Gordân Raluca Mihaela
Hriţcu Cătălin
An II A, grupa 4
Grupa de seminar A34

Algoritmica grafurilor – tema 1

Problema 1.

Un graf *G* se numeşte *rar* dacă numarul său de muchii *m* este mai mic decât $\frac{n^2}{\log n}$, unde *n* reprezintă numărul de vârfuri. O justificare este aceea că matricea de adiacenţa A a grafului, care ocupa n^2 locaţii de memorie, poate fi întotdeauna reprezentată folosind $O\left(\frac{n^2}{\log n}\right)$ locaţii de memorie astfel încât răspunsul la o întrebare "A(i,j)=1?" să se facă în $O(1)$. Descrieţi o astfel de schemă de reprezentare. **(4 puncte)**

Soluţie.

Scăderea numărului de biţi ai unei reprezentări neredundante presupune în general compresie iar decompresia nu se face în $O(1)$. De aceasta tot ce putem spera e să realizăm o grupare convenabilă a biţilor astfel încât redundanţa să fie cât mai redusă.

Fie *G* un graf cu *n* vârfuri şi *m* muchii, reprezentat prin matricea de adiacenţă $A_{n\times n}$ şi $k \in N^*$. Se observă că matricea de adiacenţă *A* stochează câte un singur bit util în fiecare din cele n^2 locaţii ale sale. Ea poate fi înlocuită de o matrice $B_{\frac{n}{k}\times\frac{n}{k}}$ care în fiecare locaţie de memorie stochează k^2 biţi utili. Practic orice element din B este o "împachetare" într-o singură locaţie de memorie a unei porţiuni de dimensiune $k\times k$ a matricei iniţiale *A*. Formula după care se "împachetează" este:

$$b[i,j] = a[ik, jk]\cdot 2^{k^2-1} + a[ik+1, jk]\cdot 2^{k^2-2} + \ldots + a[(i+1)k-1, (j+1)k-1]\cdot 2^0 =$$
$$= (\ldots((a[ik, jk]\cdot 2 + a[ik+1, jk])\cdot 2 + \ldots + a[(i+1)k-1, jk])\cdot 2 +$$
$$+ a[ik, jk+1])\cdot 2 + a[ik+1, jk+1])\cdot 2 + \ldots + a[(i+1)k-1, jk+1])\cdot 2 + \ldots$$
$$\ldots + a[ik, (j+1)k-1])\cdot 2 + a[ik+1, (j+1)k-1])\cdot 2 + \ldots + a[(i+1)k-1, (j+1)k-1].$$

De exemplu pentru *n* = 8 şi *k* = 4 avem următoarea procedură de împacherare.

Dacă $A = \begin{pmatrix} 1011 & 0010 \\ 0010 & 0001 \\ 0000 & 1001 \\ 0010 & 0110 \\ 0101 & 0100 \\ 0100 & 1100 \\ 0001 & 1000 \\ 0000 & 0001 \end{pmatrix}$ atunci $B = \begin{pmatrix} 0xB202 & 0x2196 \\ 0x5410 & 0x4C81 \end{pmatrix}$.

Algoritmul prin care se ajunge la *B* este următorul:

```
procedure pack(A, n, k, B)
begin
   for i=1 to n/k do
      for j = 1 to n/k do
         b[i,j] = 0
         for l = i*k to (i+1)*k-1 do
            for c = j*k to (j+1)*k-1 do
```

```
            B[i,j] = B[i,j]*2+A[l,c]
end
```

Pentru a răspunde la întrebarea " $A[i,j]=1$? " având la dispoziţie doar matricea B procedăm astfel:

- căutăm locaţia din B corespunzătoare lui $A[i,j]$
- extragem doar bitul care ne interesează şi îl oferim ca ieşire a algoritmului

```
function at(B, i, j)
begin
    i' = [i/k]
    j' = [j/k]
    l = i mod k
    c = j mod k
    t = l*k+c
    return al t-lea bit din B[i',j']
end
```

Funcţia execută indiferent de dimensiunea datelor de intrare două împărţiri cu rest, o înmulţire, o adunare şi extragerea unui bit dintr-un număr (se poate realiza folosind o mască dacă limbajul/hard-ul nu oferă direct această operaţie), deci timpul de execuţie este constant $O(1)$.

Până acum am omis, pentru simplitate, faptul că orice locaţie de memorie are o dimensiune finită – să o notăm cu s. Este însă necesar ca fiecare valoare $B[i,j]$ (k^2 biţi) să "încapă" într-o locaţie de memorie. Aşadar condiţia care trebuie satisfăcută de k este $k^2 \le s$.

Alegând $k^2 = \log n$, deci $k = \sqrt{\log n}$, limitarea $k^2 \le s$ devine $\log n \le s$, deci $n \le 2^s$, care este satisfăcută deoarece n este stocat la rândul lui într-o locaţie de memorie (deci $n < 2^s$).

În acest fel dimensiunea matricei B este $\frac{n}{\sqrt{\log n}} \times \frac{n}{\sqrt{\log n}} = \frac{n^2}{\log n}$. Concluzia care se poate trage este că orice graf poate fi reprezentat pe $\frac{n^2}{\log n}$ locaţii de memorie, păstrând în acelaşi timp timpul constant de acces la componente oferit de matricea de adiacenţă.

O alternativă pentru reprezentarea grafurilor rare o reprezintă listele de adiacenţă, care, deşi nu oferă timp constant de acces, oferă un timp mediu de acces de $O\left(\frac{n}{\log n}\right)$. Constanta din spatele notaţiei O este mică, deci reprezentarea este potrivită pentru grafuri rare care nu au foarte multe noduri.

Problema 2.

Diametrul unui graf este lungimea maximă a unui drum de lungime minima între doua vârfuri ale grafului. Două vârfuri care sunt extremităţile unui drum minim de lungime maximă în graf se numesc diametral opuse. Demonstraţi că urmatorul algoritm determină o pereche de vârfuri diametral opuse într-un arbore T:

- dintr-un vârf oarecare se execută o parcurgere BFS a arborelui; fie u ultimul vârf vizitat;
- din vârful u se execută o parcurgere BFS a arborelui; fie v ultimul vârf vizitat;
- return u, v.

Rămâne valabil algoritmul pentru un graf conex oarecare ? **(4 puncte)**

Soluţie.

Parcurgerea BFS este numită parcurgere în lăţime deoarece algoritmul explorează sistematic muchiile lui *G* pentru a "descoperii" fiecare nod accesibil dintr-un nod de start *s*. Algoritmul lărgeşte, uniform frontiera dintre nodurile descoperite şi cele nedescoperite, pe lăţimea frontierei. Aceasta înseamnă că algoritmul descoperă toate nodurile de la distanţa *k* faţă de *s* înainte de a descoperi vreun nod la distanţa *k*+1.

Input: G – un graf, s – nod de start
Output:
delta – distanţele minime de la s la orice nod aceesibil din s
L – lista ce conţine nodurile în ordinea parcurgerii lor

```
procedure BFS(s)
begin
   for ∀u∈V(G) do
      vizitat[u] = false
      delta[u] = ∞
   Q = {s}
   vizitat[s] = true
   delta[s] = 0
   L = ∅
   while Q≠∅ do
      u = top(Q)
      adauga(L, u)
      for ∀v∈N_G(u) do
         if vizitat[v] = false then
            vizitat[v] = true
            push(q, v)
            delta[v] = delta[u] + 1
      pop(Q)
   return u
end
```

Fie L = (u_1=**r**, u_2, … , u_n) lista obţinută în urma parcurgerii BFS. Parcurgerea BFS este o parcurgere pe nivele (pentru oricare 2 nivele i şi j cu i<j, nodurile de pe nivelul i se află înaintea celor de pe nivelul j în lista L). Rezultă că vârful u_n se află pe utimul nivel al arborelui, sau, altfel spus, u_n se află la cea mai mare distanţă posibilă faţă de ***r***.

Algoritmul precizat efectuează 2 parcurgeri BFS. Fie L = (u_1=**r**, u_2, … , u_n) lista obtinuţă în urma primei parcurgeri BFS. Demonstrăm următoarea propoziţie:

Fie T=(V, E) un arbore oarecare cu n noduri, r rădăcina arborelui T şi ***d*** diametrul arborelui.
Avem: $\forall u,v \in V$, $\exists$ un drum unic de la u la v, notat $d(u,v)$

$$\boldsymbol{d} = \max\{\, d(u,v) \mid u,v \in V \,\}$$

Propoziţie

*Există un drum de lungime **d** pentru care u_n este una din extremităţi (u_n este unul dintre cele 2 vârfuri diametral opuse determinate de algoritm).*

Demonstaţie

Distingem 3 cazuri posibile:

a) *Există un drum de lungime **d** care porneşte din **r** .*

Ştim că u_n este nodul aflat la cea mai mare distanţă faţă de ***r***. Dar cea mai mare distanţă posibilă de la **r** la un alt nod este chiar diametrul **d** al grafului. Deci, d(u_n, **r**) = d(**r**, u_n)=**d** $\Rightarrow$ (q.e.d)

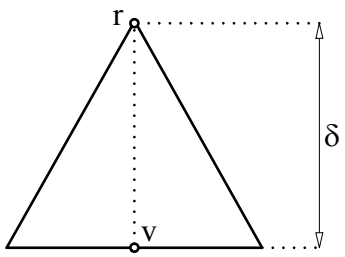


Fig. 2.1

b) *Nici un drum de lungime **d** nu pleacă din **r**, dar există un drum de lungime **d** care trece prin **r**.*

Fie A_1, A_2, …,A_m subarborii nodului ***r***, m>1. (pentru m=1 obţinem situaţia ***a)***)

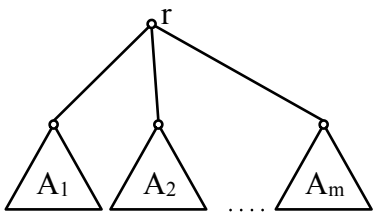


Fig. 2.2

Fie A si B cei mai înalţi 2 subarbori (h(A) ≥ h(B) , h(A) = max { h(A_i), 1 <= i <=m)).

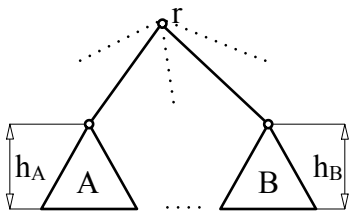


Fig. 2.3

Înălţimea arborelui cu rădăcina ***r*** va fi h(A)+1.

Diametrul arborelui va fi ***d*** = h(A)+h(B)+2 şi orice drum între un nod u de pe ultimul nivel al lui A şi un nod v de pe ultimul nivel al lui B este de lungime ***d***. Notăm această afirmaţie cu (*).

Apelând BFS(r), obţinem lista L=(u_1=**r**, u_2, … , u_n).

Cum u_n se află pe ultimul nivel al arborelui $\Rightarrow$ d(r, u_n) = h(A)+1

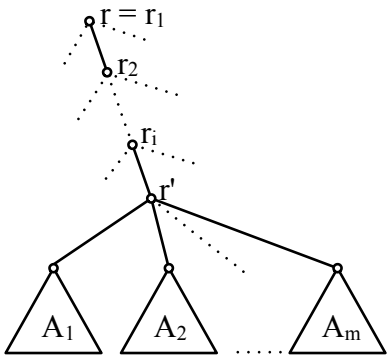
Caz (1) h(A) > h(A_i), $\forall A_i \neq A$ $\Rightarrow$ singurele noduri aflate la distanţa h(A)+1 faţă de ***r*** sunt cele de pe ultimul nivel al arborelui A.

Deci nodul u_n se află pe ultimul nivel al arborelui A. Dar din afirmaţia (*) rezultă că pentru toate nodurile u de pe ultimul nivel al lui A există cel puţin un drum de

lungime ***d*** care porneşte din u $\Rightarrow$ există cel puţin un drum de lungime ***d*** care porneşte din u_n (q.e.d.)

Caz(2) $A=A_{i1}$, $h(A_{i1}) = h(A_{i2}) = \ldots = h(A_{ik})$ cu $i1, \ldots, ik \in \{1, 2, \ldots, m\}$ $\Rightarrow$ **d**=2*h(A)+2 şi orice drum de lungime ***d*** pleacă dintr-un nod u de pe ultimul nivel al lui A_i şi ajunge într-un nod v de pe ultimul nivel al lui A_j , unde $i, j \in \{ i1, \ldots, ik\}$ şi $i \neq j$.
Dar ştim că $d(r,u_n)=h(A)+1$ $\Rightarrow$ u_n se află pe ultimul nivel al unui arbore A_i, cu $i \in \{i1,\ldots,ik\}$.
Rezultă că există un drum de lungime ***d*** pentru care u_n este una din extremităţi (q.e.d.)

c) *Nici un drum de lungime **d** nu trece prin **r*** $\Rightarrow$ arborele cu rădăcina ***r*** are forma:



unde $i \geq 1$ şi $\exists$ cel puţin un drum de lungime ***d*** care trece prin **r'**.

Fig. 2.4

Notăm cu R' arborele cu rădăcina în nodul ***r'***.
Notăm cu $A_1, A_2, \ldots, A_m$ subarborii nodului ***r'***.
Fie A şi B cei mai înalţi 2 subarbori ($h(A) \geq h(B)$, $h(A) = \max \{ h(A_i), 1 <= i <= m))$.
Avem: **d** = h(A)+h(B)+2.

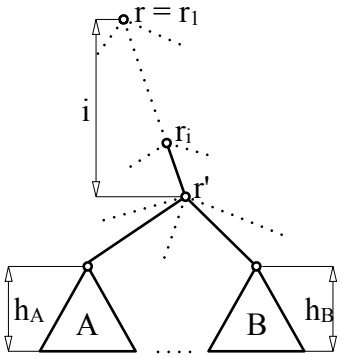


Fig. 2.5

Dar ***d*** este şi diametrul arborelui R'. Aplicând raţionamentul de la cazul b) pentru arborele R', se demonstrează că pentru orice nod u de pe ultimul nivel al arborelui R' (nivelul h(A)+1 din R'), există un drum de lungime ***d*** în R' , drum pentru care u este una din extremităţi. Notăm această afirmaţie cu (**).

Apelând BFS(r), obţinem lista $L=(u_1=r, u_2, \ldots, u_n)$, cu u_n pe ultimul nivel al arborelui cu rădăcina ***r*** (nivelul i+1+h(A)).
Demonstrăm că u_n se află de fapt pe nivelul h(A)+1 al arborelui R'.
Pentru aceasta demonstrăm prin reducere la absurd că nu există nici un nod terminal v astfel încât $v \notin R'$ şi $d(r,v) \geq i+1+h(A)$

Demonstaţie:
Presupunem că $\exists$ v, v $\notin$ R' şi d(r,v) $\geq$ i+1+h(A) $\Rightarrow$ $\exists$ j $\in$ {1,2,…,i} astfel încât v aparţine arborelui cu rădăcina r_j.

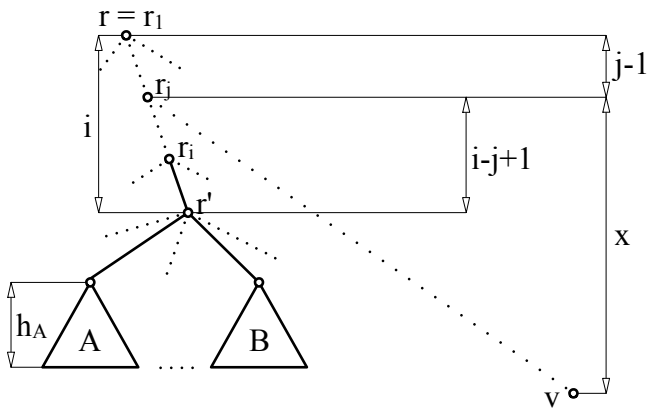


Fig. 2.6

Avem: i+1+h(A) ≤ j-1+x
i-j ≥ 0 ; i ≥ 1
h(A) ≥ h(B) ; ***d*** = h(A)+h(B)+2
Fie u un nod de pe ultimul nivel al arborelui A.
d(u, v) = d(u, r_j) + d(r_j, v)
= h(A)+1+(i-j)+1 + x $\Rightarrow$ d(u, v) ≥ h(A)+2 + 2(i-j) + h(A) + 2
Cum j-1+x ≥ i+1+h(A), avem x ≥ h(A)+2+(i-j).

Înlocuim h(A)+2 cu **d**-h(B) $\Rightarrow$ d(u, v) ≥ 2(i-j) + **d** - h(B) + h(A) + 2
Avem: i-j ≥ 0, h(A) - h(B) ≥ 0.
$\Rightarrow$ d(u, v) ≥ 0 + **d** + 0 + 2
$\Rightarrow$ d(u, v) ≥ **d** + 2
(fals pentru că diametrul ***d*** este distanţa maximă dintre 2 noduri ale arborelui)

Am demonstrat că orice nod de pe ultimul nivel al arborelui T se află de fapt pe ultimul nivel al arborelui R'. Cum u_n se află pe ultimul nivel al arborelui T, rezultă că u_n este de fapt pe nivelul h(A)+1 în arborele R'.

Afirmaţia (**) $\Rightarrow$ există un drum de lungime **d** pentru care u_n este una din extremităţi (q.e.d.)

Concluzie: a), b), c) $\Rightarrow$ după prima parcurgere BFS obţinem unul din cele 2 vârfuri diametral opuse (u_n).

După a doua parcurgere obţinem v_n, nodul aflat la cea mai mare distanţă faţă de u_n. Dar, cea mai mare distanţă faţă de u_n este diametrul grafului (din propoziţia anterioară). Deci avem că u_n şi v_n sunt două vârfuri diametral opuse.

Algoritmul precedent nu furnizează întotdeauna răspunsul corect atunci când este aplicat unui graf conex oarecare. Pentru a demonstra acest lucru considerăm următoarele contra-exemple (Fig. 2.7). În toate cele 3 cazuri se consideră nodul 1 ca nod de start pentru prima parcurgere BFS şi listele de adiacenţă ordonate crescător. În primul caz ordinea de parcurgere este 1,2,3,4 şi apoi 4,1,2,3, deci conform algoritmului 4 şi 3 sunt două vârfuri diametric opuse – afirmaţie falsă, deoarece 3 şi patru sunt adiacente (distanţa 1) iar distanţa între vârfurile 2 şi 3 este 2. Similar în

celelalte două cazuri prezentate se obţin 5 şi 4, respectiv 6 şi 5, vârfuri care sunt adiacente în timp ce diametrul ambelor grafuri este 2.

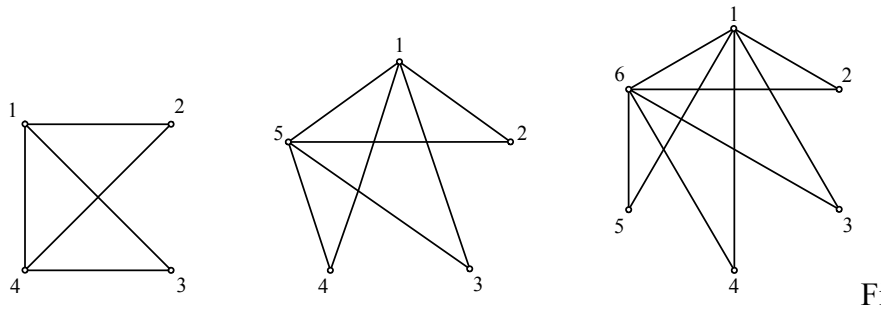


Fig. 2.7

Problema 3.

Fie *T* un arbore binar cu rădăcină. Un algoritm simplu de desenare a lui *T* poate fi descris recursiv după cum urmează.

- Folosim ca suport o grilă (liniatura unui caiet de matematică); vârfurile se plasează în punctele de intersecţie ale grilei.
- Desenăm subarborele stâng.
- Desenăm subarborele drept.
- Plasăm cele două desene unul lângă altul la distanţă pe orizontală doi şi cu rădăcinile la aceeaşi înălţime.
- Plasăm rădăcina cu un nivel mai sus la jumătatea distanţei pe orizontală dintre cei doi copii.
- Dacă avem doar un copil plasăm rădăcina cu un nivel mai sus la distanţa 1 faţă de copil (la stânga sau la dreapta după cum este acesta).

Descrieţi cum se poate asocia pentru fiecare nod *v* al arborelui *T* (folosind algoritmul de mai sus) coordonatele $(x(v), y(v))$ reprezentând punctul de pe grilă unde va fi desenat. **(3 puncte)**

Soluţie.

Pentru a asocia coordonatele fiecărui nod, realizăm mai întâi o parcurgere în postordine (subarbore stâng, subarbore drept, rădăcină) şi calculăm pentru fiecare nod *p:* $w_l[p]$ (şi $w_r[p]$) distanţa la cel mai din stânga (respectiv dreapta) fiu. Fie *p* un nod oarecare. Întâlnim 3 cazuri:

1) Nodul *p* are doi fii: *q* fiul stânga şi *r* fiul dreapta (Fig. 3.1). Atunci

$$\begin{cases} w_l[p] = w_l[q] + w_r[q] + 1 \\ w_r[p] = w_l[r] + w_r[r] + 1 \end{cases}$$

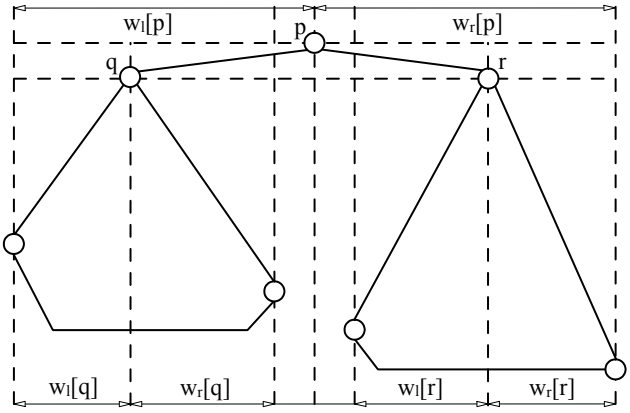


Fig 3.1

2) Nodul *p* are un singur fiu:
   a) Fie *q* fiu stânga (Fig. 3.2.a). Atunci:
$$\begin{cases} w_l[p] = w_l[q] + 1 \\ w_r[p] = \max(w_r[q] - 1, 0) \end{cases}$$
   b) Fie *q* fiu dreapta (Fig. 3.2.b). Atunci:
$$\begin{cases} w_l[p] = \max(w_l[q] - 1, 0) \\ w_r[p] = w_r[q] + 1 \end{cases}$$

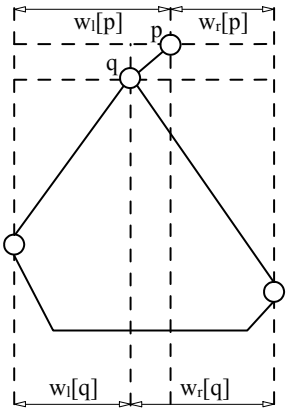


Fig 3.2.a

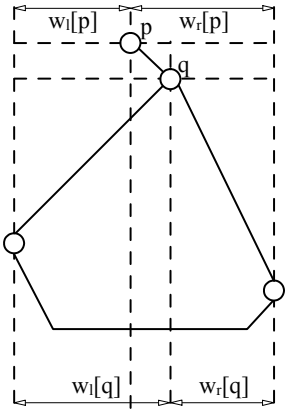


Fig 3.2.b

3) Nodul *p* nu are fii (nod terminal). Atunci $w_l[p] = w_r[p] = 0$.

Procedura recursivă care implementează această primă parcurgere este:

```
procedure postordine(p)
begin
   if p.left ≠ NULL and p.right ≠ NULL then
      postordine(p.left)
      postordine(p.right)
      wl[p] = wl[p.left]+wr[p.left]+1
      wr[p] = wl[p.right]+wr[p.right]+1
   else if p.left ≠ NULL then
      postordine(p.left)
      wl[p] = wl[p.left]+1
      wr[p] = max(wr[p.left]-1, 0)
   else if p.right ≠ NULL then
      postordine(p.right)
      wl[p] = max(wl[p.right]-1, 0)
      wr[p] = wr[p.right]+1
   else
      wl[p] = wr[p] = 0
end
```

Având calculate valorile din w_l şi w_r realizăm acum o parcurgere în preordine şi calculăm coordonatele fiecărui nod. Cazurile care apar sunt aceleaşi ca la prima parcurgere:

1) Nodul *p* are doi fii: *q* fiul stânga şi *r* fiul dreapta (Fig. 3.1). Atunci
$$\begin{cases} x[q] = x[p] - w_r[q] - 1 \\ x[q] = x[p] + w_l[q] + 1 \end{cases}$$
2) Nodul *p* are un singur fiu:

a) Fie q fiu stânga (Fig. 3.2.a). Atunci $x[q] = x[p] - 1$.

b) Fie q fiu dreapta (Fig. 3.2.b). Atunci $x[q] = x[p] + 1$.

3) Nodul p nu are fii (nod terminal), informaţiile au fost deja completate.

Algoritmul este următorul:

```
procedure preordine(p)
begin
   if p.left ≠ NULL and p.right ≠ NULL then
      x[p.left] = x[p]–wr[p.left]-1
      x[p.right] = x[p]+wl[p.right]+1
      y[p.left] = y[p.right] = y[p]-1
      preordine(p.left)
      preordine(p.right)
   else if p.left ≠ NULL then
      x[p.left] = x[p]-1
      y[p.left] = y[p]-1
      preordine(p.left)
   else if p.right ≠ NULL then
      x[p.right] = x[p]+1
      y[p.right] = y[p]-1
      preordine(p.right)
end
```

Procedura de desenare nu trebuie decât să facă iniţializări şi să ruleze cele două proceduri de parcurgere. La final tablourile x şi y vor conţine coordonatele fiecărui nod al arborelui cu rădăcina în punctul (x_0, y_0), primit ca parametru.

```
procedure draw(T, x0, y0)
begin
   postordine(T.rădăcină)
   x[T.rădăcină] = x0
   y[T.rădăcină] = y0
   preordine(T.rădăcină)
   plot(x,y)
end
```

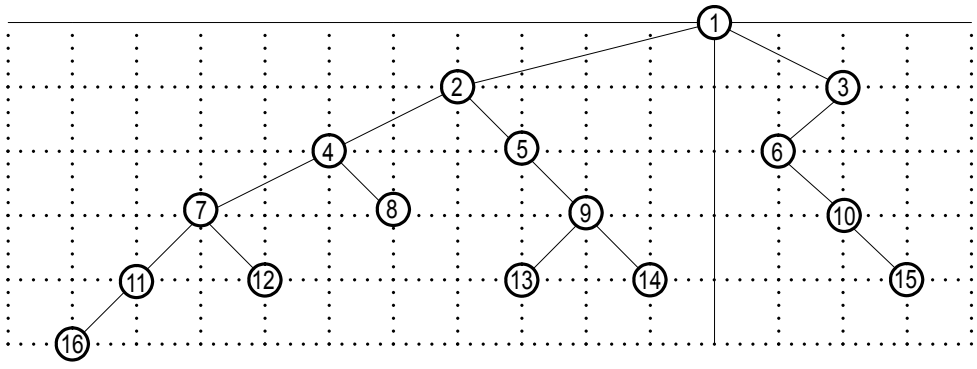


Fig.3.3

Exemplu. Fie graful din figura 3.3. După prima parcurgere valorile pentru w_l şi w_r se pot regăsi în tabelul următor:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

w_l	10	6	1	4	0	0	2	0	1	0	1	0	0	0	0	0
w_r	3	3	1	1	2	2	1	0	1	1	0	0	0	0	0	0

Coordonatele finale obţinute pentru punctul de plecare $(x_0, y_0) = (0,0)$ sunt:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
x	0	-4	2	-6	-3	1	-8	-6	-2	2	-9	-7	-3	-1	3	-10
y	0	-1	-1	-2	-2	-2	-3	-3	-3	-3	-4	-4	-4	-4	-4	-5

Problema 4.

Într-o sesiune de examene s-au înscris n studenti care trebuie să susţină examene dintr-o mulţime de m discipline. Întrucât examenele se susţin în scris, se doreşte ca toţi studenţii care dau examen la o disciplină să facă acest lucru simultan. De asemenea, regulamentul de desfăşurare a examenelor interzice ca un student să dea două examene în aceeaşi zi. Pentru fiecare student se dispune de lista disciplinelor la care doreşte să fie examinat.

Să se descrie construcţia unui graf G care să ofere răspunsul la urmatoarele două întrebări prin determinarea unor parametri asociaţi (care se vor preciza):

- care e numărul maxim de examene ce se pot organiza în aceeaşi zi ?
- care e numărul minim de zile necesare organizării tuturor examenelor?

(**3 puncte**)

Soluţie.

Fie S ={1,2,…,n} mulţimea studenţilor, D ={1,2,…,m} mulţimea disciplinelor. Pentru fiecare student *i* se dispune de lista disciplinelor la care va fi examinat: L(*i*).

Pentru a răspunde la cerinţele problemei, construim un graf G = (V, E), unde:

- V = D = mulţimea disciplinelor (un vârf al grafului reprezintă o disciplină)
- Pentru i,j $\in$ V, i $\neq$ j, avem {i,j} $\in$ E dacă există cel puţin un student care va fi examinat şi la disciplina i şi la disciplina j.

Dacă {i,j} $\in$ E, spunem despre disciplinele i şi j că sunt *incompatibile*.
Dacă {i,j} $\notin$ E, spunem despre disciplinele i şi j că sunt *compatibile*.

Cerinţa 1 : Care e numărul maxim de examene ce se pot organiza în aceeaşi zi?

Două sau mai multe examene pot fi date în aceeaşi zi dacă disciplinele respective sunt compatibile. Numărul maxim de examene ce se pot organiza în aceeaşi zi reprezintă de fapt cardinalul maxim al unei mulţimi de discipline compatibile.
O mulţime e formată din discipline compatibile $\Leftrightarrow$ e mulţime stabilă.
Deci răspunsul la cerinţa 1 va fi dat de numărul de stabilitate al grafului:
$\alpha(G)$ = max|S|, S mulţime stabilă in G.

Cerinţa 2 : Care e numărul minim de zile necesare organizării tuturor examenelor?

O organizare a tuturor examelor reprezintă o partiţie a mulţimii V cu proprietatea că oricare 2 discipline din aceeaşi submulţime sunt compatibile (submulţimile sunt mulţimi stabile de vârfuri). Semnificaţia faptului că 2 discipline sunt în aceeaşi submulţime este că cele 2 examene vor fi date în aceeaşi zi.

O asftel de partiţionare se realizează printr-o *p-colorare* a grafului, iar *p* va fi numărul de zile corespuzător organizării examenelor.
Deci răspunsul la cerinţa 2 va fi dat de numărul cromatic al grafului :

$\chi(G) = \min\{p \mid p \in \mathbf{N}^*, G \text{ are o p-colorare}\}$.

Algoritmul de construcţie pentru graful G=(V,E):

```
procedure build(G)
begin
    E = ∅
    for  i =1 to n do
        p = L(i)
        while p ≠ ∅ do
            q = succ(p)
            while q ≠ ∅ do
                E ← E ∪ {{ inf(p), inf(q) }}
                q ← succ(q)
            p ← succ(p)
end
```

Fiecare element dintr-o listă L(i) este format din :

- informaţie (o disciplină la care studentul va da examen) : inf(q)
- pointer la următorul element din lista : succ(q)

Pentru cazul cel mai nefavorabil (fiecare student va susţine examenul la toate disciplinele), complexitatea algoritmului este $O(nm^2)$.

Exemplu:

$S = \{1,2,3,4\},\ D = \{1,2,3,4,5,6\}$

$L = ((1,2,3,6),(1,3),(1,3),(1,2,4))$

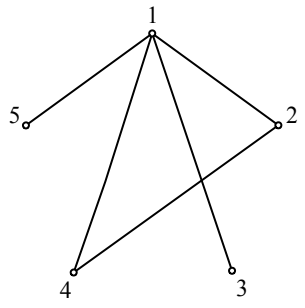


Fig. 4.1

Numărul maxim de examene ce se pot organiza în aceeaşi zi :

$\alpha(G) = 3 = |\{3, 4, 5\}| = |\{2, 3, 5\}|$

Numărul minim de zile necesare organizării tuturor examenelor?

$\chi(G) = 3 = |\{\{1\}, \{2\}, \{3,4,5\}\}|$

Problema 1

b) In continuare voi prezenta algoritmul care testeaza daca G este S-lant in timpul O(n+m):

```
procedure S_Lant
begin
        S_Lant ← true      // valoarea returnata de functie (adevarat sau fals)
        p ← A[i]
        repeat             // initializeaza toate elementele lui V cu 1
            V[p → ind] ← 1
            p ← p → urm
        until(p ≠ NULL)

        for( i ← 2, k ) do
            p ← A[i]
            repeat
                if(V[p → ind] = 0)
                then
                    S_Lant ← false
                    return
                else
                    p ← p → urm
            until(p ≠ NULL)
end
```

Analizand complexitatea algoritmului prezentat anterior, observam cu usurinta ca $T_A(n)=O(n+m)$ din cauza celor doua cicluri imbricate (for si repeat); ($k \leq n-1$).

Explicatie:

Algoritmul presupune ca G este deja S-sortat dupa gradul varfurilor din S.

Intai este creat tabloul V care reprezenta multimea varfurilor grafului G (care initial va fi initializat cu 1 (primul ciclu repeat)).

Apoi este parcurs tabloul A cu liste de adiacenta ale grafului. Elementele tabloului A sut pointeri catre liste de adiacenta.

Urmatorul pas il reprezinta parcurgerea celorlalte varfuri din S, verificand daca varfurile aflate in listele lor de adiacenta au mai fost intalnite la un pas anterior, caz in care algoritmul continua. Daca in schimb ele nu au fost gasite anterior, algoritmul se opreste cu raspunsul “false”, adica G nu este S-lant.

Problema 2:

A)

Trebuie sa aratam ca daca graful G este autocomplementar atunci G este conex. Daca G nu ar fi conex atunci el ar fi format din mai multe componente conexe. Consideram 2 varfuri din G astfel: primul varf il luam pe cel cu gradul cel mai mare (fie acesta M) si pe al doilea varful cu gradul cel mai mic (fie acesta m).Cele 2 varfuri exista evident!

Din ipoteza stim ca G este izomorf cu complementul sau.In complementul lui G = (V(G), P_2(V)-E(G)), nodul M va avea gradul cel mai mic si m va avea gradul cel mai mare din modul in care este definit complementul unui graf (mai exact m va avea n-m-1 noduri adiacente).Cum graful G este autocomplementar avem ca n-m-1 = M si deci m+M = n-1 (unde n ordinul lui G).

Daca cele doua varfuri se afla in aceeasi componenta conexa atunci un nod care nu apartine componentei conexe va avea minimum un vecin, deoarece consideram ca m>0.Daca componenta conexa considerata are n_1 varfuri, avem n_1 <= n-2. Dar m+M <= n_1 deoarece se afla in aceasi componenta conexa, de unde m+M <= n_1 <= n-2 adica m+M < n-1 ceea ce nu este adevarat.

Daca varfurile considerate se afla in componente conexe diferite, consideram n_1 si n_2 ordinul componentelor. Avem M < n_1 si m < n_2 sau M+1 <= n_1 si m+1 <= n_2 de unde M+m+2 <= n_1 + n_2 <= n, adica M+m+2 <= n, de unde M+m < n-1 care din nou nu este adevarata.

In concluzie G nu are mai multe componente conexe diferite, adica este conex.

In ce priveste ordinul sau putem avea una din situatiile:

1.Pentru fiecare 2 noduri din G intre care nu exista muchie trebuie sa existe in G alte 2 noduri intre care sa exista muchie ca atunci cand construim complementara sa se pastreze proprietatea de izomorfism.

2.Pentru fiecare 2 noduri adiacente trebuie sa existe 2 noduri intre care nu exista muchie din acelasi motiv ca la 1.

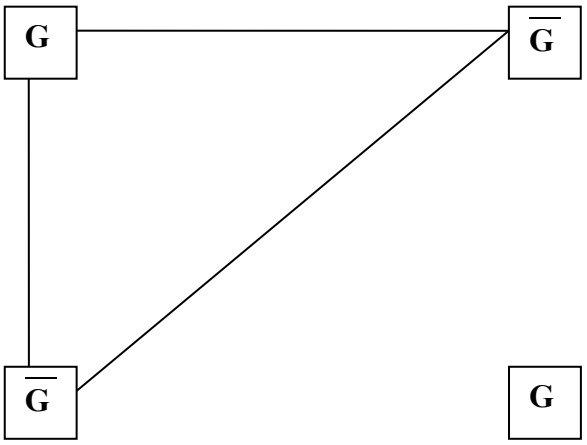
Deci numarul de varfuri (ordinul) din G trebuie sa fie multiplu de 4 (multiplu de perechi de 2 noduri a cate 2 noduri fiecare).Daca mai adaugam in G un nod astfel incat el este adiacent cu jumatate din nodurile din G in graful complementar el va fi adiacent cu cealalta jumatate deci proprietatea se pastreaza si in cazul in care ordinul lui G poate fi si multiplu de 4 plus 1.

De altfel graful G si complementarul sau prin reuniune formeaza un graf complet care are n(n-1)/2 noduri. Cum ordinul lui G este egal cu ordinul complementului avem ca ordinul lui G este n(n-1)/4. In plus stim ca n este numar natural deci n(n-1) este multiplu de 4 de unde n = 4k sau n = 4k+1.

