *b*.

Ieşire

Fişierul de ieşire AGE.OUT va conține pe o singură linie un singur număr natural ce reprezintă numărul de zile necesare pentru a organiza întâlnirea agenților. Dacă o astfel de întâlnire este imposibilă, în fişierul de ieşire se va scrie cuvântul NUă.

Exemplu

AGE.IN:	AGE.OUT
6 7 1 5 1 2 4 5 2 3 3 4 4 1 5 4 5 6	3

12.8 Interogări[22]

Enunț

Se consideră un graf care inițial este format din *P* noduri izolate, etichetate de la 1 la *P*. Se mai consideră *N* **intrări**, unde **intrare** poate însemna:

- ***comandă*** – o comandă are forma **'I+J'**, cu semnificația că în graf se adaugă muchia care uneşte nodurile **I** şi **J** (dacă **I** şi **J** erau deja unite în acel moment, nu se întreprinde nici o acțiune);
- ***interogare*** – o interogare este de forma **'I?J'**, adică se întreabă dacă în acel moment *I* şi *J* sunt în aceeaşi componentă conexă.

Se pleacă de la un graf inițial format din noduri izolate, care pe parcurs se „unifică". Pe parcurs se pun interogări dacă anumite perechi de noduri sunt sau nu în aceeaşi componentă conexă.

[22] Republica Moldova, Şcoala de Vară la Informatică, 2002

Intrare / Output

Fişierul **ENTRIES.IN** conține pe prima linie numărul *N* de intrări. Pe următoarele *N* linii se găsesc intrările, câte una pe linie. O intrare este codificată prin trei numere separate prin câte un spațiu. Primele două numere reprezintă nodurile **I** şi **J** (numere întregi, cuprinse între 1 şi *P*)**,** iar al treilea este 1 dacă intrarea este o ***comandă,*** respectiv 2 dacă intrarea este o ***interogare.***

La fiecare interogare, se scrie pe o linie separată în fişierul **ENTRIES.OUT** numărul 1 dacă nodurile referite sunt în acel moment în aceeaşi componentă conexă, respectiv numărul 0 în caz contrar.

Restricții

- $1 \le N \le 5\,000$; $1 \le P \le 10\,000\,000$
- În lista de intrări există cel puțin o interogare

Exemplu

ENTRIES.IN	**ENTRIES.OUT**
9 1 2 2 1 2 1 3 7 2 2 3 1 1 3 2 2 4 2 1 4 1 3 4 2 1 7 2	0 0 1 0 1 0

12.9 Import galactic[23]

Enunț

Odată cu apariția unui nou motor spațial ThrustoZoom, compania de import/export HyperCommodities a început comerțul cu cele mai îndepărtate galaxii din Univers. HyperCommodities doreşte să importe bunuri din unele galaxii ale sectorului Plural Z. Planetele din aceste galaxii exportă materiale din clasa *vacuuseal, transparent aluminum, digraphite,* şi *quantum steel*. Rapoartele preliminare denotă următoarele fapte:

- Fiecare galaxie conține cel puțin una şi cel mult 26 planete. Fiecare planetă din galaxie este identificată de o literă unică de la A la Z.

[23] ACM, Mid-Central programming contest,1995

- Fiecare planetă e specializată în producția şi exportul unui tip de produs. Planete diferite din aceeaşi galaxie exportă diferite produse.
- Unele perechi de planete sunt interconectate prin linii hiperspațiale de livrare. Dacă planetele A şi B sunt interconectate, ele pot comercializa produse fără restricții. Dacă planeta C este conectată la B dar nu şi la A, atunci A şi C pot de asemenea face comerț prin intermediul planetei B, dar B reține 5% din volumul tranzitat în calitate de plată pentru tranzacție. (A va primi doar 95% din volumul produsului, livrat de C şi C primeşte doar 95% din produsul livrat de A) În general, oricare două planete pot face comerț între ele, dacă există o rețea de linii de livrare, care le uneşte, dar fiecare planetă intermediară reține 5% din produsul tranzitat (care, de la planetă la planetă poate varia).
- Cel puțin câte o planetă din fiecare galaxie doreşte să construiască o linie hiperspațială pentru ThrustoZoom , care va livra produse către Pământ. Liniile ThrustoZoom sunt concepute la fel ca şi celelalte linii comerciale în interiorul galaxiilor. De exemplu, dacă planeta K deschide o linie ThrustoZoom spre Pîmânt, ea va deplasa produsele gratis, în timp ce orice planetă conectată la K, va tranzita produsele sale cu o plată de tranzit obişnuită.

