ABORDĂRI DIDACTICE ÎN APLICAREA ALGORITMULUI PRIM PENTRU DETERMINAREA ARBORELUI PARȚIAL DE COST MINIM

Marina Bostan, drd. UST

Rezumat. În lucrarea dată sunt examinate abordări didactice procesului de predare – învățare al algoritmului Prim pentru determinarea arborelui partial de cost minim asociat unui graf.

1. Noțiuni generale

În Teoria grafurilor, graful reprezintă o pereche ordonată $G = (V, E)$, unde V - este mulțimea vârfurilor, E – este mulțimea muchiilor (arcelor) grafului G. Muchia reprezintă conexiunea între două vârfuri. Muchiile pot fi orientate sau neorientate, totodată pot fi ponderate. Graful se numește conex, dacă există traseul (drum) între oarecare două vârfuri.

Ciclul reprezintă traseul care se începe și se termină în același vârf. Arborele este graful conex, care nu conține cicluri. Arborele parțial reprezintă o submulțime din mulțimea muchiilor. Dacă suma costurilor muchiilor este minimală din toate arborele parțiale ale grafului, atunci arborele dat se numește arborele parțial de cost minim.

Problema arborelui de valoare optima

Studiul arborilor este justificat de existenţa în practică a unui număr mare de probleme care pot fi modelate prin arbori. Dintre acestea amintim:

1. construirea unor reţele de aprovizionare cu apă potabilă (sau cu energie electrică sau termică etc) a unor puncte de consum, de la un punct central;
2. construirea unor căi de acces între mai multe puncte izolate;
3. desfăşurarea unui joc strategic;
4. luarea deciziilor în mai multe etape (arbori decizionali);
5. evoluţii posibile ale unui sistem pornind de la o stare iniţială;
6. construirea unei reţele telefonice radiale, a unei reţele de relee electrice;
7. legarea într-o reţea a unui număr mare de calculatoare;
8. organigramele întreprinderilor;
9. studiul circuitelor electrice în electrotehnică (grafurile de fluență etc);
10. schemele bloc ale programelor pentru calculatoare etc.

În toate problemele de mai sus se doreşte ca, dintre muchiile unui graf neorientat, să se extragă arborele optim din mulţimea tuturor arborilor care pot fi extraşi din graful dat.

Deoarece definiţia arborelui este dificil de aplicat pentru deciderea faptului că un graf este arbore sau nu (şi în special sunt greu de verificat conexitatea şi mai ales existenţa ciclurilor) există mai multe caracterizări posibile ale unui arbore.

În practică sunt utilizați algoritmii Kruskal și Prim pentru determinarea unui graf parțial de cost minim. În lucrare dată vom examina mai detaliat algoritmul Prim.

2. Aspecte didactice în aplicarea algoritmului Prim

Algoritmul **Primm** este un algoritm din teoria grafurilor care găsește arborele parțial de cost minim al unui graf conex ponderat. Înseamnă că găsește submulțimea muchiilor care

formează un arbore care include toate vârfurile și al cărui cost este minimizat. Algoritmul a fost descoperit în 1930 de către matematicianul Vojtěch Jarník și apoi, independent, de informaticienii Robert C. Prim în 1957 și redescoperit de Edsger Dijkstra în 1959. De aceea mai este numit Algoritmul DJP, algoritmul Jarník sau algoritmul Prim-Jarník.

Algoritmul Prim se utilizează atunci când numărul de muchii a arborelui examinat este destul de mare.

Etapele algoritmului:

Pasul 1. Se începe construirea arborelui parțial de cost minim de la un nod stabilit inițial.

Pasul 2. De la nodul inițial se determină muchia incidentă cu costul cel mai mic (adică ponderea muchiei cea mai mică).

Pasul 3. De la nodul selectat la fel se determină muchia de cel mai mic cost. Muchiile alese se „conectează” una lângă alta construind astfel arborele parțial de cost minim.

Pasul 4. Dacă graful conține *n* noduri și se pornește de la nodul fixat, atunci pentru construirea arborelui parțial de cost minim se efectuează *n-1* pași.

Algoritmul consideră iniţial că fiecare nod este un subarbore independent, ca şi Kruskal. Însă spre deosebire de acesta, nu se construiesc mai mulţi subarbori care se unesc şi în final ajung să formeze arbore minim de acoperire, ci există un arbore principal, iar la fiecare pas se adaugă acestuia muchia cu cel mai mic cost care uneşte un nod din arbore cu un nod din afara sa. Nodul rădăcină al arborelui principal se alege arbitrar. Când s-au adăugat muchii care ajung în toate nodurile grafului, s-a obţinut arbore minim de acoperire dorit. Abordarea seamănă cu algoritmul Dijkstra de găsire a drumului minim între două noduri ale unui graf.