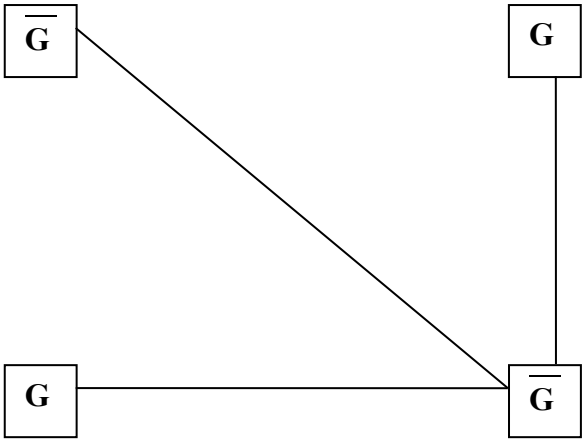
B)

Vom construi H autocomplementar astfel incat oricare ar fi G un graf dat sa avem ca G este subgraf indus in H.

Fie H astfel:



Complementul lui H este:



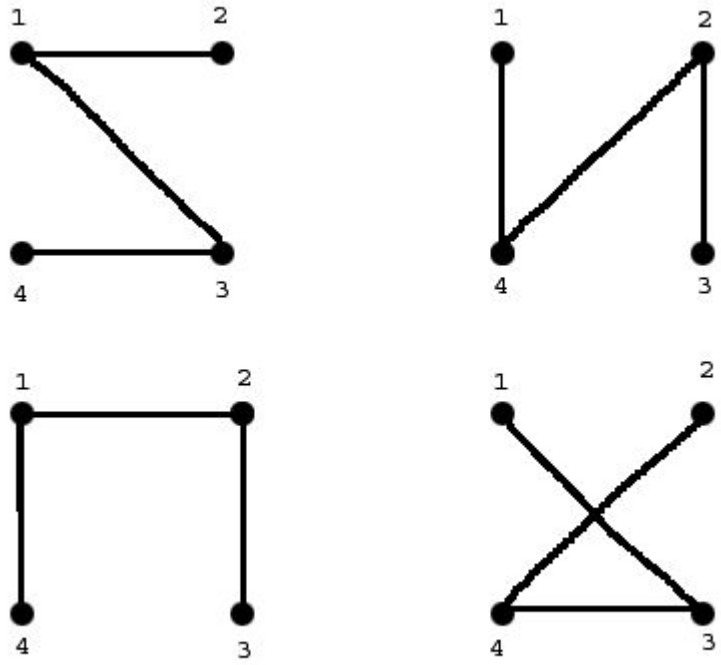
H astfel construit este evident autocomplementar, iar G este subgraf indus in H $G = (V(G), P_2(V(G)) \cap E(H))$, aceasta pentru orice G.

C)

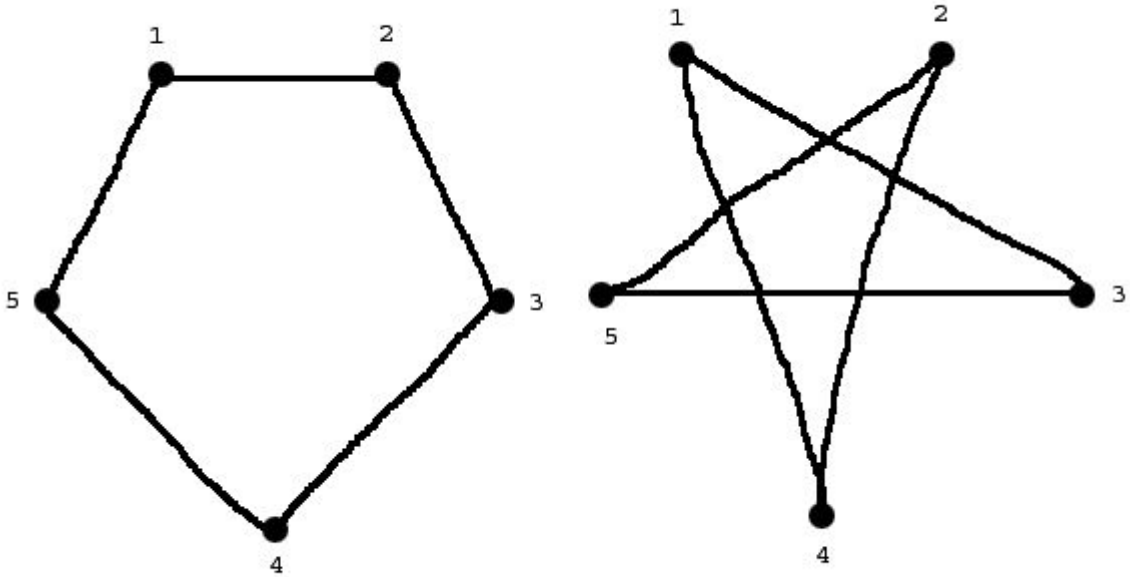
Trebuie sa consideram toate grafurile autocomplementare cu cel mult 7 varfuri. Cum ordinul unui astfel de graf trebuie sa fie multiplu de 4 sau multiplu de 4 plus 1 va trebui sa consideram grafurile cu 1, 4 sau 5 varfuri.

Cu un singur nod avem un singur graf:

Cu patru noduri avem urmatoarele grafuri :

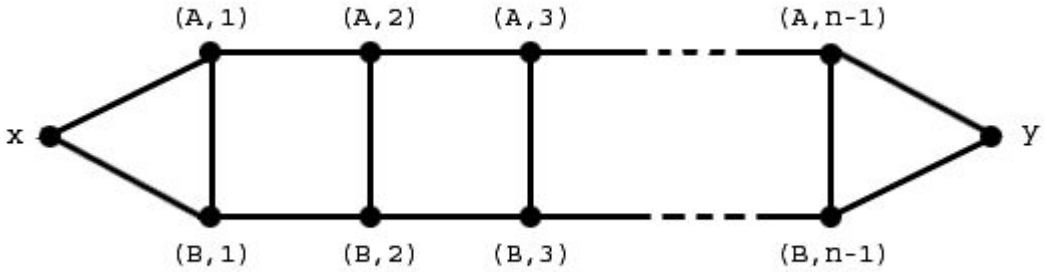


Cu cinci noduri avem grafurile :



Problema 3:

Graful G arata astfel:



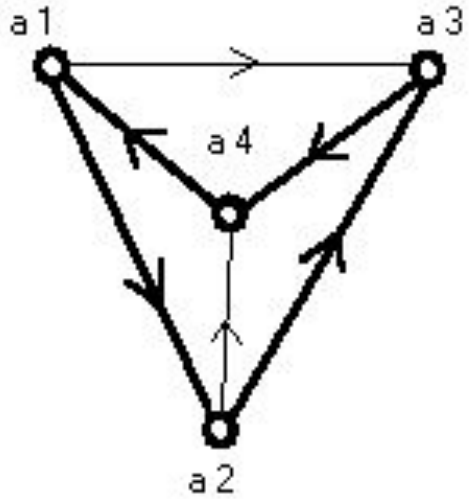
Un graf rar este un graf in care numarul de muchii m este mai mic decat (n^2 / log n), unde n este numarul de varfuri.

Aratam ca G este graf rar: G are (n-1)+(n-1)+2 = 2n noduri si (n-2)+(n-2)+(n-1)+4 = 3n-1 muchii. Deci trebuie sa aratam ca: 3n-1 <= ($4n^2$) / log (2n) care este evident adevarata (de altfel avem |E(G)| = O(|G|)), deci G este graf rar.

Acum trebuie sa estimam numarul de drumuri de la x la y. Exista maxim 2^n -1 drumuri de la nodul x la nodul y, unde n este dat de ordinul lui P_{n-1} , deci n este un intreg mare (Numarul de drumuri de la x la y poate fi determinat dat fiind asemanarea situatiei cu un arbore binar complet cu n nivele si x ca radacina si y ca frunza). Presupunerea programatorului L este deci falsa asa cum arata exemplul dat.

Problema 4

Vom nota nodurile turneului cu $a_1, a_2, \dots, a_k$.
Fie circuitul C = { $(a_1, a_2), (a_2, a_3), (a_3, a_4), \dots, (a_{n-1}, a_n), (a_n, a_1)$ }.



Fie acest exemplu simplu de turneu in care avem circuitul a_1, a_2, a_3, a_4 (cel ingrosat).

Demonstratia ceruta, "*pentru orice varf x al lui C se pot determina in timpul O(n) inca doua varfuri ale lui C, y si z a.i. (x, y, z) este un circuitt de lungime 3*" o voi arata prin intermediul unui algoritm, pe care il voi explica dupa prezentare.

Procedure seek(a, n) // *a* este un nod oarecare din C, iar *n* este numar de noduri ala lui C.

```
begin
                gasit ← false   // cand este gasit un circuit, gasit ia valoarea true
x ← a           // x ia valoarea nodului asupra caruia se efectueaza cautarea
y ← a + 1       // y ia valoarea succesorul lui a
z ← a + 2       // iar z ia valoarea succesorul lui a + 1

repeat
        if ( (x, y, z) circuit )
then                        // daca a fost gasit un circuit
        gasit ← true
else                        // daca nu s-a gasit un circuit
        y ← y + 1
        z ← z + 1
until ( gasit = true and z != x )    // defapt conditia z != x nici nu este necesara
        end.
```

Explicatie:

Plecand dintr-un nod oarecare x, algoritmul verifica initial daca intre x si urmatoarele doua noduri succesoare din circuit (y, z) exista un circuit de lungime 3, ceea ce inseamna sa existe o muche intre x si z cu directia dinspre z inspre x. Daca exista o astfel de muchie, atunci algoritmul se incheie (circuitul a fost gasit). Daca directia muchiei este dinspre x spre z, atunci trebuie cautate alte noduri y si z. Algoritmul va cauta in continuare un circuit intre x, succesorul lui y (fostul z) si succesorul lui z.

In cel mai rau caz, circuitul va fi gasit intre (x, x-1, x–2), deoarece, daca presupunem ca pana la pasul (x, x–2, x–3) nu a fost gasit un circuit, inseamna ca arcul dintre x si x-3 are directia spre x-3, ceea ce va rezulta ca la pasul urmator, (x, x-1, x-2) vom avea circuitul cautat.

Daca analizam algoritmul prezentat vom vedea ca are complexitatea in cazul cel mai defavorabil O(n).

Gordân Raluca Mihaela
Hriţcu Cătălin
An IIA, grupa 4
Grupa de seminar A34
Echipa nr. 2

Algoritmica grafurilor – tema 3

Problema 1.

Fie $G=(V,E)$ un graf cu *n* vârfuri, *m* muchii şi cu matricea de adiacenţă *A*. Dintre cele 2^m orientări posibile ale muchiilor sale considerăm una oarecare şi cu ajutorul ei construim matricea de incidenţă vârf-arc $Q \in \{0,1,-1\}^{n\times m}$ definită prin:

$(Q)_{ve}=-1$, dacă *v* este extremitatea iniţială a arcului *e*,

$(Q)_{ve}=1$, dacă *v* este extremitatea finală a arcului *e*,

$(Q)_{ve}=0$, în toate celelalte cazuri.

Demonstraţi că matricea $A+QQ^T$ este o matrice diagonală şi precizaţi semnificaţia combinatorică a elementelor ei. **(3 puncte)**

Soluţie.

Notăm matricea pătratică $n\times n$ $A+QQ^T=M$. Se observă că $M[i,j]=A[i,j]+(QQ^T)[i,j]\ \forall i,j\in V(G)$. Se disting două cazuri:

1) Dacă $i\neq j$ atunci $(QQ^T)[i,j]=\sum_{k\in E(V)} Q[i,k]\cdot Q[j,k]$. După cum *ij* poate sau nu să se regăsească în $E(V)$ avem două subcazuri:
   a) Dacă $ij\in E(G)$ atunci $A[i,j]=1$ şi, pentru $k\neq ij$ avem „*i* nu este extremitatea iniţială a arcului *k*" sau (sau inclusiv) „*j* nu este extremitatea finală a arcului *k*" afirmaţie echivalentă cu $Q[i,k]=0$ sau $Q[j,k]=0$. De aici: $Q[i,k]\cdot Q[j,k]=\begin{cases}0, \text{dacă } k\neq ij\\ -1, \text{dacă } k=ij\end{cases}$ şi deci $(QQ^T)[i,j]=\sum_{k\in E(V)} Q[i,k]\cdot Q[j,k]=-1$, iar $M[i,j]=1-1=0$.
   b) $ij\notin E(G)$ atunci $A[i,j]=0$ şi $(QQ^T)[i,j]=\sum_{k\in E(V)} Q[i,k]\cdot Q[j,k]=0$, deci $M[i,j]=0$.

   Deci matricea *M* este diagonală.
2) Dacă $i=j$ atunci avem $A[i,i]=0$ şi $(QQ^T)[i,i]=\sum_{k\in E(V)}(Q[i,k])^2=\sum_{k=ix}(-1)^2+\sum_{k=xi}(1)^2+0=d_G^-(i)+d_G^+(i)=d_G(i)$ iar $M[i,i]=d_G(i)$.

Deci $M=\underset{i=1,n}{diag}[d_G(i)]$ - matricea diagonală a gradelor.

Exemplu. Fie graful *G* desenat în Fig. 1.1 cu matricea de adiacenţă *A* (Fig 1.2). Considerăm o orientare pentru muchiile sale (Fig. 1.3). Matricea de incidenţă vârf-arc *Q* este reprezentată în Fig. 1.4 iar transpusa ei Q^T în Fig. 1.5. Înmulţind *Q* cu Q^T se

obţine matricea din Fig. 1.6. care adunată cu matricea de adiacenţă A va produce ca rezultat $M = diag[3 \quad 2 \quad 3 \quad 3 \quad 3]$, matricea diagonală a gradelor grafului G.

Fig 1.1

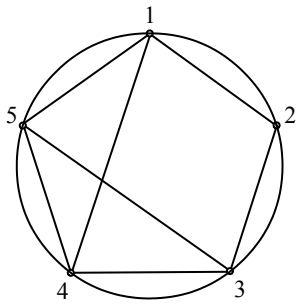


Fig. 1.2

A	1	2	3	4	5
1	0	1	0	1	1
2	1	0	1	0	0
3	0	1	0	1	1
4	1	0	1	0	1
5	1	0	1	1	0

Fig. 1.3

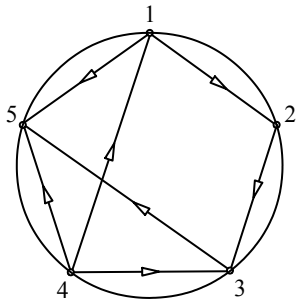


Fig 1.4

Q	12	23	34	45	35	41	15
1	-1	0	0	0	0	1	-1
2	1	-1	0	0	0	0	0
3	0	1	-1	0	-1	0	0
4	0	0	1	-1	0	-1	0
5	0	0	0	1	1	0	1

Fig 1.5

Q^T	1	2	3	4	5
12	-1	1	0	0	0
23	0	-1	1	0	0
34	0	0	-1	1	0
45	0	0	0	-1	0
35	0	0	-1	0	1
41	1	0	0	-1	0
15	-1	0	0	0	1

Fig. 1.6

QQ^T	1	2	3	4	5
1	3	-1	0	-1	-1
2	-1	2	-1	0	0
3	0	-1	3	-1	-1
4	-1	0	-1	3	-1
5	-1	0	-1	-1	3

Problema 2.

Fie G un graf oarecare şi notăm cu *b(G)* graful obţinut din G prin inserarea câte unui nou nod pe fiecare muchie. Demonstraţi că *b(G)* este un graf bipartit. (**2 puncte**).

Demonstraţi că două grafuri G şi H sunt izomorfe dacă şi numai dacă *b(G)* este izomorf cu *b(H)*. Deduceţi că testarea izomorfismului a două grafuri oarecare se reduce polinomial la testarea izomorfismului a doua grafuri bipartite **(2 puncte).**

Soluţie.

Anexa 1.

Exemplu. Fie graful G reprezentat în Fig. 2.1. Se observă imediat că adăugând un nou nod pe fiecare dintre muchii se obţine graful bipartit $b(G) = (V(G), E(G); E(b(G)))$ (Fig. 2.2). De asemenea se poate observa că fiecare nod din $E(G)$ are aritate 2.

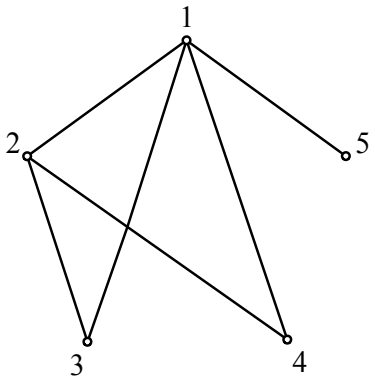


Fig 2.1

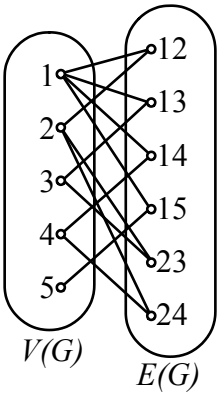


Fig 2.2

Problema 3.

Graful paianjen cu *n* vârfuri este graful care se obţine unind unul din vârfurile de grad 1 ale grafului P_3 cu toate vârfurile unui graf oarecare cu $n-3$ vârfuri, disjunct de P_3 (*n* este un întreg pozitiv mare). Dacă *G* este un graf cu *n* vârfuri reprezentat prin matricea de adiacenţă, arătaţi că se poate testa dacă este graf paianjen folosind doar $O(n)$ *probe* are matricii de adiacenţă. (o probă este un acces la un element al matricii, fără a-l memora explicit pentru utilizări ulterioare). **(4 puncte)**

Soluţie.

Orice graf paianjen are: o coada *t* (tail), un corp *b* (body), un cap *h* (head) şi $n-3$ picioare f_i (feet) (Fig. 3.1). Deoarece graful are cel puţin un picior atunci orice graf paianjen are cel putin 4 vârfuri.

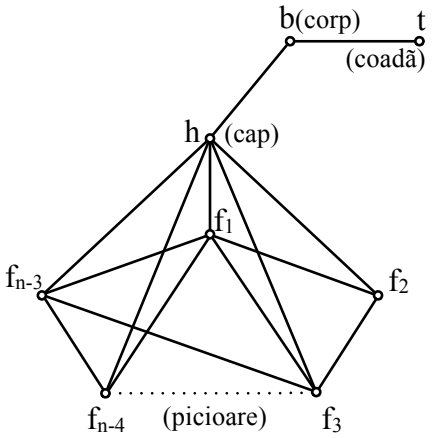


Fig. 3.1

Dacă graful *G* are 4 vârfuri ($n=4$) atunci el este graf paianjen doar dacă este izomorf cu P_4 (la P_3 se adaugă un graf cu un singur vârf). Testarea se face calculând gradul fiecărui vârf $d_G(v)\forall v\in V(G)$ şi apoi verificând să avem două vârfuri de grad 1 şi două de grad 2. Se observă că numărul de accesări ale matricii de adiacenţă este în acest caz constant egal cu $12(=4\cdot 3)$.

Să considerăm în continuare $n\geq 5$. Mai întâi considerăm un vârf oarecare $v\in V(G)$ şi calculăm gradul său $d_G(v)$ (prin $n-1$ accesări ale matricii de adiacenţă). Se poate întâlni unul din următoarele cazuri.

1) Dacă $d_G(v)=0$ atunci graful G nu poate fi graf paianjen deoarece nu este conex. La fel dacă $d_G(v)=n-1$, deoarece coada are gradul 1, corpul are gradul 2, capul are gradul $n-2$ şi gradul picioarelor nu poate depăşi $n-3$.

2) Dacă $d_G(v)=n-2$ atunci vârful v este cap. Observăm că singurul vârf neadiacent cu capul este coada, deci singurul vârf x care poate fi coadă se obţine realizând maxim $n-1$ probe. Se calculează apoi $d_G(x)$ ($n-1$ probe) şi dacă $d_G(x)\neq 1$ atunci G nu este graf paianjen. Altfel (x este coadă) fie $N_G(x)=\{y\}$, calculăm $d_G(y)$ ($n-1$ probe) şi dacă obţinem $d_G(y)=2$ atunci y este corpul şi deci G este graf paianjen. Pentru $d_G(y)\neq 2$ G nu poate fi graf paianjen.

3) Dacă $d_G(v)=1$ atunci vârful v poate fi sau coadă sau picior. Fie u singurul adiacent al vârfului v (acest vârf poate fi reţinut în timpul calculării gradului deci nu necesită probe suplimentare), se va calcula $d_G(u)$ ($n-1$ probe):
   a) Dacă $d_G(u)=2$ atunci vârful v este coadă şi u poate fi corp. Fie $N_G(u)=\{v,w\}$, atunci se calculează $d_G(w)$ şi dacă $d_G(w)=n-2$ (w este cap) atunci graful este graf paianjen, în caz contrar graful nu este graf paianjen.
   b) Dacă $d_G(u)=n-2$ atunci vârful u este cap. Algoritmul continuă ca la cazul 2) identificând pe v cu u.
   c) Dacă $d_G(u)\neq 2$ şi $d_G(u)\neq n-2$ graful nu poate fi graf paianjen.

4) Dacă $d_G(v)=2$ atunci vârful v este corp sau picior. Indiferent de caz el trebuie să aibă legătură directă cu capul (vârful de grad $n-2$). deci considerând $N_G(v)=\{z,t\}$, unul dintre vârfurile z şi t este capul. Se calculează deci gradele pentru z şi t ($2n-2$ probe) şi dacă $d_G(z)=n-2$ ori (xor) $d_G(t)=n-2$ atunci am găsit capul şi algoritmul se continuă ca la cazul 2) identificând pe v cu z (sau t). În caz contrar algoritmul se tremină cu răspus negativ.

5) Dacă $3\le d_G(v)\le n-3$ atunci vârful v este picior. Problema care se pune este determinarea unui vârf „interesant” (vârf care poate fi încadrat în unul din cazurile anterioare). Fie $H=N_G(v)$ mulţimea adiacenţilor vârfului v şi $T=V-(H\cup\{v\})$ mulţimea neadiacenţilor vârfului v. Atunci capul trebuie să fie din H iar coada (şi corpul) trebuie să fie din T. Alegem arbitrar $x\in H$ şi $y\in T$ şi cât timp H şi T sunt ambele nevide repet:
   a) Dacă x şi y sunt adiacente ($xy\in E(G)$) atunci se elimină y din T deoarece nu poate fi coada. Presupunând prin absurd că y ar fi coadă avem că de la orice picior se ajunge la y pe un drum de dimensiune minimă 3 (picior–cap–corp–coadă). Dar v,vx,x,xy,y este un drum de dimensiune 2 de la v la y (contradicţie). Se alege arbitrar un nou y din T.
   b) Dacă x şi y nu sunt adiacente atunci se elimină x din H, deoarece x nu poate fi cap decât dacă y este coada. Se alege arbitrar un nou x din H.

   Se observă că procesarea anterioră creează mulţimile T şi H folosind $n-1$ probe iar apoi repetă testarea $xy\in E(G)$ de maxim $n-1$ ori (numărul de elemente din

$T \cup H$ este iniţial $n-1$ şi scade la fiecare pas cu o unitate, în final fiind maxim egal cu unu). Cum însă este interzisă memorarea rezultatelor probelor, se va renunţa la T şi H; în schimb se consideră x şi y indici pe linia v în matricea de adiacenţă, x parcurgând nodurile adiacente cu v iar y cele neadiacente cu v. În acest fel numărul de accesări creşte de la $2n-2$ la $3n-3$ pentru această parte a cazului 5).

Dacă G este graf paianjen atunci procesarea se încheie cu H vidă şi y va conţine potenţiala coadă. Se calculează gradul lui y şi dacă $d_G(y)=1$ atunci am găsit coada şi algoritmul continuă ca la cazul 3) identificând v cu y. Dacă procesarea se încheie cu T vidă sau cu $d_G(y)\neq 1$ atunci G nu este graf paianjen (nu are coada).

Se observă că numărul maxim de probe ale matricei de adiacenţă este maximul dintre: 0 pentru cazul 1), $3n-3$ pentru cazul 2), $4n-4$ pentru 3) (subcazul b)), $5n-5$ pentru cazul 4) şi $8n-8$ pentru cazul 5). La aceste valori se mai adaugă probele făcute pentru determinarea gradului vârfului de start v ($n-1$). Deci testarea unui graf paianjen se face cu maxim $9n-9=O(n)$ probe ale matricii de adiacenţă.

Obs. Pentru a minimiza numărul de accesări ale matricii de adiacenţă s-a presupus ca fiind permisă reţinerea a două noduri adiacente cu nodul curent în timpul calculării gradului. Rutina care calculează gradul unui nod va face în continuare doar $n-1$ probe:

```
function gradExtins(v,&x,&y)
begin
   grad = 0
   for ∀u∈V(G) do
      if u != v then
         if a[u,v] = 1 then
            grad = grad + 1
            y = x
            x = u
   return grad
end
```

Exemplu.

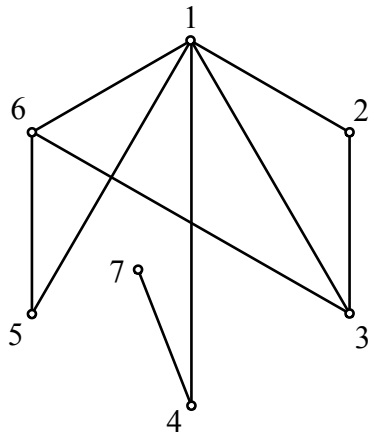


Fig. 3.2

Fie graful G cu $n=7$ muchii, dat de Fig. 3.2. Dacă începem testarea grafului cu vârful 1, obţinem $d_G(1)=5=n-2$, prin urmare vârful 1 nu poate fi decât capul paianjenului (cazul 2)). Aflăm singulul nod din graf neadiacent cu nodul 1, care este 7, posibilă coadă. Se calculează $d_G(7)=1$ şi se trece la testarea singurului adiacent, deci la vârful 4. Cum $d_G(7)=2$ am găsit şi corpul paianjenului, algoritmul se încheie cu răspuns afirmativ.

Dacă începem cu vârful 7, obţinem $d_G(7)=1$, deci 7 poate fi coadă sau picior. Fie singurul adiacent pentru 7, vârful 4. Calculăm $d_G(4)=2$ deci 7 este coadă iar 4 este picior (cazul 3).a)). Rămâne să aflăm gradul lui 1 (celălalt adiacent pentru 4) şi cum acesta este $d_G(1)=5=n-2$ tragem din concluzia că graful este graf paianjen.

Dacă începem cu vârful 2, obţinem $d_G(2)=2$, deci 2 poate fi corp sau picior (cazul 4)). Cum cei doi adiacenţi ai lui 2 sunt 1 şi 3 se calculează $d_G(1)=5=n-2$ şi $d_G(3)=3$ şi, deoarece doar una dintre valori este egală cu $n-2$ atunci vârful 1 este capul. Aflăm singulul vârf din graf neadiacent cu 1, care este 7, posibilă coadă. Se calculează $d_G(7)=1$ şi se trece la testarea singurului adiacent, deci la vârful 4. Cum $d_G(7)=2$ am găsit şi corpul paianjenului, algoritmul se încheie cu răspuns afirmativ.

Dacă începem cu vârful 3, obţinem $d_G(3)=3$, deci 3 nu este „interesant” (cazul 5)). Aflăm mulţimile $H=\{1,2,6\}$ şi $T=\{4,5,7\}$. La primul pas alegem un vârf din H şi unul din T, să presupunem că s-au ales iniţial $x=6$ iar $y=5$. Cum $65\in E(G)$ atunci se elimină 5 din T deoarece nu poate fi coadă (este la distanţă 2 de un picior). Noul $T=\{4,7\}$ iar $y=4$. Deoarece $64\notin E(G)$, 6 nu poate fi cap decât dacă 4 este coadă, se va elimina 6 din H. $H=\{1,2\}$ şi noul $x=1$. Cum $14\in E(G)$ avem că 4 nu poate fi coadă, deci se elimină din T, $T=\{7\}$ şi $y=7$. Deoarece $17\notin E(G)$ şi $27\notin E(G)$ 1 şi 2 sunt pe rând eliminate din H. În final avem $H=\varnothing$ şi cum $y=7$ şi $d_G(7)=1$ am găsit că 7 este coadă. Calculăm $d_G(4)=2$ deci 4 este picior. Rămâne să aflăm gradul lui 1 (celălalt adiacent pentru 4) şi cum acesta este $d_G(1)=5=n-2$ tragem din nou concluzia că graful este graf paianjen.

Dacă eliminăm din G vârful 7 ($G=G-7$) şi plecăm din nou din vârful 3 obţinem $d_G(3)=3$ (cazul 5)). Aflăm mulţimile $H=\{1,2,6\}$ şi $T=\{4,5\}$. Algoritmul rulează similar ca pe graful iniţial, presupunând că alegerile de la ficare pas rămân aceleaşi. La primul pas alegem un vârf din H şi unul din T, să presupunem că s-au ales iniţial $x=6$ iar $y=5$. Cum $65\in E(G)$ atunci se elimină 5 din T deoarece nu poate fi coadă (este la distanţă 2 de un picior). Noul $T=\{4\}$ iar $y=4$. Deoarece $64\notin E(G)$, 6 nu poate fi cap decât dacă 4 este coadă, se va elimina 6 din H. $H=\{1,2\}$ şi noul $x=1$. Cum $14\in E(G)$ avem că 4 nu poate fi coadă, deci se elimină din T. T devine $\varnothing$ şi algoritmul se încheie răspunzând că graful nu este graf paianjen (nu are coadă).

Problema 4.

Asociem unui arbore binar t de ordin n cu rădăcina r un drum P_{3n} orientat procedând astfel: fiecărui nod v al lui T i se asociază trei noduri cu acelaşi nume v pe care le desemnăm prin v_1 , v_2 , v_3 ; dacă v nu are în T descendent stâng, atunci se introduce arcul v_1v_2 în P_{3n} ; dacă v nu are în T descendent drept, atunci se introduce

arcul v_2v_3 în P_{3n}; dacă descendentul stâng al lui *v* în *T* este *w*, atunci se introduc în P_{3n} arcele v_1w_1 şi w_3v_2; dacă descendentul drept al lui *v* în *T* este *w*, atunci se introduc în P_{3n} arcele v_2w_1 şi w_3v_3.

Dacă se parcurge drumul P_{3n} de la extremitatea iniţială r_1 la extremitatea finală r_3 şi se listează numele vârfurilor în ordinea parcurgerii lor se obţine un şir în care numele fiecărui vârf al lui *T* apare exact de trei ori.

Demonstraţi că: dacă din acest şir se reţine doar prima apariţie a fiecărui nume se obţine parcurgerea *pre-order* a arborelui *T*; dacă din acest şir se reţine doar a doua apariţie a fiecărui nume se obţine parcurgerea *in-order* a arborelui *T*; dacă din acest şir se reţine doar a treia apariţie a fiecărui nume se obţine parcurgerea *post-order* a arborelui *T*. (**3 puncte**)

Soluţie.

Fie *L* lista obţinută în urma parcurgerii drumului P_{3n} de la extremitatea iniţială (r_1) la extremitatea finală (r_2).

Fie *v* un vârf oarecare al arborelui *T*. Fie *A* şi *B* subarborii săi.

Demonstrăm următoarea proprietate (P(*v*)) :

În lista *L*, vârful *v* apare:

- O dată înaintea tuturor nodurilor din subarborele stâng (*A*),
- O dată după toate nodurile din subarborele stâng (*A*) şi înaintea celor din subarborele drept (*B*),
- O dată după toate nodurile din subarborele drept (*B*).

(*L* este de forma : $L = \ldots\ v_1\ \underbrace{\ldots\ldots}_{A}\ v_2\ \underbrace{\ldots\ldots}_{B}\ v_3\ \ldots$)

Demostraţie:

Inducţie după numărul *m* de nivele ale arborelui:

1) $m=0$

   Fie *u* singurul nod al arborelui.

   Avem $L = u_1 u_2 u_3. \rightarrow$ Are loc P(*u*).

2) $m=1$

   a) *u* are numai fiu stâng :

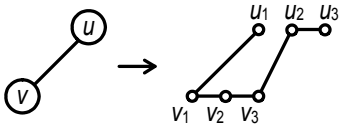


Fig. 4.1

   Vom obţine: $L = u_1\ \underbrace{v_1v_2v_3}_{A}\ u_2\ u_3\ . \rightarrow$ Are loc P(*u*).

   b) *u* are numai fiu drept :

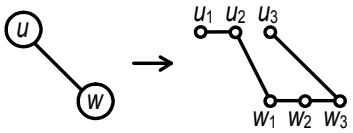


Fig. 4.2

Vom obţine: $L = u_1 \;\; u_2 \;\; \underbrace{w_1 w_2 w_3}_{B} \;\; u_3 \; . \rightarrow$ Are loc P(u).

c) u are 2 fii :

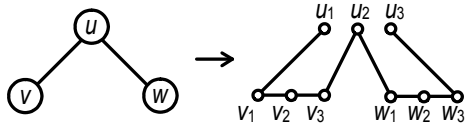


Fig. 4.3

Vom obţine: $L = u_1 \; \underbrace{v_1 v_2 v_3}_{A} \;\; u_2 \;\; \underbrace{w_1 w_2 w_3}_{B} \;\; u_3 \; . \rightarrow$ Are loc P(u).

3) Presupunem proprietatea adevărată pentru rădăcina unui arbore cu mai puţin de m nivele şi o demonstrăm pentru rădăcina unui arbore cu m nivele ($m>1$). Fie aceasta u .

Considerăm doar cazul în care u are şi subarbore stâng şi subarbore drept. Celelate 2 cazuri (numai subarbore stâng sau numai subarbore drept) se demonstrează similar.

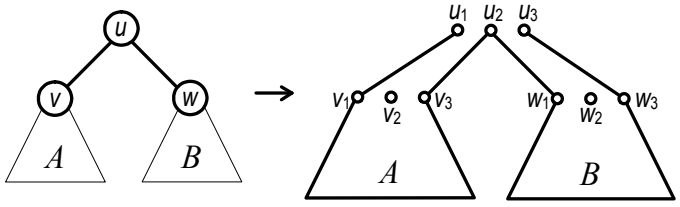


Fig. 4.4

Notăm:

A_v = lista nodurilor din subarborele stâng al nodului v (ordinea în listă este dată de parcurgerea drumului construit cu aceste noduri),

B_v = lista nodurilor din subarborele drept al nodului v,

A_w = lista nodurilor din subarborele stâng al nodului w,

B_w = lista nodurilor din subarborele drept al nodului w.

Deoarece arborii cu rădăcinile v şi w au cel mult m-1 nivele, conform ipotezei inductive → nodurile din A formează lista $L_A = v_1 \;\; A_v \;\; v_2 \;\; B_v \;\; v_3$; nodurile din B formează lista $L_B = w_1 \;\; A_w \;\; w_2 \;\; B_w \;\; w_3$.

Considerăm arborele cu rădăcina u → la drumurile formate din nodurile subarborilor A şi B adaugăm muchiile $u_1 v_1$, $v_3 u_2$, $u_2 w_1$ şi $w_3 u_3$ → un nou drum care va avea prima muchie $u_1 v_1$; urmează apoi muchiile formate cu nodurile din A, apoi muchiile $v_3 u_2$ şi $u_2 w_1$, apoi muchiile formate cu nodurile din B, apoi muchia $w_3 v_3$.

Primul nod din lista L va fi u_1, urmat de v_1. În nodul v_3 nu se ajunge decât după ce au fost parcurse toate nodurile din A (conform ipotezei inductive). Din v_3 se ajunge în u_2, apoi în w_1 (parcurgând muchiile $v_3 u_2$ şi $u_2 w_1$). În nodul w_3 nu se ajunge decât după ce au fost parcurse toate nodurile din B (conform ipotezei inductive). Ultima muchie parcursă este $w_3 v_3$, deci ultimul nod din listă va fi v_3 .

Deci lista va avea următoarea formă:

$L = u_1 \; \underbrace{v_1 ... v_2 ... v_3}_{A} \;\; u_2 \;\; \underbrace{w_1 ... w_2 ... w_3}_{B} \;\; u_3 \; . \rightarrow$ Are loc P(u).

1), 2), 3) → q.e.d.

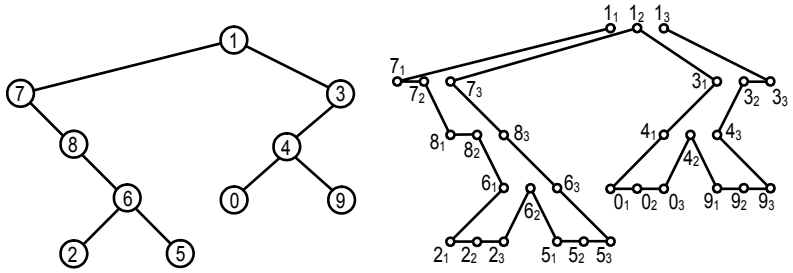
Deci, $\forall$ v un vârf din T (A şi B subarborii săi), u apare în lista L :

- O dată înaintea tuturor nodurilor din subarborele stâng (A),

- O dată după toate nodurile din subarborele stâng (*A*) şi înaintea celor din subarborele drept (*B*),
- O dată după toate nodurile din subarborele drept (*B*).

a) Din lista *L* se reţine doar prima apariţie a fiecărui nod → fiecare nod va apărea înaintea tuturor nodurilor din subarborele stâng şi subarborele drept → se obţine parcurgerea *pre-order* a arborelui.
b) Din lista *L* se reţine doar a doua apariţie a fiecărui nod → fiecare nod va apărea după toate nodurile din subarborele stâng şi înaintea tuturor nodurilor din subarborele drept → se obţine parcurgerea *in-order* a arborelui.
c) Din lista *L* se reţine doar a treia (ultima) apariţie a fiecărui nod → fiecare nod va apărea după toate nodurile din subarborele stâng şi subarborele drept → se obţine parcurgerea *post-order* a arborelui.