HyperCommodities a asociat o valoare relativă (un număr real mai mic decât 10) fiecărui produs exportat de planete. Valoarea numerică este direct proporțională cu importanța produsului. Produsele mai valoroase pot fi vândute cu un profit mai mare în magazine. Se cere să se determine care planetă este cea mai convenabilă pentru export (va asigura un profit maxim), dacă se iau în considerare plățile de tranzit.

Input

Fişierul de intrare conține descrierea uneia sau a mai multor galaxii. Fiecare descriere începe cu o linie care conține un număr întreg N care specifică numărul planetelor în galaxie. Următoarele N lini conțin descrierea fiecărei planete, care este formată din:

1. Litera care corespunde planetei.
2. Spațiu.
3. Valoarea relativă a produsului exportat către Pământ în forma *d.dd*.

4. Spațiu.
5. Un şir de caractere şi/sau caracterul '*': literele din şir indică liniile de comerț către aceste planete iar '*' indică acordul planetei de a deschide o linie ThrustoZoomcăte Pământ.

Ieşire

Pentru fiecare descriere a galaxiei fişierul deieşire va conține o singură linie citită ca "Import from P" unde P este litera ce corespunde planetei cu cel mai valoros export, ținând cont de pierderile cauzate de tranzit. Dacă mai multe planete pot oferi un export la fel de valoros, se va afişa indicele primei planete, în ordine lexicogafică.

Exemplu:

Input

```
1
F 0.81 *
5
E 0.01 *A
D 0.01 A*
C 0.01 *A
A 1.00 EDCB
B 0.01 A*
10
S 2.23 Q*
A 9.76 C
K 5.88 MI
E 7.54 GC
M 5.01 OK
G 7.43 IE
I 6.09 KG
C 8.42 EA
O 4.55 QM
Q 3.21 SO
```

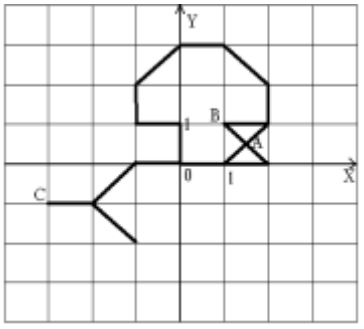
Output

```
Import from F
Import from A
Import from A
```

12.10 Incendiator[24]

Enunț:

Pe o rețea rectangulară cu distanța unitară între liniile vecine a fost introdus un system de coordinate cartezian. Originea sistemului se află în unul din punctele de intersecție a liniilor rețelei, axele lui sunt orientate paralel liniilor.



[24] Finala .campion, 2007.

Pe rețea a fost plasată o figură din chibrituri. Au fost folosite chibrituri de 2 tipuri:

- Chibritele de lungime 1 au fost plasate de-a lungul liniilor rețelei.
- Chibritele de lungime $\sqrt{2}$ au fost plasate pe diagonalele celulelor rețelei.

Ionuț vrea să ardă figura. El o poate aprinde doar într-un punct, cu coordonate întregi (de exemplu în punctul A de pe desen aprinderea este interzisă, iar în punctele B şi C - permisă).
Se ştie că chibriturile ard uniform, dar fiecare chibrit are viteza sa proprie de ardere. Chibritul poate arde concomitent în mai multe locuri (de exemplu, când se aprinde de la vecinii săi din ambele părți sau la mijlocul unui chibrit diagonal focul trece la alt chibrit intersectat şi se extinde în ambele părți).

Secrieți un program, care va deteremina, în ce punct trebuie aprinsă figura, pentru ca să ardă într-un timp minim.