Pentru o implementare eficientă, următoarea muchie de adăugat la arbore trebuie să fie uşor de selectat. Vârfurile care nu sunt în arbore trebuie sortate în funcţie de distanţa până la acesta (de fapt costul minim al unei muchii care leagă nodul dat de un nod din interiorul arborelui). Se poate folosi pentru aceasta o structură de heap. Presupunând că (u, v) este muchia de cost minim care uneşte nodul u cu un nod v din arbore, se vor reţine două informaţii:

- d[u] = w[u,v] distanţa de la u la arbore
- p[u] = v predecesorul lui u în drumul minim de la arbore la u.

La fiecare pas se va selecta nodul u cel mai apropiat de arborele principal, reunind apoi arborele principal cu subarborele corespunzător nodului selectat. Se verifică apoi dacă există noduri mai apropiate de u decât de nodurile care erau anterior în arbore, caz în care trebuie modificate distanţele dar şi predecesorul. Modificarea unei distanţe impune şi refacerea structurii de heap.

Complexitatea algoritmului

Iniţializările se fac în O(|V|). Bucla principală while se execută de |V| ori. Procedura GetMin() are nevoie de un timp de ordinul O(lg|V|), deci toate apelurile vor dura O(|V|lg|V|). Bucla for este executată în total de O(|E|) ori, deoarece suma tuturor listelor de adiacenţă este

2|E|. Modificarea distanţei, a predecesorului, şi refacerea heapului se execută într-un timp de O(1), O(1) şi respectiv O(lg|V|). Deci în total bucla interioară for durează O(|E|lg|V|).

În consecinţă, timpul total de rulare este O(|V|lg|V|+|E|lg|V|), adică O(|E|lg|V|). Aceeaşi complexitate s-a obţinut şi pentru algoritmul Kruskal. Totuşi, timpul de execuţie al algoritmului Prin se poate îmbunătăţi până la O(|E|+|V|lg|V|), folosind heap-uri Fibonacci.

Utilizarea softului matematic Maple18 în implementarea algoritmilor pentru determinarea arborelui de valoare optimă a unui graf.

Softul Maple este un sistem de calcul algebric (CAS) dezvoltat de firma Maplesoft (http://maplesoft.com), și care poate fi utilizat și la soluționarea unor probleme din Teoria Grafurilor. În acest scop se folosesc diverse pachete: Combinatorics, GraphTheory, LinearAlgebra, Optimization, Plots, etc.

Pachetul **GraphTheory** reprezintă o colecție de comenzi destinate pentru crearea, construirea, manipularea și testarea grafurilor și determinarea proprietăților acestora. Apelarea pachetului GraphTheory se face cu ajutorul comenzii ***with(GraphTheory)***, ca rezultat se obține lista tuturor funcțiilor apelabile care țin de examinarea și cercetarea proprietăților grafurilor.

În această lucrare vom descrie comenzile utilizate pentru implementarea algritmelor susnumite.

Pentru crearea unui graf folosim funcția **Graph(V, E, w)**, unde parametrii indicați au următoare semnificație:
V – (opțional) lista nodurilor (numere întregi, simboluri, șiruri de caractere);
E – (opțional) mulțimea muchiilor (arcelor);
X – (opțional) simbolul, cu costuri sau fără costuri.

O listă de numere întregi ***V***, este o listă de simboluri sau șiruri de caractere, care specifică nodurile. Fiecare nod trebuie să fie un număr întreg, simbol sau un șir. O mulțime ***E*** specifică mulțimea de muchii. O muchie neorientată între nodurile ***i*** și ***j*** este introdusă ca o mulțime de două noduri. O muchie ponderată este introdusă ca o muchie, unde ***w*** - greutatea muchiei, care este un număr întreg sau zecimal.

Comanda **MinimalSpanningTree(G)**, unde **G** este graf, determină arborele de cost minim pentru graful **G**, adică este un arbore a cărui suma costurilor de muchii este cât posibil de mica. Problema constă în aceia că acest arbore de cost minim nu este unic, Maple îl alege doar unul, care și-l prezentăm.

Pentru reprezentare grafică a grafului utilizăm comanda **DrawGraph(G, style=s);** unde parametrii utilizați sunt:
G – graful;
s - (opțional) circle, tree, bipartite, spring, planar.

