GÎRDEA MARTA
URICARU RALUCA
A34
(grupa 4A)

TEMA NR. 1
4 martie 2003

1. *Un graf se numeşte rar dacă numărul 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 adiacenţă A a grafului, care ocupă* n^2 *locaţii de memorie, poate fi întotdeauna reprezentată folosind O(* $\frac{n^2}{\log n}$ *) 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.*

Soluţie:

Având în vedere faptul că în matricea de adiacenţă A a unui graf fiecare element A(i, j) poate lua doar valorile 0 şi 1, este suficient un singur bit pentru a memora aceste informaţii.

Pentru reducerea spaţiului ocupat de matricea de adiacenţă se propune următoarea schemă de reprezentare:

- *matricea A este împărţită în matrice pătratice de dimensiune* $\lceil\sqrt{\log n}\rceil X \lceil\sqrt{\log n}\rceil$ *;*
- *numărul acestor matrice este* $\frac{n^2}{\lceil\log n\rceil}$*;*
- *se creează o nouă matrice pătratică A' de dimensiune* $\lceil\frac{n}{\lceil\sqrt{\log n}\rceil}\rceil \mathrm{X} \lceil\frac{n}{\lceil\sqrt{\log n}\rceil}\rceil$*;*
- *fiecare element al matricei A' este o codificare a unei matrice de dimensiune* $\lceil\sqrt{\log n}\rceil X \lceil\sqrt{\log n}\rceil$ *după cum urmează:*
    - *o astfel de matrice are log n elemente ce pot lua doar valorile 0 şi 1;*
    - *dacă matricea este parcursă pe linii se obţine o segvenţă de log n biti;*
    - *fiecare astfel de segvenţă poate fi reprezentarea binară a unui număr între 0 şi* $2^{\log n} = n$*;*
    - *fie B o matrice de dimensiune* $\lceil\sqrt{\log n}\rceil X \lceil\sqrt{\log n}\rceil$ *obţinută prin partiţionarea lui A ca mai sus, cu* $B(i,j) = A(k_1\lceil\sqrt{\log n}\rceil + i,\ k_2\lceil\sqrt{\log n}\rceil + j)$ *şi fie s secventa de biti obtinută prin parcurgerea matricei B pe linii; atunci* $A'(k_1, k_2) = m$*, unde m este numărul a cărui reprezentare binară este s;*
    - *având în vedere că reprezentarea binară a numerelor naturale este unică, dacă* $B_1 \neq B_2$*, atunci* $s_1 \neq s_2$ *şi* $m_1 \neq m_2$*;*
- *obţinerea răspunsului la întrebarea: „A(i, j) = 1 ?": valoarea lui A(i,j) se poate obţine din matricea A' astfel:*

- *din modul de partiţionare a lui A se constată că elementul A(i,j) se obţine din procesarea elementului A'(*$\left\lceil \frac{i}{\left\lceil \sqrt{\log n} \right\rceil} \right\rceil$*,* $\left\lceil \frac{j}{\left\lceil \sqrt{\log n} \right\rceil} \right\rceil$*);*
- *reprezentarea binară a elementului A'(* $\left\lceil \frac{i}{\left\lceil \sqrt{\log n} \right\rceil} \right\rceil$ *,* $\left\lceil \frac{j}{\left\lceil \sqrt{\log n} \right\rceil} \right\rceil$ *)este succesiunea liniilor matricii B, unde B(p,q) = A(i,j), cu p = i mod* $\left\lceil \sqrt{\log n} \right\rceil$ *şi q = j mod* $\left\lceil \sqrt{\log n} \right\rceil$ *, adică bitul de pe poziţia (i mod* $\left\lceil \sqrt{\log n} \right\rceil$ *)**$\left\lceil \sqrt{\log n} \right\rceil$ *+ j mod* $\left\lceil \sqrt{\log n} \right\rceil$ *din reprezentarea binară a lui A'(*$\left\lceil \frac{i}{\left\lceil \sqrt{\log n} \right\rceil} \right\rceil$*,* $\left\lceil \frac{j}{\left\lceil \sqrt{\log n} \right\rceil} \right\rceil$*), numărând de la stânga la dreapta;*