Intrare

Prima linie a fişierului de intrare conține un număr natural N — numărul de chibrite. Urmează N linii, fiecare din ele conținând câte 5 numere întregi X_{i1}, Y_{i1}, X_{i2}, Y_{i2}, T_i, separate prin spațiu - coordonatele capetelor chibritului cu indicele i şi timpul de ardere a lui în condiția ca e aprins doar din o singură parte. Lungimile corecte ale chibriturilor (1 sau $\sqrt{2}$), conexitatea figurii formate şi lipsa chibriturilor care se suprapun este garantată.

Ieşire

Fişierul de ieşire va conține coordonatele punctului în care trebuie de aprins figura pentru ca ea să ardă într-un timp minim, apoi timpul de ardere a figurii cu cel puțin 2 semne după virgulă, separate prin spațiu. Dacă există mai multe soluții, poate fi prezentată oricare dintre ele.

Restricții:

$1 \le N \le 40$; $-200 \le X_i, Y_i \le 200$; $0 \le T \le 10^7$.

Exemple

*.in	*.out
1 0 0 1 1 1	0 0 1.00
5 0 0 0 1 1 1 0 0 1 10 0 0 1 0 1 0 0 1 1 1 2 2 1 1 1	0 0 3.25
3 1 1 1 2 10 1 2 2 2 10 1 1 2 2 50	2 2 35.00

12.11 Reconstrucţia arborilor[25]

Enunț

Mihai şi Sandu joacă următorul joc.

Mihai desenează pe o foaie un **arbore** cu N vârfuri numerotate de la 1 la N. Apoi el lichidează vârfurile după cum urmează: caută, care **frunză** (vârf de putere 1) este numerotat cu cel **mai mic** număr , **lichidează această frunză** şi scrie numărul nodului **de la care a rupt** această frunză. El repetă operația până când rămâne doar un nod în graf şi o listă de (N-1) numere.

Apoi Sandu încearcă să reconstruiască arborele utilizând doar lista . În caz de succes el câştigă jocul.

Scrieți un program, care l-ar ajuta pe Sandu să câştige.

Intrare

Prima linie a fişierului conține un număr întreg N, $2 \le N \le 100.000$, numărul de vîrfuri în arbore.

Linia a doua conține N-1 numere separate prin spațiu. Acestea sunt numerele scrise de Mihai.

Ieşire

[25] Croația, Olimpiada națională la Informatică, 2003

In N-1 linii ale fişierului de ieşire veţi înscrie muchiile arborelui **în ordine arbitrară**. Fiecare muchie va fi descrisă de 2 vârfuri (în ordine arbitrară) care o formează.

Exemple:

list.in 5 4 3 4 5	**list.in** 7 1 2 2 2 1 7	**list.in** 10 6 2 7 9 2 9 4 4 10
list.out 1 4 2 3 3 4 4 5	**list.out** 3 1 1 7 2 1 4 2 5 2 6 2	**list.out** 8 4 7 5 3 2 9 6 9 4 10 4 1 6 2 9 2 7

12.12 Relaţii romantice

Enunț

La anul II al universității cineva a lansat un studiu al relațiilor romantice între studenți. În scopuri didactice, o relație romantică poate exista între un băiat şi o fată. Scopul studiului constă în determinarea setul maximal de studenți, care nu sunt în relații romantice. Rezultatul numărul de studenți în acest set.

Scrieți un program, care calculează setul maximal de studenți, care nu sunt în relații romantice.

Intrare

Fişierul de intrare conține un set de date text în forma următoare:

numarul de studenți

şi descrierea fiecărui student, în formatul

student_id: (numărul_de_relații) student_id$_1$ student_id$_2$ **…**student_id$_3$

sau **student_id:**(0)

Ieşire
Fişierul de ieşire va conține câte un număr pentru fiecare set de date din fişierul input.

Restrictii : $n \leq 500$.