Opțiunea stilului în comanda **DrawGraph** pentru a afișa graful inițial într-un stil specific. Există câteva stiluri diferite susținute pentru afișarea unui graf: cerc, arbore, bipartit și planar.

Comanda **PrimsAlgorithm(G, animate)**, unde **G** este graf, utilizează algoritmul lui **Prim** pentru determinarea arborelui de cost minim în mod interactiv-real.

Vom examina cum se aplică algoritmul Prim la soluționarea unei probleme concrete.

Problema. Determinarea arborelui de cost minim într-un graf.

O firmă-distribuitor de apă potabilă dorește să-și facă un oraș-depozit din 7 orașe, pentru care se cunosc toate distanțele de drumuri existente. Se cere de a găsi un traseu optim de la depozit către celelalte orașe, astfel încât distanța totală pe care o va parcurge pentru a distribui apă potabilă în toate celelalte 6 orașe să fie minimă. Să se precizeze care ar fi orașul în care să fie depozitul, pentru ca toate celelalte orașe să fie ușor accesibile.

Firma distribuitor trebuie să elaboreze un algoritm care să determine un traseu minimal pentru a distribui apă potabilă în 6 orașe și să determine care din orașele ar trebuie să fie depozit

Soluție. Dacă să asociem 7 orașe cu vârfuri unui graf, iar lungimile drumurilor cu ponderile muchiilor, se poate formula problema de a determina arborele de cost minim pentru graful dat.

Aplicând algoritmul Prim să se determine arborele de cost minim pentru graful dat:

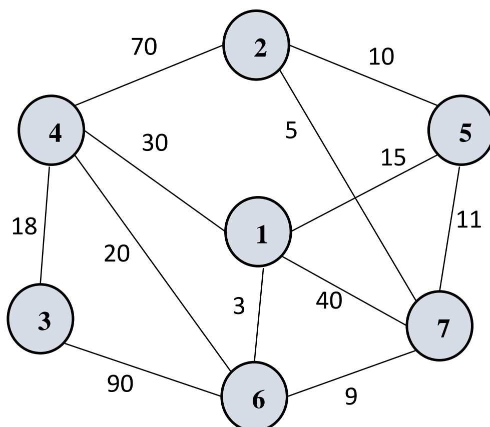

Figura 1. Graful initial G_1 cu 7 noduri

Metoda 1. ***Rezolvare manuală***

1) Selectăm nodul inițial **1** și alegem o muchie incidentă cu costul minimal $\{1, 6\} = 3$.

2) De la nodul găsit **6** iarăși alegem o muchie cu cel mai mic cost $\{6, 7\} = 9$.

3) Repetăm același algoritm până nu va fi examinate toate nodurile. Altfel obținem următorul tabel de examinare a grafului dat:

Numărul de pași	Muchia	Componenta conexă	Costul
Pasul 1	{1} [1, 6]	{1, 6}	C=3
Pasul 2	[6, 7]	{1, 6, 7}	C=3+9=12

Pasul 3	[7, 2]	{1, 2, 6, 7}	C=12+5=17
Pasul 4	[2, 5]	{1, 2, 5, 6, 7}	C=17+10=27
Pasul 5	[5, 4]	{1, 2, 4,5, 6, 7}	C=27+20=47
Pasul 6	[4, 3]	{1, 2, 3, 4, 5, 6, 7}	C=47+18=65

Am obținut costul minim al arborelui parțial egal cu 65 cu următoare reprezentare grafică:

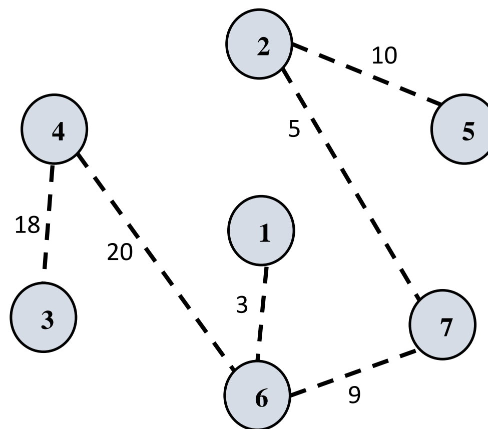

Figura 2. Arborele partial de cost minim pentru graful G_1

Metoda 2. ***Rezolvare în sistemul Maple 18***

1) Se apelează pachetul **GraphTheory:** *with*(*GraphTheory*)