Observaţie: Având în vedere faptul că numănul n nu se împarte întotdeauna exact la $\left\lceil \sqrt{\log n} \right\rceil$ *, este posibil ca matricele de la extremităţile dreapta, respectiv jos ale matricei A matricele obţinute prin patriţionare să nu aibă dimensiunile cerute, ci dimensiuni mai mici.*

De exemplu, o matrice de dimendiune 5 X 5 trebuie patriţionată în matrici de dimensiune 2 X 2. Coloana a cincea va fi deci împărţită în 3 matrice, două de dimensiune 2 X 1 şi una de dimensiune 1 X 1. Aceste matrice vor fi „extinse" la matrice de dimensiune 2 X 2 astfel:

Fie B_1 *o matrice de dimensiune 2 X 1 (de exemplu,* $B_1(0,0) = A(0,4)$ *şi* $B_1(1,0) = A(1,4)$ *şi* B_2 *matricea de dimensiune 2 X 2 la care va fi extinsă* B_1*; atunci* $B_2(0,0) = B_1(0,0)$*,* $B_2(1,0) = B_1(1,0)$*, iar* $B_2(0,1)$ *şi*$B_2(1,1)$ *pot avea oricare din valorile 0 şi 1.* $B_2(0,0)$ $B_2(0,1)$*,* $B_2(1,0)$*,* $B_2(1,1)$ *este reprezentarea binară a valorii elementului A'(2,0).*

Am văzut mai sus cum se face extragerea dintr-un element al lui A' a bitului care reprezintă valoarea lui A(i, j). De acolo ne dăm seama că nu se va încerca niciodată extragerea unui bit care nu corespunde unui element din matricea A, deci valorile elementelor adăugate la extinderea matricelor B nu au importanţă.

- *complexităţi:*
  - *matricea A' are* $\frac{n^2}{\lceil \log n \rceil}$ *elemente, deci complexitatea spaţiu a reprezentării sale este O(*$\frac{n^2}{\log n}$*);*
  - *operaţiile de calcul al elementului corespunzător lui A(i,j) din A' se fac prin efectuarea unor împărţiri sau calcule de resturi, operaţii ce necesită timp constant O(1);*
  - *accesul la elementele matricei A' se face în timp constant O(1);*
  - *extragerea unui bit din reprezentarea binară a unui element din A' se poate face în timp constant O(1) prin operaţii logice pe biţi (utilizându-se eventual o mască);*

  În concluzie,răspunsul la întrebarea: „A(i, j) = 1 ?" se poate obţine în timp constant (independent de n) O(1).

2. *Diametrul unui graf este lungimea maximă a unui drum de lungime minimă între două vârfuri ale grafului. Două vârfuri care sunt extremităţile unui drum minim de lungime maximă în graf se numesc diametral opuse. Demonstraţi că următorul 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 executa o parcurgere BFS a arborelui; fie v ultimul vârf vizitat;*
- *Return u, v.*

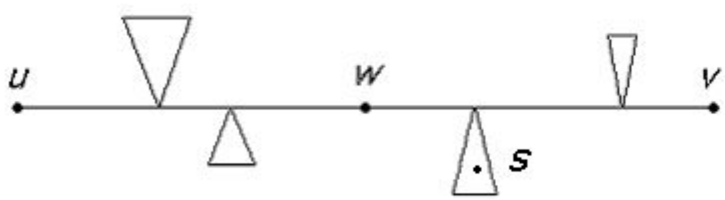
Rămâne valabil algoritmul pentru un graf conex arecare?

Soluţie*[¯]*:

Atunci când se aplică BFS unui graf se obţine un arbore numit ***arbore de lăţime*** *(care nu parcurge întotdeauna toate muchiile grafului iniţial). Prin noţiunea de arbore de lăţime întelegem un arbore cu rădăcina în nodul de start al BFS-ului şi care are proprietatea că drumul de la oricare nod la rădăcină este cel mai mic drum dintre cele două noduri, în graful iniţial. Aceasta înseamnă că algoritmul descoperă nodurile de la distanţa k faţă de rădăcină înainte să descopere nodurile de la distanţa k+1. Ultimul nod descoperit de BFS este cel mai din dreapta nod de pe frontieră. În cazul în care graful iniţial este arbore, arborele lăţime va conţine toate muchiile sale (drumul între două noduri din graful iniţial este unic deci, automat, minim).*

Primul BFS din algoritmul de mai sus, aplicat pentru un nod de start oarecare din arborele iniţial, se poziţionează pe cel mai din dreapta nod de pe frontieră, notat cu u, iar ultimul nod vizitat de al doilea BFS este cel mai depărtat nod de u, şi anume v. Trebuie să demonstrăm că distanţa de la u la v este de fapt diametrul arborelui iniţial, şi anume cel mai lung drum dintre două noduri ale arborelui.