Exemplu

input	output
7 0: (3) 4 5 6 1: (2) 4 6 2: (0) 3: (0) 4: (2) 0 1 5: (1) 0 6: (2) 0 1 3 0: (2) 1 2 1: (1) 0 2: (1) 0	5 2

12.13 UP[26]

Enunț:
Penru a susține cu succes testului de angajare în funcția de programator la Compania Macrosoft, competitorul trebuie să poată rezolva probleme din următoarele patru compartimente: programarea dinamică, Backtracking-ul, Teoria Grafurilor şi Geometria Computațională. Pregătirea cea mai bună este cea făcută în baza unui îndrumar (culegere de probleme), editat de Compania Macrosoft.
Nivelul de pregătire al competitorului la fiecare din teme este determinat de un indice notat prin *UP*, care ia valori de la *1* la *L*. În procesul testării, pentru fiecare problemă e determinată valoarea minimă a indicelui *UP* al programatorului pentru fiecare din compartimentele, necesare ca el să poată rezolva problema dată. Sunt

[26] Rusia, Olimpiada Naţională la Informatică, 2000

cuoscute şi valorile indicelui *UP*, pe care programatorul le va obține după rezolvarea ei. Dacă înainte de rezolvare, programatorul are un *UP* mai înalt la unele compartimente, acest nivel nu va micşorat. Pentru rezolvarea unei probleme care măreşte indicele *UP* la cel puțin un compartiment, programatorul cheltuie 2 ore, în caz contrar – 1 oră.
Trebuie să alcătuiți un plan de studii, care ar avea nu mai mult de *T* ore, conform căruia un programator începător (*UP=1* la toate compartimentele) ar atinge nivelul *L* la toate compartimentele, rezolvând tot odată cât mai multe probleme.

Intrare
Prima linie a fişierului de intrare conține un întreg *T* – numărul de ore rezervate pentru pregătire. A doua linie conține numărul *L* ($2 \le L \le 16$), iar cea de a treia – numărul problemelor *M* ($1 \le M \le 500$, $2 \le T \le M$). Fiecare din următoarele *M* linii conține câte 8 numere, care descriu o problemă. Primele patru numere determină *UP* necesare pentru rezolvarea ei la fiecare din compartimente. Următoarele patru indică *UP*, până la care pot creşte indicii *UP* la fiecare din compartimente după rezolvarea problemei.

Ieşire
În cazul posibilității atingerii nivelului *L* la toate compartimentele, prima linie a fişierului de intrare va conține numărul maxim de probleme rezolvate. Cea de a doua linie va conține consecutivitatea problemelor rezolvate, în ordinea rezolvării, fără repetări. Dacă nivelul *L* nu poat fi atins, va fi afişat un singur număr 0.

Exemplu

Intrare	**Ieşire**
7 5 6 2 1 1 1 2 4 5 5 1 1 1 1 3 1 1 1 3 3 3 3 3 3 3 3 1 3 1 1 5 5 5 5 2 2 2 2 2 2 2 2 1 2 3 4 2 3 4 5	4 2 1 4 3

12.14 Alchimistul[27]

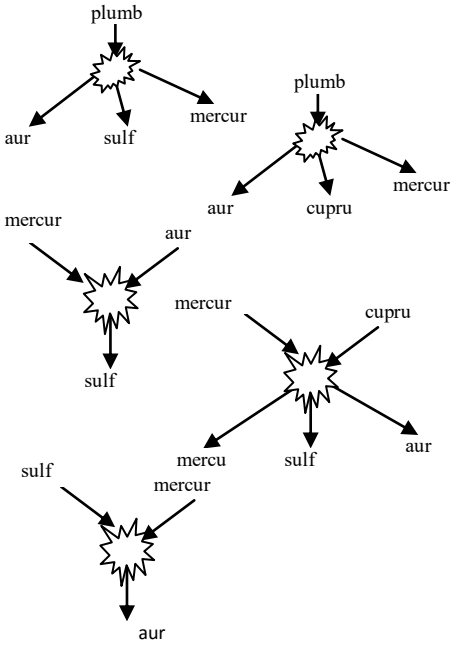
Alchimistul Ion a descoperit Piatra Filozofală, cu ajutorul căreia pot fi realizate mai multe reacții alchimice de transformare a unor substanțe în altele. Masa fiecărui reagent la începutul reacției şi masa fiecărei din substanțele obținute în rezultatul reacției sunt egale cu câte 1 gram. (În alchimie legea conservării masei poate fi încălcată).
Inițial Ion are un gram de plumb. Cu ajutorul Petrei filozofale Ion poate transforma substanța sa în alte substanțe, care ulterior de asemenea pot fi prelucrate cu ajutorul Petrei filozofale. Efectuând reacțiile una după alta, Ion vrea să obțină cât mai mult aur.
Scrieți un program, care, având o descriere a reacțiilor posibile, îl va ajuta pe Ion să obțină cât mai mult aur.