Exemplu. Fie arborele :



L = **1 7** 7 **8** 8 **6 2** 2 2 6 **5** 5 5 6 8 7 1 **3 4 0** 0 0 4 **9** 9 9 4 3 3 1
L = 1 7 **7** 8 **8** 6 2 **2** 2 **6** 5 **5** 5 6 8 7 **1** 3 4 0 **0** 0 **4** 9 **9** 9 4 **3** 3 1
L = 1 7 7 8 8 6 2 2 **2** 6 5 5 **5 6 8 7** 1 3 4 0 0 **0** 4 9 9 **9 4** 3 **3 1**

Gordân Raluca Mihaela
Hriţcu Cătălin
An IIA, grupa 4
Grupa de seminar A34
Echipa nr. 2

Algoritmica grafurilor – tema 5

Problema 1.

Să se arate că un graf G este bipartit dacă şi numai dacă orice subgraf indus H al lui G satisface proprietatea $2\alpha(H) \geq |H|$. **(3 puncte)**

Soluţie – Lupta împotriva pirateriei software.

În America pirateria software a atins proporţii alarmante. Pierderile companiilor din industrie sunt atât de mari încât este nevoie de un răspuns hotărât din partea autorităţilor. Se ştie că scena este formată din milioane de persoane (actori) fiecare având un rol bine precizat:

- *Distribuitorii* (suppliers) sunt în general angajaţi ai companilor dezvoltatoare de software. Ei au acces la noile produse înainte ca acestea să apară pe piaţă şi le furnizează iniţiaţilor.
- *Iniţiaţii* (crackers, ripers, packers, testers, coders, etc) sunt cei care care sparg protecţiile, elimină disfuncţionalităţile, testează şi împachetează versiuni pirat ale programelor. Ei sunt foarte greu de prins deoarece nu pun direct în circulaţie rodul muncii lor, folosindu-se în acest scop de intermediari.
- *Intermediarii* (couriers) întreţin situri web sau menţin liste de discuţii prin care pun în circulaţie şi răspândesc ilegal software.
- *Utilizatorii finali* (users) sunt cei care utilizează efectiv programele însă nu se bucură de încredere în cadrul scenei.

Observaţiile agenţilor FBI infiltraţi în rândul intermediarilor relevă că:

- Relaţiile dintre actorii implicaţi sunt bazate pe încredere (obs. relaţia de încredere nu este simetrică). Un actor va deveni suspicios şi nu va oferi informaţii importante unei persoane în care nu are suficientă încredere.
- Scena este împărţită în grupuri rivale neexistând încredere în cei din afara grupului. Fiecare grup are un actor principal C, care se bucură de cea mai mare încredere între toţi ceilalţi membrii.
- Fiecare actor se ascunde în spatele unor identitiţi virtuale şi pentru a-l putea prinde trebuie aflată mai întâi identitatea lui reală.

Atacul asupra fiecărui grup va începe prin capturarea acestei persoane C. La momentul actual fiecare grup este infiltrat de un agent care se bucură de încredere din partea unui număr restrâns de actori. Rolul agentului este de a determina identitatea persoanei C prin întrebarea unei persoane din grup. La rândul ei această persoană va întreba o altă persoană din grup şi aşa mai departe, până când întrebarea ajunge la C. Un actor va răspunde însă la întrebare doar dacă „drumul” pe care a sosit aceasta este de suspiciune minimă. Altfel el va înţelege tentativa de prindere a actorului principal şi agentul va fi desconspirat.

Odată actorul principal prins, agenţii FBI se vor folosi de identitatea sa virtuală pentru a prinde restul iniţiaţilor grupului. Ei vor întreba pe rînd fiecare iniţiat despre identitatea sa reală. Întrebarea va circula de la actor la actor pe un drum care să

stârnească cât mai puţină suspiciune pentru cel întrebat. Odată aflate identităţile tuturor iniţiaţilor aceştia vor fi prinşi şi activitatea grupului va fi întreruptă.
Cerinţe. Având ca intrare relaţiile de încredere între membrii unui grup de piraţi software date în procente, un agent A şi o listă de iniţiaţi, să se determine C, actorul care se bucură de încrederea cea mai mare în interiorul grupului, să se găsească un drum de suspiciune minimă de la A la C, apoi să se determine toate drumurile de suspiciune minimă de la C la fiecare iniţiat.
Indicaţii. Se determină vârful C prin calcularea maximului dintre sumele procentelor de încrede care intră în fiecare vârf. Se transformă apoi fiecare relaţie de încredere *i* în relaţie de suspiciune prin transformarea $s = 100 - i$ şi se inversează sensul tuturor arcelor. Acum ponderea arcului *uv* este suspiciunea pe care o stârneşte *u* atunci când pune o întrebare lui *v*. Se determină drumul de lungime minimă de la A la C şi apoi drumurile de lungime minimă de la A la fiecare iniţiat din listă.

Problema 2.

Demonstraţi că într-un graf bipartit *G* cu *n* vârfuri şi *m* muchii avem inegalitatea $4m \le n^2$. (**2 puncte**)

Descrieţi un algoritm care să testeze dacă un graf cu n vârfuri şi m muchii este complementarul unui graf bipartit în timpul $O(n+m)$ (**3 puncte**).

Soluţie.

Considerăm $V = \{1, 2, ..., n\}$. Fie N numărul maxim de mulţimi *st*-inevitabile disjuncte două câte două. Fie *D(s,t)* drumul în *G* de la *s* la *t*, de lungime $d = d(s,t)$.

$$D(s,t) = s,\ su_1,\ u_1,\ u_1u_2,\ u_2,\ u_2u_3,\ \ldots\ldots,\ u_{d-1},\ u_{d-1}t,\ t.$$

Demonstrăm pe rând că:

1) $N \le d$
2) $N \ge d$

1) Demonstrăm prin reducere la absurd că în orice mulţime *st*-inevitabilă trebuie să existe cel puţin o muchie de pe drumul de la *s* la *t*.

Fie *A* o multime de muchii *st*-inevitabilă $\Rightarrow \exists\ S \subset V$ astfel încât $s \in S$, $t \notin S$ şi
$A = \{e \in E \mid e = uv,\ u \in S,\ v \notin S\}$
Presupunem că $\forall\ e \in E(\ D(s,t)\ ),\ e \notin A$.

$s \in S$
$su_1 \notin A \Rightarrow$ trebuie ca $u_1 \in S$
Dar $u_1u_2 \notin A \Rightarrow u_2 \in S$
$u_2u_3 \notin A \Rightarrow \ldots \Rightarrow u_{d-1} \in S$
$u_{d-1}t \notin A \Rightarrow t \in S$ (contradicţie)

Deci, în orice mulţime *st*-inevitabilă trebuie să existe cel puţin o muchie de pe drumul de la *s* la $t \Rightarrow \exists$ cel mult *d* mulţimi *st*-inevitabile disjuncte două câte două ($N \le d$)

2) Demonstrăm că există măcar *d* mulţimi *st*-inevitabile disjuncte două câte două.

Construim mulţimile $A_0, A_1, \ldots, A_{d-1}$ astfel:
$\forall\ i \in \{0,1, ..., d-1\}\quad A_i = \{\ uv \in E \mid d(s,u) = i,\ d(s,v) = i+1\ \}$.

A_i este *st*-inevitabilă: $\exists\ S_i = \{\ w \in V\ \mid d(s,w) \le i\ \} \subset V$ astfel încât $s \in S_i$ $(d(s,s)=0)$, $t \notin S_i$ $(\ d(s,t)=d > i\ )$ şi $A_i = \{e \in E \mid e=uv,\ u \in S_i,\ v \notin S_i\ \}$.

Mulţimile A_i sunt disjuncte două câte două:
Fie $i, j \in \{0,1, ..., d-1\}$, $i \ne j$.
Avem $A_i = \{uv \in E \mid d(s,u) = i,\ d(s,v) = i+1\}$, $A_j = \{uv \in E \mid d(s,u) = j,\ d(s,v) = j+1\ \}$
$\Rightarrow A_i \cap A_j = \varnothing$.
Deci există măcar d mulţimi *st*-inevitabile disjuncte două câte două ($N \ge d$).

1), 2) $\Rightarrow$ $N = d$ (numărul maxim de mulţimi *st*-inevitabile disjuncte două câte două este egal cu distanţa în G de la s la t).

Se poate determina o familie de astfel de mulţimi cu ajutorul unei parcugeri BFS a lui G din s. Fie T arborele BFS asociat grafului G, cu s nod de plecare. Observăm că A_i = mulţimea muchiilor ce au o extremitate pe nivelul i al arborelui T şi cealaltă extremitate pe nivelul $i+1$. Mulţimea S_i corespunzătoare este formată din nodurile aflate pe nivelele 0, 1, ... , i ale arborelui T.
Următorul algoritm determină o familie de d mulţimi *st*-inevitabile disjuncte două câte două. Algoritmul foloseşte o parcurgere BFS în care pentru fiecare nod se reţine nivelul său în arborele BFS (arborele nu este construit efectiv). Algoritmul se opreşte atunci când se ajunge într-un nod de pe nivelul $d+1$ (nod aflat faţă de s la o distanţă > $d(s,t)$). În acest moment toate mulţimile A_i au fost construite.

```
procedure st_inevitabile()
begin
   C = { s }
   for  v = 1 to n do
      nivel[v] = ∞
   while ( C ≠ ∅ )  do
      v = top( C )
      for ( u ∈ Adj(v) )  do
         push( C, u )
         nivel[u] = nivel[v] + 1
         if ( nivel[t] ≥ nivel[u] ) then
            A[ nivel[u] ] = A[ nivel[u] ] ∪ {u,v}
         else
            return
end.
```

Exemplu. Pentru graful reprezentat în Fig. 2.1 alături de arborele parcurgerii sale BFS cele 3 mulţimi 1-11 inevitabile sunt:

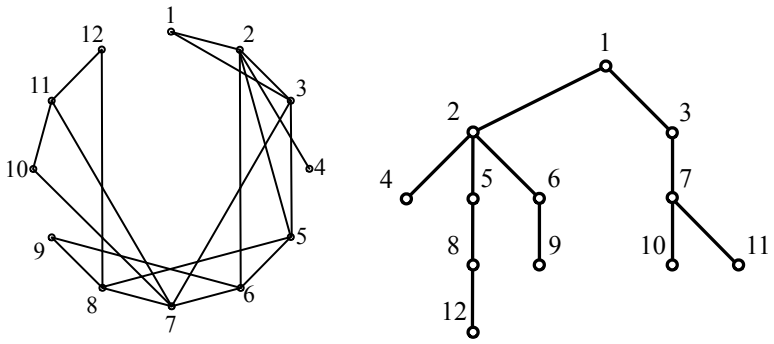


Fig. 2.1

$A_0 = \{ \{1,2\}, \{1,3\} \}$, $S_0 = \{1\}$
$A_1 = \{ \{2,4\}, \{2,5\}, \{2,6\}, \{3,7\} \}$, $S_1 = \{1, 2, 3\}$
$A_2 = \{ \{5,8\}, \{6,9\}, \{7,10\}, \{7,11\} \}$, $S_2 = \{1, 2, 3, 4, 5, 6, 7\}$

Problema 3.

Arătaţi că orice graf G cu m muchii are un graf parţial H bipartit şi cu cel puţin $\frac{m}{2}$ muchii. **(2 puncte)**

Soluţie.

a) Fie A maximală în (în raport cu incluziunea) satisfăcând proprietăţile: $v \in A, [A]_G$ este conex (1), $N = N_G(A) \neq \varnothing$ (2) şi $R = V - (A \cup N) \neq \varnothing$ (3). Demonstrăm prin reducere la absurd că atunci orice vârf din R este adiacent cu orice vârf din N.

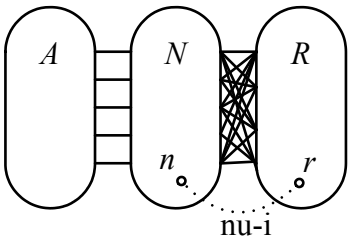


Fig. 3.1

Presupunem că $\exists n \in N$, $\exists r \in R$ astfel încât $nr \notin E(G)$ (Fig. 3.1). Atunci putem construi $A' = A \cup \{n\}$ cu

$$N' = N_G(A') = (N - \{n\}) \cup (N_G(n) \cap R) \text{ şi}$$
$$R' = V - (A' \cup N') = R - N_G(n).$$

Demonstrăm că A' satisface relaţiile (1), (2) şi (3).

(1) Deoarece $\exists n \in N_G(A)$ rezultă că $\exists a \in A$ astfel încât $an \in E(G)$. Dar $[A]_G$ este conex deci şi $[A']_G = [A \cup \{n\}]_G$

(2) Demonstrăm că $N - \{n\} \neq \varnothing$. Presupunând contrariul avem că $N = \{n\}$ şi deoarece graful G este conex avem în mod necesar că $nr \in E(G)$, ceea ce contrazice o presupunere anterioară. Deci $N - \{n\} \neq \varnothing$ şi prin urmare şi $N' \neq \varnothing$.

(3) Deoarece $r \notin N_G(n)$ şi $R' = R - N_G(n)$ avem că $r \in R'$, de unde $R' \neq \varnothing$.

Cum A' satisface relaţiile (1), (2) şi (3) şi $A \subset A'$, minimalitatea mulţimii A este contrazisă, deci orice vârf din R este adiacent cu orice vârf din N.

b) Fie graful G care este $\{C_k\}_{k \geq 4}$ -free şi A ce satisface relaţiile (1), (2), (3) şi este maximală în raport cu incluziunea. Demonstrăm prin reducere la absurd că A este clică în G.

Presupunem că $\exists n_1, n_2 \in N$, $n_1 \neq n_2$ şi $n_1 n_2 \notin E(G)$. Cum orice vârf din R este adiacent cu orice vârf din N şi $R \neq \varnothing$ avem că $\exists r \in R$ cu proprietatea că $n_1 r \in E(G)$ şi $n_2 r \in E(G)$. Deoarece $n_1, n_2 \in N_G(A)$ rezultă că $\exists a_1, a_2 \in A$ astfel încât $a_1 n_1, a_2 n_2 \in E(G)$. Dacă $a_1 = a_2$ atunci $(a_1, a_1 n_1, n_1, n_1 r, r, r n_2, n_2, n_2 a_1, a_1)$ este un circuit de lungime 4 în G (Fig. 3.2). Dacă $a_1 \neq a_2$ atunci există $\delta_G(a_1, a_2)$ un drum în G de la a_1 la a_2 de lungime cel puţin 1 (deoarece $[A]_G$ este conex). Dar în acest caz $(a_1, a_1 n_1, n_1, n_1 r, r, r n_2, n_2, n_2 a_2, a_2) + \delta_G(a_1, a_2)$ este un circuit de lungime cel puţin 5 în G (Fig. 3.3). Am obţinut că G are un circuit de lungime cel puţin 4, ceea ce contrazice faptul că G este $\{C_k\}_{k \geq 4}$-free. Prin urmare A este clică în G.

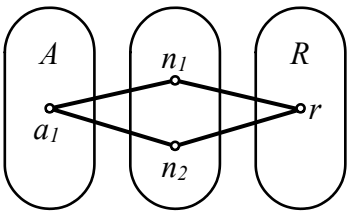


Fig . 3.2

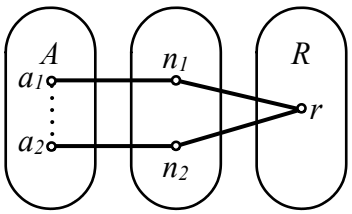


Fig. 3.3

c) Fie G un graf $\{C_k\}_{k \geq 4}$-free, regulat şi conexe şi fie $n = |V(G)|$. Demonstrăm prin re ducere la absurd că graful G este complet.

Presupunem că $\exists v \in V(G)$ cu $d_G(u) < n-1$. Deci $\exists A'$ maximală cu proprietăţile (1), (2) şi (3). Am demonstrat la punctul a) că atunci orice vârf din R este adiacent cu orice vârf din N. În plus deoarece G este $\{C_k\}_{k \geq 4}$-free atunci N este clică. Deci orice nod n din N este adiacent cu toate celelalte noduri din N ($|N|-1$) cu toate nodurile din R ($|R|$) şi cu cel puţin 1 nod din A. Cum A, N şi R sunt disjuncte obţinem

$$d_G(n) \geq |N| + |R| \quad (*).$$

Fie r un nod oarecare din R. Ştim că r nu are adiacenţi din A deci nu este adiacent decât cu toate nodurile din N ($|N|$) şi cu celelate nodurile din R (maxim $n-1$):

$$d_G(r) \leq |N| + |R| - 1 \quad (**).$$

Din relaţiile (*) şi (**) avem că $d_G(n) \neq d_G(n)$ ceea ce intră în contradicţie cu faptul că graful G este regulat. Deci graful G este complet.

Problema 4.

Demonstraţi că în orice graf conex $G = (V, E)$ există o mulţime stabilă S astfel încât graful bipartit $H = (S, V-S; E')$ este conex, unde $E' = E - \wp_2(V-S)$. Deduceţi că $\alpha(G) \geq \frac{|G|-1}{\Delta(G)}$ pentru orice graf conex G. **(3 puncte)**

Soluţie.

Fie $G = (V,E)$ un graf 3-regulat oarecare $\Rightarrow \forall\ v \in V$, $d_G(v) = 3$ (orice nod are exact 3 vecini).

Putem determina un circuit par în G astfel: pornind dintr-un nod oarecare (r), facem o parcurgere DFS (construind în acelaşi timp arborele DFS prin relaţia *părinte*) şi ne oprim la primul nod (u) pentru care au fost vizitaţi toţi vecinii săi. Cu ajutorul informaţiilor reţinute în vectorul *parinte* construim drumul de la u la vecinii săi (v, w si *t=parinte[u]*), aflaţi pe drumul de la r la u (deoarece au fost deja parcurşi).

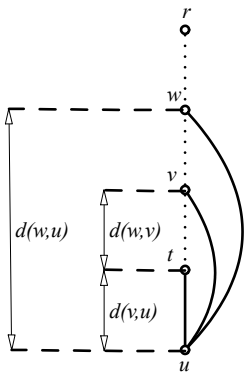


Fig. 4.1

Distingem următoarele cazuri:

1) *d(v,u)* este un număr impar ⇒ adăugând muchia *uv* la drumul de la *v* la *u*, obţinem un circuit par (lungimea sa fiind 1+*d(v,u)*)
2) *d(w,u)* este un număr impar ⇒ adăugând muchia *uw* la drumul de la *w* la *u*, obţinem un circuit par (lungimea sa fiind 1+*d(w,u)*)
3) *d(v,u)* şi *d(w,u)* sunt numere pare ⇒ *d(w,v)* este pară ⇒ adăugând muchiile *vu* şi *uw* la drumul de la *w* la *v*, obţinem un circuit par (lungimea sa fiind 2+*d(w,v)*)

Algoritmul de determinare a circuitului par este următorul:

```
procedure CircuitPar()
begin
   for (v ∈ V) do
      vizitat[v] = 0
      parinte[1] = -1
   DFS(1, u)   /* procedura va determina primul nod u */
               pentru care au fost vizitaţi toţi vecinii săi */
   t = parinte[u]
   v ← NG(u) – {t}
   w ← NG(u) – {t, v}
   /* Cazul 1) */
   i = u
   C1 = ( )     /* va retine circuitul u, ut, ..., v, vu */
   d = 0        /* d va retine lungimea drumului */
   while ( i ≠ v ) do
      j = parinte[i]
      adaug( C1, i );  adaug( C1, ij )
      d = d+1
      i = j
      if ( d = impar ) then
         adaug(C1, v); adaug(C1, vu); adaug(C1, u)
         return C1
   /* Cazul 2) */
   C2 = C1     /* va retine circuitul u, ut, ..., w, wu */
   while ( i ≠ w ) do
      j = parinte[i]
      adaug( C2, i );  adaug( C2, ij )
      d = d+1
```

```
        i = j
        if ( d = impar ) then
            adaug(C2, w); adaug(C2, wu); adaug(C2, u)
            return C2
    /* Cazul 3) */
    C = C2 – C1
    adaug(C, w); adaug (C, wu); adaug (C, u)
    adaug (C, uv); adaug(C, v)
    return C
end.

procedure DFS(i, u)
begin
    vizitat[i] = 1
    este_u = true
    for ( j ∈ Adj( i ) ) do
        if ( vizitat[j] = 0 ) then
            then  este_u = false
            parinte[ j] = i
            DFS(j, u)
    if  (este_u = true ) then
        u = i
        return
end.
```

Exemple.

Cazul 1) Fie graful din Fig. 4.2. Circuitul obţinut de algoritm este

C = (6, 65, 5, 54, 4, 43, 3, 36, 6)

şi are lungimea 4 – circuit par.

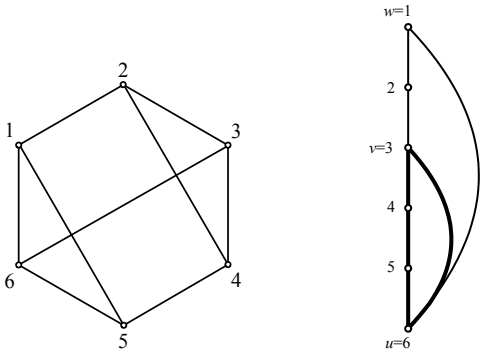


Fig. 4.2

Cazul 2) Fie graful din Fig. 4.3. Circuitul obţinut de algoritm este

C = (4, 43, 3, 32, 2, 21, 1, 14, 4)

şi are lungimea 4 – circuit par.

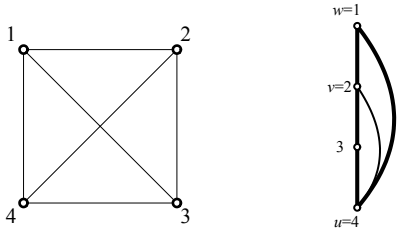
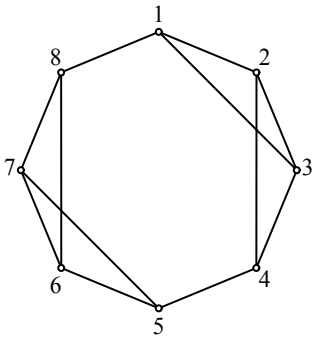


Fig 4.3

Cazul 3) Fie graful din Fig. 4.4. Circuitul obţinut de algoritm este

C = (6, 65, 5, 54, 4, 43, 3, 31, 1, 18, 8, 86, 6)

şi are lungimea 6 – circuit par.



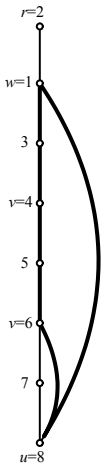


Fig. 4.4

Gordân Raluca Mihaela
Hriţcu Cătălin
An IIA, grupa 4
Grupa de seminar A34
Echipa nr. 2

Algoritmica grafurilor – tema 5

Problema 1.

Să se arate că un graf G este bipartit dacă şi numai dacă orice subgraf indus H al lui G satisface proprietatea $2\alpha(H) \geq |H|$. **(3 puncte)**

Soluţie.

Direct. Fie $G = (S, T; E)$ un graf bipartit.
$S \cup T = V$, $S \neq \emptyset$, $T \neq \emptyset$, $S \cap T = \emptyset$, S şi T sunt mulţimi stabile de vârfuri.

Fie $A \subseteq V$, $A \neq \emptyset$, $H = G[A]$. Trebuie să demonstrăm că $\alpha(H) \geq \frac{|A|}{2}$.

Caz 1. Dacă $A \subseteq S$ sau $A \subseteq T$, atunci $\alpha(H) = |A| \geq \frac{|A|}{2}$.

Caz 2. $A \cap S = S' \neq \emptyset$, $A \cap T = T' \neq \emptyset \Rightarrow \{S', T'\}$ este o bipartiţie pentru A.
$\forall u, v \in S' \Rightarrow u, v \in S \Rightarrow uv \notin E(G)$. Dar $E(G) \supseteq E(H) \Rightarrow uv \notin E(H) \Rightarrow S'$ este mulţime stabilă de vârfuri din H. Analog pentru mulţimea T'.
Deci $H = (S', T'; E(H))$ este graf bipartit.

$$\left.\begin{array}{l} |S'| + |T'| = |A| \Rightarrow |S'| \geq \frac{|A|}{2} \text{ sau } |T'| \geq \frac{|A|}{2} \\ \text{dar } S' \text{ şi } T' \text{ sunt mulţimi stabile} \end{array}\right| \Rightarrow \alpha(H) \geq \frac{|A|}{2}.$$

Reciproc. Fie $G = (V(G), E(G))$ un graf oarecare. Ştim că orice subgraf indus H al lui G ($H=G[A]$) satisface proprietatea $2\alpha(H) \geq |H|$.

Presupunem că G nu e bipartit $\Rightarrow$ există cel puţin un circuit de lungime impară.
Fie C circuit, cu $|C| = 2n + 1$, $n \geq 1$. Notăm vârfurile din C cu $x_1, x_2, ..., x_{2n+1} \Rightarrow$
$C = (x_1, x_1x_2, x_2, x_2x_3, x_3, ..., x_{2n}x_{2n+1}, x_{2n+1}, x_{2n+1}x_1, x_1)$

Fie $H = G[V(C)]$. Demonstrăm că $\alpha(H) \leq n$.

În cel mai favorabil caz, $E(H) = E(C)$ (în H nu există alte muchii decât cele care formează circuitul). Se observă că 2 vârfuri x_i şi x_{i+1} nu pot face parte din aceeaşi mulţime stabilă (vârfurile sunt adiacente). La fel pentru x_{2n+1} şi x_1.
Construim o mulţime stabilă de vârfuri A ($A \subseteq V(H)$) de cardinal maxim:

$$\left.\begin{array}{l} x_1 \in A \\ x_1x_2 \notin E(H) \end{array}\right| \Rightarrow x_2 \notin A \qquad \left.\begin{array}{l} x_3 \in A \\ x_3x_4 \notin E(H) \end{array}\right| \Rightarrow x_4 \notin A \quad .....$$

$$\left.\begin{array}{l} x_{2n-1} \in A \\ x_{2n-1}x_{2n} \notin E(H) \\ x_{2n+1}x_1 \notin E(H) \end{array}\right| \Rightarrow x_{2n} \notin A \text{ şi } x_{2n+1} \notin A$$

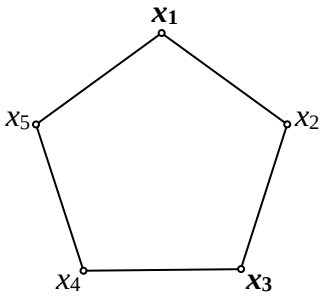
Deci $A = \{ x_1, x_3, ..., x_{2n-1} \}$, $|A| = n \Rightarrow \alpha(H) = n$

Dacă $E(H) \supset E(C)$ (în H există şi alte muchii decât cele care formează circuitul), atunci numărul de stabilitate al grafului H poate fi chiar mai mic decât n (unele vârfuri vor fi eliminate din A).

Deci $\alpha(H) \le n$.
Dar H este subgraf indus al lui $G \Rightarrow 2\alpha(H) \ge |H| = 2n+1$ $\Big| \Rightarrow$ contradicţie $\Rightarrow$
$\Rightarrow$ presupunerea făcută este falsă $\Rightarrow$ G este graf bipartit.

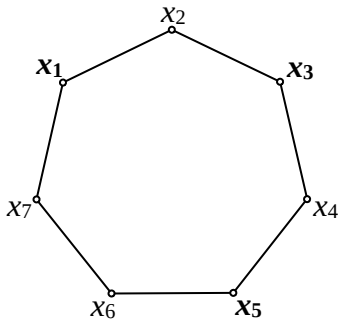
Exemplu

Fig. 1.1.



$n = 2$, $A = \{x_1, x_3\}$

Fig. 2.1.



$n = 3$, $A = \{x_1, x_3, x_5\}$

Problema 2.

a) Demonstraţi că într-un graf bipartit G cu n vârfuri şi m muchii avem inegalitatea $4m \le n^2$. (**2 puncte**)

b) Descrieţi un algoritm care să testeze dacă un graf cu n vârfuri şi m muchii este complementarul unui graf bipartit în timpul $O(n+m)$ (**3 puncte**).

Soluţie.

a) Fie $G = (S, T; E)$ un graf bipartit cu n vârfuri şi m muchii. Avem $|S| = k$ şi deci $|T| = n-k$. Fiecare vârf din S poate fi unit printr-o muchie cu fiecare vârf din T prin urmare în G sunt maxim $k(n-k)$ muchii, deci $m \le -k^2 + kn$. Considerăm funcţia $f : (0, n) \to R$ definită prin $f(x) = -x^2 + nk$. Ea este continuă şi derivabilă pe $(0, n)$ şi $f'(x) = -2k + n$. Dacă punem condiţia ca $f'(x) = 0$ obţinem punctul $x = \frac{n}{2}$ pentru care $f(\frac{n}{2}) = \frac{n^2}{4}$, care după cum se observă din Fig. 2.1 este singurul punct de maxim. Avem $f(x) \le \frac{n^2}{4}$ $\forall x \in (0, n)$ şi în particular pentru $x = k \in \{1,2,...,n-1\}$ avem că $m \le -k^2 + kn \le \frac{n}{4}$. Deci pentru orice graf G bipartit avem satisfăcută inegalitatea cerută: $4m \le n^2$.

	0		n/2		n
$f(x)$	0	↗	$n^2/4$	↘	0
$f'(x)$	+ + + + + + + + +	0	- - - - - - - - - -		

Fig. 2.1.

Altă soluţie ar fi sa arătăm că: $k(n-k) \le \frac{n^2}{4} \Leftrightarrow 4kn - 4k^2 - n^2 \le 0 \Leftrightarrow 4k^2 + n^2 - 4kn \ge 0 \Leftrightarrow (n-2k)^2 \ge 0$ (adevărat).

b) Fie $H = (V, E)$ un graf cu n vârfuri şi m muchii, reprezentat prin liste de adiacenţă; fie $G = \overline{H} = (V, E')$. Considerăm $V = \{1,2,...,n\}$.

Notăm $x = |E'| \Rightarrow x + m = \frac{n(n-1)}{2}$. Pentru ca G să fie graf bipartit, trebuie să aibă loc: $4x \le n^2 \Leftrightarrow \frac{n(n-1)}{2} - m \le \frac{n^2}{4} \Leftrightarrow \frac{2n^2-2n}{4} - m \le \frac{n^2}{4} \Leftrightarrow \frac{n^2-2n}{4} \le m \Leftrightarrow \frac{n(n-2)}{4} \le m$. Dacă această relaţie nu are loc, atunci cu siguranţă G nu e bipartit.

Pas1. Determinăm m folosind listele de adiacenţă $\Rightarrow$ complexitatea $O(n+m)$.

```
function determin_m()
begin
m = 0
for  v = 1 to n do
   for ( u ∈ Adj(v) )do  m = m + 1
return m
end.
```

Pas2. Dacă $m < \frac{n(n-2)}{4}$, atunci algoritmul se termină deoarece G nu poate fi graf bipartit. Altfel, se execută pasul 3.

Pas3. Construim graful G şi verificăm dacă e bipartit.

Ştim cu siguranţă că $m \ge \frac{n(n-2)}{4} \Rightarrow 4m \ge n^2 - 2n \Rightarrow n^2 \le 4m + 2n \Rightarrow n^2 = O(m+n)$ (n^2 şi $4m+2n$ sunt considerate funcţii având argumentele n şi m). Deci orice algoritm de complexitate $O(n^2)$ va fi şi de complexitate $O(m+n)$.

Algoritmul de construcţie a grafului G (reprezentat prin matricea de adiacenţă a):

1. Iniţial: $a[u,v] = 1$, pentru orice $u \ne v$, iar $a[v,v] = 0$.
2. Setăm $a[u,v] = 0$, pentru orice u,v cu $uv \in E$, construind în acelasi timp arborele BFS corespunzător grafului G, arbore reprezentat printr-un vector $nivel[1..n]$. Algoritmul de mai jos realizează de fapt o parcurgere BFS pentru graful G.

```
procedure construiesteG()
begin
for  v = 1 to n do nivel[v] = -1
for  v = 1 to n do
   if  nivel[v] = -1 then BFS(v)
end.

procedure BFS(i)
begin
C = { i }
nivel[i] = 0
while ( C ≠ ∅ )  do
   v = top( C )
   for ( u ∈ Adj(v) )  do
      a[u,v] = a[v,u] = 0    (muchia uv ∈ E(H), deci uv ∉ E(G))
   for  u = 1 to n do
```

```
            if (a[u,v]=1 and nivel[u]=-1) then
                push( C, u )
                nivel[u] = nivel[v] + 1
        end.
```

Algoritmul de construcţie a grafului G are complexitatea $O(n^2)$.

Graful G fiind construit, mai trebuie să verificăm dacă e bipartit.
Fie S = mulţimea vârfurilor de pe nivelele pare din G
T = mulţimea vârfurilor de pe nivelele impare din G.
Avem: $S \neq \emptyset$ (S poate fi vidă doar dacă $n = 0$), $T \neq \emptyset$ (T poate fi vidă doar dacă graful H este complet şi în acest caz orice vârf din S poate fi mutat în T), $S \cap T = \emptyset$, $S \cup T = V$. Pentru a avea $G = (S, T; E')$ bipartit, mai trebuie ca S şi T să fie mulţimi stabile de vârfuri din $G \Leftrightarrow$ oricare 2 vârfuri de pe acelaşi nivel sunt neadiacente.

```
procedure bipartit(G)
begin
for  v = 1 to n-1 do
    for u = v+1 to n do
        if (a[u,v]=1 and nivel[u]=nivel[v]) then
            return false
return true
end.
```

Algoritmul de verificare dacă G este bipartit are complexitatea $O(n^2)$.
Deci, pasul 3 are complexitatea timp $O(n^2) + O(n^2) = O(n^2) = O(m+n)$

Algoritmul care rezolvă problema:

```
procedure complementarul_este_bipartit(H)
begin
m = determin_m()
if m < n(n-2)/4 then
    return false
else
    construiesteG()
    return bipartit(G)
end.
```

Complexitatea totală a algoritmului este: $O(m+n) + O(1) + O(m+n) = O(m+n)$.

Exemplu

n=7, m=14
$n(n\text{-}2) = 7 * 5 = 35$
$4m = 4 * 14 = 56 > 35$ $\Rightarrow m \geq \frac{n(n-2)}{4}$ $\Rightarrow G$ ar putea fi bipartit

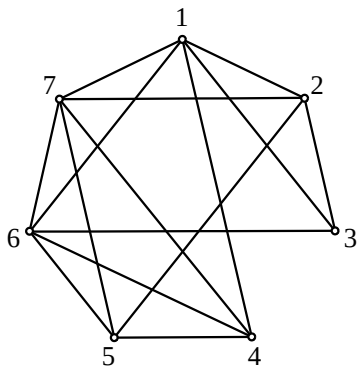


Fig. 2.2.Graful *H*

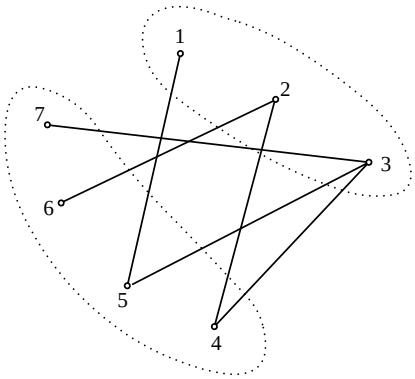


Fig. 2.3. Graful *G*

Construim graful *G:*

a =

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	0	0	0	0	1	0	0
2	0	0	0	1	0	1	0
3	0	0	0	1	1	0	1
4	0	1	1	0	0	0	0
5	1	0	1	0	0	0	0
6	0	1	0	0	0	0	0
7	0	0	1	0	0	0	0

nivel =

1	**2**	**3**	**4**	**5**	**6**	**7**
0	4	2	3	1	5	3

$\Rightarrow S$={1,2,3},T={4,5,6,7}; G=($S,T;E'$) bipartit

Problema 3.

Arătaţi că orice graf G cu m muchii are un graf parţial H bipartit şi cu cel puţin $\frac{m}{2}$ muchii. **(3 puncte)**

Soluţie.