Pentru simplificare vom considera drumul de la u la v ca fiind de lungime 2L. (cazul când acest drum are lungime impară se tratează similar). Privim arborele sub altă formă : aşezăm drumul de la u la v în linie dreaptă, ca o axă, iar ceilalţi subarbori îi "agăţăm" de nodurile de pe axă, în semiplanul superior determinat de aceasta sau în cel inferior. Considerăm w ca fiind un nod aflat la mijlocul drumului dintre u şi v, la distanţa l de cele 2 noduri.



Notăm cu s nodul ales aleator ca nod de start pentru primul BFS. Observăm că s trebuie să fie în dreapta nodului w, deoarece, dacă s ar fi în stânga, am obţine drumul de la s la v mai mare decât drumul de la s la u, ceea ce contrazice ipoteza, întrucat u era cel mai înepărtat nod de s.

Fie T un subarbore pendant dintr-un nod x aflat pe drumul de la u la v, la stânga lui w. Pentru fiecare astfel de subarbore obţinem urmatoarea relaţie:

$$adâncime(T) \le d(u,x)$$

Dacă această relaţie nu ar fi adevarată, am obţine următorul rezultat :

[¯] *BIBLIOGRAFIE: csce.unl.edu/~vimod/CSCE423/4Sol.pdf* *adâncime (T) + d(x,s) ≥ d(u,x) + d(x,s).*

Deci ar exista un nod z de pe T astfel încât

d(z,x) + d(x,s) ≥ d(u,s) => d(z,s) ≥ d(u,s)

ceea ce contrazice ipoteza că u este cel mai departat nod de s, aşa cum am obţinut din primul BFS.

Fie T' un subarbore pendant dintr-un nod y aflat pe drumul de la u la v, la dreapta lui w. Pentru fiecare astfel de subarbore obţinem următoarea relaţie:

adâncime (T') ≤ d(y,v).

Dacă această relaţie nu ar fi adevărată, am obţine următorul rezultat :

adâncime (T') + d(y,u) ≥ d(u,y) + d(y,v).

Deci ar exista z', un nod de pe T' astfel încât d(u,z) ≥ d(u,v) ceea ce contrazice ipoteza că v este cel mai depărtat nod de u, aşa cum am obţinut din al doilea BFS, aplicat pentru nodul u .

Deci pentru fiecare x si y aleşi ca mai sus, şi T respectiv T', obţinem relaţiile:

adâncime (T) ≤ d(u,x)

şi

adâncime (T') ≤ d(y,v)

din care avem

adâncime (T) + d(x,w) ≤ d(u,x) + d(x, w) = d(u,w) =L

si

adâncime (T') + d(w,y) ≤ d(w,y) + d(y,v) = d(w,v) = L.

Deci pentru oricare nod z de pe subarborele T obţinem d(z,w) ≤ L şi pentru oricare nod z' de pe subarborele T' obţinem d(z',w) ≤ L. Deci

d(z,w) + d(w, z') ≤ 2L, de unde d(z,z') ≤ 2L

oricare ar fi z si z' două noduri aflate pe doi astfel de subarbori ca cei aleşi mai sus.

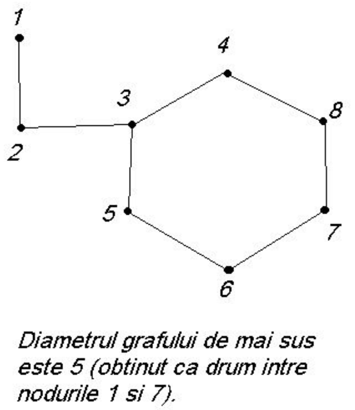
Dacă am alege 2 subarbori pendanti de aceeaşi parte a lui w am obtine acelaşi rezultat. De asemenea este evident că orice două noduri aflate pe drumul de la u la v au distanta între ele mai mică decât d(u,v).