Intrare
Prima linie a fişierului de intrare conține numărul întreg K— numărul de substanțe participante la reacții ($1 \le K \le 6$).
Cea de a doua linie conține lista substanțelor (aurul şi plumbul sunt prezente în toate listele). Denumirea substanțelor nu depăşeşte 10 simboluri.
A treia linie conține numărul întreg L — numărul de tipuri de reacții realizate de Piatra Filozofală ($1 \le L \le 100$).
Urmează L descrieri ale reacțiilor. Fiecare descriere e formată din două linii:

- Prima linie – substanțele care intră în reacție.
- Linia doi – substanțele rezultante.

Output
Programul va scrie în fişierul de ieşire un număr — cantitatea obținută de aur, sau, în cazurile în care Ion poate obține orice cantitate de aur – textul QUANTUM SATIS

[27] Rusia, Olimpiada națională la Informatică, 1999

input	output	explicaţii
5 plumb aur sulf mercur cupru 5 plumb aur sulf mercur plumb aur cupru mercur mercur aur sulf mercur cupru mercur sulf aur sulf mercur aur	3	

12.15 Paza prezidenţială[28]

Enunț:
Odată demult era o republică a absurdului. Era acolo tot ce-i rebuie unei republici, chiar şi preşedinte cu castel. Planul castelului este un dreptunghi, împărțit în M x N pătrate unitare. Unele pătrate sunt pereți, altele libere. Fiecară pătrat liber e numit odaie. Preşedintele, fiind paranoic, a pus în unele odăi fântâni ascunse (cu aligatori la fund). Apoi a decis să se pună peste tot unde era posibil în castel gardieni. Ceea ce nu era de loc simplu. Gardienii sunt antrenați să tragă imediat ce văd pe cineva. Preşedintele trebuie să plaseze gardienii astfel, încât ei să nu se vadă – altfel se împuşcă reciproc! Suplimentar, ei nu pot fi plasați în odăile cu fântâni. Fiecare odaie are nu mai mult de un gardian. Doi gardieni din odăi diferite se văd reciproc, dacă odăile sunt în aceeaşi linie sau coloană şi între ele nu sunt pereți

[28] CEOI, 2002

Determinați numărul maxim de gărzi în castel şi indicați o amplasare a lor.

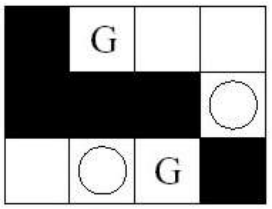
Intrare
Prima linie din input va conține numerele întregi M şi N – dimensiunile castelului ($1 \le M, N \le 200$). Linia i din următoarele M linin conține N numere întregi $a_{i;1}$; … ; $a_{i;N}$, separate prin spațiu, unde:

- $a_{i;j} = 0$ – pătratul [i; j] reprezintă o odaie liberă (fără fântână);
- $a_{i;j} = 1$ – pătratul [i; j] reprezintă o odaie cu fântână;
- $a_{i;j} = 2$ – pătratul [i; j] reprezintă un perete;

Indicele i denotă liniei, iar indicele j –coloana.

Ieşire
Prima linie a fişierului de ieşire va conține numărul maximal K de gardieni, ce pot fi plasați în castel. Următoarele K linii vor conține o posibilă amplasare a gărzilor, pe fiecare linie fiind scrise coordonatele unui gardian.