2) Se definește graful G_1 cu 7 noduri și 11 muchii ponderate:

G1 := *Graph*(7, {[{1, 4}, 30], [{1, 5}, 15], [{1, 6}, 3], [{1, 7}, 40], [{2, 4}, 70], [{2, 5}, 10], [{2, 7}, 5], [{3, 4}, 18], [{3, 6}, 90], [{4, 6}, 20], [{5, 7}, 11], [{6, 7}, 9]})

Graph 2: an undirected weighted graph with 7 vertices and 12 edge(s)

3) Apelăm procedura pentru determinarea arborelui de cost minim: *T1* := *MinimalSpanningTree*(*G1*)

Graph 2: an undirected weighted graph with 7 vertices and 6 edge(s)

4) Construim graful inițial și arborele de cost minim obținut: *DrawGraph*([*G1*, *T1*], *style* = *circle*)

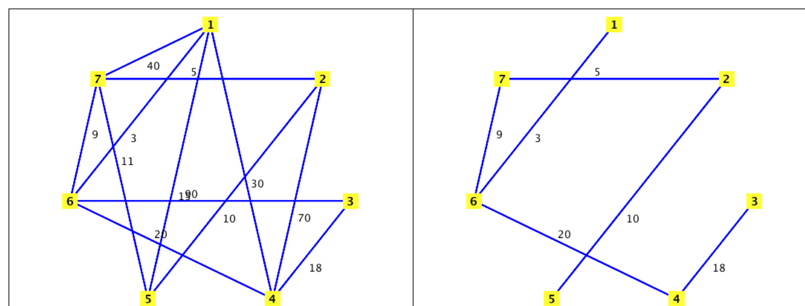

Figura 3. Graful G_1 și arborele partial de cost minim pentru graful G_1

în aplicația Maple

Apelăm procedura interactivă pentru determinarea arborelui de cost minim prin algoritmul *Prim*: *PrimsAlgorithm*(*G1*, *animate*)

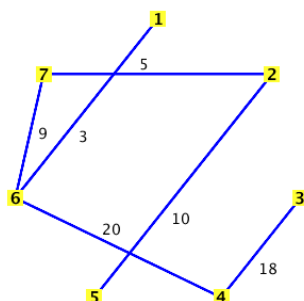

Figura 4. Arborele final parțial de cost minim pentru algoritmul Prim

Observăm că obținem aceleași arborele de cost minim și prin metoda manuală și prin rezolvarea în sistemul Maple18 utilizând doi algoritmi diferiți.

Metoda 3. ***Rezolvarea în C***

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]= {0}
,min,mincost=0,cost[10][10];
void main() {
  clrscr();
  printf("\n Enter the number of nodes:");
  scanf("%d",&n);
  printf("\n Enter the adjacency matrix:\n");
  for (i=1;i<=n;i++)
   for (j=1;j<=n;j++) {
        scanf("%d",&cost[i][j]);
        if(cost[i][j]==0)
           cost[i][j]=999;
  }
  visited[1]=1;
  printf("\n");
  while(ne<n) {
        for (i=1,min=999;i<=n;i++)
          for (j=1;j<=n;j++)
           if(cost[i][j]<min)
            if(visited[i]!=0) {
                min=cost[i][j];
                a=u=i;
                b=v=j;
            }
            if(visited[u]==0 || visited[v]==0) {
                printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
                mincost+=min;
                visited[b]=1;
            }
            cost[a][b]=cost[b][a]=999;
  }
  printf("\n Minimun cost=%d",mincost);
  getch();
 }
```

După executarea programului introducem numărul de noduri grafului dat și matricea ponderilor asociată grafului dat, ca în rezultat obținem costul arborelui parțial egal cu 65. Sesizăm că obținem același rezultat.

```
 Enter the number of nodes:7

 Enter the adjacency matrix:
0 0 0 30 15 3 40
0 0 0 70 10 0 5
0 0 0 18 0 90 0
30 70 18 0 0 20 0
15 10 0 0 0 0 11
3 0 90 20 0 0 9
40 5 0 0 11 9 0