Demonstraţia acestei probleme se va face constructiv (din punct de vedere algoritmic). Astfel prezentăm întâi un simplu algoritm greedy pentru această problemă, urmând ca apoi să demonstrăm finititudinea şi corectitudinea sa.

```
function graf_parţial_bipartit(G)
begin
   S := {1,...,n} // = V(G)
   schimb := 0
   u := 1
   repeat
      if u∈S then
         if |N_G(u)∩S|>|N_G(u)∩(V\S)| then
            S := S \ {u}
            schimb := 0
      else
         if |N_G(u)∩(V\S)|>|N_G(u)∩S)| then
            S := S ∪{u}
```

```
            schimb := 0
        u := (u mod n)+1
        schimb := schimb+1
    until schimb=n
    T := V \ S
    return S,T
end.
```

În continuare vom presupune graful *G* conex şi cu cel puţin două vârfuri. Dacă acestă condiţie nu este satisfăcută se împarte graful iniţial în componete conexe şi se elimină vârfurile izolate, urmând ca algoritmul să fie rulat pentru fiecare componetă conexă în parte. Dacă pentru componeta conexă G_i cu m_i muchii se obţin prin algoritm S_i şi T_i clase ale unei 2-partiţii atunci $S=\bigcup_{i\le k} S_i \cup I$ şi $T=\bigcup_{i\le k} T_i$ sunt două clase de bipartiţie (în mod sigur nu unice) ale grafului H căutat (am notat cu *k* numărul de componente conexe ale lui *G* şi cu *I* mulţimea vârfurilor izolate).

Algoritmul porneşte considerând toate vârfurile în acceaşi componentă ($S=V$, $T=\varnothing$). Se parcurg vârfurile în mod repetat (circular), şi la fiecare pas se testează dacă vârful curent are mai mulţi adiacenţi în clasa căreia îi aparţine decât în cealaltă clasă. Când $u\in S$ se testează dacă $|N_G(u)\cap S|>|N_G(u)\cap T|$, respectiv dacă $|N_G(u)\cap T|>|N_G(u)\cap S|$ atunci când $u\in T$. În caz afirmativ vârful este trecut în cealaltă clasă, astfel pe tot parcursul algoritmului $S\cup T=V$ şi $S\cap T=\varnothing$. Bucla *repeat* se execută cel puţin o dată, când se realizează în mod sigur o schimbare ($|N_G(u)\cap V|=d_G(u)>0=|N_G(u)\cap\varnothing|$ deoarece am presupus graful conex), astfel încât după prima execuţie $S,T\neq\varnothing$. Acest invariant este păstrat pe toată execuţia ulterioară a funcţiei. Într-adevăr presupunând că vârful *u* este singurul vârf al unei clase, atunci $|N_G(u)\cap\{u\}|=0\le|N_G(u)\cap(V-\{u\})|$ deci *u* nu va fi mutat. În concluzie ceea ce obţinem la sfârşitul algoritmului este o 2-partiţie a grafului *G*.

Fie graful parţial $H=(S,T;E')$ unde $E'=\{uv\in E|(u\in S\wedge v\in T)\vee(u\in T\wedge v\in S)\}$ obţinut din *G* prin păstrarea doar a muchiilor de la *S* la *T*. Pentru a demonstra că *H* are cel puţin $\frac{m}{2}$ muchii ($|E'|\ge\frac{m}{2}$) vom arăta prin echivalenţă că $|E'|\ge|E\setminus E'|$, adică în graful *G* rămân cel puţin la fel de multe muchii câte se elimină. Deoarece *H* este bipartit avem că $|E'|=\sum_{u\in S}|N_H(u)|=\sum_{u\in S}|N_G(u)\cap T|=\sum_{u\in T}|N_G(u)\cap S|$. În plus $|E\setminus E'|=\frac{1}{2}\left(\sum_{u\in S}|N_G(u)\cap S|+\sum_{u\in T}|N_G(u)\cap T|\right)$. Să observăm acum că algoritmul se termină doar după ce a parcurs toate cele *n* vârfuri ale lui *G* şi nu a realizat nici o schimbare. Deci la final avem satisfăcute relaţiile

$$\forall u\in S,|N_G(u)\cap S|\le|N_G(u)\cap T|\ \text{ şi }\ \forall u\in T,|N_G(u)\cap T|\le|N_G(u)\cap S|.$$

Cele două sume din expresia lui $|E\setminus E'|$ pot fi majorate astfel încât:

$$|E\setminus E'|\le\left(\sum_{u\in S}|N_G(u)\cap T|+\sum_{u\in T}|N_G(u)\cap S|\right)/2=\sum_{u\in S}|N_G(u)\cap T|=|E'|.$$

Deci în cazul în care se termină, algoritmul va furniza un răspuns corect. Este deci critic să demonstrăm că el se termină întotdeauna.

Notăm cu $m(S,T)=\left|\{uv\in E|u\in S\wedge v\in T\}\right|$ numărul muchiilor de la un vârf din din *S* la un vârf din *T* (la final $m(S,T)=|E'|$). Este evident că $m(S,T)$ satisface relaţia:

$$m(S,T)=\sum_{u\in S}|N_G(u)\cap T|=\sum_{u\in T}|N_G(u)\cap S|.$$

Iniţial $m(S,T)=0$ deoarece la acel punct $T=\varnothing$. De asemenea se observă că pentru a continua, algoritmul trebuie să realizeze o schimbare de elemente la cel mult *n*–1 paşi. Fie *S* şi *T* înaintea schimbării şi vârful curent $v\in S$ .Fie $S'=S\setminus\{v\}$ şi $T'=T\cup\{v\}$ cele două mulţimi după realizarea schimbării. Avem

$$m(S',T')=\sum_{u\in S'}|N_G(u)\cap T'|=\sum_{u\in S}|N_G(u)\cap T|-|N_G(v)\cap T|+|N_G(v)\cap S|.$$

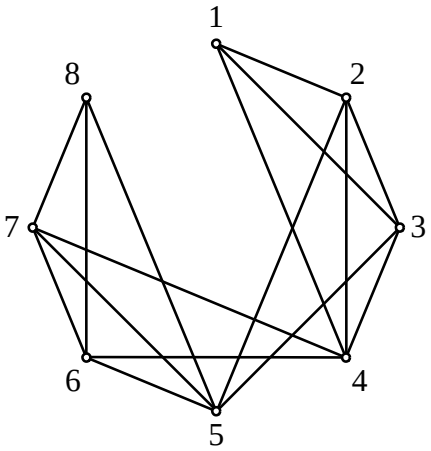
Pentru ca schimbarea să aibă loc trebuie să însă să avem $|N_G(v)\cap S|>|N_G(v)\cap T|$ care ne conduce la concluzia că $m(S,T)>m(S',T')$. Astfel $m(S,T)$ creşte la fiecare $n-1$ paşi cel puţin cu o unitate şi cum $m(S,T)\le m$ pentru orice *S* şi *T,* atunci algoritmul efectuează cel mult $(n-1)(m+1)$ paşi, deci este finit.

În concluzie am arătat un mod în care pentru orice graf *G* cu *m* muchii putem obţine un graf parţial bipartit şi cu cel puţin $\frac{m}{2}$ muchii.

Exemplu.

Să urmărim execuţia funcţiei *graf_parţial_bipartit* având ca intrare graful *G* cu $m=8$ muchii reprezentat în Fig. 3.1.

Fig 3.1



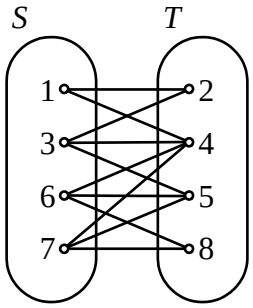


Fig 3.2

Iniţial $S=\{1,2,...,n\}$, $T=\varnothing$, schimb = 0 şi $u=1$. Deoarece $u\in S$ se va testa dacă el are mai mulţi adiacenţi în *S* decât în *T*, şi deoarece acest lucru este adevărat $3>0$ *u* este mutat în *T*. La fel şi $u=2$ deoarece are 3 adiacenţi în *S* şi doar 1 în *T*. Vârful $u=3$ are 2 adiacenţi în *S* şi tot ataţia în *T* deci va rămâne în *S*. Vârful $u=4$ are 3 adiacenţi în *S* şi doar 2 în *T* deci va fi mutat în *T* şi la fel şi $u=5$ deoarece are 4 adiacenţi în *S* şi doar 1 în *T*. Vârfurile $u=6$ şi $u=7$ nu vor fi mutate deoarece au numarul de adiacenţi în *T* egal cu numarul de adiacenţi în *S* (2). În fine $u=8$ va fi mutat în *T* deoarece are acolo doar un adiacent în timp ce în *S* are 2. Astfel după o primă parcurgere a tuturor vârfurilor se obţine $S=\{3,6,7\}$ şi $T=\{1,2,4,5,8\}$. Se testează din nou vârful $u=1\in T$ şi se obţin 2 adiacenţi în *T* şi doar unu în *S* deci vârful va fi mutat din nou în *S*. Vârful $u=2\in T$ are 2 adiacenţi în *T* şi toţi atâţia în *S*, $u=3\in S$ are un adiacent în *S* şi 3 în *T*, $u=4\in T$ are un adiacent în *T* şi 4 în *S* iar nodurile $u=6\in S$ şi $u=7\in S$ au amândouă câte un adiacent în *S* şi câte 3 adiacenţi

în *T*, în fine $u = 8 \in T$ are un adiacent în *T* şi 2 în *S*, prin urmare nici unul din aceste vârfuri nu vor fi mutate. Cum s-au parcurs toate nodurile fără a face vreo schimbare algoritmul se încheie cu $S = \{1,3,6,7\}$ şi $T = \{2,4,5,8\}$, şi $\|H\| = 11 \geq \frac{16}{2} = 8$ (Fig. 3.2).

Problema 4.

a) Demonstraţi că în orice graf conex $G = (V, E)$ există o mulţime stabilă *S* astfel încât graful bipartit $H = (S, V - S; E')$ este conex, unde $E' = E - \wp_2(V - S)$.

b) Deduceţi că $\alpha(G) \geq \frac{|G|-1}{\Delta(G)}$ pentru orice graf conex *G*. **(3 puncte)**

Soluţie.

a) Demostraţia se poate face constructiv: folosind o parcurgere BFS a vârfurilor grafului *G*, construim graful bipartit *H* şi mulţimea stabilă *S*. Conexitatea grafului *H* (subgraf parţial al lui *G*), va rezulta din algoritmul de construcţie.

Fie *G* = (*V*,*E*) un graf conex oarecare, $V = \{1,2,...,n\}$, *G* reprezentat prin listele de adiacenţă *AdjG*(*v*), $v \in V$.

Iniţial, *H* = *G* (listele de adiacenţă corespunzătoare lui *H* sunt: *AdjH*(*v*)=*AdjG*(*v*)). *H* va deveni apoi o "copie aproximativă" a lui *G* obţinută printr-o parcurgere BFS puţin modificată, în sensul că pentru fiecare nod *v* de pe un nivel par (al arborelui BFS corespunzător lui *H*) se va impune condiţia ca *v* să nu fie adiacent cu nici un alt nod de pe acelaşi nivel.

Dacă este îndeplinită această condiţie, se continuă parcurgerea grafului.

Dacă există *u* astfel încât *nivel*[*u*] = *nivel*[*v*] şi $uv \in E(H)$, atunci *nivel*[*v*] devine *nivel*[*v*]-1 (nivelul părintelui său). În final, muchia *vw* va fi eliminată din graful *H* (*w* este părintele vârfului *v* în "arborele BFS" construit).

Mulţimea stabilă *S* va fi formată din toate nodurile aflate pe nivele pare în "arborele BFS" construit.

```
procedure construieste(H, S)
begin
S = ∅
for  v = 1 to n do
   AdjH(v) = AdjG(v)
   nivel[v] = -1
C = { 1 }
while ( C ≠ ∅ )  do
   w = top( C )
   for ( v ∈ AdjH(w) )  do
      if nivel[v] = -1 then
         push( C, v )
         nivel[v] = nivel[w] + 1
         if (nivel[v] = par) then
            if ( ∃ u∈V , nivel[u]=nivel[v]  and uv∈E(H)) then
               nivel[v] = nivel[v]-1
            else
               /* nivel[v] rămâne un număr par */
               S = S ∪ {v}
```

```
/* mai trebuie să eliminăm din H muchiile dintre oricare 2 vârfuri de pe
acelaşi nivel (impar) */
for  v = 1 to n-1 do
   for u = v+1 to n do
      if (v ∈ AdjH(u) and nivel[u]=nivel[v]) then
         elimina(AdjH(u), v)
         elimina(AdjH(v), u)
end.
```

Algoritmul de mai sus rezolvă corect problema:

- *S* este mulţime stabilă de vârfuri deoarece am impus condiţia ca oricare 2 vârfuri de pe acelaşi nivel par să fie neadiacente. În plus, oricare 2 vârfuri de pe nivele pare diferite nu pot fi adiacente (în orice arbore BFS, un vârf poate fi adiacent numai cu vârfurile de pe un nivel imediat superior, imediat inferior sau de pe nivelul curent).
- Graful construit: $H=(S, V-S; E')$ este bipartit.
  Dacă $n=1$, atunci $E'=E=\emptyset \Rightarrow H$ bipartit.
  Fie $n \geq 2$. Avem $S \neq \emptyset$ $(1 \in S)$, $V-S \neq \emptyset$ (există cel puţin un vârf adiacent cu 1, care va fi pe nivelul impar 1), $S \cap (V-S) = \emptyset$, $S \cup (V-S) = V$, S şi $(V-S)$ sunt mulţimi stabile $\Rightarrow H$ bipartit.
- Graful *H* este conex.
  Ştim că *G* este conex. *H* a fost construit pornind de la o copie a lui *G*, din care am eliminat apoi unele muchii. Trebuie să arătăm că eliminarea acestor muchii nu influenţează conexitatea grafului *H*. Algoritmul elimină doar muchiile dintre 2 vârfuri aflate pe acelaşi nivel impar (fie *k* acest nivel). Dar orice vârf *v* aflat pe un nivel impar rămâne adiacent cu:
  1) un vârf *w* de pe nivelul *k*-1 , dacă nivelul lui *v* a fost impar de la început
  2) sau un vârf *u* de pe nivelul *k*+1 , dacă nivelul lui *v* a fost iniţial par şi s-a făcut modificarea *nivel*[*v*] = *nivel*[*v*]-1. Deoarece *u* şi *v* sunt acum pe nivele diferite, muchia ce le uneşte va rămâne în graful *H*.

  Deci graful *H* rămâne conex după eliminarea muchiilor dintre oricare 2 vârfuri de pe acelaşi nivel impar.

  Altă metodă prin care se poate demonstra că *H* este conex se bazează pe observaţia că orice muchie ce va fi eliminată face parte dintr-un circuit: fie *v* pe nivelul *k*, *w* părintele său (pe nivelul *k*+1 sau *k*-1), *u* pe nivelul *k* astfel încât *uv* este muchie. Există un drum în *H* de la *u* la *w* ce nu conţine muchii între 2 vârfuri de pe acelaşi nivel. Deci adăugând muchia *uv* la drumul considerat, obţinem un circuit. Eliminând această muchie din *H*, graful rămâne conex.

Exemplu

nivel[1] = 0, nivel[2] = 1, nivel[3] = 1, nivel[4] = 2
nivel[5] = 2, dar {5,4} este muchie $\Rightarrow$ nivel[5] = 1
nivel[6] = 2
nivel[7] = 2, dar {7,4} este muchie $\Rightarrow$ nivel[7] = 1
nivel[8] = 2, nivel[9] = 3, nivel[10] = 3, nivel[11] = 2
nivel[12] = 2, dar {12,11} este muchie $\Rightarrow$ nivel[12] = 1
nivel[13] = 2, nivel[14] = 3, nivel[15] = 3
S = {1, 4, 6, 8, 11, 13}
$V-S$ = {2, 3, 5, 7, 9, 10, 12, 14, 15}

Muchiile eliminate vor fi: {2,5}, {2,7}, {5,12}.

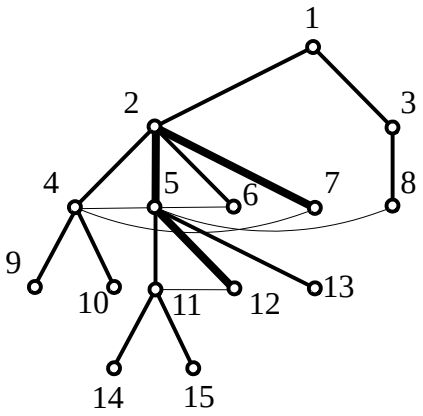


Fig. 4.1.Graful *G*

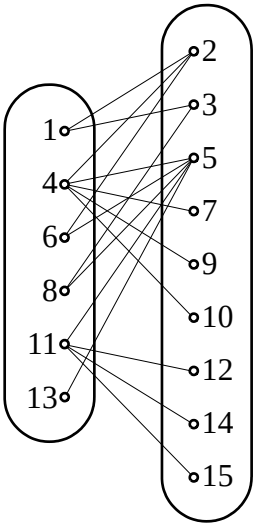


Fig. 4.2 Graful *H*

b) Fie *G* un graf conex oarecare (cu $|G| = n$).

$\Rightarrow$ există o mulţime stabilă *S* astfel încât graful bipartit $H = (S, V\text{-}S; E')$ este conex .

Fie $m = |E'|$, $k = |S|$.

m = suma gradelor vârfurilor din *S* = suma gradelor vârfurilor din *T*

$$m = \sum_{v \in S} d_H(v) = \sum_{v \in S} d_G(v)$$

$\forall\, v \in S$, avem $d_G(v) \le \Delta(G) \Rightarrow m \le \Delta(G) * k \quad (k = |S|)$

Avem $k \ge \dfrac{m}{\Delta(G)} \Rightarrow k \ge \dfrac{n-1}{\Delta(G)}$

Dar *H* este conex$\Rightarrow$ are cel puţin *n*-1 muchii ($m \ge n$-1) $\Rightarrow \alpha(H) \ge \dfrac{n-1}{\Delta(G)}$

$k = |S|$, *S* este mulţime stabilă $\Rightarrow \alpha(H) \ge k$ (q.e.d.)

Gordân Raluca Mihaela
Hriţcu Cătălin
An IIA, grupa 4
Grupa de seminar A34
Echipa nr. 2

Algoritmica grafurilor – tema 6

Problema 1.

Pentru $d \in N^*$ se consideră graful $G_d = K_2 \times K_2 \times ... \times K_2$.

a) Să se determine ordinul, dimensiunea şi diametrul lui G_d. (**2 puncte**)

b) Să se arate că G_d este bipartit şi să se determine $\alpha(G_d)$. (**2 puncte**)

Soluţie.

$G_1 = K_2$ (Fig. 1.1), avem $V(G_1) = \{0,1\}$ şi $E(G_1) = \{\{0,1\}\}$.

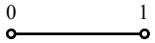


Fig. 1.1

$G_2 = K_2 \times K_2$. (Fig. 1.2) avem $V(G_2) = V(G_1) \times V(G_1) = \{(i,j) | i,j \in \{0,1\}\}$.

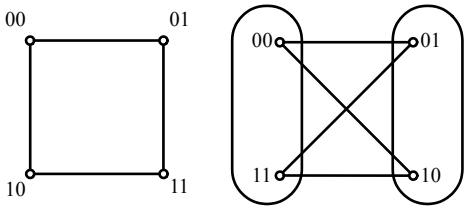


Fig. 2.1

Notaţie. Pentru a simplifica scrierea asociem fiecărui *n*-uplu $(i_1, i_2, ..., i_n)$ cuvântul $i_1 i_2 ... i_n \in \{0,1\}^n$. Spre exemplu în locul tripletului (i, j, k) vom nota simplu ijk.

Prin urmare $V(G_2) = V(G_1) \times V(G_1) = \{ij | i, j \in \{0,1\}\}$. Conform definiţiei produsului cartezian a două grafuri avem $\{i_1 i_2, j_1 j_2\} \in E(G_2)$ dacă şi numai dacă $i_1 = j_1$ şi $i_2 j_2 \notin E(G_1)$, sau $i_2 = j_2$ şi $i_1 j_1 \notin E(G_1)$. Dar $G_1 = K_2$ şi deci $uv \notin E(G_1)$ dacă şi numai dacă $u \neq v$. Deci $E(G_2) = \{\{i_1 i_2, j_1 j_2\} | (i_1 = j_1 \wedge i_2 \neq j_2) \vee (i_2 = j_2 \wedge i_1 \neq j_1)\}$.

Demonstrăm prin inducţie după *d* următoarea formă pentru G_d:

$$(*) \begin{cases} V(G_d) = \{i_1 i_2 ... i_d | i_1, i_2, ..., i_d \in \{0,1\}\} \\ E(G_d) = \{\{i_1 i_2 ... i_d, j_1 j_2 ... j_d\} | \exists r \in \{1,2,...,d\}\, i_r \neq j_r \wedge i_k = j_k, \forall k \in \{1,2,...,d\}, k \neq r\} \end{cases}$$

Baza inducţiei fiind demonstrată în observaţiile anterioare vom presupune relaţia (*) adevărată pentru G_d şi vom demonstra că ea este valabilă şi pentru G_{d+1}.

$G_{d+1} = G_d \times K_2$, deci $V(G_{d+1}) = V(G_d) \times V(K_2) = \{i_1 i_2 ... i_d i_{d+1} | i_1, i_2, ..., i_d i_{d+1} \in \{0,1\}\}$, conform notaţiilor folosite. Pentru $E(G_{d+1})$ avem următoarea relaţie recursivă:

$$E(G_{d+1}) = \{\{i_1 i_2 .. i_d i_{d+1}, j_1 j_2 .. j_d j_{d+1}\} | (i_1 i_2 .. i_d = j_1 j_2 .. j_d \wedge i_{d+1} \neq j_{d+1}) \wedge (i_{d+1} = j_{d+1} \wedge \{i_1 i_2 .. i_d, j_1 j_2 .. j_d\} \in E(G_d))\}$$

care, prin aplicarea ipotezei inductive pentru $E(G_d)$, devine:

$$E(G_{d+1}) = \{\{i_1 i_2 .. i_d i_{d+1}, j_1 j_2 .. j_d j_{d+1}\} | \exists r \in \{1,2,...,d+1\}\ i_r \neq j_r \wedge i_k = j_k, \forall k \in \{1,2,...,d+1\}, k \neq r\}.$$

Prin urmare orice graf G_d va avea forma impusă de realaţiile (*).

Notaţie. Pentru a uşura scrierea pentru $i, j \in \{0,1\}$ şi $i \neq j$ vom nota $i = \bar{j}$ sau $j = \bar{i}$. Avem deci $\bar{0} = 1$ şi $\bar{1} = 0$, folosind în plus convenţia că $i = \bar{\bar{i}}$. Pentru $i = i_1 i_2 ... i_d \in V(G_d)$ şi $1 \leq k \leq d$ vom nota $i_{\langle k \rangle} = i_1 i_2 ... \bar{i}_k ... i_d$ (de exemplu $1011_{\langle 2 \rangle} = 1111$). De asemenea pentru $i = i_1 i_2 ... i_d \in V(G_d)$ şi $1 \leq k_1, k_2, ..., k_s \leq d$ vom nota $i_{\langle k_1 \rangle \langle k_2 \rangle ... \langle k_s \rangle} = i_1 i_2 ... \bar{i}_{k_1} ... \bar{i}_{k_2} ... \bar{i}_{k_s} ... i_d$ (de exemplu $1011_{\langle 1 \rangle \langle 3 \rangle \langle 2 \rangle} = 0101$). Se deduce imediat că $i_{\langle k \rangle \langle l \rangle} = i_{\langle l \rangle \langle k \rangle}$ şi $i = i_{\langle k \rangle \langle k \rangle}$.

Observaţia 1.1. Merită remarcat că nu am impus nici o restricţie asupra indicilor $k_1, k_2, ..., k_s$, care, în general pot să nu fie distincţi. Totuşi pentru orice $1 \leq k_1, k_2, ..., k_s \leq d$ există $1 \leq \alpha < \alpha_2 < ... < \alpha_t \leq s$ cu $s \leq t$ şi $k_{\alpha_1}, k_{\alpha_2}, ..., k_{\alpha_p}$ distincte două câte două, astfel încât $i_{\langle k_1 \rangle \langle k_2 \rangle ... \langle k_s \rangle} = i_{\langle k_{\alpha_1} \rangle \langle k_{\alpha 2} \rangle ... \langle k_{\alpha_t} \rangle}$.

Fie funcţia[1] $h_d : \{0,1\}^d \times \{0,1\}^d \to \{0,1,...,d\}$ dată prin

$$h_d(i, j) = \left| \left\{ k \,|\, k \in \{1,2,...,d\}, i_k \neq j_k \right\} \right| \quad \forall i = i_1 i_2 ... i_d, j = j_1 j_2 ... j_d \in \{0,1\}^d .$$

Proprietatea 1.1. Pentru orice două vârfuri $i, j \in V(G_d)$ avem $d_{G_d}(i, j) = h_d(i, j)$.

Vom demonstra această proprietate prin inducţie finitară după valorile funcţiei h_d. Pentru $h_d(i, j) = 0$ avem $\forall k \in \{1,...,d\}$, $i_k = j_k$ care implică $i = j$. $D = \{i\}$ este un drum de lungime 0 de la *i* la *i* deci $d_{G_d}(i, j) = 0$.

Pentru $h_d(i, j) = 1$ avem $\exists! r \in \{1,...,d\}$ $i_r \neq j_r$ şi $\forall k \in \{1,...,d\}$, $k \neq d$, $i_k = j_k$. Din (*) avem că $ij \in E(G_d)$, deci $D = \{i, ij, j\}$ este un *ij*-drum de lungime 1 de la *i* la *j* şi cum $i \neq j$ obţinem că $d_{G_d}(i, j) = 1$.

Presupunem proprietatea adevărată pentru orice $i', j' \in V(G_d)$ astfel încât $h_d(i', j') \leq n - 1$ şi o demonstrăm pentru orice $i, j \in V(G_d)$ pentru care $h_d(i, j) = n$.

Fie deci $\forall i, j \in V(G_d)$ astfel încât $h_d(i, j) = n$. Atunci $\exists 1 \leq k_1 < k_2 < ... < k_n \leq d$ pentru care $i_{k_r} \neq j_{k_r}$ $\forall r \leq n$, cele *n* poziţii prin care *i* diferă de j conform definiţiei funcţiei h_d. Atunci drumul

$$D = \left\{ i, ii_{\langle k_1 \rangle}, i_{\langle k_1 \rangle}, i_{\langle k_1 \rangle} i_{\langle k_1 \rangle \langle k_2 \rangle}, ..., i_{\langle k_1 \rangle \langle k_2 \rangle ... \langle k_{n-1} \rangle} i_{\langle k_1 \rangle \langle k_2 \rangle ... \langle k_n \rangle}, i_{\langle k_1 \rangle \langle k_2 \rangle ... \langle k_n \rangle} = j \right\}$$

este un *ij*-drum de lungime *n* în G_d deoarece $\forall i \in V(G_d)$, $\forall k \leq d$, $ii_{\langle k \rangle} \in E(G_d)$.

Rămâne de demonstrat că nu există un drum de lungime mai mică.

Presupunem prin absurd că ar exista un asemenea *ij*-drum *D'* de lungime $m < n$. Atunci din (*) obţinem că *D'* are forma:

$$D' = \left\{ i, ii_{\langle k_1 \rangle}, i_{\langle k_1 \rangle}, i_{\langle k_1 \rangle} i_{\langle k_1 \rangle \langle k_2 \rangle}, ..., i_{\langle k_1 \rangle \langle k_2 \rangle ... \langle k_{m-1} \rangle} i_{\langle k_1 \rangle \langle k_2 \rangle ... \langle k_m \rangle}, i_{\langle k_1 \rangle \langle k_2 \rangle ... \langle k_m \rangle} = j \right\}$$

Avem că $i_{\langle k_1 \rangle \langle k_2 \rangle ... \langle k_m \rangle} = j$ şi folosind observaţia 1.1 obţinem că $i_{\langle k_{\alpha_1} \rangle \langle k_{\alpha_2} \rangle ... \langle k_{\alpha_p} \rangle} = j$ cu $p \leq m$ şi $k_{\alpha_1}, k_{\alpha_2}, ..., k_{\alpha_p}$ distincte două câte două. De aici obţinem că $i_{k_{\alpha_r}} \neq j_{k_{\alpha_r}}$,

[1] distanţa hamming

$\forall r \le p$ şi $i_k = j_k$ $\forall k \ne k_{\alpha_1}, k_{\alpha_2}, ..., k_{\alpha_p}$ şi deci $h_d(i,j) = p \le m < n = h_d(i,j)$, contradicţie. Cu aceasta proprietatea 1.1 este demonstrată.

Proprietatea 1.2. Se obţine ca o consecinţă imediată a proprietăţii 1.1 că $ij \in E(G_d)$ dacă şi numai dacă $h(i,j) = 1$.

Observaţia 1.2. Este uşor de arătat că h_d satisface relaţia de recurenţă următoare:

$$h_{d+1}(i_1 i_2 ... i_d i_{d+1}, j_1 j_2 ... j_d j_{d+1}) = h_d(i_1 i_2 ... i_d, j_1 j_2 ... j_d) + h_1(i_{d+1}, j_{d+1}),$$

$$\text{unde } \forall i, j \in \{0,1\} \; h_1(i,j) = \begin{cases} 1, i \ne j \\ 0, i = j \end{cases}.$$

a) *Ordinul grafului* G_d este egal numărul de *d*-uple cu valori în $\{0,1\}$

$$|G_d| = |\{0,1\}^d| = 2^d .$$

Din (*) avem imediat că pentru orice vârf $i_1 i_2 ... i_d \in G_d$ că $N_{G_d}(i) = \{i_{\langle 1 \rangle}, i_{\langle 2 \rangle}, ..., i_{\langle d \rangle}\}$. Deci graful este *d*-regulat şi $2 \cdot \|G_d\| = \sum_{i \in G_d} d_{G_d}(i) = |G_d| \cdot d$ de unde *dimensiunea grafului* G_d este

$$\|G_d\| = 2^{d-1} d .$$

Conform proprietăţii 1.1 pentru orice $i, j \in V(G_d)$ avem $d_{G_d}(i,j) = h_d(i,j)$. Cum h_d nu poate lua valori mai mari decât *d* şi, pentru orice $i = i_1 i_2 ... i_d \in V(G_d)$ există $j = \bar{i}_1 \bar{i}_2 .. \bar{i}_d \in V(G_d)$ pentru care $h_d(i,j) = d$, *diametrul grafului* G_d este

$$d(G_d) = \max_{i,j \in V(G_d)} d_{G_d}(i,j) = \max_{i,j \in V(G_d)} h_d(i,j) = d .$$

b) Demonstrăm prin inducţie matematică după *d* faptul că G_d este bipartit şi dacă $G_d = (S_d, T_d; E(G_d))$ avem $|S_d| = |T_d| = \frac{|G_d|}{2} = 2^{d-1}$. În plus la fiecare pas al inducţiei vom pune în evidenţă un cuplaj cu 2^{d-1} muchii pentru G_d.

Începem prin a testa proprietăţile de mai sus pentru grafurile G_1, G_2 şi G_3.

Pentru G_1 avem $S_1 = \{0\}$ şi $T_1 = \{1\}$ (Fig. 1.1), evident $|S_1| = |T_1| = 2^0 = 1$ şi $\{\{0,1\}\}$ este un cuplaj cu o muchie pentru G_1.

Pentru G_2 avem $S_2 = \{00,11\}$ şi $T_2 = \{01,10\}$ (Fig. 1.2). Avem $|S_2| = |T_2| = 2^1 = 2$ şi $\{\{00,01\},\{11,10\}\}$ un cuplaj cu 2 muchii în G_2.

Pentru G_3 avem $S_3 = \{000,110,011,101\}$ şi $T_2 = \{001,111,010,100\}$ (Fig. 1.3). Avem $|S_3| = |T_3| = 2^2 = 4$ şi $\{\{000,001\},\{110,111\},\{011,010\},\{101,100\}\}$ un cuplaj cu 4 muchii în G_2.

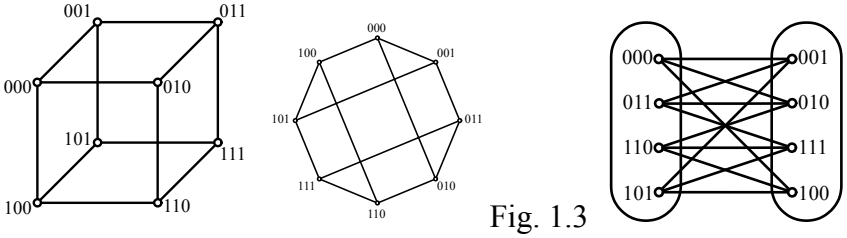


Fig. 1.3

Presupunem că G_d este bipatit, $G_d = (S_d, T_d; E(G_d))$ şi avem $|S_d| = |T_d| = 2^{d-1}$. Atunci pentru G_{d+1} definim următoarele două mulţimi

$$S_{d+1} = (S_d \times S_1) \cup (T_d \times T_1) \text{ şi } T_{d+1} = (S_d \times T_1) \cup (T_d \times S_1).$$

Vom demonstra că aceste mulţimi sunt clase de bipartiţie pentru G_{d+1}. Se observă imediat că $S_{d+1}, T_{d+1} \neq \varnothing$ deoarece $S_d, S_1, T_d, T_1 \neq \varnothing$. Deoarece $S_1 \cap T_1 = S_d \cap T_d = \varnothing$ avem $S_{d+1} \cap T_{d+1} = ((S_d \times S_1) \cup (T_d \times T_1)) \cap ((S_d \times T_1) \cup (T_d \times S_1)) =$
$= ((S_d \times S_1) \cap (S_d \times T_1)) \cup ((S_d \times S_1) \cap (T_d \times S_1)) \cup ((T_d \times T_1) \cap (S_d \times T_1)) \cup ((T_d \times T_1) \cap (T_d \times S_1)) =$
$= \varnothing \cup \varnothing \cup \varnothing \cup \varnothing = \varnothing$. În fine $V(G_1) = S_1 \cup T_1$, $V(G_d) = S_d \cup T_d$ şi $V(G_{d+1}) = V(G_d) \times V(G_1) = (S_d \times S_1) \cup (T_d \times T_1) \cup (S_d \times T_1) \cup (T_d \times S_1) = S_{d+1} \cup T_{d+1}$. Prin urmare rămâne de demonstrat stabilitatea claselor partiţiei găsite.

Dacă $S_d = \{s_1, s_2, ..., s_{2^{d-1}}\}$ şi $T_d = \{t_1, t_2, ..., t_{2^{d-1}}\}$ atunci avem

$$S_{d+1} = \{s_1 0, s_2 0, ..., s_{2^{d-1}} 0, t_1 1, t_2 1, ..., t_{2^{d-1}} 1\} \text{ şi}$$
$$T_{d+1} = \{s_1 1, s_2 1, ..., s_{2^{d-1}} 1, t_1 0, t_2 0, ..., t_{2^{d-1}} 0\} .$$

Deoarece S_d este stabilă avem că $\forall k, r$, $1 \le k, r \le 2^{d-1}$, $k \neq r$ $\{s_k, s_r\} \notin E(G_d)$. Conform proprietăţii 1.2. $h_d(s_k, s_r) > 1$ de unde $h_{d+1}(s_k 0, s_r 0) = h_d(s_k, s_r) + h_1(0,0) > 1$ (s-a folosit aici observaţia 1.2) ceea ce conduce, folosind din nou proprietatea 1.2., la $\{s_k 0, s_r 0\} \notin E(G_{d+1})$ (1). Analog se arată că $\{t_k 1, t_r 1\} \notin E(G_{d+1})$ (2). Pentru că $S_d \cap T_d = \varnothing$ avem $\forall k, r$, $1 \le k, r \le 2^{d-1}$, $s_k \neq t_r$ ceea ce implică $h_d(s_k, t_r) > 0$. Dar $h_{d+1}(s_k 0, t_r 1) = h_d(s_k, t_r) + h(0,1) > 0 + 1 = 1$ şi folosind proprietatea 1.2. se obţine $\{s_k 0, t_r 1\} \notin E(G_{d+1})$ (3). Relaţiile (1), (2) şi (3) ne asigură că mulţimea S_{d+1} este stabilă. În mod similar se demonstrează că şi T_{d+1} este stabilă, deci graful G_{d+1} este bipartit. Mai departe $|S_{d+1}| = |S_d \times S_1| + |T_d \times T_1| = 2^{d-1} \cdot 1 + 2^{d-1} \cdot 1 = 2^d$ şi $|T_{d+1}| = |S_d \times T_1| + |T_d \times S_1| = 2^{d-1} \cdot 1 + 2^{d-1} \cdot 1 = 2^d$, deci $|S_{d+1}| = |T_{d+1}| = 2^d$. În plus

$$C_{d+1} = \{\{s_1 0, s_1 1\}, \{s_2 0, s_2 1\}, ..., \{s_{2^{d-1}} 0, s_{2^{d-1}} 1\}, \{t_1 1, t_1 0\}, \{t_2 1, t_2 0\}, ..., \{t_{2^{d-1}} 1, t_{2^{d-1}} 0\}\}$$

este un cuplaj în G_{d+1} având 2^d muchii.

Prin urmare $\forall d \ge 1$ G_d este bipartit şi în plus conţine un cuplaj cu 2^{d-1} muchii, deci numărul de cuplaj $\nu(G_d) \ge 2^{d-1}$. Dar numărul de cuplaj într-un graf nu poate depăşi jumătate din numărul vârfurilor deci $\nu(G_d) = 2^{d-1}$. Aplicând teorema lui König grafului bipartit G_d obţinem că $\nu(G_d) + \alpha(G_d) = 2^d$ şi deci $\alpha(G_d) = 2^{d-1}$.