Exemplu

input	output	Explicaţie
3 4 2 0 0 0 2 2 2 1 0 1 0 2	2 1 2 3 3	

12.16 Metroul[29]

Metroul din Londra reprezintă un sistem complex, care transportă zilnic milioane de pasageri (fig. 1). Din punctul de vedere al pasagerului, metroul poate fi tratat ca o mulțime de linii de tren şi o mulțime de stații. În scopuri didactice, vom nota liniile de tren prin literele mari A,

[29] Olimpiada Republicană la Informatică, 2003

B, C, D ş.a.m.d. ale alfabetului latin, în total n linii, iar staţiile – prin numerele naturale 1, 2, 3, ..., în total m staţii (fig. 2).

Fig. 1

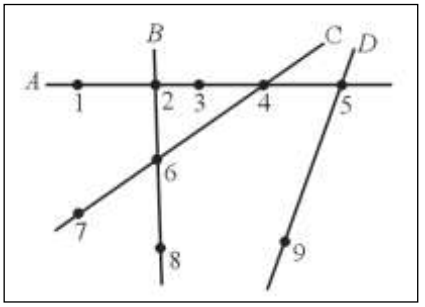


Fig. 2

Liniile de tren şi staţiile respective au fost proiectate în aşa fel, încît pasagerul, care pleacă din orice staţie x, să poată ajunge în oricare altă staţie y. Evident, în cazurile în care staţiile se află pe linii diferite, pasagerul este nevoit să facă una sau mai multe transbordări, schimbînd trenul în staţiile în care se întîlnesc două sau mai multe linii de tren.

De exemplu, pentru a ajunge din staţia 1 în staţia 8 (fig. 2), pasagerul poate să facă o singură transbordare în staţia 2 sau două transbordări – prima în staţia 4 şi a doua în staţia 6.

Elaboraţi un program, care, cunoscînd planul metroului, staţia de plecare x şi staţia de sosire y, calculează numărul minim de transbordări.

Intrare Fişierul text METROU.IN conţine pe prima linie numerele naturale n, m, x, y separate prin spaţiu. Fiecare din următoarele n linii ale fişierului conţine numere de staţii separate prin spaţiu. Linia a 2-a a fişierului de intrare conţine numerele de staţii ale liniei de tren A, linia a treia a fişierului de intrare conţine numerele de staţii ale liniei de tren B ş.a.m.d.

Ieşire
Fişierul text METROU.OUT va conține pe o singură linie numărul minim de transbordări.

Exemplu.

METROU.IN

```
4 9 1 8
1 2 3 4 5
2 6 8
7 6 4
5 9
```

METROU.OUT

```
1
```

Restricții. $2 \le n \le 26$, $3 \le m \le 250$, $x \ne y$. Timpul de execuție nu va depăşi 5 secunde. Fişierul sursă va avea denumirea METROU.PAS, METROU.C sau METROU.CPP.

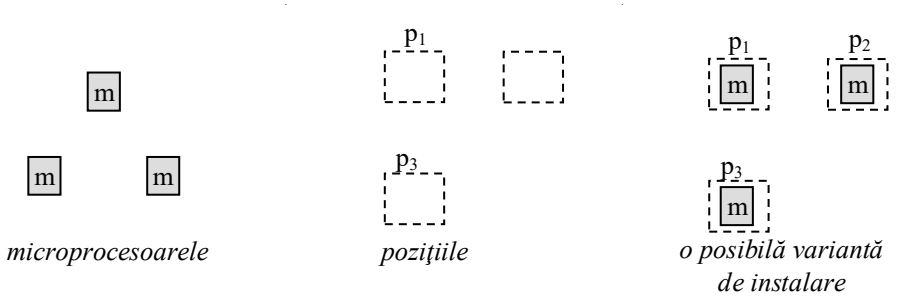
12.17 Multimicroprocesoare[30]

Este cunoscut faptul, că capacitatea de prelucrare a calculatoarelor moderne poate fi mărită prin includerea în componența acestora a mai multor microprocesoare. În astfel de calculatoare, un microprocesor prelucrează datele numerice, un alt microprocesor prelucrează informația grafică, un al treilea procesor prelucrează informația sonoră ş.a.m.d. Pentru a face schimb de date între ele, microprocesoarele sunt reunite prin conductoare.