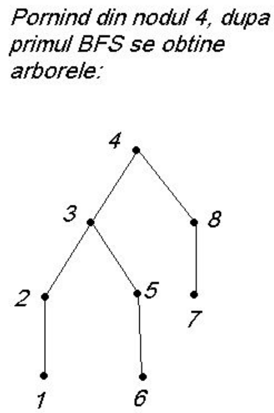
În concluzie, oricum am alege două noduri din arbore, drumul dintre ele nu poate fi mai mic decât drumul de la u la v. Deci distanţa de la u la v este diametrul arborelui initial (faptul că am impus lungimea drumului de la u la v ca fiind egala cu 2L nu restrâange rezultatul obţinut, întrucât, dacă acest drum ar avea lungimea de forma 2L-1, am alege w la distanta L de u si L-1 de v sau invers şi am urma aceiaşi paşi ca pentru situaţia anterioară).

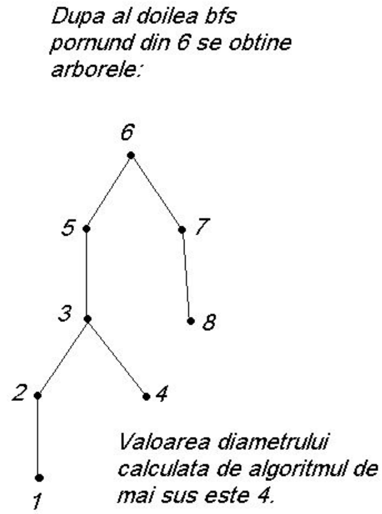
q.e.d.

*În cazul în care graful iniţial este un **graf conex oarecare**, algoritmul de mai sus **nu mai dă întotdeauna răspunsul corect**, întrucât între oricare două noduri putem avea mai mult de un drum (deci putem avea cicluri) iar arborele lăţime obţinut va păstra numai primele drumuri găsite de la rădăcina sa la celelalte noduri. Pentru oricare două noduri se va reţine numai un drum şi acesta diferă pentru fiecare arbore laţime obtinut (în funcţie de nodul de plecare).*

*Vom da un **exemplu** în care, pentru un graf oarecare, în funcţie de nodul de pornire şi de ordinea de alegere a nodurilor adiacente, obţinem drumuri distincte în arbori de lăţime diferiţi, între două noduri, iar dacă vom calcula diametrul după algoritmul de mai sus vom obţine valori diferite.*







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 grila (limiatura unui caiet de matematică); vârfurile se plaseaza îin punctele de intersecţie ale grilei.*
- *Desenăm subarborele stâng.*
- *Desenăm subarborele drept.*
- *Plasăm cele două desene unul lângă altul la distanţa pe orizontala doi şi cu rădăcinile la aceeasi înalţime.*
- *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 pot 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.

Soluţie:

Fie T un arbore cu n noduri numerotate (pentru simplificare) de la 1 la n. Pentru fiecare nod din acesta arbore vom memora valoarea sa, legătura către părinte, catre fiul stang şi catre fiul drept. Dacă unul dintre aceştia nu există, legătura va fi NIL.

Se vor mai utiliza de asemenea 4 vectori suplimentari:

- *x, unde x[i] este coordonata pe orizontală a nodului i;*
- *y, , unde y[i] este coordonata pe verticală a nodului i;*
- *lărgimeStânga, unde lărgimeStânga[i] este distanţa pe x de la i la cel mai din stânga nod al subarborelui cu rădăcina i;*
- *lărgimeDreapta, unde lărgimeDreapta [i] este distanţa pe x de la i la cel mai din dreapta nod al subarborelui cu rădăcina i;*