Exemplu. Pentru graful G_4 reprezentat în Fig. 1.4 avem:
$|G_4| = 2^4 = 16$, $\|G_4\| = 2^3 \cdot 3 = 24$, $d(G_d) = 4$ şi $\alpha(G_d) = 2^3 = 8$.
Graful este bipartit cu
$S_4 = \{0000, 0011, 0110, 0101, 1100, 1111, 1010, 1001\}$ şi
$T_4 = \{0001, 0010, 0111, 0100, 1101, 1110, 1011, 1000\}$.

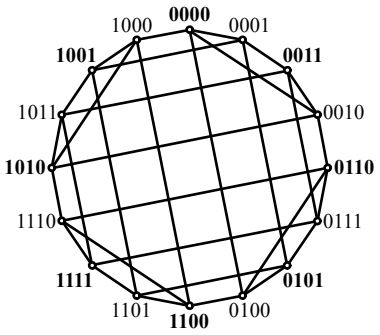


Fig. 1.4

Problema 2.

Un graf cu cel puţin trei vârfuri se numeşte *confidenţial conex* dacă pentru orice trei vârfuri distincte *a*,*b*,*c* ale grafului există un drum de la *a* la *b* astfel încât niciunul dintre vârfurile interne ale acestui drum (dacă există astfel de vârfuri) nu este *c* sau un vecin al lui *c*. Un exemplu banal de graf confidenţial conex este graful K_n cu $n \geq 3$.

Demonstraţi că un graf conex $G=(V,E)$, cu cel puţin ***patru*** vârfuri şi care nu-i complet, este confidenţial conex dacă şi numai dacă au loc următoarele două condiţii:

1) Pentru orice vârf *v* mulţimea $\overline{N}(v)=\{w \in V \mid w \neq v, vw \notin E\}$ este nevidă şi induce un graf conex.
2) Orice muchie a grafului este conţinută într-un C_4 indus în graf sau este muchia din mijlocul unui P_4 indus în graf. (**4 puncte**)

Soluţie.

Din punct de vedere formal definiţia unui graf confidenţial conex se poate scrie astfel:
Definiţie. Un graf $G=(V,E)$ cu $|G| \geq 3$ se numeşte *confidenţial conex* dacă
$\forall a,b,c \in V$ distincte (*) $\exists D$ *ab*-drum în *G* astfel încât $\forall u \in Int(D)$, $u \in \overline{N}(c)$.

Proprietatea 2.1. Condiţia 1) este evident echivalentă cu

(1') $\forall c \in V$, $\overline{N}(c) \neq \varnothing$ şi

(1") $\forall a,b \in [\overline{N}(c)]_G$ $\exists D$ *ab*-drum în $[\overline{N}(c)]_G$.

Proprietatea 2.2. Condiţia 2) este echivalentă cu

(2') $\forall ab \in E$, $\exists a' \in N_G(a)$ astfel încât $a' \in \overline{N}(b)$.

Direct. Presupunem proprietate 2) este satisfăcută. Fie $\forall ab \in E$, dacă $ab \in C_4$ indus în *G* atunci $\exists a',b' \in V$ astfel încât $aa',bb',a'b' \in E$ şi $ab',ba' \notin E$. Dacă $ab \in P_4$ indus în *G* atunci $\exists a',b' \in V$ astfel încât $aa',bb' \in E$ şi $ab',ba',a'b' \notin E$. Indiferent de caz $\exists a' \in N_G(a)$ astfel încât $ba' \notin E$ deci (2') este satisfăcută.

Invers. Presupunem că proprietatea (2') este satisfăcută. Fie $ab \in E$ oarecare. Putem aplica (2') pentru *a* şi obţinem $\exists a' \in N_G(a)$ astfel încât $a' \in \overline{N}(b)$, iar pentru *b* tot din (2') obţinem $\exists b' \in N_G(b)$ astfel încât $b' \in \overline{N}(a)$. Deci $aa',bb' \in E$ şi $ab',ba' \notin E$. Dacă $a'b' \in E$ atunci $[\{a,b,b',a'\}]_G = C_4$, altfel $[\{a,b,b',a'\}]_G = P_4$, *ab* fiind muchie interioară.

Direct. Fie $G=(V,E)$ un graf confidenţial conex cu cel puţin patru vârfuri şi care nu este complet. Demonstrăm mai întâi că pentru orice $c \in V$, $\overline{N}(c) \neq \varnothing$.

Presupunem prin absurd că $\exists c \in V$, $\overline{N}(c) = \varnothing$, deci $ua \in E$, $\forall u \in V$, $u \neq c$. Fie $a, b \in V$ două vârfuri oarecare distincte, diferite de *c*. Fie *D* un *ab*-drum oarecare în *G*, dacă $D = (a = u_0, u_0u_1, u_1, ..., u_m = b)$ are lungimea $m > 1$, atunci $\forall u_i \in Int(D)$ avem $cu_i \in E$ deci $u_i \in \overline{N}(c)$. Cum graful *G* este confidenţial conex rezultă că există *D* un *ab*-drum de lungime 1 de la *a* la *b*, deci $ab \in E$. Cum *a* şi *b* au fost alese distincte oarecare rezultă că graful *G* este complet, contradicţie. Deci $\forall c \in V$, $\overline{N}(c) \neq \varnothing$.

Tot prin reducere la absurd demonstrăm şi că $\forall c \in V$, $\left[\overline{N}(c)\right]_G$ este conex. Presupunem că $\exists c \in V$ pentru care $\left[\overline{N}(c)\right]_G$ este neconex, deci $\exists a, b \in \left[\overline{N}(c)\right]_G$ pentru care nu există niciun *ab*-drum în $\left[\overline{N}(c)\right]_G$. Cum însă $\forall D$ *ab*-drum în *G* nu este *ab*-drum şi în $\left[\overline{N}(c)\right]_G$ trebuie să avem $\exists u \in Int(D)$, $u \notin \overline{N}(c)$, ceea ce contrazice faptul că graful este confidenţial conex. Deci condiţia 1) este satisfăcută.

Fie $\forall ab \in E$. Fie $c \in \overline{N}(a) \neq \varnothing$. Cum graful G este confidenţial conex $\exists D$ *ac*-drum în *G* astfel încât $\forall u \in Int(D)$, $u \in \overline{N}(b)$. Dacă $D = \{a = u_0, u_0u_1, u_1, ..., u_m = c\}$ cu $m \geq 2$ (deoarece $ac \notin E$), atunci putem alege $a' = u_1 \in N_G(a)$ şi din $u_1 \in Int(D)$ obţinem că $a' \in \overline{N}(b)$. Condiţia (2') este astfel satifsfăcută şi, prin echivalenţă şi condiţia 2).

Invers. Fie $G = (V, E)$ un graf cu cel puţin patru vârfuri şi necomplet şi care satisface 1) şi 2). Fie trei vârfuri distincte oarecare $a, b, c \in V$.

Cazul 1. Dacă $ab \in E(G)$ (Fig. 2.1) atunci $\exists D = \{a, ab, b\}$ *ab*-drum în *G* cu $Int(D) = \varnothing$ deci (*) este satisfăcută.

Cazul 2. Dacă $ab \notin E(G)$, facem discuţie după poziţia lui *c*.

Subcazul 2.1. $c \notin N_G(a) \wedge c \notin N_G(b)$ (Fig. 2.2). Deoarece $a \in \overline{N}(c)$ şi $b \in \overline{N}(c)$ din condiţia (1") rezultă că $\exists D$ *ab*-drum în $\left[\overline{N}(c)\right]_G$, deci $\forall u \in D$, $u \in \overline{N}(c)$, relaţia (*) fiind satisfăcută.

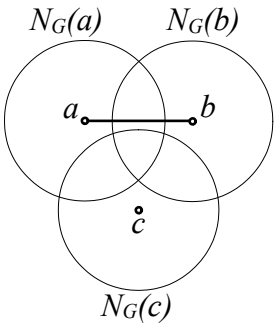


Fig. 2.1

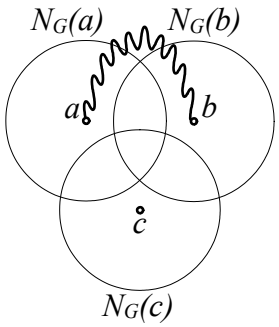


Fig. 2.2

Subcazul 2.2. $c \in N_G(a) \wedge c \notin N_G(b)$ (Fig. 2.3). Deoarece $ac \in E$ din (2') rezultă că $\exists a' \in N_G(a)$ astfel încât $a' \in \overline{N}(c)$. Dar şi $b \in \overline{N}(c)$ deci din (1") rezultă că $\exists D$ *a'b*-drum în $\left[\overline{N}(c)\right]_G$, deci $\forall u \in D$, $u \in \overline{N}(c)$. Alegem $D' = (a, aa', D)$ pentru care $Int(D') = D - \{a\}$, deci *D'* satisface relaţia (*).

Subcazul 2.3. $c \notin N_G(a) \wedge c \in N_G(b)$, analog 2.2.

Subcazul 2.4. $c \in N_G(a) \wedge c \in N_G(b)$ (Fig. 2.4). Deoarece $ac \in E$ din (2') rezultă că $\exists a' \in N_G(a)$ astfel încât $a' \in \overline{N}(c)$. La fel $bc \in E$ şi deci $\exists b' \in N_G(b)$ astfel încât

$b' \in \overline{N}(c)$. Deci aplicând (1") pentru a' şi b' obţinem că $\exists D$ *a'b'*-drum în $\left[\overline{N}(c)\right]_G$. Alegem $D' = (a, aa', D, b'b, b)$ pentru care $Int(D') = D$. Cum $\forall u \in D$, $u \in \overline{N}(c)$ rezultă că D' satisface relaţia (*).

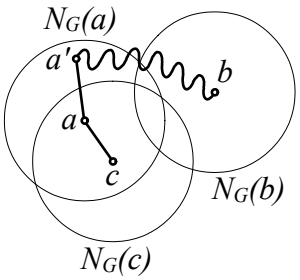


Fig. 2.3.

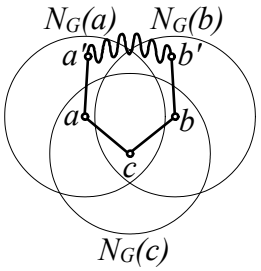


Fig. 2.4.

În concluzie am demonstrat că pentru trei vârfuri distincte oarecare $a, b, c \in V$ avem un *ab*-drum D care satisface (*), deci graful *G* este *confidenţial conex*.

Problema 3.

În problema 2-SAT se dau: o mulţime de variabile booleene $U = \{x_1, x_2, ..., x_n\}$ şi o mulţime de clauze $C = \{C_1, C_2, ..., C_m\}$, unde fiecare clauză C_i este disjuncţia a doi literali $C_i = v_i \vee w_i$, literalii reprezentând variabile sau negaţiile acestora. Problemei i se asociază un digraf *G* cu $V(G) = \{x_1, x_2, ..., x_n, \overline{x}_1, \overline{x}_2, ..., \overline{x}_n\}$ (adică toţi literalii posibili) şi în care pentru fiecare clauză $C_i = v_i \vee w_i$ se adaugă arcele $\overline{v}_i w_i$ şi $v_i \overline{w}_i$ (folosind, evident, convenţia referitoare la dubla negare). Demonstraţi că există o atribuire a valorilor de adevărat şi fals pentru variabilele booleene, astfel încât fiecare clauză să fie adevărată, dacă şi numai dacă digraful *G* are proprietatea că pentru orice $i \in \{1, ..., n\}$ $\overline{x}_i$ şi x_i aparţin la componente tari conexe diferite. (**4 puncte**)

Argomentaţi complexitatea timp de $O(n+m)$ pentru testarea proprietăţii de mai sus. (**2 puncte**)

Soluţie.

Avem: $U = \{x_1, x_2, ..., x_n\}$, $\overline{U} = \{\overline{x_1}, \overline{x_2}, ..., \overline{x_n}\}$ şi $C = \{C_1, C_2, ..., C_m\}$.

$G = (V(G), E(G))$, $V(G) = U \cup \overline{U}$,

$E(G) = \{vw \mid v, w \in V(G), \exists C_i \in C : (C_i = \overline{v} \vee w) \vee (C_i = v \vee \overline{w})\}$

a) Există o atribuire a valorilor 1 (adevărat) şi 0 (fals) pentru $x_1, x_2, ..., x_n$ astfel încât orice clauză e adevărată dacă şi numai dacă digraful *G* are proprietatea: $\forall i \in \{1, 2, ..., n\}$, $\overline{x_i}$ şi x_i aparţin la componente tare conexe diferite.

Direct. Există o atribuire a valorilor 1 (adevărat) şi 0 (fals) pentru $x_1, x_2, ..., x_n$ astfel încât orice clauză e adevărată. Fie această atribuire $f : \{x_1, x_2, ..., x_n\} \to \{0,1\}$. Extindem funcţia f : $f : U \cup \overline{U} \cup C \to \{0,1\}$, $f(\overline{x}) = \overline{f(x)}$, $f(x \vee y) = f(x) \vee f(y)$. Presupunem prin reducere la absurd că $\exists i \in \{1, 2, ..., n\}$, astfel încât $\overline{x_i}$ şi x_i aparţin la

aceeaşi componentă tare conexă. Rezultă că există în G un drum de la x_i la $\overline{x_i}$ şi un drum de la $\overline{x_i}$ la x_i:

$D(x_i, \overline{x_i}) : x_i, x_i a_1, a_1, a_1 a_2, a_2, ..., a_k, a_k \overline{x_i}, \overline{x_i}$

$D(\overline{x_i}, x_i) : \overline{x_i}, \overline{x_i} b_1, b_1, b_1 b_2, b_2, ..., b_l, b_l x_i, x_i$

1. Presupunem $f(x_i) = 1$

$x_i a_1 \in E(G) \Rightarrow \exists C_j = \overline{x_i} \vee a_1, f(C_j) = 1 \Rightarrow f(a_1) = 1$

$x_i a_1 \in E(G) \Rightarrow \exists C_j = \overline{a_1} \vee a_2, f(C_j) = 1 \Rightarrow f(a_2) = 1 \ldots\ldots$

$\Rightarrow f(\overline{x_i}) = 1$ (fals)

2. Presupunem $f(x_i) = 0 \Rightarrow f(\overline{x_i}) = 1$

Se demonstrează ca la punctul ***1.*** că ajungem la $f(x_i) = 1$ (fals)

Deci $\overline{x_i}$ şi x_i nu aparţin la aceeaşi componentă tare conexă. (q.e.d)

Reciproc. Ştim că digraful G are proprietatea: $\forall i \in \{1,2,...,n\}$, $\overline{x_i}$ şi x_i aparţin unor componente tare conexe diferite. Trebuie să găsim o atribuire a valorilor 1 şi 0 pentru $x_1, x_2, ..., x_n$ (o funcţie f definită ca la demonstraţia directă), astfel încât orice clauză din C să fie adevărată ($f(C_j) = 1, \forall j \in \{1,...,m\}$).

Fie G_1 , G_2 , …, G_t componentele tare conexe ale digrafului G. Construim digraful $H = (V(H), E(H))$ astfel:

$V(H) = \{G_1, G_2, ..., G_t\}$, $E(H) = \{G_i G_j \mid i \neq j \in \{1,...,t\}, \exists x \in G_i, \exists y \in G_j : xy \in E(G)\}$

$(x \in G_i \xleftrightarrow{not} x \in V(G_i))$

Propoziţia 1. $\forall i \neq j \in \{1,...t\}$, în H nu pot exista simultan arcele $G_i G_j$ şi $G_j G_i$.

Demonstraţie: Presupunem prin reducere la absurd că există ambele arce în G.

$G_i G_j \in E(H) \Rightarrow \exists x \in G_i, \exists y \in G_j : xy \in E(G)$

$G_j G_i \in E(H) \Rightarrow \exists z \in G_j, \exists v \in G_i : zv \in E(G)$

$y, z \in G_j \Rightarrow \exists D_{yz}$, $zv \in E(G)$, $x, v \in G_i \Rightarrow \exists D_{vx} \Rightarrow$ putem construi în G un drum de la y la x. Dar $xy \in E(G) \Rightarrow$ există în G un drum de la x la y. Deci x şi y ar trebui să aparţină aceleiaşi componente tare conexe (fals).

Propoziţia 2. Nu există în G un drum D de la x la y, $x \neq y$, $x, y \in G_i$, astfel încât D conţine vârfuri ce nu aparţin componentei tare conexe G_i .

Demonstraţie: Presupunem prin reducere la absurd că ar exista un astfel de drum:

$D(x, y) : x, xa_1, a_1, a_1 a_2, a_2, ..., a_k, ..., y$, $a_k \notin G_i$ Rezultă că există în G un drum de la x la a_k ($D(x, a_k)$) şi există în G un drum de la a_k la y ($D(a_k, y)$)

Dar $x, y \in G_i \Rightarrow \exists D(y, x)$. $D(y, x) * D(x, a_k) = D(y, a_k) \Rightarrow$ există în G drum de la a_k la y şi de la y la a_k . Deci a_k şi y ar trebui să aparţină aceleiaşi componente tare conexe (fals).

Propoziţia 3. Nu există circuite în H.

Demonstraţie: Presupunem că ar exista în H un circuit: $G_1, G_1G_2, G_2, ..., G_k, ..., G_1$. Fie $x, y \in G_1$, $x \neq y$. Putem construi un drum de la x la y ce conţine noduri din alte componente tare conexe: $G_2, ..., G_k, ...$ (fals deoarece contrazice ***Propoziţia 2***)

Fie digraful H, $V(H) = \{G_1, G_2, ..., G_t\}$. Putem considera vârfurile din H ordonate astfel încât să fie indeplinită urmatoarea proprietate:

Proprietate: Pentru $\forall vw \in E(G), v \in G_i, w \in G_j$, avem $i \leq j$.

Trebuie să demonstrăm că există o astfel de ordine.

Observăm că $vw \in E(G), v \in G_i, w \in G_j \Rightarrow \exists G_iG_j \in E(H)$. Din ***Propoziţia 1.*** $\Rightarrow$ $G_jG_i \notin E(H)$.

Fie lista $L = G_1, G_2, ..., G_j, ..., G_i, ...G_t$, astfel încât $\exists vw \in E(G), v \in G_i, w \in G_j$ şi $i > j$. Distingem 2 cazuri:

1. Nu există în H drum de la G_j la G_i care să conţină componente G_k, cu $j < k < i$. Deoarece ştim că $G_jG_i \notin E(H)$, putem interschimba poziţiile celor 2 componente, obţinând astfel o ordonare ce îndeplineşte proprietatea.
2. Există în H un drum de la G_j la G_i care să conţină componente G_k, cu $j < k < i$. Adăugând la acest drum arcul G_iG_j, obţinem un circuit în H (ceea ce contrazice ***Propoziţia 3***).

Deci există o ordonare $G_1, G_2, ...G_t$ a vârfurilor din H astfel încât $\forall vw \in E(G), v \in G_i, w \in G_j$, avem $i \leq j$.

Putem construi acum funcţia $f : U \cup \overline{U} \cup C \to \{0,1\}$, astfel încât $f(C_j) = 1, \forall j \in \overline{1,m}$. Se observă că toţi literalii dintr-o componentă tare conexă G_i trebuie să aibă aceeasi valoare (demonstraţia e similară cu cea de la implicaţia directă a teoremei). Deasemenea, observăm că $\forall vw \in E(G), \exists C_i \in C, C_i = (\overline{v} \vee w) = (v \Rightarrow w)$. Dar $v \Rightarrow w$ este adevărată dacă $w = 1$ sau $w = 0 \wedge v = 0$. Deci, pentru a construi funcţia *f* vom parcurge componentele tare conexe în ordinea stabilită anterior şi pentru fiecare G_i vom încerca să atribuim tuturor literalilor w din aceasta componentă valoarea 1 (conform observaţiei de mai sus, toate clauzele de forma $v \Rightarrow w$ vor fi adevărate) (ramura *else* a algoritmului de mai jos) . În caz contrar, tuturor literalilor din G_i le vom atribui valoarea 0.

```
procedure determina_atribuire()
begin
for  i = 1 to n do
   vizitat[ x_i ] = 0
   vizitat[ x_i_bar ] = 0
for  i = t downto 1 do
    if ( ∃v ∈ G_i  astfel încât vizitat[ v_bar ]=1) then
       for (all  v ∈ G_i ) do  f(v) = 0
                               vizitat[ v ] = 1
   else
```

for (all $v \in G_i$) **do** $f(v) = 1$

vizitat[v] = 1

end.

Propoziţia 4. Funcţia construită de algoritmul de mai sus satisface $f(C_j) = 1, \forall j \in \overline{1,m}$.

Demonstraţie:

Notăm $ctc(x) = i$ faptul că x este în componenta tare conexă G_i.

Folosim metoda reducerii la absurd:

Presupunem că $\exists l \in \overline{1,m}$, astfel încât $f(C_l) = 0$

Fie $C_l = v \vee w \Rightarrow f(v) = f(w) = 0$

Considerăm $v \in G_{i1}, w \in G_{i2}$.

$f(v) = 0 \Rightarrow \bar{v}$ a fost deja vizitat (altfel, atunci când $\bar{v}$ va fi vizitat, ar trebui ca $f(\bar{v}) = 0$, ceea ce nu se poate) $\Rightarrow j = ctc(\bar{v}) > ctc(v) = i1$

Analog $f(w) = 0 \Rightarrow k = ctc(\bar{w}) > ctc(w) = i2$.

Deoarece $C_l = v \vee w$, există în G arcele $\bar{v}w$ şi $\bar{w}v$.

Cum $\bar{v}w \in E(G)$, din ***Proprietate*** rezultă că $j = ctc(\bar{v}) \le ctc(w) = i2$. Dar $i2 < k$, deci avem $j < k$.

Analog,din $\bar{w}v \in E(G)$ rezultă că $k = ctc(\bar{w}) \le ctc(v) = i1$. Folosind $i1 < j$, rezultă $k < j$.

(contradicţie)

Deci $f(C_j) = 1, \forall j \in \overline{1,m}$.

b) Pentru testarea proprietăţii trebuie să parcurgem următorii paşi:

Pas1 Construim digraful G pornind de la mulţimile U şi C (digraful va fi reprezentat prin liste de adiacenţă exterioare).

$U = \{x_1, x_2, ..., x_n\}$, $\bar{U} = \{\bar{x_1}, \bar{x_2}, ..., \bar{x_n}\}$, $V(G) = U \cup \bar{U}$.

Iniţializăm listele de adiacenţă cu $\varnothing$, operaţie ce are complexitatea *O(n)* (numărul de vârfuri din G este $2n$). Apoi, pentru fiecare clauză $C_i = v \vee w$, adăugăm v la Adj($\bar{w}$) şi w la Adj($\bar{v}$); rezultă complexitatea *O(m)* (numărul de muchii din G va fi $2m$). Deci, complexitatea primului pas va fi *O(n+m)*.

Pas2 Determinăm componentele tare conexe ale grafului $G=(V, E)$. Folosim algoritmul de parcurgere DFS, ce are complexitatea *O(n+m)*.

Algoritmul:

```
procedure DFS(G)
begin
   for (all v∈V) do
      culoare[v] = 0
   k = 0        //prima culoare folosita va fi k+1=1
   timp = 0   //va marca momentul in care e parcurs fiecare varf
   for (all v∈V) do
```

```
        if (culoare[v]=0) then
            k=k+1
            DFS_recursiv(v)
end.

procedure DFS_recursiv(v)
begin
    timp = timp+1
    culoare[v] = k
    for  ( all u∈Adj(v) ) do
        if ( culoare[u] = 0 ) then
            DFS_recursiv(u)
    timp = timp+1
    timpFinal[v] = timp
end.

procedure CompTareConexe(G)
begin
    DFS(G)
    G' = inversul digrafului G
    DFS(G') /*considerand varfurile in ordinea descrescatoare a
              timpilor finali de vizitare */
end.
```

Complexitatea unei parcurgeri DFS este *O(n+m)*. Construcţia digrafului *G'* necesită o parcurgere a tuturor muchiilor din *G*, deci va avea complexitatea *O(n+m)*. Rezultă că pasul 2 va avea complexitatea *O(n+m)*.

Pas3 Verificăm dacă are loc proprietatea.

```
function ok()
begin
for  i = 1 to n do
    if culoare[ x_i ] = culoare[ x_i_bar ] then
        return false
return true
end.
```

Complexitatea acestui pas este *O(n)*.

Deci testarea proprietăţii necesită timpul *O(n+m) + O(n+m) + O(n) = O(n+m)*.

Gordân Raluca Mihaela
Hriţcu Cătălin
An IIA, grupa 4
Grupa de seminar A34
Echipa nr. 2

Algoritmica grafurilor – tema 7

Problema 1.

Gossip Problem. Într-un grup de n "doamne", fiecare cunoaşte o parte dintr-o bârfă pe care celelalte n-o cunosc. Ele comunică prin telefon şi orice apel telefonic între orice două doamne are ca efect faptul că fiecare din ele va afla tot ce cunoaşte cealaltă.

a) Descrieţi o schemă de a da telefoanele astfel încât într-un număr minim $f(n)$ de apeluri telefonice, fiecare "doamnă" va afla tot ce ştiu celelalte. Indicaţie: Arătaţi că $f(2)=1$, $f(3)=3$, $f(4)=4$ şi pentru $n>4$ $f(n)=2n-4$ (uşor, indicând scheme de telefonare cu aceste numere de apeluri). Încercaţi să argumentaţi că $2n-4$ este chiar numărul minim. (**2 puncte** pentru descrierea schemei, **1 punct** pentru demonstrarea optimalităţii)

b) Modelaţi problema în limbajul teoriei grafurilor: schemei de telefonare îi va corespunde un şir de muchii iar cunoaşterea comună se va exprima printr-o condiţie referitoare la existenţa unor drumuri speciale cu elementele din şirul considerat. (**1 punct**)

Soluţie.

a) Vom descrie mai întâi o schemă de telefonare urmând ca mai apoi să demonstrăm că numărul de apeluri telefonice este minim.

Pentru $n=2$ cele două doamne vor purta o singură convorbire (altfel nu află una ce ştie celalta), deci $f(2)=1$ (Fig. 1.1) iar secvenţa de convorbiri este $S=\{D_1,D_2\}$.

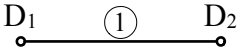


Fig. 1.1

Pentru $n=3$ fie $\{D_1,D_2\}$ prima convorbire, apoi urmează o convorbire $\{D_1,D_3\}$ sau $\{D_2,D_3\}$ prin care D_1 şi D_3 (respectiv D_2 şi D_3) ating conoaşterea globală. Pentru ca şi D_2 (respectiv D_1) să ştie tot mai este nevoie de încă un telefon $\{D_1,D_2\}$ sau $\{D_2,D_3\}$ (respectiv $\{D_1,D_2\}$ sau $\{D_1,D_3\}$). Analog pentru primă convorbire $\{D_1,D_3\}$ sau $\{D_2,D_3\}$. În concluzie $f(3)=3$ iar secvenţele care încep cu $\{D_1,D_2\}$ sunt următoarele:

$S_1=\{D_1,D_2\},\{D_1,D_3\},\{D_1,D_2\}$, $S_2=\{D_1,D_2\},\{D_1,D_3\},\{D_2,D_3\}$,

$S_3=\{D_1,D_2\},\{D_2,D_3\},\{D_1,D_2\}$, $S_4=\{D_1,D_2\},\{D_2,D_3\},\{D_1,D_3\}$ (Fig. 1.2).

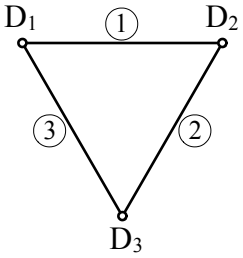


Fig. 1.2

Pentru $n=4$ fie $\{D_1, D_2\}$ prima convorbire. Dacă următoarea convorbire este $\{D_1, D_3\}$, $\{D_1, D_4\}$, $\{D_2, D_3\}$ sau $\{D_2, D_4\}$ secvenţa nu poate avea mai puţin de 5 convorbiri. Deci singura alegere minimă este $\{D_3, D_4\}$ ca a doua convorbire. În acest caz încă două convorbiri disjuncte $\{D_1, D_3\}$ şi $\{D_2, D_4\}$, sau $\{D_1, D_4\}$ şi $\{D_2, D_3\}$ produc secvenţele de lungime 4:

$S_1 = \{D_1, D_2\}, \{D_3, D_4\}, \{D_1, D_3\}, \{D_2, D_4\}$, $S_2 = \{D_1, D_2\}, \{D_3, D_4\}, \{D_2, D_4\}, \{D_1, D_3\}$,

$S_3 = \{D_1, D_2\}, \{D_3, D_4\}, \{D_1, D_4\}, \{D_2, D_3\}$, $S_4 = \{D_1, D_2\}, \{D_3, D_4\}, \{D_2, D_3\}, \{D_1, D_4\}$.

Numărul minim de telefoane este deci $f(4)=4$ (Fig. 1.3).

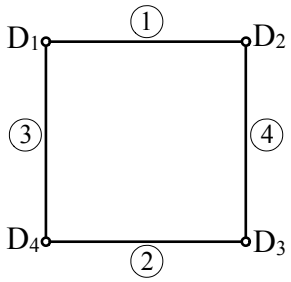


Fig. 1.3

Pentru $n>4$ fie următoarea schemă generală. Mai întâi prima doamnă vorbeşte cu toate doamnele de la 5 la *n*, apoi cu doamnele 2,3 şi 4 prin cele 4 telefoane de la cazul $n=4$ (Fig. 1.3) şi în final când doamnele 1,2,3,4 au ajuns la cunoştinţa globală doamna 1 vorbeşte din nou cu toate doamnele de la 5 la *n*. Secvenţa de $2n-4$ telefoane corespunzătoare acestei scheme este (Fig. 1.4):

$$S_n = \{D_1, D_5\}, \ldots, \{D_1, D_n\}, \{D_1, D_2\}, \{D_3, D_4\}, \{D_1, D_4\}, \{D_2, D_3\}, \{D_1, D_5\}, \ldots, \{D_1, D_n\}$$

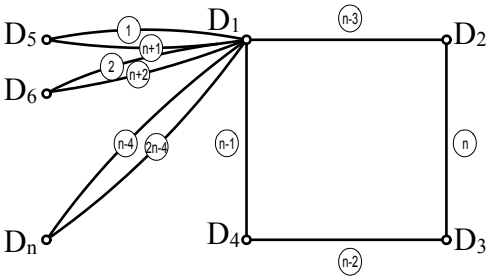


Fig. 1.4

Observaţia. 1.1. Dacă *S* este o secvenţă prin care *n* doamne ajung la cunoaşterea globală atunci şi $\overline{S}$, inversa secvenţei *S* are această proprietate.

Vom arăta în continuare prin inducţie că sunt necesare cel puţin $2n-4$ conversaţii pentru ca fiecare doamnă să afle tot ce ştiu celelalte. Cum pentru $n \le 4$ acestă afirmaţie este adevărată, vom considera că orice secvenţă S_k, $k<n$ cu proprietatea cerută, are cel puţin $2k-4$ elemente (notăm $|S_k| \ge 2n-4$). Demonstrăm prin inducţie că orice secvenţă S_n prin care *n* doamne ajung la cunoştinţa globală satisface $|S_n| \ge 2n-4$.

Cazul 1. Există o doamnă D_i care poartă o singură conversaţie $\{D_i, D_j\}$ în S_n. Această conversaţie este însă precedată de cel puţin $n-2$ conversaţii prin care s-au adunat toate bârfele celorlalte (după conversaţie D_i trebuie să ştie totul). La fel după această conversaţie trebuie să existe încă cel puţin $n-2$ conversaţii prin care D_j răspândeşte (direct sau indirect) celorlalte $n-2$ doamne cunoştinţele doamnei D_i. Prin urmare în acest caz $|S_n| \geq 2n-3 > 2n-4$.

Observaţia. 1.2. Dacă nu suntem în cazul 1 atunci fiecare doamnă poartă cel puţin două conversaţii.

Cazul 2. Există o conversaţie $\{D_i, D_j\}$ care se repetă în S_n. Atunci eliminând toate apariţiile conversaţiei $\{D_i, D_j\}$ din S_n şi înlocuind în toate apariţiile lui D_i cu D_j obţinem o secvenţă S_{n-1} care satisface ipoteza inductivă, deci avem $|S_{n-1}| \geq 2n-6$ iar $|S_n| = |S_{n-1}| + 2 \geq 2n-4$.

Observaţia. 1.3. Dacă nu suntem în cazul 2 atunci nu există convorbiri care se repetă.

Cazul 3. Există o conversaţie $\{D_i, D_j\}$ cu toate că D_i a atins cunoaşterea globală. Presupunem că $\{D_i, D_j\}$ este ultima conversaţie a doamnei D_i (dacă nu e adevărat alegem alt D_j). Fie de asemenea $\{D_i, D_k\}$ prima conversaţie a doamnei D_i. Conform observaţiei 1.3 obţinem că $D_j \neq D_k$. Dacă eliminăm din S_n $\{D_i, D_j\}$ şi $\{D_i, D_k\}$ şi înlocuim orice apariţie a lui D_i cu D_k obţinem S_{n-1} care satisface ipoteza inductivă, deci avem $|S_{n-1}| \geq 2n-6$ iar $|S_n| = |S_{n-1}| + 2 \geq 2n-4$.

Observaţia. 1.4. Dacă nu suntem în cazurile 1,2 sau 3 atunci dacă o convorbire $\{D_i, D_j\}$ este finală pentru D_i, ea este finală şi pentru D_j. Aplicând observaţia 1.1 obţinem că dacă o convorbire $\{D_i, D_j\}$ este iniţială pentru D_i, atunci ea este iniţială şi pentru D_j.

Cazul 4. Există o doamnă D_i care poartă doar două conversaţii telefonice $\{D_i, D_j\}$ şi $\{D_i, D_k\}$. Putem presupune că $\{D_i, D_j\}$ a avut loc înaintea conversaţiei $\{D_i, D_k\}$. De asemenea vom presupune că nici unul din cazurile anterioare nu poate fi aplicat. Fie $\{D_j, D_j'\}$ următoarea convorbirea pe care o poartă doamna D_j după $\{D_i, D_j\}$ (aceasta există deoarece $\{D_i, D_j\}$ este convorbire iniţială şi pentru D_j, orice doamnă purtând cel puţin două conversaţii). De asemenea fie $\{D_k, D_k'\}$ convorbirea pe care o poată D_k înaintea convorbirii $\{D_i, D_k\}$ (aceasta există deoarece $\{D_i, D_k\}$ este convorbire finală şi pentru D_k, orice doamnă purtând cel puţin două conversaţii) (Fig 1.5). Cum cunoştinţele doamnei *a* pot ajunge la celelalte $n-3$ doamne ($\{1,...,n\} \setminus \{D_i, D_j, D_k\}$) decât prin $\{D_j, D_j'\}$ rezultă că este nevoie de cel puţin încă $n-3$ conversaţii. În mod similar este nevoie de încă cel puţin $n-3$ conversaţii pentru a culege cunoştinele celorlalte $n-3$ doamne pentru $\{D_k, D_k'\}$. Dacă aceste mulţimi de conversaţii nu ar fi disjuncte atunci am putea să aplicăm cazul 3. Deci conversaţiile sunt disjuncte şi deci avem satisfăcut $|S_n| \geq 2(n-3)+2 = 2n-4$.

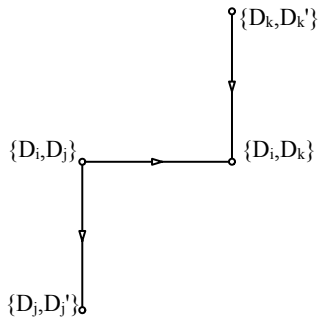


Fig 1.5.

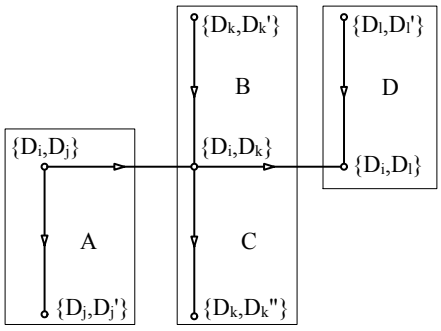


Fig. 1.6.

Cazul 5. Dacă orice doamnă ar purta cel puţin 4 conversaţii, atunci numărul de total conversaţii ar fi jumătate din sama numărul tuturor conversaţiilor pe care le face fiecare doamnă, care este cel puţin $4n$. Deci $|S_n| \geq 4n/2 = 2n > 2n-4$.