Presupunem, că un calculator personal conține n microprocesoare, notate prin m1, m2, ..., mn. Prin rij vom nota numărul de conductoare între microprocesoarele mi şi mj. Evident, $r_{ij} = r_{ij}$ şi $r_{ii} = 0$.

În interiorul calculatorului, microprocesoarele m_1, m_2, ..., m_n pot fi instalate în anumite poziții, notate prin p_1, p_2, ..., p_n. Prin d_{kl} vom nota distanța între pozițiile p_k şi p_l (vezi desenul). Evident, $d_{kl} = d_{lk}$ şi $d_{kk} = 0$.

[30] Olimpiada Republicană la Informatică, 2009



Elaborați un program care determină pozițiile în care vor fi instalate microprocesoarele, cu condiția ca lungimea sumară a conductorilor să fie minimală.

Input. Fişierul text MULTI.IN conține pe prima lini numărul întreg n. Fiecare din următoarele n linii ale fişierului de intrare conține câte n numere întregi, separate prin spațiu. Linia $(i+1)$ a fişierului de intrare conține numerele întregi ri1, ri2, ..., rin.

În continuare, în fişierul de intrare se conțin n linii cu câte n numere întregi, separate prin spațiu. Linia $(n+k+1)$ a fişierului de intrare conține numerele întregi dk1, dk2, ..., dkn, separate prin spațiu.
Output. Fişierul text MULTI.OUT va conține pe prima linie numerele întregi $q_1, q_2, ..., q_n$, separate prin spațiu. Aceste numere indică pozițiile în care sunt instalate microprocesoarele, respectiv, m_1, m_2, ..., m_n, cu condiția că lungimea sumară a conductorilor este minimală. Linia a doua a fişierului de ieşire va conține numărul întreg L_{min} – lungimea sumară minimală a conductorilor ce reunesc microprocesoarele. Dacă problema admite mai multe variante de instalare, în fişierul de ieşire se va indica doar una din ele.

Exemplu.

MULTI.IN

```
3
0 10 50
10 0 100
50 100 0
0 1 2
1 0 3
2 3 0
```

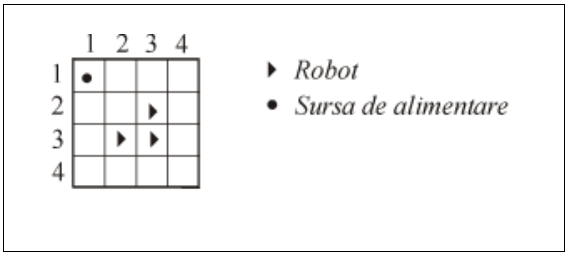
MULTI.OUT

```
3 2 1
230
```

Restricţii. $3 \le n \le 9$, $0 \le r_{ij} \le 1000$, $1 \le d_{kl} \le 1000$. Timpul de execuție nu va depăşi 2,0 secunde. Programul va folosi cel mult 32 Megaocteți de memorie operativă. Fişierul sursă va avea denumirea MULTI.PAS.

12.18 Roboţii 2[31]

Pe un teren de dimensiunile n × m, divizat în pătrățele identice cu laturi de lungimea unu, lucrează q roboti. Roboții se deplasează pe teren prin "sărituri", exact la fel ca cele ale "calului" din jocul de şah. Pentru o bună funcționare, o dată pe zi toți roboții trebuie concomitent să se adune în pătrățelul cu coordonatele (s, t), unde ei sînt conectați la o sursa de alimentare (vezi desenul). Menționăm, că în unele cazuri, în funcție de poziția inițială a roboților, nu toți din ei ar putea să se adune în pătrățelul (s, t).



Prin lungimea drumului parcurs de un robot vom înțelege numărul de "sărituri" efectuate de robot în procesul deplasării din pătrățelul curent în pătrățelul cu coordonatele

[31] Baraje pentru selectarea lotului național la Informatică, 2004

(s, t). Evident, lungimea totală a drumurilor parcurse de roboți se va calcula prin însumarea lungimilor de drumuri parcurse de fiecare robot.

Scrieți un program care calculează minimul lungimii totale L a drumurilor parcurse de roboți pentru a se aduna în pătrățelul (s, t).