Un algoritm mai detaliat pentru desenarea arborelui este:

```
procedure DeseneazăArbore(T)
begin
   /*se pleacă din rădăcină*/
   p ← T
   /*dacă nodul curent este frunză*/
   if  (p→stânga = NIL and  p→dreapta =NIL )
      /*acesta este desenat iniţial în colţul din stânga jos, adică la
      coordonatele 0, 0*/
      x[p→val] ← 0
      y[p→val] ← 0
      /*largimea stânga/dreapta a unui arbore cu un singur nod este 0*/
      lărgimeStânga[p→val] ← 0
      lărgimeDreapta[p→val] ← 0
   /*dacă există fiu stânga*/
   if  (p→stânga ≠  NIL)
   then
      /*acesta este desenat*/
      DeseneazăArbore(p→stânga)
      /*dacă fiul stânga este unicul fiu*/
      if  (p→ dreapta =  NIL)
      then
          /*nodul este desenat la distanţă 1 pe orizontală de cel mai
          din dreapta nod al subarborelui cu rădăcina în fiul său stâng
          (în dreapta acestuia)*/
          x[p→val] ← x[p→stânga→val]+lărgimeDreapta[p→stânga→val]+1
          /*pe nivelul imediat superior*/
          y[p→val] ← y[p→stânga →val] + 1
          /*lărgime stânga este lărgimea totala a subarborelui stâng +
          1*/
          lărgimeStânga[p→val] ← lărgimeDreapta[p→stânga →val] +
                               lărgimeStânga[p→stânga →val] + 1
          /*neexistând subarbore drept, lărgimea dreapta este 0*/
          lărgimeDreapta[p→val] ← 0
          return
   if  (p→ dreapta ≠  NIL)
   /*dacă există fiu dreapta*/
   then
      /*acesta este desenat*/
      DeseneazăArbore(p→ dreapta)
      /*dacă fiul dreapta este unicul fiu*/
      if  (p→ stânga =  NIL)
      then
          /*nodul este desenat la distanţă 1 pe orizontală de cel mai
          din stânga nod al subarborelui cu rădăcina în fiul său
          drept )în stânga acestuia)*/
          x[p→val] ←x[p→dreapta→val]-lărgimeStânga[p→dreapta→val]-1
          /*dacă se obţin coordonate negative se translează întreg
          subarborele astfel încât coordonata minimă să fie 0*/
          if (x[p→val] < 0)
          then
                TranslOrizontala(p, -x[p→val])
          y[p→val] ← y[p→stânga →val] + 1
          /*lărgime dreapta este lărgimea totala a subarborelui drept +
          1*/
          lărgimeDreapta [p→val] ← lărgimeDreapta[p→stânga →val] +
                               lărgimeStânga[p→stânga →val] + 1
```

```
        /*neexistând subarbore stâng, lărgimea stânga este 0*/
        lărgimeStânga [p→val] ← 0
        return

   /*se aduc rădăcinile pe acelaşi nivel*/
   translY = y[p→ stânga→val] - y[p→ dreapta→val]
   /*subarborele cu adâncimea mai mica este „ridicat” la nivelul
   celuilalt subarbore*/
   if (translY < 0)
   then
      TranslVerticală(p→stânga, -translY)
      else
            TranslVerticală(p→dreapta, translY)
   /*rădăcina se plasează pe nivelul superior*/
   y[p→val] ← y[p→ dreapta→val] + 1

   /*subarborii trebuie aşezaţi la distanţa 2 pe orizontală, deci
   rădăcinile lor trebuie plasate la distanţa
   lărgimeDreapta[p→stânga→val] + 2 + lărgimeStânga[p→ dreapta→val]
   una de cealaltă*/
   translX = x[p→ stânga→val] - x[p→ dreapta→val] +
           lărgimeDreapta[p→stânga→val] + 2 + lărgimeStânga[p→
           dreapta→val]
   TranslOrizontala(p→ dreapta, translX)
   x[p→val] ← (x[p→ dreapta→val] + x[p→ stanga→val]) / 2
   /*lărgime dreapta este lărgimea totala a subarborelui drept + 1*/
   lărgimeDreapta [p→val] ← lărgimeDreapta[p→stânga →val] +
                            lărgimeStânga[p→stânga →val] + 1
   /*lărgime stânga este lărgimea totala a subarborelui stâng + 1*/
   lărgimeStânga[p→val] ← lărgimeDreapta[p→stânga →val] +
                           lărgimeStânga[p→stânga →val] + 1
end

procedure TranslOrizontală(T, translX)
begin
   if (T≠ NIL)
   then
      /*translare rădăcină*/
      x[T→val] ← x[T→val] + translX
      /*translare fii*/
      TranslOrizontală(T→ stânga, translX)
      TranslOrizontală(T→ dreapta, translX)
end

procedure TranslVerticală(T, translY)
begin
   if (T≠ NIL)
   then
      /*translare rădăcină*/
      y[T→val] ← y[T→val] + translY
      /*translare fii*/
      TranslVerticală (T→ stânga, translY)
      TranslVerticală(T→ dreapta, translY)
end
```