Cazul 6. Dacă fiecare doamnă poartă cel puţin 3 conversaţii, atunci dacă nu se poate aplica 5, avem cel puţin o doamnă D_i care poartă exact 3 conversaţii. Fie acestea în ordinea cronologică $\{D_i, D_j\}$, $\{D_i, D_k\}$ şi $\{D_i, D_l\}$ (Fig. 1.6). Presupunem că nici unul din cele cinci cazuri anterioare nu se aplică. Fie $\{D_j, D_j'\}$ următoarea convorbirea pe care o poartă doamna D_j după $\{D_i, D_j\}$. Fie $\{D_k, D_k'\}$ convorbirea pe care o poată D_k înaintea convorbirii $\{D_i, D_k\}$ şi $\{D_k, D_k''\}$ convorbirea pe care o poartă D_k după aceea. În fine fie $\{D_l, D_l'\}$ convorbirea pe care o poartă D_k după $\{D_i, D_l\}$. Notăm cu A mulţimea conversaţiilor ce urmează după $\{D_i, D_j\}$, cu B mulţimea conversaţiilor ce preced $\{D_i, D_k\}$, cu C cea a conversaţiilor ce urmează după $\{D_i, D_k\}$ şi în fine D este mulţimea conversaţiior ce preced $\{D_i, D_l\}$.

Vom presupune că mulţimile A, B, C şi D sunt disjuncte două câte două, cu excepţia lui B şi C pentru care avem $B \cap C = \{D_i, D_j\}$. Într-adevăr presupunând că ele nu ar fi disjuncte se demonstrează relativ uşor că $|S_n| \geq 2n-4$, de exemplu $A \cap B = \{D_n, D_m\}$ atunci prin eliminarea conversaţiilor $\{D_i, D_j\}$ şi $\{D_i, D_l\}$ şi modificând $\{D_i, D_k\}$ în $\{D_k, D_l\}$ obţinem o secvenţă S_{n-1} care satisface ipoteza inductivă, deci avem $|S_{n-1}| \geq 2n-6$ iar $|S_n| = |S_{n-1}| + 2 \geq 2n-4$. În mod similar se demostrează şi în celelalte cazuri avem $|S_n| \geq 2n-4$.

Putem deci presupune în continuare, fără a restrânge din generalitate, că mulţimile A, B, C şi D sunt disjuncte două câte două (exceptînd desigur B şi C). Se observă că pentru a aduce la cunoştinţă tuturor celor $n-1$ doamne secretul doamnei D_i sunt necesare cel puţin $n-1$ telefone, deci $|A \cup C \cup \{\{D_i, D_l\}\}| \geq n-1$, de unde $|A \cup C| \geq n-2$. Aplicând acelaşi raţionament asupra secvenţei inverse (observaţia 1.1) se obţine că $|B \cup D| \geq n-2$. Deci numărul de convorbiri satisface şi în acest ultim caz $|S_n| \geq 2n-4$. În concluzie optimalitatea schemei prezentate a fost demonstrată.

b) Considerăm un multigraf G în care vârfurile sunt cele n doamne iar muchiile sunt etichetate cu numere din intervul $\{1,...,m\}$ cu semnificaţia: există muchia D_iD_j şi este etichetată cu t dacă o conversaţie $\{D_i, D_j\}$ a avut loc la momentul t. În concluzie

având o secvenţă de *m* convorbiri *S* se poate ajunge la multigraful *G* prin etichetarea unei muchii oarecare din din digraf D_iD_j cu *t*, poziţia pe care conversaţia $\{D_i, D_j\}$ o are în sevenţa *S*. Dacă nu există o asemenea convorbire în secvenţă atunci muchia D_iD_j nu va aparţine lui *G*. Condiţia care trebuie pusă pentru ca o doamnă D_i să atingă cunoaşterea globală este următoarea:

Plecând din vârful D_i să existe un drum cu muchiile etichetate în ordine descrescătoare la oricare alt vârf D_j din *G*.

Se observă că existenţa unui asemenea drum este echivalentă cu existenţa unei subsecvenţe în *S* prin care doamna D_i ar fi aflat cunoştinţele doamnei D_j.

Exemplu. Pentru $n = 5$ considerăm următoarea secvenţă de 6 apeluri telefonice: $S = \{D_1, D_4\}, \{D_2, D_3\}, \{D_2, D_5\}, \{D_1, D_2\}, \{D_4, D_5\}, \{D_1, D_3\}$. Multigraful *G* corespunzător acestei secvenţe poate fi observat în Fig. 1.7.

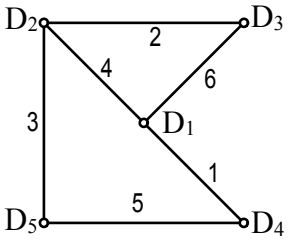


Fig. 1.7.

Vrem să arătăm că schema prezentată satisface cerinţele problemei.

Pentru $i = 1$ avem $(D_1, \{D_1, D_2\}, D_2)$ drum direct până la 2, $(D_1, \{D_1, D_3\}, D_3)$ drum până direct la 3, $(D_1, \{D_1, D_4\}, D_4)$ drum direct până la 4 şi $(D_1, \{D_1, D_2\}, D_2, \{D_2, D_5\}, D_5)$ drum cu muchii etichetate în ordine descrescătoare până la 5. Se observă că $(D_1, \{D_1, D_4\}, D_4, \{D_4, D_5\}, D_5)$ sau $(D_1, \{D_1, D_3\}, D_3, \{D_3, D_2\}, ...)$ sunt drumuri până la 5 care nu satisfac această condiţie.

Pentru $i = 2$ avem $(D_2, \{D_1, D_2\}, D_1)$ drum direct până la 1, $(D_2, \{D_2, D_3\}, D_3)$ drum direct până la 2, $(D_2, \{D_1, D_2\}, D_1, \{D_1, D_4\}, D_4)$ drum cu muchii etichetate în ordine descrescătoare până la 4, şi $(D_2, \{D_2, D_5\}, D_5)$ drum direct până la 5.

Pentru $i = 3$ avem $(D_3, \{D_1, D_3\}, D_1)$ drum direct până la 1, $(D_3, \{D_2, D_3\}, D_2)$ drum până direct la 2, $(D_3, \{D_1, D_3\}, D_1, \{D_1, D_4\}, D_4)$ drum cu muchii etichetate în ordine descrescătoare până la 4 şi $(D_3, \{D_1, D_3\}, D_1, \{D_1, D_2\}, D_2, \{D_2, D_5\}, D_5)$ drum cu muchii etichetate în ordine descrescătoare până la 5.

Pentru $i = 4$ avem $(D_4, \{D_1, D_4\}, D_1)$ drum direct până la 1, $(D_4, \{D_4, D_5\}, D_5, \{D_2, D_5\}, D_2)$ drum cu muchii etichetate în ordine descrescătoare până la 2, $(D_4, \{D_4, D_5\}, D_5, \{D_2, D_5\}, D_2, \{D_2, D_3\}, D_3)$ drum cu muchii etichetate în ordine descrescătoare până la 3 şi $(D_4, \{D_4, D_5\}, D_5)$ drum direct până la 5.

Pentru $i = 5$ avem $(D_5, \{D_4, D_5\}, D_4, \{D_1, D_4\}, D_1)$ drum cu muchii etichetate în ordine descrescătoare până la 1, $(D_5, \{D_2, D_5\}, D_2)$ drum direct până la 2, $(D_5, \{D_2, D_5\}, D_2, \{D_2, D_3\}, D_3)$ drum cu muchii etichetate în ordine descrescătoare până la 3 şi $(D_5, \{D_4, D_5\}, D_4)$ drum direct până la 4.

Problema 2.

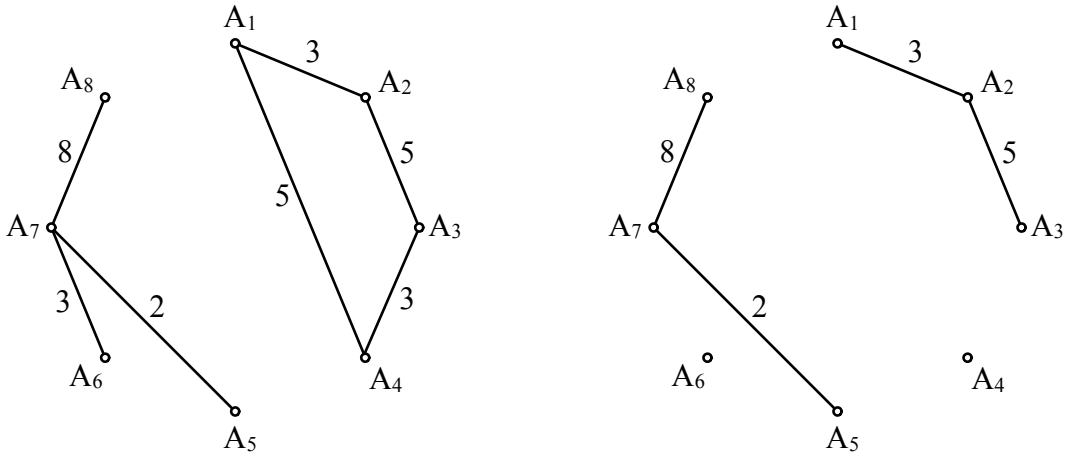
Fie D un digraf şi două funcţii definite pe mulţimea arcelor sale, $a : E(D) \to R_+$ şi $b : E(D) \to R_+^*$. Descrieţi un algoritm eficient pentru determinarea unui circuit C^* în D astfel încât

$$\frac{a(C^*)}{b(C^*)} = \min\left\{\frac{a(C)}{b(C)}; C \text{ circuit în } D\right\}$$

(**4 puncte**).

Soluţie.

RALUCA !!!!!!!!!!!!!!!



Problema 3.

Fie $A_1, A_2, ..., A_n$ submulţimi distincte ale unei mulţimi de n elemente S. Demonstraţi că există un element x în mulţimea S astfel încât $A_1 - \{x\}$, $A_2 - \{x\}$, ..., $A_n - \{x\}$ să fie elemente distincte. (**2 puncte**)

Argomentaţi complexitatea timp de $O(n+m)$ pentru testarea proprietăţii de mai sus. (**2 puncte**)

Soluţie.

Raluca !!!!!!!!!!

Problema 4.

Fie G un graf şi $c : E(G) \to R_+$, o funcţie de capacitate a muchiilor. Oricărui drum din graf cu măcar o muchie i se asociază *locul îngust* ca fiind muchia de capacitate minimă. Descrieţi un algoritm eficient care să determine pentru două vârfuri distincte ale grafului drumul cu locul îngust cel mai mare (dintre toate drumurile de la s la t în graful G). (**4 puncte**)

Soluţie.

Această problemă se poate rezolva cu ajutorul algoritmului lui *Djikstra*, modificând modul de calculare al costului unui drum. Astfel costul unui drum nu mai este calculat ca suma costurilor muchiilor componente ci ca minimul dintre capacităţi.

Acest lucru este posibil deoarece relaţia de minim este asociativă şi deci costul asociat unui drum nu va fi influenţat de ordinea de considerare a muchiilor. Complexitatea acetui algoritm este $O(m \log n)$, în cazul folosirii optimizării aduse de Jhonson.

Totuşi este mult mai simplu să construim un algoritm Greedy pentru această problemă. Să considerăm graful $G = (V, E)$ şi $s, t \in V$ două vârfuri distincte. Se pune problema determinării drumului de capacitate maximă de la *s* la *t*.

Pasul 1. Iniţial se vor sorta muchiile grafului în ordinea descrescătoare a capacităţilor. Avem deci o ordonare $e_1, e_2, ..., e_m$ pentru muchii astfel încât $c(e_1) \geq c(e_2) \geq ... c(e_m)$. Această sortare se poate face prin interclasare sau heap-sort în $O(m \log m)$.

Pasul 2. Vom determină *T*, pădurea parţială de capacitate maximă care conţine pe *s* şi pe *t* şi are număr minim de vârfuri. Această operaţie se poate face prin intermediul algoritmului lui *Kruskal* la care adăugăm o condiţie de oprire suplimentară, pentru *s* şi *t* aparţin aceleiaşi componente, din motive de eficienţă. Algoritmul este următorul:

```
function ArborePartial(G,T)
begin
   T=∅
   k = 1
   while  k<=m and (s şi t ∉ aceleeşi componente) do
      if <T∪ e_i >G nu are circuite then
         T = T ∪{e_i}
      k = k+1
   return T
end.
```

Corectitudinea acestui pas provine din corectitudinea algoritmului lui Kruskal. Acesta calculează un arbore parţial de cost minim. Singura modificare adusă este condiţia de oprire, astfel încât ceea ce se obţine este o pădure ce nu va conţine în mod necesar toate cele *n* vârfuri, dar va conţine în mod sigur pe *s* şi *t* în aceeaşi componente. Există deci în mod sigur un subarbore *T'* care conţine pe *s* şi pe *t*.

Complexitatea acestui pas depinde foarte mult de implementarea structurilor de date pentru reprezentarea mulţimilor disjuncte. Cea mai rapidă implementare cunoscută pentru această este folosind union-find-uri cu uniune după rang şi comprimarea drumului.

Pentru a implementa eficient un union-find trebuie să reţinem, pentru fiecare nod *x* pe lângă valoarea parent[x], rangul său, unde rang[x] este numărul de muchii al celui mai lung drum de la *x* la o frunză descendentă. Atunci când unim două mulţimi, transformăm rădăcina cu rang mai mare în părinte pentru rădăcina cu rang mai mic. De asemenea atunci când se caută reprezentantul unei mulţimi se va realiza o compactare a drumului, toate nodurile de pe drumul de la nodul care se caută la rădăcina componentei sale sunt făcute să arate direct spre rădăcină. Fie următoarerele 4 rutine care implementează cele explicate mai sus:

```
procedure New(x)
begin
   parent[x] = x
   rang[x] = 0
```

```
end.

function Find(x)
begin
   if x≠parent[x] then
      p[x] = find(p[x])
   return p[x]
end.

procedure Merge (x,y)
begin
   if rang[x]>rang[y] then
      parent[y] = x
   else
      parent[x] = y
      if rang[x] = rang[y] then
         rang[y] = rang[y]+1
end.

procedure Union(x,y)
begin
   Merge(Find(x),Find(y))
end.
```

Folosind o astfel de implementare pentru mulţimi disjuncte timpul pentru algoritmul lui Kruskal este de ordiniul $O(m\alpha(m,n)) = O(m\log m)$, unde cu α am notat funcţia lui Ackermann, o funcţie cu o creştere *foarte* lentă. Cum o condiţie de oprire nu poate decât să scadă complexitatea avem complexitatea timp pentru acest pas de $O(m\log m)$.

Pasul 3. Se determină în pădurea T obţinută drumul *unic* de la p la t. Această operaţie se poate face în $O(n+m)$ printr-o parcurgere sistematică a muchiilor, de exemplu cu DFS. Corectitudinea acestui pas (şi deci a algoritmului) se obţine prin inducţie. Fie $D = (p = u_0, u_0u_1, u_1, ..., u_{k-1}, u_{k-1}u_k, u_k = t)$ drumul de la p la t furnizat de algoritm. Fie un drum $D' = (p = v_0, v_0v_1, v_1, ..., v_{l-1}, v_{l-1}v_l, v_l = t)$ de capacitate minimă. Fie $e = v_iv_{i+1}$ prima muchie pentru care $u_iu_{i+1} \neq v_iv_{i+1}$. Dacă muchia e a fost parcursă la pasul 2 şi nu a fost adăugată la T atunci $T \cup e$ ar fi avut un circuit C ce conţine e, dar cum muchiile au fost parcurse în ordine descrescătoare a capacităţii deci drumul $C - e$ are o capacitate cel puţin egală cu e, $C - e$ fiind în acelaşi timp v_iv_{i+1}-drum în D. Aplicând un raţionament similar pentru orice muchie e obţinem prin inducţie că $|D| \geq D'$, deci D este minimal.

Complexitatea totală a algoritmului este $O(m\log m) = O(m\log n^2) = O(m\log n)$, aceeaşi cu cea a algoritmului *Dijkstra* schiţat iniţial.

Exemplu. Fie graful G din Fig. 4.1 $s = 1$ şi $t = 8$. Se ordonează muchiile apoi se adaugă pe rând al T muchii care nu adaugă circuite grafului. Pădurea obţinută după pasul 2 conţine muchiile $\{12,34,67,37,23,78\}$, listate în ordinea în care ele sunt

adăugate de algoritm. Parcurgând DFS această pădure plecând de la 1 şi până la 8 se obţine drumul $D = \{1,12,2,23,3,37,7,78,8\}$ care are capacitatea 4, care este maximă pentru orice drum de la 1 la 8.

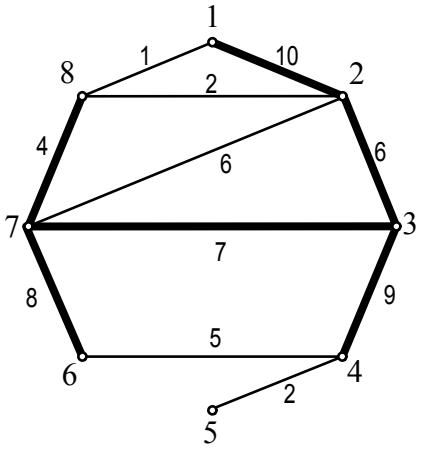


Fig 4.1

Gordân Raluca Mihaela
Hriţcu Cătălin
An IIA, grupa 4
Grupa de seminar A34
Echipa nr. 2

Algoritmica grafurilor – tema 8

Problema 1.

Fie G un graf conex şi o funcţie de cost $c: E(G) \to R$. Vom numi *tăietură* orice mulţime A de muchii ale lui G cu proprietatea că există o bipartiţie $(S, V(G) - S)$ a mulţimii vârfurilor lui G astfel încât A este mulţimea muchiilor lui G cu extremităţile în clase diferite ale bipartiţiei.

a) Arătaţi că dacă funcţia de cost are proprietatea că orice tăietură are o unică muchie de cost minim, atunci există un unic arbore parţial de cost minim. (**2 puncte**)
b) Deduceţi că, dacă funcţia de cost c este injectivă, atunci G are un unic arbore parţial de cost minim. (**1 punct**)
c) Sunt adevărate reciprocile afirmaţiilor a) şi b)? (**1 punct**)

Soluţie.

a) Presupunem că avem satisfăcută proprietatea din enunţ:

(IP) $\forall A$ tăietură, $\exists e^* \in A$ astfel încât $c(e) < e^* \ \forall e \in A - \{e^*\}$.

Vrem să arătăm că există un unic arbore paţial de cost minim:

(C) $\exists T^* \in T_G$ astfel încât $c(T^*) < c(T) \ \forall T \in T_G - \{T^*\}$.

Fie T^* un arbore parţial de cost minim pentru G. Presupunem prin absurd că există $T' \neq T^*$ astfel încât $c(T^*) = c(T')$, deci şi T' ar fi arbore parţial de cost minim. Fie $e \in E(T'), e \notin E(T^*)$, aceasta există deoarece $T' \neq T^*$. Deoarece T^* este arbore rezultă că $T^* + e$ are exact un circuit C. Fie $e' \in C - e$ oarecare, $\{e, e'\}$ este o tăietură în $T^* + e$ deci conform (IP) $c(e') > c(e)$ sau $c(e) > c(e')$.

Cazul 1. Dacă $\exists e' \in C - e$ pentru care avem $c(e') > c(e)$, atunci construim $T_1 = T^* + e - e'$ arbore cu $c(T_1) = c(T^*) + c(e) - c(e') < c(T^*)$, contradicţie cu faptul că T^* este arbore parţial de cost minim.

Cazul 2. Dacă $\forall e' \in C - e$ avem $c(e) > c(e')$ atunci alegem $e'' \in C - e \subseteq T^*$, $e'' \notin T'$ (e'' există pentru că altfel întreg circuitul C ar fi indus în a arborele T'). Deoarece T' este arbore rezultă că $T' + e''$ are exact un circuit C' cu $e \in C'$ şi $c(e) > c(e'')$. Construim $T_2 = T' + e'' - e$ arbore cu $c(T_2) = c(T') + c(e'') - c(e) < c(T')$, contradicţie cu faptul că T' este arbore parţial de cost minim.

În concluzie dacă orice tăietură are o unică muchie de cost minim, atunci există un unic arbore parţial de cost minim.

b) Presupunem că funcţia de cost $c: E(G) \to R$ este injectivă. Avem deci că $\forall e_i, e_j \in E(G)$ cu $i \neq j$ $c(e_i) \neq c(e_j)$ şi prin urmare există $e_1, e_2, ..., e_m$ o ordonare a muchiilor din $E(G)$ astfel încât $c(e_1) < c(e_2) < ... < c(e_m)$. Fie A o tăietură oarecare în G, $A = \{e_{k_1}, e_{k_2}, ..., e_{k_p}\}$, pentru care însă $c(e_{k_1}) < c(e_{k_2}) < ... < c(e_{k_m})$, deci $\exists e^* = e_{k_1} \in A$

astfel încât $c(e) < e^* \quad \forall e \in A - \{e^*\}$. Aplicând implicaţia de la a) obţinem că G are un unic arbore parţial de cost minim.

c) Ambele reciproce sunt false. Fie G un arbore cu funcţia de cost asociată $c : E(G) \to R$ constantă: $c(e) = c, \forall e \in E(G)$. Este evident că G este unicul arbore parţial al său $T_G(G) = \{G\}$, deci G are un unic arbore parţial de cost minim. Dacă $|E(G)| \geq 2$ atunci $\forall e_i, e_j \in E(G)$ cu $i \neq j$ dar $c(e_i) = c(e_j)$ deci c nu este injectivă. Dacă $|E(G)| \geq 3$ deoarece G este conex obţin că $\exists v \in G$ cu $d_G(v) = 2$, deci $\exists A = E(v, V - v)$ tăietură cu $|A| \geq 2$. Dar deoarece $c(e) = c, \forall e \in E(G)$ se obţine că toate muchiile tăieturii A (cel puţin două) au cost minim. În concluzie reciprocele afirmaţiilor a) şi b) nu sunt adevărate.

Problema 2.

Considerând o numerotare fixată a celor $m > 0$ muchii ale unui graf conex $G = (V, E)$ de ordin n. Pentru orice submulţime de muchii A considerăm $x^A \in GF^m$ vectorul m-dimensional cu elemente 0,1 definit prin $x_i^A = 1 \Leftrightarrow e_i \in A$ (vectorul caracteristic). GF^m este spaţiul vectorial peste corpul GF (cu elementele 0 şi 1, şi operaţiile de adunare şi înmulţire modulo 2).

a) Demonstraţi că mulţimea vectorilor caracteristici ai tuturor tăieturilor grafului G, la care adăugăm şi vectorul nul, formează un subspaţiu vectorial X al lui GF^m. (**1 punct**)
b) Demonstraţi că vectorii caracteristici ai mulţimilor muchiilor circuitelor grafului G generează un subspaţiu vectorial U al lui GF^m ortogonal pe X. (**1 punct**)
c) Arătaţi că $\dim(X) \geq n - 1$. (**1 punct**)
d) Arătaţi că $\dim(U) \geq m - n + 1$. (**1 punct**)
e) Deduceţi că $\dim(X) = n - 1$ şi că $\dim(U) = m - n + 1$. (**1 punct**)

Soluţie.

Anexa 1.

Problema 3.

Arătaţi că orice arbore cu gradul maxim $t > 0$ are cel puţin t vârfuri pendante. (**2 puncte**)

Soluţie.

Anexa 2.

Problema 4.

Fie $T = (V, E)$ un arbore cu rădăcina r (un vârf oarecare) şi $parent(v)$ părintele nodului $v \in V$, $v \neq r$. Un cuplaj M al lui T se numeşte *propriu* dacă orice vârf expus v (relativ la M) în T are un frate w (două vârfuri sunt fraţi dacă au acelaşi părinte) astfel încât $w\, parent(v) \in M$.

a) Demonstraţi că orice cuplaj propriu este de cardinal maxim. (**1 punct**)

b) Arătaţi că pentru orice arbore cu n vârfuri, dat prin listele de adiacenţă, se poate construi în timpul $O(n)$ un cuplaj propriu. (**2 puncte**)

Soluţie.

a) Fie $T=(V,E)$ un arbore şi M un cuplaj propriu cu rădăcina r. Vom demonstra că M are cardinal maxim arătând, prin reducere la absurd că nu există drumuri de creştere în T relativ la M.

Presupunem că $\exists P=(u_0,u_0u_1,u_1,...,u_{k-1}u_k,u_k)$ drum de creştere în T relativ la M. Avem deci pentru P satisfăcute următoarele proprietăţi:

1) $\forall u \in Int(P)$, $\exists v \in P$ astfel încât $uv \in M \cap P$

2) $u_0, u_k \in E(M)$.

Cum $u_0 \neq u_k$ (T nu are circuite, fiind arbore) avem $u_0 \neq r$ sau $u_k \neq r$. Presupunem că $u_0 \neq r$, atunci forma drumului P este reprezentată în Fig. 4.1.

Deoarece $u_0 \in E(M)$ şi M este cuplaj propriu avem că $\exists w$ frate pentru u_0 ($parent(u_0)=parent(w)=u_1$) astfel încât $wu_1 \in M$. Dar folosind 1) obţinem că $u_1u_2 \in M$ şi deci $d_M(u_1)=2$, ceea ce contrazice faptul că M este cuplaj. În concluzie M este cuplaj de cardinal maxim.

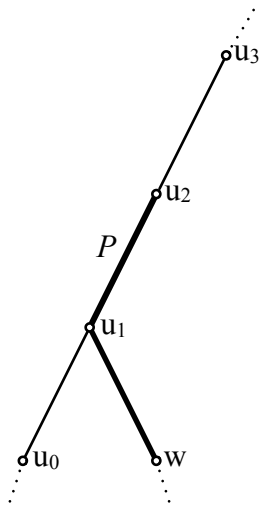


Fig. 4.1.

b) Determinarea unui cuplaj propriu se poate realiza cu ajutorul unei parcurgeri în adancime (DFS) a arborelui T. Iniţial cuplajul construit M este vid, şi la M se adaugă pe rând muchii cu păstrarea proprietăţii de cuplaj. Dacă la întroarcerea din parcuregerea în adâncime se descoperă două vârfuri i şi j expuse relativ la M în relaţia părinte copil atunci muchia ij este adăugată la M. Fie următoarea implementare recursivă a algoritmului prezentat mai sus:

```
procedure DFSCuplajPropriu (T,M,i)
begin
   for j∈Adj(i) do
      parent[j] = i
      DFSCuplajPropriu(T,M,j)
      if (saturat[i]=0 and saturat[j]=0) then
         M = M∪{ij}
         saturat[i] = 1
```

```
            saturat[j] = 1
end

procedure CuplajPropriu(T,M,r)
begin
    for i∈V(T) do
        saturat[i] = 0
    M = ∅
    DFSCuplajPropriu(T,M,r)
end
```

Corectitudinea algoritmului este imediată. Proprietatea că M este cuplaj se demonstrază prin inducţie după numărul de elemente din M. Pentru $|M|=0$ avem că $M=\varnothing$ care este cuplaj. Dacă la un anumit pas k M_k este cuplaj atunci la pasul următor $M_{k+1}=M_k \bigcup e$ unde $e=ij$ este o muchie cu ambele extremităţi expuse relativ la M_k , şi $d_{M_{k+1}}(i)=d_{M_k}(i)+1=1$ şi $d_{M_{k+1}}(j)=1$, deci şi M_{k+1} este cuplaj. Rămâne de arătat că M, cuplajul obţinut la finalul algoritmului, este cuplaj propriu. Fie v un vârf expus relativ la M în T. Asta înseamnă că la întoarcerea în $u=parent(v)$, u era deja saturat. Cum la parcuregerea în adâncime întoarcerea dintr-un nod nu se face decât atunci când s-au parcurs toţi adiacenţii, nodul u a fost saturat de către un alt fiu al său w, deci w este frate pentru v astfel încât $uw\in M$. Cum alegerea nodului v a fost arbitrară obţinem că M este într-adevăr un cuplaj propriu.

Complexitatea timp a acestui algoritm este $O(|E|+|V|)$ dată de parcuregera DFS, dar ţinând seama că T este arbore avem că $|E|=n-1$ dacă $|V|=n$. În concluzie timpul de execuţie a acestui algoritm este $O(n)$, liniar în raport cu numărul de noduri.

Gordân Raluca Mihaela
Hriţcu Cătălin
An IIA, grupa 4
Grupa de seminar A34
Echipa nr. 2

Algoritmica grafurilor – tema 9

Problema 2.

Numim cuplaj *de grad maxim* în graful *G*, un cuplaj *M* cu suma gradelor vârfurilor saturate de *M* maximă printre toate cuplajele grafului.

a) Arătaţi că un cuplaj de grad maxim este de cardinal maxim (**2 puncte**)
b) Demonstraţi că există în graful *G* un cuplaj care saturează toate vârfurile de grad maxim dacă şi numai dacă orice cuplaj de grad maxim are aceeaşi proprietate. (**2 puncte**)
c) Demonstraţi că dacă mulţimea vârfurilor de grad maxim ale grafului G induce un graf bipartit, atunci *G* are un cuplaj care saturează toate vârfurile de grad maxim. (**2 puncte**)
d) Deduceţi că mulţimea muchiilor unui graf bipartit *G* poate fi partiţionată în $\Delta(G)$ cuplaje. (**2 puncte**)

Soluţie.

Fie funcţia de pondere $w: E \to R_+$ dată prin $w(uv) = d_G(u) + d_G(v),\ \forall uv \in E$. Fie *M* un cuplaj pentru *G*, avem $w(M) = \sum_{uv \in M} (d_G(u) + d_G(v)) = \sum_{u \in S(M)} d_G(u)$. Este deci satisfăcută următoarea proprietate:

Proprietatea 2.1 *M* este cuplaj de grad maxim în G dacă şi numai dacă *M* este cuplaj de pondere maximă în *G* – cu ponderea *w* dată anterior.

Definiţie. Numim *drum de creştere a ponderii* în *G* relativ la cuplajul *M* un drum alternat cu cel puţin o muchie pentru care suma gradelor extremităţilor expuse este mai mare decât suma gradelor extremităţilor saturate.

Observaţie. Încercăm să obţinem mai multe informaţii despre forma drumurilor de creştere a ponderii. Facem o discuţie după forma drumurilor alternate (Fig. 2.1).

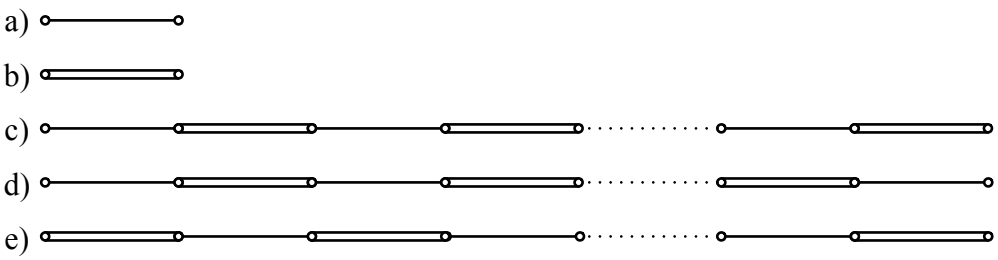


Fig. 2.1

a) P – muchie care nu-i din *M*. P nu are vârfuri saturate, definiţia este satisfăcută deci P este drum de creştere a ponderii.
b) P – muchie din *M*. P nu are vârfuri expuse, definiţia nu este satisfăcută deci P nu este drum de creştere a ponderii.

c) P – drum alternat de lungime pară, deci are o extremitate expusă u_e şi una saturată u_s . P este drum de creştere a ponderii dacă şi numai dacă $d_G(u_e) > d_G(u_s)$.
d) P – drum alternat de lungime impară, care este drum de creştere. Deoarece are ambele extremităţi expuse *P* este drum de creştere a ponderii.
e) P – drum alternat de lungime impară, care nu este drum de creştere. Deoarece are ambele extremităţi saturate *P* nu este drum de creştere a ponderii.

Proprietatea 2.2 *P* este *drum de creştere a ponderii* în *G* relativ la cuplajul *M* dacă şi numai dacă:

i. P este *drum de creştere* în *G* relativ la cuplajul *M*.
ii. sau P este drum alternat par, pentru care $d_G(u_e) > d_G(u_s)$, unde cu u_e am notat extremitatea expusă şi cu u_s cea saturată.

Teoremă. Fie un graf *G* şi *M* un cuplaj, *M* este de grad maxim în *G* dacă şi numai dacă *M* nu admite drumuri de creştere a ponderii.
Demonstraţie.
Direct. Fie *M* un cuplaj de grad maxim în *G*. Presupunem prin absurd că *M* admite un drum de creştere a ponderii. Deci $\exists P$ drum de creştere a ponderii în *G* relativ la *M*.
Caz 1. Dacă *P* este *drum de creştere* în *G* relativ la cuplajul *M* atunci pentru $M_1 = M \Delta P$ avem ponderea $w(M_1) = w(M) + d_G(u_{e_1}) + d_G(u_{e_2})$, deci $w(M_1) > w(M)$, contradicţie.
Caz 2. Dacă P este drum alternat par, pentru care $d_G(u_e) > d_G(u_s)$, avem $M_1 = M \Delta P$ cu ponderea $w(M_1) = w(M) - d_G(u_e) + d_G(u_s)$, şi din nou, $w(M_1) > w(M)$ - contradicţie. Deci *M* nu admite drumuri de creştere a ponderii.
Invers. Presupunem că *M* este un cuplaj ce nu admite drumuri de creştere. Fie M^* un cuplaj de grad maxim în *G*. Fie graful parţial $H = \langle M \Delta M^* \rangle_G$. Evident $d_H(v) \le 2$ $\forall v \in V(G)$ şi deci componentele conexe ale lui *H* sunt drumuri (eventual de lungime 0) sau circuite şi avem următoarele posibilităţi (Fig. 2.2 – muchiile duble sunt din *M* iar cele îngroşate sunt din M^*)

a) ∘
b)
c)
d)
e)

Fig. 2.2

Cazul a) $w(E(C) \cap M) = w(E(C) \cap M^*) = 0$.
Cazul b) Dacă $d_G(u_e) > d_G(u_s)$ atunci este reprezentat un drum de creştere a ponderii pentru *M*, ceea ce contrazice ipoteza. Dacă $d_G(u_e) < d_G(u_s)$ atunci este reprezentat un drum de creştere a ponderii pentru M^*, contradicţie cu prima parte a acestei teoreme - M^* este cuplaj de grad maxim în *G* şi nu are drum de creştere a ponderii. Deci $d_G(u_e) = d_G(u_s)$ de unde $w(E(C) \cap M) = w(E(C) \cap M^*)$.

Cazul c) În Fig. 2.2.c) este reprezentat un drum de creştere pentru *M*, contradicţie.
Cazul d) În Fig. 2.2.d) este reprezentat un drum de creştere pentru M^*, contradicţie.
Cazul e) Se observă că $w(E(C)\cap M)=w(E(C)\cap M^*)=\sum_{v\in V(C)} d_G(v)$.

Pentru fiecare componentă conexă *C* este deci satisfăcută proprietatea $w(E(C)\cap M)=w(E(C)\cap M^*)$. Sumând după fiecare componetă conexă se obţine $\sum_{u\in S(M)} d_G(u)=\sum_{v\in S(M^*)} d_G(v)$ de unde $w(M)=w(M^*)$, deci şi *M* este un cuplaj de grad maxim în *G*. Teorema este astfel demonstrată.

a) Fie *M* un cuplaj de grad maxim. Demonstrăm prin reducere la absurd că *M* este cuplaj de cardinal maxim. Presupunem că există P drum de creştere pentru *M*.

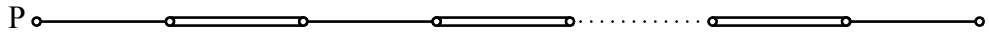


Fig. 2.3

Fie u_{e_1} şi u_{e_2} cele două extremităţi expuse ale acestui drum. Fie $M_1 = M\Delta P$, avem satisfăcută relaţia $w(M_1)=w(M)+d_G(u_{e_1})+d_G(u_{e_2})$, de unde rezultă că $w(M_1)>w(M)$, contradicţie cu faptul că *M* un cuplaj de grad maxim. Deci *M* este cuplaj de cardinal maxim.

b) Implicaţia inversă este imediată. Dacă orice cuplaj de grad maxim saturează toate vârfurile de grad maxim, cum există întotdeauna cel puţin un cuplaj de grad maxim, acesta este un cuplaj ce saturează toate vârfurile de grad maxim.
Demonstraţia implicaţiei directe este mai complicată. Presupunem că $\exists M_1$ cuplaj care saturează toate vârfurile de grad maxim. Fie următorul algoritm care transformă M_1 într-un cuplaj de grad maxim cu păstrarea proprietăţii de mai sus.

```
function cuplaj_maxim(M1)
begin
   M = M1
   while există P drum de creştere a ponderii pt. M do
      M = MΔP
   return M
end.
```

La fiecare iteraţie a ciclului *while* ponderea cuplajului curent creşte cu cel puţin o unitate. Cum ponderea maximă a unui cuplaj este $2|E|$ (dacă toate vârfurile sunt saturate) atunci algoritmul este finit executând maxim $2|E|$ paşi. Corectitudinea este dată de către terema de caracterizare a cuplajelor de grad maxim cu drumuri creştere a ponderii demonstrate precedent. Neajunsul major al acestui algoritm este că timpul de execuţie este exponenţial dat de necesitatea căutării unui drum de creştere a ponderii.