 Edge 1:(1 6) cost:3
 Edge 2:(6 7) cost:9
 Edge 3:(7 2) cost:5
 Edge 4:(2 5) cost:10
 Edge 5:(6 4) cost:20
 Edge 6:(4 3) cost:18
 Minimun cost=65
```

Figură 5. Rezultatul programului în C utilizând algoritmul Prim

Concluzii

Determinarea unui arbore parțial de cost minim este o problemă cu aplicații în foarte multe domenii: rețele, clustering, prelucrare de imagini. Studierea, testarea și analiza algoritmilor studiate cu ajutorul softurilor specializate da studenților facilitează procesul de alegere algoritmului potrivit în soluționarea problemelor din viața reală. Eficacitatea și funcționalitatea algoritmului utilizat se evidențiază prin aplicarea softurile specializate (Maple 18, Teoria grafurilor), ceea ce permite vizualizarea interactivă algoritmilor aplicate, verificarea soluțiilor obținute, testarea algoritmului prin modificarea datelor de intrare, compararea diferitor algoritmi, individualizarea algoritmilor. Toate aceste momente contribuie la dezvoltarea gândirii logice și critice a studenților și le permite programarea algoritmilor aplicate cu ajutorul limbajelor de programare.

Bibliografie

1. Corlat S., Gremalschi A. Grafuri: Metodologia predării în cadrul instruirii de performanță la disciplinile Matematică & Informatică: [pentru uzul studenților]; AȘM, Univ. de Stat din Tiraspol, 2014, 158 p, ISBN 978-9975-76-122-2.
2. Chiriac L., Bostan M. Aspecte didactice în predarea algoritmilor pentru determinarea drumurilor minime în grafuri. CAIM 2017, Iași, September 14–17, 2017.
3. Tomescu I. Combinatorică şi teoria grafurilor. Editura Universității din Bucureşti, 1990
4. Bang-Jensen, Gutin G. Digraphs Theory, Algorithms and Applications. Springer-Verlag, 2007.
5. Bomdy J.A., Murty U.S.R. Graph Theory. Springer, 2007.

6. Ore O. Theory of graphs. American Mathematical Society Colloquium Publications, Vol. XXXVIII, American Mathematical Society, 1962, 270 p.
7. Хаггарти Р. Дискретная математика для программистов. Перевод с английского, Москва, Техносфера, 2003, 320 с., ISBN 5-94836-016-4.
8. Bârză S., Morgan L-M. Algoritmica grafurilor. Bucureşti: Ed. Fundaţiei România de Mâine, 2008, 148 p., ISBN 978-973-163-147-9 [vizitat 05.02.2018] http://www.ocpiilfov.ro/ocpi_ilfov/Man.pdf.
9. http://www.maplesoft.com/products/maple/new_features/maple18/Graph_Theory.aspx [vizitat 12.02.2018].

ALGEBRA AJUTĂ GEOMETRIA

Laurenţiu Calmuţchi, dr. hab., prof. univ., UST
Dorin Afanas, dr. conf. univ., UST

Rezumat. În acest articol se aplică metoda algebrică de rezolvare a problemelor geometrice.
Cuvinte-chee: problemă de construcţie; soluţie; cercetare.
Abstract. This article the algebraic method to solving geometric problems.
Keywords: problem of construction; solution; research.

Pe parcursul secolelor geometria a servit bază nu numai a matematicii, dar şi a altor ştiinţe. Anume în geometrie au apărut primele teoreme şi primele demonstraţii. Însăşi legile gândirii matematice s-au format cu ajutorul geometriei. Multe probleme geometrice au contribuit la apariţia a noi direcţii ştiinţifice şi invers, multe probleme ştiinţifice au fost rezolvate cu ajutorul metodelor geometrice.

Un rol deosebit de important în dezvoltarea multor ramuri matematice revine problemelor geometrice de construcţie. Este suficient să ne amintim de problemele cuadraturii discului, dublării cubului şi trisecţiei unghiului, care se cereau a fi rezolvate doar cu ajutorul riglei şi a compasului. Aceste probleme,apărute încă în secolul IV î.e.n., au devenit clasice şi abia la sfârşitul secolului al XIX-lea s-a demonstrat că ele nu pot fi rezolvate cu ajutorul riglei şi a compasului. Cercetările asupra acestor probleme au dezvoltat mai multe ramuri ale matematicii, mai cu seamă algebra,care la rîndul său a permis de a rezolva pe o cale mai simplă, iar uneori chiar şi imposibil de a rezolva pe altă cale unele probleme geometrice.

Vom considera că elevii au făcut cunoştinţă cu construirea segmentului, determinat de următoarele formule:

1. $x = a + b$;
2. $x = a - b \ (a > b)$;
3. $x = na, unde\ n \in N$;
4. $x = \frac{a}{n}\ , unde\ n \in N$;
5. $x = \frac{m}{n}a, \quad unde\ n, m \in N$;