Date de intrare. Fişierul text ROBOTII.IN conține pe prima linie numerele naturale n, m, s, t, q, separate prin spațiu. Următoarele q linii ale fişierului de intrare conțin cîte două numere naturale x, y separate prin spațiu – coordonatele fiecărui robot.

Date de ieşire. Dacă toți roboții se pot aduna în pătrățelul (s, t), în fişierul de ieşire ROBOTII.OUT se va scrie pe o singură linie numărul natural L – minimul lungimii totale a drumurilor parcurse de roboți. În caz contrar, dacă cel puțin unul din roboți nu poate să ajungă în pătrățelul (s, t), în fişierul de ieşire se va scrie pe o singură linie valoarea -1.

Exemplul 1.

ROBOTII.IN

```
4 4 1 1 3
2 3
3 2
3 3
```

ROBOTII.OUT

```
6
```

Restricții. $2 \le n, m, s, t \le 250$; $1 \le q \le 10000$. Timpul de execuție nu va depăşi 3 secunde. Fişierul sursă va avea denumirea ROBOTII.PAS, ROBOTII.C sau ROBOTII.CPP.

Bibliografie

1. Sedgewick Th, *Algorithms in C*, 2001, Addison Wesley

2. Gibbons Alan, *Algorithmic ghaph theory*, 1999, Addison Wesley

3. Липский В., *Комбинаторика для программистов*, 1988, Мир, Москва

4. Новиков Ф.А., *Дискретная математика для программистов*, 2001, Питер, Санкт Петербург

5. Майника Э., *Алгоритмы оптимизации на сетях и графах*,1981, Мир,Москва

6. Кристофидес П., *Теория графов. Алгоритмический подход*, 1978, Мир, Москва

7. Cormen Th., Leiserson Ch., Rivest R., *Introducere în algoritmi.* Agora, Cluj, 2001.

8. Cristian A.Giumale, *Introducere în analiza algoritmilor. Teorie şi aplicaţie*. Polirom, Iaşi, 2004

9. Pătruţ Bogdan. *Programarea calculatoarelor*. Teora, Bucureşti, 1998.

10. Cerchez Em., Şerban M. *Programarea în limbajul C/C++ pentru liceu. Vol III. Teoria Grafurilor*. Polirom, Iaşi, 2006.

11. Пападимитриу Х., Стайглц К., *Комбинаторная оптимизация. Алгоритмы и сложность*. Москва, Мир, 1985

12. Corlat S., Corlat A. *Grafuri. Noțiuni, algoritmi, implementări.* Chişinău, Biotehdesign, 2012

13. Gremalschi A. ş. a. *Broşura Olimpiadei Republicane la Informatică.* 2003 – 2013.

Abrevieri şi notaţii

$\vee$	- disjuncţia (operaţia logică SAU [OR])
$\wedge$	- conjuncţia (operaţia logică ŞI [AND])
$\neg$	- negaţia (operaţia logică NU [NOT])
$A \Rightarrow B$	- din A rezultă B.
$A \Leftrightarrow B$	- A este echivalent cu B
$x \in X, x \notin X$	- x aparţine X (x nu aparţine X)
$\{x \in X : Q\}$	- submulţimea elementelor x din X, care satisfac condiţia Q
$A \subseteq B$	- A se conţine în B (A este submulţime a mulţimii B)
$\wp(X)$	- mulţimea tuturor submulţimilor mulţimii X
$\lvert X \rvert$	- cardinalul mulţimii X
$a_1, ..., a_n$	- secvenţă din n elemente
(a, b)	- pereche ordonată
$A \times B$	- produs cartezian al mulţimilor A şi B; mulţimea tuturor perechilor posibile $(a,b) : a \in A, b \in B$
$a \leftarrow b$	- valoarea elementului b este atribuită elementului a .
$i \uparrow$	- valoarea elementului i este incrementată cu 1.
$i \downarrow$	- valoarea elementului i este decrementată cu 1.
$a \rightleftarrows$	- valorile elementelor a şi b sunt interschimbate $(a \leftarrow b) \wedge (b \leftarrow a)$