Demonstrăm în plus că plecând cu un cuplaj M_1 ce saturează toate vârfurile de grad maxim, cuplajul obţinut în urma algoritmului păstrează această proprietate. Folosim inducţie după numărul de execuţii ale ciclului *while*. Cuplajul iniţial are proprietatea, deci putem presupune că după *k* operaţii M_k saturează toate vârfurile de

grad maxim. Presupunem prin absurd că $M_{k+1} = M_k \Delta P_k$ nu satisface proprietatea, deci $\exists v \in V(G), d_G(v) = \Delta(G)$ astfel încât $v \in S(M_k)$ şi $v \in E(M_{k+1})$.

Caz 1. Dacă P_k este drum de creştere pentru M atunci $E(M_{k+1}) = E(M_k) \setminus \{u_{e_1}, u_{e_2}\}$, deci deoarece $v \in S(M_k)$ avem $v \in S(M_{k+1})$, contradicţie.

Caz 2. Dacă P este drum alternat par, pentru care $d_G(u_e) > d_G(u_s)$, unde cu u_e am notat extremitatea expusă şi cu u_s cea saturată, avem $E(M_{k+1}) = (E(M_k) \setminus \{u_e\}) \cup \{u_s\}$. Dacă $v \neq u_s$ atunci $v \in S(M_{k+1})$, contradicţie. Dacă $v = u_s$ atunci $d_G(u_e) > d_G(u_s)$ contradicţie cu faptul că $d_G(v) = \Delta(G)$.

În concluzie M_2 cuplajul obţinut în urma algoritmului este de grad maxim şi saturează toate vârfurile de grad maxim. Vom demonstra prin reducere la absurd că orice alt cuplaj de grad maxim are această proprietate. Presupunem că există un cuplaj de grad maxim M_3 pentru care $\exists v \in V(G), d_G(v) = \Delta(G), v \in E(M_3)$ ($v \in S(M_2)$). Fie graful parţial $H = \langle M_2 \Delta M_3 \rangle_G$. Evident $d_H(v) \leq 2 \quad \forall v \in V(G)$ şi deci componentele conexe ale lui H sunt drumuri (eventual de lungime 0) sau circuite şi avem următoarele posibilităţi (Fig. 2.4 componenta conexă la care aparţine v – muchiile duble sunt din M_2 iar cele îngroşate sunt din M_3).

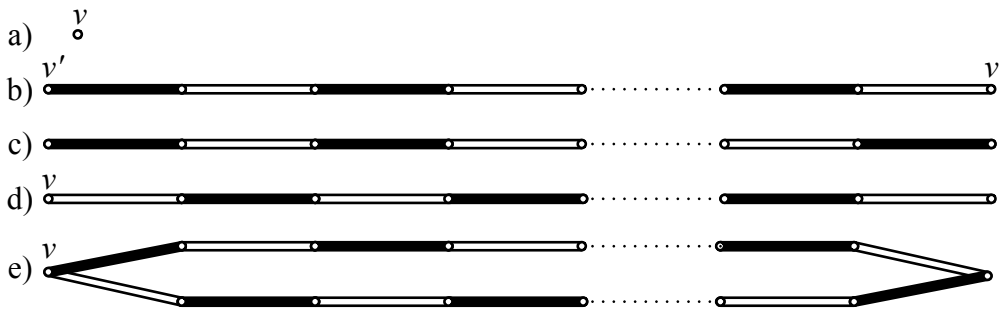


Fig. 2.4

Cazul a) Se observă că $v \notin S(M_2)$, contradicţie.

Cazul b) Deoarece $d_G(v) = \Delta(G)$, avem $d_G(v) \geq d_G(v')$. Dacă $d_G(v) = d_G(v')$ atunci $d_G(v') = \Delta(G)$ şi $v' \in E(M_2)$ contradicţie cu faptul că M_2 saturează toate vârfurile de de grad maxim. Dacă $d_G(v) > d_G(v')$ atunci în Fig. 2.4.b) este reprezentat un drum de creştere a ponderii relativ la M_3, contradicţie cu faptul că M_3 este cuplaj de grad maxim.

Cazul c) Avem $v \in E(M_3) = \varnothing$, contradicţie.

Cazul d) Drumul reprezentat în Fig. 2.4.d) este drum de creştere relativ la M_3, contradicţie cu faptul că M_3 este cuplaj de grad maxim (deci şi de cardinal maxim).

Cazul e) Se observă că $v \notin E(M_3)$, contradicţie.

Deci orice cuplaj de grad maxim saturează toate vârfurile de grad maxim.

c) Folosind punctul b) putem să demonstrăm prin echivalenţă că dacă mulţimea vârfurilor de grad maxim ale grafului G induce un graf bipartit atunci orice cuplaj de grad maxim pentru G saturează toate vârfurile de grad maxim.

Fie G un graf pentru care mulţimea vârfurilor de grad maxim induce un graf bipartit. Presupunem prin absurd că există M un cuplaj de grad maxim, $\exists v \in V$,

$d_G(v)=\Delta(G)$ astfel încât $v\in E(M)$. Demonstrăm că se poate construi un drum de creştere a ponderii pentru *M* plecând din $u_0=v$. Dacă $\Delta(G)=1$ afirmaţia c) este evident adevărată, deci vom considera în continuare că $\Delta(G)\geq 2$.

Pas 0. $d_G(u_0)=\Delta(G)\geq 2$ rezultă că $\exists u_1\in V$, $u_0u_1\in E$, dar $v\in E(M) \Rightarrow u_0u_1\notin M$

Pas 1.

1.1. Dacă $u_1\in E(M)$ atunci (u_0,u_0u_1,u_1) este drum de creştere pentru *M*, gata.

1.2. Dacă $u_1\in S(M)$ atunci există $u_2\in S(M)$, $u_1u_2\in M$.

Pas 2.

2.1. Dacă $d_G(u_2)<\Delta(G)$ rezultă că $d_G(u_0)>d_G(u_2)$, deci $(u_0,u_0u_1,u_1,u_1u_2,u_2)$ este drum de creştere a ponderii pentru *M*, gata.

2.2. Dacă $d_G(u_2)=\Delta(G)\geq 2$ atunci $\exists u_3\in V$, $u_2u_3\in E$, $u_2u_3\notin M$.

2.2.1. Dacă $u_3=u_0$ (Fig. 2.5) atunci

2.2.1.1. Dacă $d_G(u_1)<\Delta(G)$ atunci $(u_0,u_0u_2,u_2,u_2u_1,u_1)$ este drum de creştere a ponderii pentru *M*, gata.

2.2.1.2. Dacă $d_G(u_1)=\Delta(G)$ atunci u_0, u_1 şi u_2 sunt toate vârfuri de grad maxim şi $(u_0,u_0u_1,u_1,u_1u_2,u_2,u_2u_0,u_0)$ este un circuit impar – contradicţie cu faptul că mulţimea vârfurilor de grad maxim ale grafului *G* induce un graf bipartit.

2.2.2. Dacă $u_3\neq u_0$ (Fig. 2.6) atunci execuţia algoritmului se continuă cu pasul 1, cu $u_1\leftarrow u_3$. Fie următoarea funcţie ce implementează această idee:

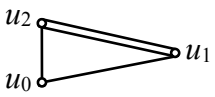


Fig. 2.5.

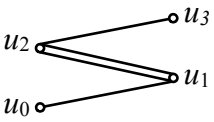


Fig. 2.6

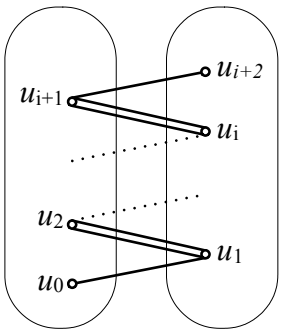


Fig. 2.7

```
function drum_de_creştere_a_ponderii(G,M,v)
begin
   if Δ(G)=1 then
      let v'∈Adj(v)
      return (v,vv',v')
   u[0] = v
   i = 1
   let u[i]∈Adj(v)
   gata = false
   while true do
      if u[i]∈E(M) then
         P = (u[0],u[0]u[1],u[1], ..., u[i])
         gata = true
      else
         let u[i]u[i+1]∈M
         if dG(u[i+1])<Δ(G) then
            P = (u[0],u[0]u[1],u[1], ..., u[i+1])
            gata = true
         else
            let u[i+1]u[i+2]∈E, u[i+1]u[i+2]∉M
            if ∃u[2k], 2k<=i, u[2k]=u[i+2] then
               if dG(u[i])<Δ(G)
                  P = (u[0],u[0]u[1], ..., u[2k-1]u[2k],
                        ,u[2k],u[2k]u[i+1],u[i+1],u[i+1]u[i],u[i])
                  gata = true
               else
                  throw `Circuit impar`
      i = i+2
   return P
end.
```

Să observăm că vârfurile $u_0, u_1, ...u_i, u_{i+1}$, obţinute după fiecare pas complet (gata = false) al algoritmului sunt distincte şi de grad maxim (Fig. 2.7). Cum numărul maxim de vârfuri este finit $|V|$, algoritmul va executa maxim $\frac{|V|}{2}$ paşi, deci un număr finit. El se va termina fie când a găsit un drum de creştere a ponderii pentru M – contradicţie cu faptul că M este cuplaj de grad maxim, fie când a găsit un circuit impar în subgraful indus de vârfurile de grad maxim care este bipartit – contradicţie.

În concluzie presupunearea făcută este falsă deci orice cuplaj de grad maxim pentru G saturează toate vârfurile de grad maxim. Din b) obţinem că G are un cuplaj care saturează toate vârfurile de grad maxim.

d) Folosind punctul anterior putem scrie un algoritm care să construiască cele $\Delta(G)$ cuplaje disjuncte. Plec cu graful iniţial G bipartit, deci şi mulţimea vârfurilor de grad maxim este bipartită. Conform punctului c) există M un cuplaj care saturează toate vârfurile de grad maxim, considerăm acesta primul dintre cele $\Delta(G)$ cuplaje şi eliminăm muchiile sale din G. Obţinem astfel G' un graf bipartit şi care are

$\Delta(G') = \Delta(G) - 1$. Repetăm procedeul de mai sus până obţinem graful nul. Algoritmul care implementează această idee este următorul:

```
function Delta_cuplaje_disjuncte (G')
begin
    G = G'
    i = Δ(G)
    repeat
        găseşte M[i] cuplaj pentru G care saturează
            toate vârfurile de grad maxim
        E(G) = E(G) \ M[i]
        i = i-1
    until (i=0)
    return M[1],M[2],...,M[Δ(G)]
end
```

Algoritmul rulează ciclul *repeat* de exact $\Delta(G)$ ori, la fiecare pas construind un cuplaj disjunct de toate cele precedente, a căror muchii au fost eliminate din *G*. Un asemenea cuplaj există deoarece la fiecare pas graful G este bipartit, iar mulţimea vârfurilor de grad maxim induce un subgraf bipartit, deci putem aplica rezultatul de la punctul c). În concluzie mulţimea muchiilor unui graf bipartit poate fi partiţionată în $\Delta(G)$ cuplaje.

Gordân Raluca Mihaela
Hriţcu Cătălin
An IIA, grupa 4
Grupa de seminar A34
Echipa nr. 2

Algoritmica grafurilor – tema 10

Problema 1.

Fie G un graf conex cu n vârfuri şi T_G familia arborilor săi parţiali. Se consideră graful $H=(\mathsf{T}_G, E(H))$ unde $T_1T_2 \in E(H) \Leftrightarrow |E(T_1)\Delta E(T_2)|=2$.

a) Demonstraţi că H este conex şi are diametrul cel mult $n-1$. (**2 puncte**)
b) Demonstraţi că pentru orice funcţie de cost c pe mulţimea muchiilor grafului G, mulţimea arborilor parţiali de cost c minim induce un subgraf conex în H. (**2 puncte**).

Soluţie.

Avem echivalenţele $T_1T_2 \in E(H) \Leftrightarrow |E(T_1)\Delta E(T_2)|=2 \Leftrightarrow |(E(T_1)\setminus E(T_2))\cup(E(T_2)\setminus E(T_1))|=2$ dar $(E(T_1)\setminus E(T_2))\cap(E(T_2)\setminus E(T_1))=\varnothing$ şi deci $T_1T_2 \in E(H) \Leftrightarrow |(E(T_1)\setminus E(T_2))|+|(E(T_2)\setminus E(T_1))|=2$. T_1 şi T_2 sunt însă arbori parţiali şi avem $|E(T_1)|=|E(T_2)|=n-1$, deci $|(E(T_1)\setminus E(T_2))|=|(E(T_2)\setminus E(T_1))|$. Este satisfăcută:

Proprietatea 1.1. $T_1T_2 \in E(H) \Leftrightarrow |(E(T_1)\setminus E(T_2))|=|(E(T_2)\setminus E(T_1))|=1$

Proprietatea 1.2. $\forall T_1, T_2 \in \mathsf{T}_G$ avem $|E(T_1)\Delta E(T_2)| \equiv 0 \bmod 2$.

Demonstraţie. Presupunem prin absurd că proprietatea nu ar fi satisfăcută. Există deci $T_1, T_2 \in \mathsf{T}_G$ pentru care $|E(T_1)\Delta E(T_2)| \equiv 1 \bmod 2$ deci $|(E(T_1)\setminus E(T_2))|+|(E(T_2)\setminus E(T_1))|=2k+1 \Leftrightarrow 2|(E(T_1)\setminus E(T_2))|=2k+1$, contradicţie (cardinalul unei mulţimi este număr natural).

Notaţie. Notăm $\forall T_1, T_2 \in \mathsf{T}_G$, $k(T_1,T_2)=\dfrac{|E(T_1)\Delta E(T_2)|}{2}$.

Observaţia 1.1.

$k(T_1,T_2)=0 \Rightarrow |E(T_1)\Delta E(T_2)|=0 \Rightarrow T_1=T_2$

$k(T_1,T_2)=1 \Rightarrow |E(T_1)\Delta E(T_2)|=2 \Rightarrow T_1T_2 \in E(H)$.

$k(T_1,T_2)>1 \Rightarrow T_1T_2 \notin E(H)$.

$\max\limits_{T_1,T_2\in\mathsf{T}_G} k(T_1,T_2)=\dfrac{1}{2}\max\limits_{T_1,T_2\in\mathsf{T}_G}|E(T_1)\Delta E(T_2)| \le \dfrac{2(n-1)}{2}=n-1$.

a) Vom demonstra prin inducţie finitară după valoarea lui $k(T_1,T_2)$ că $\forall T_1, T_2 \in \mathsf{T}_G \ \exists D \in \mathsf{D}_{T_1,T_2}$ şi $d_H(T_1,T_2)=k(T_1,T_2)$ (*).

Pentru $k(T_1,T_2)=0 \Rightarrow T_1=T_2$, $D=(T_1)\in \mathsf{D}_{T_1,T_2}$ şi $d_H(T_1,T_1)=0$.

Pentru $k(T_1,T_2)=1 \Rightarrow T_1T_2 \in E(H), D=(T_1,T_1T_2,T_2)\in \mathsf{D}_{T_1,T_2}$ şi $d_H(T_1,T_1)=1$.

Presupunem afirmaţia (*) adevărată $\forall T_1, T_2 \in \mathsf{T}_G$ cu $k(T_1, T_2) < k$, $k > 1$ şi demonstrăm că ea rămâne adevărată şi pentru $\forall T_1, T_2 \in \mathsf{T}_G$ $k(T_1, T_2) = k$. Deoarece $k > 1$ avem că $|(E(T_2) \setminus E(T_1))| > 1$ deci $\exists e_2 \in E(T_2) \setminus E(T_1)$. Deoarece T_1 este arbore rezultă că $T_1 + e_2$ are exact un circuit *C*. $\exists e_1 \in E(T_1) \setminus E(T_2)$ astfel încât $e_1 \in C$, altfel *C* ar fi un circuit indus pentru T_2 care este arbore (contradicţie). Deci graful $T_3 = T_1 + e_2 - e_1$ are $n-1$ muchii şi nu are circuite ($T_1 + e_2$ avea exact un circuit *C* şi am eliminat $e_1 \in C$). Deci $T_3 = T_1 + e_2 - e_1 \in \mathsf{T}_G$, în plus $k(T_1, T_3) = 1 < k$ şi $k(T_3, T_2) = k - 1 < k$. Aplicând prima parte a ipotezei inductive pentru $k(T_1, T_3)$ şi $k(T_3, T_2)$ obţinem că $\exists D_1 \in \mathsf{D}_{T_1, T_3}$ şi $\exists D_2 \in \mathsf{D}_{T_3, T_2}$ deci $\exists D = D_1 \circ D_2 \in \mathsf{D}_{T_1, T_2}$. Cea de-a doua parte a ipotezei inductive arată că $d_H(T_1, T_3) = 1$ şi $d_H(T_3, T_2) = k - 1$, deci $d_H(T_1, T_2) \le k(T_1, T_2)$.

Rămâne de demonstrat că $d_H(T_1, T_2) \ge k(T_1, T_2)$, pentru aceasta considerăm $\forall T_1' \in N_H(T_1)$, deci $\left|E(T_1) \Delta E\left(T_1'\right)\right| = 2 \Leftrightarrow E(T_1) \setminus E\left(T_1'\right) = \{e_1\}$, $E\left(T_1'\right) \setminus E(T_1) = \{e_1'\}$ şi $T_1' = T_1 + e_1' - e_1$. De aici $d_H\left(T_1', T_2\right) \ge k - 1$ şi apoi $d_H(T_1, T_2) \ge k$. Am arătat că $d_H(T_1, T_2) = k(T_1, T_2)$ $\forall T_1, T_2 \in \mathsf{T}_G$.

Din proprietatea (*) rezultă că *H* este conex şi că $\operatorname{diam}(\mathrm{H}) = \max_{T_1, T_2 \in \mathsf{T}_G} d_G(T_1, T_2) = \max_{T_1, T_2 \in \mathsf{T}_G} k(T_1, T_2) \le n - 1$ (observaţia 1.1).

b) Fie T_G^* mulţimea arborilor parţiali de cost *c* minim. Vom demonstra într-un mod similar cu a) că $\forall T_1, T_2 \in \mathsf{T}_G$ $\exists D \in \mathsf{D}_{T_1, T_2}$ în $\left\langle \mathsf{T}_G^* \right\rangle_H$ (**).

Pentru $k(T_1, T_2) = 0 \Rightarrow T_1 = T_2$, $D = (T_1) \in \mathsf{D}_{T_1, T_2}$ în $\left\langle \mathsf{T}_G^* \right\rangle_H$.

Pentru $k(T_1, T_2) = 1 \Rightarrow T_1 T_2 \in E(H), D = (T_1, T_1 T_2, T_2) \in \mathsf{D}_{T_1, T_2}$ în $\left\langle \mathsf{T}_G^* \right\rangle_H$.

Presupunem afirmaţia (**) adevărată $\forall T_1, T_2 \in \mathsf{T}_G^*$ cu $k(T_1, T_2) < k$, $k > 1$ şi demonstrăm că ea rămâne adevărată şi pentru $\forall T_1, T_2 \in \mathsf{T}_G^*$, $k(T_1, T_2) = k$. Fie $e^* = \min_{e \in E(T_1) \Delta E(T_2)} c(e)$. Presupunem că $e^* \in E(T_2) \setminus E(T_1)$ (cazul când $e^* \in E(T_1) \setminus E(T_2)$ se tratează analog). Cum T_1 este arbore rezultă că $T_1 + e^*$ are exact un circuit *C*. $\exists e_1 \in E(T_1) \setminus E(T_2)$ astfel încât $e_1 \in C$, altfel *C* ar fi un circuit indus pentru T_2 care este arbore (contradicţie). Deci graful $T_3 = T_1 + e^* - e_1$ are $n-1$ muchii şi nu are circuite ($T_1 + e_2$ avea exact un circuit *C* şi am eliminat $e_1 \in C$). În plus $c(T_3) = c(T_1) + c(e^*) - c(e_1)$, dar $c(e^*) \le c(e_1)$ ceea ca implică $c(T_3) \le c(T_1)$, unde $T_1 \in \mathsf{T}_G^*$, şi deci $T_3 \in \mathsf{T}_G^*$. În plus $k(T_1, T_3) = 1 < k$ şi $k(T_3, T_2) = k - 1 < k$ şi aplicând ipoteza inductivă avem că $\exists D_1 \in \mathsf{D}_{T_1, T_3}$ şi $\exists D_2 \in \mathsf{D}_{T_3, T_2}$ în $\left\langle \mathsf{T}_G^* \right\rangle_H$ deci $\exists D = D_1 \circ D_2 \in \mathsf{D}_{T_1, T_2}$ în $\left\langle \mathsf{T}_G^* \right\rangle_H$. Am demonstrat astfel că pentru orice funcţie de cost *c* pe mulţimea muchiilor grafului *G*, mulţimea arborilor parţiali de cost *c* minim induce un subgraf conex în *H*.

Problema 2.

Fie $H=(V,E)$ un digraf şi $ts \in E$ un arc fixat al său. Se colorează toate arcele lui *H* cu galben, roşu şi verde arbitrar, cu singura condiţie ca arcul *ts* să fie galben (se poate întâmpla să nu avem arce roşii sau verzi). Demonstraţi algoritmic că are loc exact una din următoarele situaţii:

i) există un circuit în graful $G(H)$ (nu se ţine seama de orientare) cu arce galbene sau verzi care conţine arcul *ts* şi toate arcele galbene ale sale au aceeaşi orientare.

ii) există o partiţie (S,T) a lui *V* astfel încât $s \in S$, $t \in T$, toate arcele de la *S* la *T* sunt roşii şi toate arcele de la *T* la *S* sunt roşii sau galbene. (**2 puncte**)

Soluţie.

Anexa 1.

Problema 3.

Fie $G=(V,E)$ un graf. O mulţime de vârfuri $A \subseteq V$ se numeşte *m*-independentă dacă există un cuplaj *M* al lui *G* astfel încât $A \subseteq S(M)$. Demonstraţi că dacă *A* şi *B* sunt mulţimi m-independente şi $|A|<|B|$, atunci $\exists b \in B-A$: $A \cup \{b\}$ este m-independentă (mulţimile m-independente maximale au acelaşi cardinal). (**4 puncte**)

Soluţie.

Anexa 2.

Problema 4.

Cuplaje stabile în grafuri bipartite. Fie graful complet bipartit $K_{n,n}=(B,F;E)$, unde $B=\{b_1,b_2,...,b_n\}$ şi $F=\{f_1,f_2,...,f_n\}$. Dacă *M* este un cuplaj perfect în $K_{n,n}$ (fiecare *b* este cuplat cu exact un *f*), vom folosi notaţia: $b_i f_i \in M \Leftrightarrow f_i \in M(b_i) \Leftrightarrow b_i \in M(f_i)$. Vom presupune că $\forall b \in B$ are o ordonare a preferinţelor sale pe *F*: $f_{i_1} <_b f_{i_2} <_b ... <_b f_{i_n}$ şi $\forall f \in F$ are o ordonare a preferinţelor sale pe *B*: $b_{i_1} <_f b_{i_2} <_f ... <_f b_{i_n}$. Un cuplaj perfect *M* al lui $K_{n,n}$ se numeşte *stabil* dacă:

$\forall b \in B$ dacă $f <_b M(b)$, atunci $M(f) <_f b$ şi, de asemenea,

$\forall f \in F$ dacă $b <_f M(f)$, atunci $M(b) <_b f$.

Să se arate că pentru orice ordonări ale preferinţelor există un cuplaj stabil şi să se construiească unul în $O(n^3)$. (**4 puncte**)

Soluţie.

Definiţie. O pereche $(b,f) \in B \times F$ se numeşte *pereche instabilă* dacă $f <_b M(b)$ şi $b <_f M(f)$.

Definiţie. Un cuplaj perfect *M* al lui $K_{n,n}$ este *instabil* dacă $\exists b \in B, \exists f \in F$ astfel încât $f <_b M(b)$ şi $b <_f M(f)$, deci are o pereche instabilă.

Cea mai simplă abordare a acestei probleme ar fi să plecăm cu un cuplaj perfect *M* „la întâmplare”, să căutăm o pereche instabilă (b,f) şi să schimbăm cuplajul $M := (M \setminus \{bM(b), fM(f)\}) \cup \{bf, M(b)M(f)\}$. Algoritmul ar fi următorul:

```
procedure CuplajeStabileSimpluDarIncorect(M)
begin
   while ∃(b,f) pereche instabilă do
      M = M\{bM(b),M(f)f}∪{bf,M(b)M(f)}
end
```

Problema cu acest algoritm este că, deşi elimină la fiecare pas o pereche instabilă, poate să creeze altele şi să nu se termine niciodată. O idee mai bună este să pornim cu M vidă şi să adăugăm muchii până obţinem un cuplaj stabil. Pentru simplitatea expunerii vom considera mulţimea B ca fiind o mulţime de băieţi, şi F o mulţime de fete. Dacă există muchia $bf \in M$ vom spune că b şi f sunt căsătoriţi. Dacă $b \in E(M)$ ($f \in E(M)$) vom spune că b (respectiv f) este necăsătorit(ă). La fiecare pas un băiat necăsătorit face o propunere către prima fată din lista sa de propuneri, în ordinea preferinţelor. Dacă fata este nemăritată atunci ea va accepta propunerea (de aceea algoritmul poate fi clasificat drept Greedy – din punctul de vedere al fetelor). Dacă ea este măritată şi preferă pe noul pretendent actualului soţ atunci ea va accepta propunerea şi soţul ei va deveni necăsătorit. Altfel ea va respinge propunerea şi băiatul continuă cu următoarea preferinţă din listă. Nici un băiat nu va cere în căsătorie de două ori aceeaşi fată. Odată ce o fată acceptă o propunere ea va rămâne măritată până la sfârşitul algoritmului (însă nu în mod necesar cu acelaşi soţ). Algoritmul se încheie atunci când nu mai există băieţi necăsătoriţi, şi deci cuplajul care s-a obţinut este perfect.

```
procedure CuplajeStabileCorect(M)
begin
   M = ∅
   while ∃b∈B necăsătorit do
      fie f∈F prima fată nealeasă din lista de preferinţe a lui b
      if f nemăritată then
         M = M ∪ {bf}
      else if b<fM(f) then
         M = M ∪ {bf}
end
```

Să arătăm corectitudinea acestui algoritm. În primul rând să arătăm că alegerea din prima linie a buclei *while* se poate face întotdeauna. Să presupunem (prin absurd) că există b necăsătorit pentru care lista preferinţelor a fost epuizată. Înseamnă că el a făcut propuneri pe rând fiecărei fete şi fie a fost refuzat (deci fata era căsătorită), fie a fost acceptat pentru ca mai apoi să fie înlocuit de un soţ mai bun. În orice caz acum toate cele n fete sunt căsătorite, deci cum sunt doar n băieţi, b este căsătorit – contradicţie.

La fiecare pas un băiat va elimina un element din lista sa de preferinţe. Cum fiecare din cei n băieţi are câte n preferinţe numărul de execuţii ale buclei *while* este de cel mult n^2 – de aici finititudinea algoritmului.

Acum să arătăm stabilitatea cuplajului obţinut. Să presupunem că după ce algoritmul se termină $\exists (b, f) \in B \times F$ o pereche instabilă, echivalent cu $f <_b M(b)$ şi $b <_f M(f)$. Deoarece b este căsătorit cu $M(b)$ şi fiindcă băieţii aleg fetele în ordinea

preferinţelor şi $f <_b M(b)$ înseamnă că *b* a făcut o propunere lui *f* înainte de a face propunerea lui $M(b)$. Dacă *f* a respins această propunere atunci $M_i(f) <_f b$ la acel moment şi rămâne valabilă şi acum deoarece $M(f) < ... < M_i(f) <_f b$ (fetele măritate aleg un alt băiat doar dacă este preferat soţului). Dar $M(f) <_f b$ contrazice ipoteza. Dacă *b* a fost acceptat şi apoi schimbat de *f* cu un alt soţ mai bun atunci avem din nou $M(f) <_f b$ contradicţie. Prin urmare cuplajul obţinut este stabil.

Complexitatea algoritmului depinde foarte mult de structurile de date utilizate pentru stocarea informaţiei. De aceea s-a preferat algoritmul cu complexiatea timp cea mai bună chiar dacă spaţiul ocupat este puţin mai mare.

Preferinţe – băieţi: trebuie să aflu uşor care este preferinţa curentă şi să trec repede la următoarea. Se pot folosi şi liste înlănţuite dar am preferat utilizarea unei matrici pătratice (*n*x*n*) *preferinţă* cu următoarea semnificaţie: *preferinţă* $[b,i] = f$ dacă *f* ocupă lucul *i* în topul preferinţelor lui *b*. Prin urmare

$$preferinţă[b,1] <_b preferinţă[b,2] <_b ... preferinţă[b,n].$$

În plus pentru fiecare băiat vom reţine coloana curentă în tabloul *preferinţa* folosind un tablou unidimensional *curentă*, cu semnificaţia *curentă*[*b*]=*i*, dacă *b* a încercat toate preferinţele de la *preferinţă* [*b*,1] până la *preferinţă* $[b,i-1]$. În acest mod preferinţa curentă este *preferinţa*[b,*curentă*[*b*]] şi poate fi determniată în $O(1)$. Trecerea la următoarea preferinţă se face incrementând *curentă*[*b*] cu o unitate, deci tot în timp constant $O(1)$.

Preferinţe – fete: trebuie să pot face uşor compararea a doi pretendenţi. De aceea vom folosi o matrice pătratică (*n*x*n*) numită *rang* cu următoarea semnificaţie: $rang[f,b_1] < rang[f,b_2]$ dacă şi numai dacă $b_1 <_f b_2$. Comparaţia se face în $O(1)$.

Necăsătorit – pentru a afla cât mai repede un băiat necăsătorit se foloseşte lista înlănţuită *necăsătorit* – folosită ca o stivă. Extragerea şi introducerea unui element într-o stivă (pop, top şi push) se face în timpul $O(1)$.

Măritată – pentru a afla repede dacă o fată este măritată şi care esta soţul ei se va folosi tabloul *măritată* cu următoarea semnificaţie: *măritată*[*f*]=0 dacă *f* este nemăritată şi *măritată*[*f*]=*b* dacă f este măritată cu *b*. În final *măritată* va conţine cuplajul stabil furnizat de algoritm.

```
procedure CuplajeStabileRapid(M)
begin
   for b=n to 1 do
      curentă[b]=1
      push(b,necăsătorit)
   for f=1 to n do
      măritată[f]=0
   while not(empty(necăsătorit)) do
      b = top(necăsătorit)
      f = preferinţă[b,curentă[b]]
      curentă[b] = curentă[b]+1
      if măritată[f]=0 then
         pop(necăsătorit)
         măritată[f] = b
      else if rang[f,b]<rang[f,măritată[f]] then
```

```
            pop(necăsătorit)
            push(măritată[f], necăsătorit)
            măritată[f] = b
    M = ∅
    for f=1 to n do
        M = M ∪ {{f, măritată[f]}}
end
```

Numărul de iteraţii al buclei *while* este cel mult n^2 şi la fiecare iteraţie operaţiile care se execută durează timp constant $O(1)$. Prin urmare complexitatea timp a algoritmului este $O(n^2)$, mult mai bună decât $O(n^3)$ specificată în enunţ. Complexitatea spaţiu este tot $O(n^2)$, dată de reţinerea celor două tablouri $n \times n$ *preferinţă* şi *rang*.

Exemplu. Fie $n = 5$ şi listele de preferinţă:

$b_1 : f_3 <_{b_1} f_2 <_{b_1} f_5 <_{b_1} f_4 <_{b_1} f_1$ $\quad$ $f_1 : b_1 <_{f_1} b_4 <_{f_1} b_3 <_{f_1} b_5 <_{f_1} b_2$

$b_2 : f_2 <_{b_2} f_4 <_{b_2} f_5 <_{b_2} f_1 <_{b_2} f_3$ $\quad$ $f_2 : b_3 <_{f_2} b_1 <_{f_2} b_2 <_{f_2} b_5 <_{f_2} b_4$

$b_3 : f_2 <_{b_3} f_3 <_{b_3} f_1 <_{b_3} f_4 <_{b_3} f_5$ $\quad$ $f_3 : b_4 <_{f_3} b_5 <_{f_3} b_2 <_{f_3} b_1 <_{f_3} b_3$

$b_4 : f_2 <_{b_4} f_3 <_{b_4} f_1 <_{b_4} f_5 <_{b_4} f_4$ $\quad$ $f_4 : b_2 <_{f_4} b_5 <_{f_4} b_3 <_{f_4} b_5 <_{f_4} b_1$

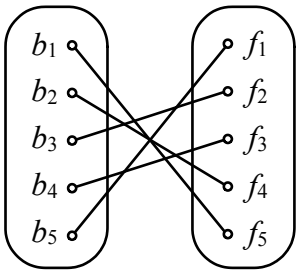
$b_5 : f_1 <_{b_5} f_2 <_{b_5} f_4 <_{b_5} f_5 <_{b_5} f_3$ $\quad$ $f_5 : b_2 <_{f_5} b_1 <_{f_5} b_3 <_{f_5} b_5 <_{f_5} b_4$

Matricile *preferinţă* şi *rang* vor arăta astfel:

$$\textit{preferinţă}=\begin{pmatrix} 3 & 2 & 5 & 4 & 1 \\ 2 & 4 & 5 & 1 & 3 \\ 2 & 3 & 1 & 4 & 5 \\ 2 & 3 & 1 & 5 & 4 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix} \qquad \textit{rang}=\begin{pmatrix} 1 & 5 & 3 & 2 & 4 \\ 2 & 3 & 1 & 5 & 4 \\ 4 & 3 & 5 & 1 & 2 \\ 5 & 2 & 3 & 1 & 4 \\ 2 & 1 & 3 & 5 & 4 \end{pmatrix}$$

Iniţial *curentă*$= (1 \quad 1 \quad 1 \quad 1 \quad 1)$, *necăsătorit*$= (1,2,3,4,5)$, *măritată*$= (0 \quad 0 \quad 0 \quad 0 \quad 0)$. Cum stiva *necăsătorit* nu este vidă se extrage primul element $b = 1$, acesta propune prima sa preferinţă *preferinţă*[1,1]=3, şi aceasta acceptă fiind necăsătorită. Elementul 1 este scos din stivă şi *măritată*[3]=1. Se trece la următorul element din stivă $b = 2$, acesta propune prima sa preferinţă *preferinţă*[1,2]=2, care acceptă deoarece este necăsătorită. Elementul 2 este scos din stivă şi *măritată*[2]=2. Se trece la $b = 3$, acesta propune prima sa preferinţă *preferinţă*[1,3]=2. Deoarece $f = 2$ este căsătorită şi *rang*[2,3]=1<3=*rang*[2,2], 3 va fi acceptat, va fi scos din stiva *necăsătorit*, 2 va fi adăugat în această stivă, şi *măritată*[2]=3. Acum configuraţia este *curentă*$= (2 \quad 2 \quad 2 \quad 1 \quad 1)$, *necăsătorit*$= (2,4,5)$, *măritată*$= (0 \quad 3 \quad 1 \quad 0 \quad 0)$. Se continuă cu $b = 2$ acesta va face următoarea propunere *preferinţă*[2,2]=4 şi deoarece 4 nu este căsătorită va fi accceptat, 2 va fi scos din stivă şi *măritată*[4]=2. Acum $b = 4$ care nu a mai făcut propuneri deci va propune prima preferinţă: *preferinţă*[4,1]=2 dar va fi respins deoarece *măritată*[2]=3 şi *rang*[2,4]=5>1=*rang*[2,3]. El va continua să facă propuneri următoarei fete *preferinţă*[4,2]=3. Cum măritată [3]=1 şi rang[3,4]=1<4=rang[3,1] el va fi acceptat, va fi scos din stiva *necăsătorit*, 1 va fi adăugat în această stivă, şi *măritată*[3]=4. Acum *curentă*$= (2 \quad 3 \quad 2 \quad 3 \quad 1)$, *necăsătorit*$= (1,5)$, *măritată*$= (0 \quad 3 \quad 4 \quad 2 \quad 0)$.

Avem $b=1$, acesta face cea de-a doua cere în căsătorie către *preferinţă*[1,2]=2 care însă va refuza deoarece *măritată*[2]=3 şi *rang*[2,1]=3>1=*rang*[2,3]. Băiatul 1 va facă următoarea cerere *preferinţă*[1,3]=5 şi deoarece *măritată*[5]=0 cei doi se căsătoresc. Ultima cerere o face $b=5$ fetei $f=1$ şi este acceptat deoarece *măritată*[1]=0. Deci în final *măritată*=$(5 \quad 3 \quad 4 \quad 2 \quad 1)$ şi cuplajul stabil obţinut *M* este:



Gordân Raluca Mihaela
Hriţcu Cătălin
An IIA, grupa 4
Grupa de seminar A34
Echipa nr. 2

Algoritmica grafurilor – tema 11

Problema 1.

Se dispune de un algoritm care primind la intrare un graf *G* şi o funcţie de pondere nenegativă pe mulţimea muchiilor acestuia, returnează un cuplaj perfect în graful *G* de pondere minimă (printre toate cuplajele perfecte ale grafului; dacă *G* nu are cuplaj perfect se anunţă acest lucru). Arătaţi că se poate utiliza acest algoritm pentru determinarea eficientă a cuplajului de cardinal maxim într-un graf oarecare.(**3 puncte**)

Soluţie.

Fie CuplajPerfect(G,p) algoritmul care primind la intrare graful *G* şi funcţia de pondere *p*, returnează un cuplaj perfect în *G* de pondere minimă.

$G = (V, E)$ graf oarecare, $|V| = n$, $|E| = m$

Definim o funcţie de pondere $f : E \to \mathbf{Z}_+, f(e) = 1, \forall e \in E$ şi aplicăm algoritmul CuplajPerfect(G,f). Dacă există cuplaje perfecte în *G*, atunci orice cuplaj perfect este chiar cuplaj de cardinal maxim (nici un alt cuplaj nu poate avea cardinalul >*n*/2), deci şi cuplajul returnat de algoritm este de cardinal maxim.

Dacă nu există cuplaje perfecte în *G*, atunci construim un graf G' astfel: dacă $n \equiv 0 (\mathrm{mod}\, 2)$, atunci $G' = K_n$ (G' se obţine adăugând la *G* toate muchiile posibile), iar dacă $n \equiv 1 (\mathrm{mod}\, 2)$, atunci adăugăm la *G* un vârf fictiv v ($V(G' = V(G) \cup \{v\}$) şi toate muchiile posibile (vom avea $G' = K_{n+1}$).

Construim o funcţie de pondere pe mulţimea muchiilor grafului G':

$$p(e) = \begin{cases} 1, e \in E(G') \cap E(G) \\ 2, e \in E(G') - E(G) \end{cases}$$

(muchiile care sunt şi în *G* au pondere 1, iar muchiile fictive au pondere 2)

Fie $M_{pf\,\min}$ cuplajul returnat de algoritmul CuplajPerfect(G′,p). Graful G' fiind complet şi de grad par, are cu siguranţă cuplaje perfecte, deci are un cuplaj perfect de pondere minimă. Avem: $\forall M_{pf}$ cuplaj perfect în G', $p(M_{pf}) \geq p(M_{pf\,\min})$.

Notăm $M = M_{pf\,\min} \cap E(G)$. *M* este cuplaj în *G*. Demonstrăm că *M* este cuplaj de cardinal maxim în *G*.

Demonstraţie:

Presupunem că *M* nu este cuplaj de cardinal maxim în *G*. Fie M' cuplajul de cardinal maxim. $|M'| > |M| \Rightarrow |M'| - |M| > 0$

Cuplajul M' saturează exact $2 * |M'|$ vârfuri din *G* (deci şi din G'). Cum $|G'|$ este un număr par, rezultă că numărul vârfurilor expuse relativ la M' în G' este par şi, în plus, subgraful indus de mulţimea vîrfurilor expuse este un subgraf complet (*H*). Deci putem construi în *H* un cuplaj perfect M_H = CuplajPerfect(*H*,p$_{/E(H)}$).

Mulţimea de muchii $M_{pf} = M' \cup M_H$ este un cuplaj perfect în graful G' (rezultă din construcţia celor două mulţimi).

Avem $|M_{pf}| = |M_{pf\min}| = \frac{|G'|}{2}$ şi $p(M_{pf}) \geq p(M_{pf\min})$.

Putem scrie $M_{pf\min} = (M_{pf\min} - M) \cup M$ şi $|M_{pf\min}| = |M_{pf\min} - M| + |M|$.

Cum $M = M_{pf\min} \cap E(G)$, datorită ponderii alese, avem $p(M_{pf\min}) = |M| * 1 + |M_{pf\min} - M| * 2$. (*)

Analog scriem $M_{pf} = (M_{pf} - M') \cup M'$ şi $|M_{pf}| = |M_{pf} - M'| + |M'|$.

Cum $M' \subseteq E(G)$ şi $(M_{pf} - M') \cap E(G) = \varnothing$ (altfel, M' nu ar fi cuplaj de cardinal maxim în *G*), avem $p(M_{pf}) = |M'| * 1 + |M_{pf} - M'| * 2$. (**)

Din relaţiile (*) şi (**) rezultă:

$p(M_{pf\min}) - p(M_{pf}) = (|M| - |M'|) * 1 + (|M_{pf\min} - M| - |M_{pf} - M'|) * 2$.

Dar $|M_{pf}| = |M_{pf\min}| \Leftrightarrow |M_{pf\min} - M| - |M_{pf} - M'| = |M'| - |M|$.

Rezultă: $p(M_{pf\min}) - p(M_{pf}) = (|M| - |M'|) * 1 + (|M'| - |M|) * 2 = |M'| - |M| > 0$, deci $p(M_{pf\min}) > p(M_{pf})$ (contrazice faptul că $M_{pf\min}$ este cuplaj perfect de pondere minimă în G').

Deci presupunerea făcută este falsă, iar cuplajul $M = M_{pf\min} \cap E(G)$ este cuplaj de cardinal maxim în *G*.

Problema 2.

Arătaţi că se poate determina, într-o matrice cu elemente 0 şi 1 dată, o mulţime de cardinal maxim de elemente egale cu 0 şi care să nu se găsească pe aceeaşi linie sau coloană, cu ajutorul unui algoritm de flux maxim (pe o reţea convenabil definită). (**3 puncte**)

Soluţie.

Fie *B* o matrice oarecare din $\{0,1\}^{n\times m}$ şi $\overline{B} \in \{0,1\}^{n\times m}$, $\overline{B}_{ij} = \neg B_{ij}; i = \overline{1,n}; j = \overline{1,m}$ (matricea complementară). Considerăm matricea simetrică din $\{0,1\}^{(n+m)\times(n+m)}$, $A(B) = \begin{pmatrix} 0_{n\times n} & \overline{B}_{n\times m} \\ \overline{B}^T_{m\times n} & 0_{m\times m} \end{pmatrix}$. Matricea $A(B)$ este o matrice simetrică şi reprezintă matricea de adiacenţă a unui graf bipartit $G(B)$. Se observă că orice graf bipartit $G = (S, T; E)$ cu $|S| = n$ şi $|T| = m$ are matricea de adiacenţă de forma $A(B) = \begin{pmatrix} 0_{n\times n} & B_{n\times m} \\ B^T_{m\times n} & 0_{m\times m} \end{pmatrix}$ şi deci corespondenţa între *B* o matrice din $\{0,1\}^{n\times m}$ şi *G* graful bipartit asociat este biunivocă.

B:		c_1	...	c_j	...	c_m
	l_1	1	...	0	...	0
	...	...	...	...	...	...
	l_i	1	...	0	...	1
	...	...	...	...	...	...
	l_n	0	...	0	...	1

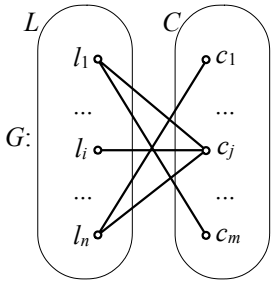


Fig. 2.1

Practic pentru o matrice $B \in \{0,1\}^{n \times m}$ se consideră $G = (L, C; E)$ unde $L = \{l_1, l_2, ..., l_n\}$ liniile matricii B, $C = \{c_1, c_2, ..., c_m\}$ coloanele matricii B şi $E = \{l_i c_j \mid B_{ij} = 0\}$ (Fig. 2.1). A determina o mulţime de cardinal maxim de elemente egale cu 0 şi care să nu se găsească pe aceeaşi linie sau coloană este echivalent cu a găsi un cuplaj de cardinal maxim în graful bipartit asociat. Dacă M este un astfel de cuplaj atunci $\{B_{ij} \mid l_i c_j \in M\}$ este mulţimea căutată.

Determinarea cuplajului de cardinal maxim într-un graf bipartit $G = (L, C; E)$, $|L| = n$, $|C| = m$ se poate reduce la o problemă de flux în reţea. Fie reţeaua $R = (G', s, t, c)$, $G' = (V', E')$ digraf, $V' = L \cup C \cup \{s, t\}$ şi $E' = E_1 + E_2 + E_3$ unde $E_1 = \{sl_i \mid l_i \in L\}$, $E_2 = \{c_j t \mid c_j \in C\}$ şi $E_3 = \{l_i c_j \mid l_i \in L, c_j \in C, l_i c_j \in E\}$. În plus avem $c : E' \to \boldsymbol{Z}_+$ definită prin $c(e) = \begin{cases} 1, e \in E_1 \cup E_2 \\ \infty, e \in E_3 \end{cases}$ (Fig. 2.2).

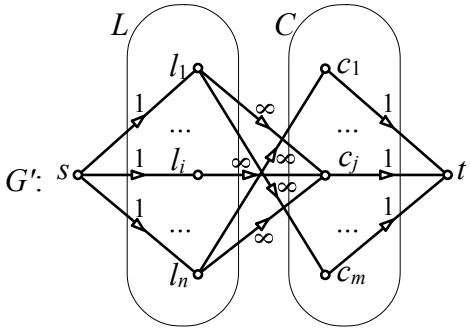


Fig. 2.2

Fie x un flux întreg în R ($x_{uv} \in \boldsymbol{Z}_+$, $\forall uv \in E'$). Fie $\forall l_i \in L$ avem $0 \le x_{sl_i} \le 1$ şi $x_{sl_i} \in \boldsymbol{Z}_+$, deci $x_{sl_i} \in \{0,1\}$ şi analog $\forall c_j \in C$, $x_{c_j t} \in \{0,1\}$. Aplicând legea de conservare a fluxului în $\forall l_i \in L$ obţinem că $\forall c_j \in C$ $0 \le x_{l_i c_j} \le 1$, deci $x_{l_i c_j} \in \{0,1\}$. Deci orice mulţime de arce $\{l_i c_j \mid l_i \in L, c_j \in C, x_{l_i c_j} = 1\}$ induce în graful G un cuplaj $M(x)$. Şi reciproc, orice cuplaj M în G induce o mulţime de arce neadiacente în G'; dacă pentru orice astfel de arc $l_i c_j$ ($l_i \in L, c_j \in C$) se consideră fluxul $x_{l_i c_j} = 1$, de asemenea $x_{sl_i} = 1$, $\forall l_i \in L$, $x_{c_j t} = 1$, $\forall c_j \in C$ şi luând $x = 0$ pe toate celelalte arce atunci fluxul construit are valoarea $|M|$.

În concluzie putem determina un cuplaj de cardinal maxim în G aflând un flux întreg maxim în reţeaua R. Deci problema iniţială a determinării unei mulţimi de cardinal maxim de elemente egale cu 0 care să nu se găsească pe aceeaşi linie sau

coloană într-o matrice *B* se reduce la o problemă de flux maxim în reţea, care poate fi rezolvată cu algoritmul Ahuja-Orlin în timp $O(nm + n^2 \log n) = O(n^3)$.

Exemplu.

Fie matricea *B* cu 4 linii şi 5 coloane dată în Fig. 2.3.a pentru care graful bipartit corespunzător este reprezentat în Fig. 2.3.b. Pentru a afla un cuplaj de cardinal maxim în acest graf se consideră reţeaua *R* din Fig. 2.4.a şi se aplică un algoritm de flux minim. Presupunând că acest algoritm întoarce fluxul x^*, mulţimea de de elemente egale cu 0 din matricea *B* este reprezentată în Fig. 2.4.b.

B:	l_1	l_2	l_3	l_4	l_5
c_1	0	1	1	0	1
c_2	0	1	0	1	1
c_3	1	0	1	1	1
c_4	1	1	1	1	0

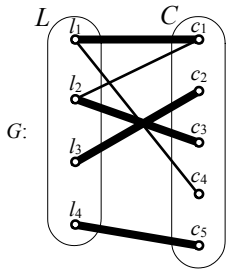
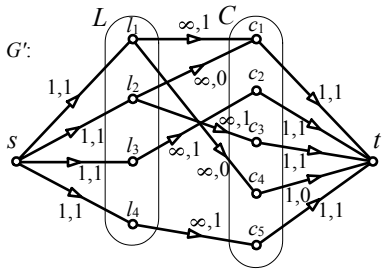


Fig. 2.3



	l_1	l_2	l_3	l_4	l_5
c_1	***0***	1	1	0	1
c_2	0	1	***0***	1	1
c_3	1	***0***	1	1	1
c_4	1	1	1	1	***0***

Fig 2.4.

Problema 3.

Digraful $G = (V, E)$ descrie topologia interconectării într-o reţea de procesoare. Pentru fiecare procesor $v \in V$ se cunoaşte încărcarea sa $load(v) \in R^+$. Se cere să se determine (cu ajutorul unei probleme de flux maxim) un plan de echilibrare statică a încărcării procesoarelor: se va indica pentru fiecare procesor ce cantitate de încărcare va trimite şi la ce procesor, astfel încât, în final, toate procesoarele să aibă aceeaşi încărcare. (**4 puncte**)

Soluţie.

Pentru ca toate procesoarele să fie echilibrate încărcarea lor finală trebuie să fie egală cu media aritmerică a încărcărilor iniţiale $m = \sum_{i \in V} load(i) / |V|$. În funcţie de această valoare procesoarele (vârfuri în digraful *G*) se pot afla în una din cele trei clase de echivalenţă: $V^+ = \{i \in V | load(i) > m\}$ (procesoare supraîncărcate), $V^- = \{i \in V | load(i) < m\}$ (procesoare subîncărcate) şi $V^0 = \{i \in V | load(i) = m\}$ (procesoare „echilibrate"). Plecând de la digraful *G* se construieşte reţeaua $R = (H, c, s, t)$ astfel: $H = (V(H), E(H))$, $V(H) = V \cup \{s, t\}$, $E(H) = E \cup E_1 \cup E_2$ unde $E_1 = \{si | i \in V^+\}$ şi $E_2 = \{jt | j \in V^-\}$. În plus $c : E(H) \to R^+$ definită astfel:

$$c(ij)=\begin{cases} load(j)-m, i=s \\ m-load(i),\ j=t \\ \infty, \text{altfel} \end{cases} \text{ (Fig. 3.1).}$$

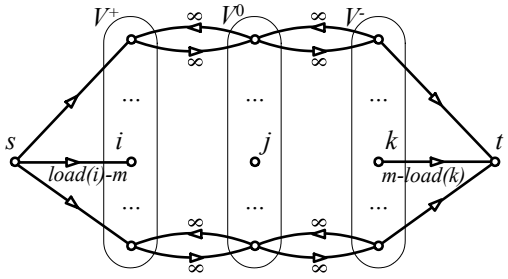


Fig. 3.1.

Fie $v_e=\sum_{i\in V^+}(m-load(i))=\sum_{j\in V^-}(load(j)-m)=c(\{s\},H\setminus\{s\})=c(H\setminus\{t\},t)$. Dacă x^* fluxul maxim în reţeaua R are valoarea v_e atunci echilibrarea procesoarelor este posibilă. Să considerăm pe rând cele trei cazuri:

Cazul 1. Dacă $v(x^*)=v_e$ atunci x^*_{ij}, $\forall ij\in E$ este cantitatea de încărcare pe care o trimite fiecare procesor în reţea pentru a se obţine echilibrarea încărcărilor. La final $newload(i)=load(i)+\sum_{ji\in E}x^*_{ji}-\sum_{ij\in E}x^*_{ij}$.

Cazul 1.1 Dacă $i\in V^+$ atunci aplicând legea conservării fluxului în i obţinem $x^*_{si}+\sum_{ji\in E}x^*_{ji}-\sum_{ij\in E}x^*_{ij}=0$. Dar $v(x^*)=v_e=c(\{s\},H\setminus\{s\})$ deci aplicând teorema Ford-Fulkerson (flux maxim - secţiune minimă) obţinem că $(\{s\},H\setminus\{s\})$ este secţiune minimă şi deci $x^*_{si}=c_{si}=load(i)-m$. Deci $newload(i)=load(i)+\sum_{ji\in E}x^*_{ji}-\sum_{ij\in E}x^*_{ij}=m$, la final procesorul i va fi echilibrat.

Cazul 1.2 Dacă $i\in V^-$ aplicând legea conservării fluxului în i obţinem $\sum_{ji\in E}x^*_{ji}-\sum_{ij\in E}x^*_{ij}-x^*_{it}=0$. Dar $v(x^*)=v_e=c(H\setminus\{t\},t)$ deci aplicând teorema din nou Ford-Fulkerson obţinem că $(H\setminus\{t\},t)$ este secţiune minimă în R şi deci $x^*_{it}=c_{it}=m-load(i)$. Deci $newload(i)=\sum_{ji\in E}x^*_{ji}-\sum_{ij\in E}x^*_{ij}-(m-load(i))=m$, la final procesorul i va fi echilibrat.

Cazul 1.3 Dacă $i\in V^0$ aplicând legea conservării fluxului în i obţinem $\sum_{ji\in E}x^*_{ji}-\sum_{ij\in E}x^*_{ij}=0$ şi $newload(i)=load(i)=m$ deci la final procesorul i va fi echilibrat.

Cazul 2. Dacă $v(x^*)<v_e$ atunci demonstrăm că echilibrarea nu poate avea loc. Pentru orice flux x în R avem $v(x)\le v(x^*)<v_e$. Deoarece $v(x)=v_e<c(\{s\},H\setminus\{s\})$, din teorema Ford-Fulkerson obţinem că $(\{s\},H\setminus\{s\})$ nu este secţiune minimă deci $\exists i\in V^+$ astfel încât $x_{si}<c_{si}=load(i)-m$. Dar aplicând legea conservării fluxului în i obţinem că $x_{si}=\sum_{ij\in E}x_{ij}-\sum_{ji\in E}x_{ji}$ deci $m<load(i)+\sum_{ji\in E}x_{ji}-\sum_{ij\in E}x_{ij}$. Avem $m<newload(i)$ astfel procesorul i este supraîncărcat, şi asta pentru orice flux x în R.

Deci în acest caz nu există o modalitate de echilibrare a încărcării tuturor procesoarelor.

Caz 3. Dacă $v(x^*) > v_e = c(\{s\}, H \setminus \{s\})$ atunci considerând c^* capacitatea minimă a unei secţiuni obţinem că $c^* > c(\{s\}, H \setminus \{s\})$ (flux maxim - secţiune minimă), contradicţie cu minimalitatea capacităţii c^*.

Deci pentru a afla un plan de echilibrare a încărcărilor în digraful G este suficient să aflăm un flux maxim în reţeaua R. Dacă valoarea acestui flux x^* este egală cu v_e atunci echilibrarea se poate realiza şi x_{ij}^* esta încărcarea care trebuie trimisă pe fiecare arc din G pentru a se ajunge la echilibru. Dacă valoarea fluxului maxim x^* este mai mică decât v_e atunci echilibrarea nu se poate realiza. Problema iniţială a determinării unui plan de echilibrare statică a încărcării procesoarelor se reduce astfel la o problemă de flux maxim în reţea, care poate fi rezolvată cu algoritmul Ahuja-Orlin în timpul $O(nm + n^2 \log n) = O(n^3)$.

Exemplu.

$load(1) = 3$ $load(3) = 5$ $load(5) = 0$ $load(7) = 4$

$load(2) = 4$ $load(4) = 1$ $load(6) = 5$ $load(8) = 2$

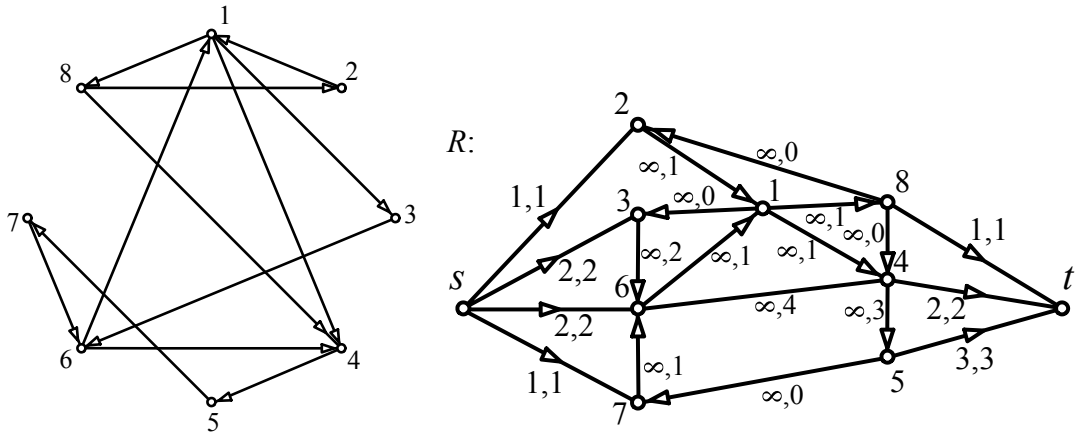


Fig. 3.2

Problema 4.

Să se determine fluxul de valoare maximă în reţeaua din figura de mai jos (explicând funcţionarea algoritmului lui Edmonds-Karp):

(**4 puncte**)

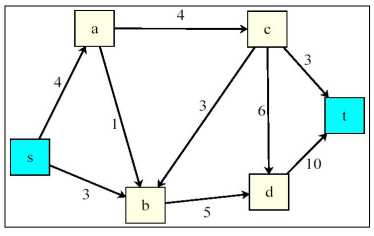


Figure 1: Etichetele arcelor reprezinta capacităţile

Soluţie.

Algoritmul Edmonds-Karp:

procedure FluxMaximSectiuneMinima(R)

begin

```
    x = flux initial
    etichetez s: (e1[s], s2[s], e3[s]) = (NULL, *, ∞)
    C = (s)  /*coada varfurilor etichetate si necercetate */
    while ( C ≠ ∅ ) do
        begin
        i = pop(C)
        etichetare(i) /* etichetez vârfuri neetichetate inca din */
                      /* N+(i) si N-(i) ca la algoritmul Ford-Fulkerson*/
                      /* si le adaug la sfarsitul cozii C */
        if (t a primit eticheta) then
            begin
            cresc fluxul cu e3[t] pe drumul de crestere
            C = ∅  /* sterg toate etichetele */
            etichetez  s  cu (NULL, *, ∞)
            C = (s)
            end
    end
    S ← multimea varfurilor etichetate
    T ← V – S
    return x, (S,T)
end.
```

Etichetarea unui vârf : $i \to e = (e_1[i], e_2[i], e_3[i])$, unde $e_1[i] = j$ este vârful aflat înaintea lui i de pe drumul de creştere, $e_2[i]$ ne spune dacă arcul ji este direct sau invers, iar $e_3[i]$ este capacitatea reziduală a drumului de creştere găsit.

Listele de adiacenţă pentru graful $\overline{G}$ sunt următoarele:

$Adj(s) = (a, b)$ $\qquad$ $Adj(c) = (a, b, d, t)$

$Adj(a) = (s, b, c)$ $\qquad$ $Adj(d) = (b, c, t)$

$Adj(b) = (s, a, c, d)$ $\qquad$ $Adj(t) = (c, d)$

Fluxul iniţial: $x(e) = 0, \forall e \in E$

	sa	sb	ab	ac	cb	bd	cd	ct	dt
x	0	0	0	0	0	0	0	0	0
c	4	3	1	4	3	5	6	3	10

Prima etichetare

$s \to (NULL, *, \infty)$ $\qquad$ $C = (s)$

etichetare(s) $\qquad$ $C = \phi$

$a \to (s, direct, 4 = \min(\infty, 4 - 0))$ $\qquad$ $C = (a)$

$b \to (s, direct, 3 = \min(\infty, 3 - 0))$ $\qquad$ $C = (a, b)$

etichetare (a) $\qquad$ $C = (b)$

$c \to (a, direct, 4 = \min(4, 4 - 0))$ $\qquad$ $C = (b, c)$

etichetare (b) $\qquad$ $C = (c)$

$d \to (b, direct, 3 = \min(3, 5 - 0))$ $\qquad$ $C = (c, d)$

etichetare (c) $C = (d)$

$t \to (c, direct, 3 = \min(4, 3 - 0))$

Am etichetat $t \Rightarrow$ cresc fluxul cu 3 pe drumul de creştere (s, sa, a, ac, c, ct) şi C devine $\varnothing$.

	sa	sb	ab	ac	cb	bd	cd	ct	dt
x	3	0	0	3	0	0	0	3	0

A doua etichetare

$s \to (NULL, *, \infty)$ $C = (s)$

etichetare(s) $C = \phi$

$a \to (s, direct, 1 = \min(\infty, 4 - 3))$ $C = (a)$

$b \to (s, direct, 3 = \min(\infty, 3 - 0))$ $C = (a, b)$

etichetare (a) $C = (b)$

$c \to (a, direct, 1 = \min(1, 4 - 3))$ $C = (b, c)$

etichetare (b) $C = (c)$

$d \to (b, direct, 3 = \min(3, 5 - 0))$ $C = (c, d)$

etichetare (c) $C = (d)$

etichetare (d) $C = \phi$

$t \to (d, direct, 3 = \min(3, 10 - 0))$

Am etichetat $t \Rightarrow$ cresc fluxul cu 3 pe drumul de creştere (s, sb, b, bd, d, dt) şi C devine $\varnothing$.

	sa	sb	ab	ac	cb	bd	cd	ct	dt
x	3	3	0	3	0	3	0	3	3

A treia etichetare

$s \to (NULL, *, \infty)$ $C = (s)$

etichetare(s) $C = \phi$

$a \to (s, direct, 1 = \min(\infty, 4 - 3))$ $C = (a)$

etichetare (a) $C = \phi$

$b \to (a, direct, 1 = \min(1, 1 - 0))$ $C = (b)$

$c \to (a, direct, 1 = \min(1, 4 - 3))$ $C = (b, c)$

etichetare (b) $C = (c)$

$d \to (b, direct, 1 = \min(1, 5 - 3))$ $C = (c, d)$

etichetare (c) $C = (d)$

etichetare (d) $C = \phi$

$t \to (d, direct, 1 = \min(1, 10 - 3))$

Am etichetat $t \Rightarrow$ cresc fluxul cu 1 pe drumul de creştere $(s, sa, a, ab, b, bd, d, dt)$ şi C devine $\varnothing$.

	sa	sb	ab	ac	cb	bd	cd	ct	dt
x	4	3	1	3	0	4	0	3	4

A patra etichetare

$s \to (NULL, *, \infty)$ $C = (s)$

etichetare(s) $C = \phi$

Nu există drumuri de creştere,deci fluxul curent este de valoare maximă: $v(x) = 7$.

Gordân Raluca Mihaela
Hriţcu Cătălin
An IIA, grupa 4
Grupa de seminar A34
Echipa nr. 2

Algoritmica grafurilor – tema 12

Problema 1.

Fie *v* valoarea fluxului maxim în reţeaua $R=(G,c,s,t)$. Demonstraţi că există *k* *st*-drumuri în *G*, $P_1,...,P_k$ ($0\le k\le |E(G)|$), şi numerele reale nenegative $v_1,...,v_k$, astfel încât $x: E(G)\to \boldsymbol{R}$, definit pentru orice arc *ij* prin $x_{ij}=0+\sum_{t:ij\in P_t} v_t$, este flux în *R* de valoare maximă *v*. (**4 puncte**)

Soluţie.

Fie reţeaua $R=(G,c,s,t)$ şi *v* valoarea fluxului maxim în *R*. Vom considera cele *k* *st*-drumuri ca fiind drumuri directe de creştere în *R*. La final vom considera $\forall i\in 1,k$ $v_i=r(P_i)$ capacitatea reziduală a drumului de creştere. Cum drumurile de creştere pot conţine însă şi arce inverse vom elimina aceste arce introducând, eventual, noi drumuri de creştere directe. Atunci când obţinem un drum de creştere ce are şi arce inverse vom parcurge aceste arce de la *t* la *s*, eliminându-le pe rând. Fie $P_1,...,P_{k-1}$ toate drumuri de creştere, şi x^{k-1} fluxul obţinut după cele $k-1$ creşteri succesive ($x_{ij}=0+\sum_{t:ij\in P_t} v_t$). Dacă la pasul *k* se obţine un drum de creştere P_k ce are şi arce inverse atunci se procedează astfel:

Pasul 1. Fie *ab* cel mai apropiat de *t* arc invers din P_k. Deoarece P_k este drum de creştere $x_{ab}^{k-1}>0$ deci $\exists P_j$, $j<k$ drum de creştere astfel încât $ab\in P_j$ (Fig. 1.1).

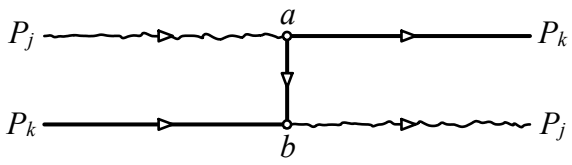


Fig. 1.1

Pasul 2.

Cazul 1. Dacă $v_j>r(P_k)$ (Fig. 1.2) se scade v_j cu $r(P_k)$ şi se construieşte un nou drum $P_{k+1}=s\xrightarrow{P_j}a\xrightarrow{P_k}t$. P_{k+1} are toate arcele directe (*ab* era cel mai apropiat de *t* arc invers din P_k) şi este drum de creştere relativ la noul flux obţinut după ce v_j a scăzut. Capacitatea reziduală a drumului P_{k+1} este $v_{k+1}=r(P_{k+1})=r(P_k)$. În fine drumul P_k devine $s\xrightarrow{P_k}b\xrightarrow{P_j}t$.

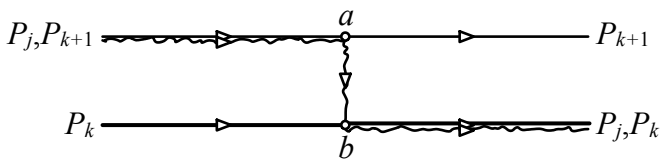


Fig. 1.2

Cazul 2. Dacă $v_j \le r(P_k)$ (Fig. 1.3) drumul P_j va fi înlocuit cu un nou drum $s \xrightarrow{P_j} a \xrightarrow{P_k} t$. Noul P_j are toate arcele directe şi este drum de creştere, capacitatea sa reziduală rămânând neschimbată v_j . Drumul P_k devine $s \xrightarrow{P_k} b \xrightarrow{P_j} t$ şi $v_k = r(P_k)$. Dacă $v_j < r(P_k)$ atunci mai există un drum $P_{j'}$, $j' < k$ drum de creştere astfel încât $ab \in P_{j'}$ pentru care repetăm pasul 2 cu $v_k = v_k - v_j > 0$.

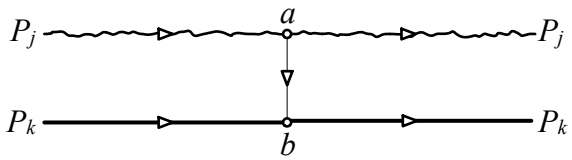


Fig. 1.3

De observat că procedând în acest mod nu am modificat valoarea fluxului obţinut. Obţinem un drum de creştere care are o muchie inversă mai puţin, deci putem repeta algoritmul pentru următorul arc invers, până P_k devine drum direct.

Fie următorul algoritm care calculează valoarea fluxului maxim şi găseşte k drumuri de creştere directe corespunzătoare acestui flux:

```
function Drumuri_directe_de_creştere(G,c,s,t)
begin
   x = 0; k = 0
   while ∃P drum de creştere do
      x = x ⊕ r(P)
      v = r(P)
      while P are arce inverse do
         let ab=cel mai apropiat de t arc invers in P
         r = v; j = 0
         while r>0 do
            if ab∈Pj then
               if r<vj then
                  vj = vj – r
                  k = k + 1 // introduc un drum nou
                  Pk-1 = s – (Pj) – a – (P) – t
                  vk-1 =  r
                  r = 0
               else
                  Pj = s – (Pj) – a – (P) – t
                  r = r - vj
               P = s – (P) – b – (Pj) – t
            j = j+1
      // P este acum drum cu arce directe
      Pk = P; vk = v;
      k = k + 1
   return P0,P1,...,Pk-1
end
```

Se observă că faptul că $P_0,P_1,\ldots,P_k$ sunt drumuri directe de creştere este un invariant pentru algoritm, deci în final vom obţine *k st*-drumuri $P_0,P_1,\ldots,P_k$ şi $v_1,...,v_k$, astfel încât fluxul $x_{ij}=0+\sum_{t:ij\in P_t} v_t$, este flux de valoare maximă în *R*.

Problema 2.

Numim $GP-descompunere$ a grafului complet K_n orice mulţime $\mathsf{A}=\{B_1,...,B_{k(\mathsf{A})}\}$, unde: fiecare B_i este un subgraf bipartit complet al lui K_n, orice două grafuri B_i şi B_j au mulţimile de muchii disjuncte şi $\bigcup_{i=1,k(\mathsf{A})} E(B_i)=E(K_n)$. Arătaţi că orice $GP-descompunere$ A a lui K_n satisface inegalitatea $k(\mathsf{A})\geq n-1$. (**4 puncte**)

Soluţie.

Anexa 1.

Problema 3.

Fie $G=(V,E)$ un graf şi $f:V\to V$ cu proprietatea că $\forall uv\in E: f(u)f(v)\in E$.
a)Demonstraţi că $\omega(G)\leq|f(V)|$.
b)Arătaţi că pentru orice graf $G=(V,E)$ există funcţii *f* cu proprietatea de mai sus şi astfel încât $|f(V)|\leq\Delta(G)+1$. (**4 puncte**)

Soluţie.

Anexa 2.

Problema 4.

Fie $G=(V,E)$ un graf. Numim *partiţie specială* orice bipartiţie (S,T) a lui *V* astfel încât subgraful indus de *T* în *G* este neconex şi subgraful indus de *S* în complementarul grafului *G* este neconex.

a) Arătaţi că graful circuit C_n ($n\geq 3$) nu are partiţii speciale.

b) Descrieţi un algoritm polinomial care să testeze dacă un graf dat are partiţii speciale. (**2 puncte**)

Soluţie.

a) Presupunem că graful $G=C_n$ ($n\geq 3$) are o partiţie specială: (S,T) astfel încât $[T]_G$ este neconex şi $[S]_{\overline{G}}$ este neconex.

$[T]_G$ neconex $\Rightarrow$ componentele sale conexe sunt drumuri induse în C_n : $P_1,P_2,...,P_k,k\geq 2$ astfel încât $|P_i|\geq 1,\forall i=\overline{1,k}$, $\forall i\neq j$, P_i şi P_j sunt disjuncte şi nu există muchie *uv* în C_n cu $u\in P_i$ şi $v\in P_j$.

Rezultă că $[S]_G$ este format din drumurile $Q_1,Q_2,...,Q_k$ având aceleaşi proprietăţi ca $P_1,P_2,...,P_k$:

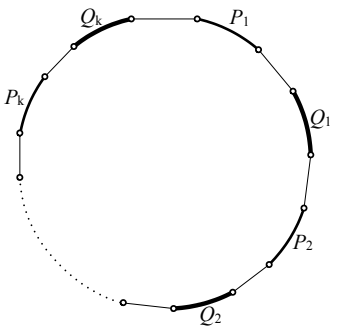


Fig 4.1

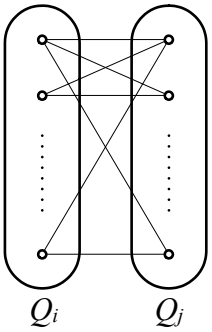


Fig4.2

$\forall i, j \in \{1,...,k\}, i \neq j$, avem: $\forall u \in Q_i, v \in Q_j \Rightarrow uv \notin E(G)$

Rezultă că $\forall i, j \in \{1,...,k\}, i \neq j$, $\forall u \in Q_i, v \in Q_j \Rightarrow uv \in E(\overline{G})$. Altfel spus, în $\overline{G}$, $V(Q_i) \cup V(Q_j)$ induce un subgraf bipartit complet (Fig4.2). Rezultă că există drum în $\overline{G}$ între oricare 2 vârfuri din $V(Q_1) \cup V(Q_2) \cup ... \cup V(Q_k) = S \Rightarrow [S]_{\overline{G}}$ este conex (fals). Deci presupunerea făcută este falsă $\Rightarrow$ graful C_n ($n \geq 3$) nu are partiţii speciale.

b) Deoarece nu am găsit un algoritm polinomial care să testeze dacă un graf dat are partiţii speciale vom prezenta unul exponenţial pentru această problemă. Vom genera prin backtracking pe rând toate cele 2^n bipartiţii ale mulţimii V şi vom testa dacă una dintre acestea este o partiţie specială pentru G. S-a preferat o versiune nerecursivă de backtracking gestiunea stivei căzând în totalitate în sarcina programatorului. O bipartiţie (S,T) a lui V va fi reprezentată prin vectorul S (stivă) cu elemente de 0 şi 1 cu semnificaţia $v \in S$ dacă $s(v)=1$ şi $v \in T$ dacă $s(v)=0$, $\forall v \in V$. Pentru a testa dacă (S,T) este partiţie specială pentru G se folosesc două parcurgeri DFS, pentru determinararea numărului de componente conexe din $\langle T \rangle_G$ respectiv $\langle S \rangle_{\overline{G}}$.

```
function Partiţii_speciale_backtracking(G)
begin
   let Ḡ complementarul lui G
   k = 1
   S[k] = -1
   while k>0 do
      S[k]=S[k] + 1
      if S[k]<=1 then
         if k=n then
            if Partitie_speciala(G, Ḡ ,S) then
               return true
         else
            k = k + 1
            S[k] = -1
      else
         k = k – 1
   return false
```

```
end

function Partiţie_speciala(G, Ḡ ,S)
begin
   if DFS_componente_conexe(⟨S⟩_Ḡ )=1 then
      return false
   for i = 1 to n do
      T = not(S)
   if DFS_componente_conexe(⟨T⟩_G )=1 then
      return false
   return true
end
```

Complexitatea timp a algoritmului este $2^n \cdot O(n+m) = O(2^n)$ exponenţială în raport cu numărul de vârfuri.