Funcţionarea algoritmului:

Soluţia prezentată mai sus este un algoritm Divide-et-Impera: pentru un arbore se desenează întâi subarborele stâng, apoi subarborele drept, după care se asamblează cele două desene.

Pasul de bază:

- *fiecărei frunze i se vor atribui iniţial coordonatele (0, 0);*
- *lărgimea unui arbore cu un singur nod este evident 0.*

Dacă nodul v are doar fiul stâng, fie acesta w, el va fi desenat astfel:

- *este desenat fiul stâng w;*
- *v trebuie desenat pe nivelul imediat superior, deci y[v] =y[w] + 1;*
- *v trebuie desenat în dreapta fiului său, la distanţă 1 pe orizontală faţă de acesta, adică la la distanţă 1 pe orizontală faţă de nodul cel mai din dreapta (dpdv al structurii desenului) din subarborele cu rădăcina w; după cum am văzut mai sus, distanţa dintre acesta din urmă şi w este lărgimeDreapta[w], deci distanţa dintre v şi w va fi lărgimeDreapta[w]+1, iar x[v] = x[w] + lărgimeDreapta[w]+1;*
- *lărgimeDreapta[v] = 0, deoarece v nu are fiu drept, şi din construcţie se observă că v este cel mai din dreapta nod (dpdv al structurii desenului) din subarborele a cărui rădăcină este.*
- *, nodul cel mai din stânga al subarborelui cu rădăcina v, care este şi nodul cel mai din stânga al subarborelui cu rădăcina w se găseşte la distanţa lărgimeStânga[w] de w, aflându-se în stânga acestuia, iar w se află la lărgimeDreapta[w] + 1 în stânga lui v.*

Dacă nodul v are doar fiul drept, fie acesta w, el va fi desenat astfel:

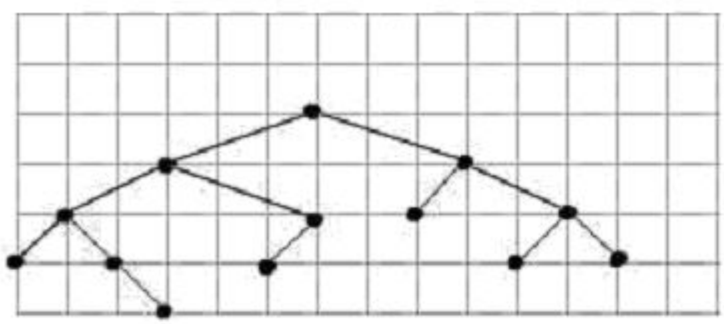
- *este desenat fiul drept w;*
- *v trebuie desenat pe nivelul imediat superior, deci y[v] =y[w] + 1;*
- *v trebuie desenat în stânga fiului său, la distanţă 1 pe orizontală faţă de acesta, adică la la distanţă 1 pe orizontală faţă de nodul cel mai din stânga din subarborele cu rădăcina w; după cum am văzut mai sus, distanţa dintre acesta din urmă şi w este lărgimeStânga[w], deci distanţa dintre v şi w va fi lărgimeStânga [w]+1, iar x[v] = x[w] - lărgimeStânga [w]-1.*
- *în cazul în care se admit doar coordonate numere naturale iar coordonata x[v] obţinută este negativă, întreg arborele va fi translat pe orizontală cu /x[v] /, astfel încât toate coordonatele să rămână numere naturale; prin urmare, în cazul de faţă, x [v]devine 0;*
- *lărgimeStânga [v] = 0, deoarece v nu are fiu stâng, şi din construcţie se observă că v este cel mai din stânga nod (dpdv al structurii desenului) din subarborele a cărui rădăcină este.*
- *lărgimeDreapta[v] = lărgimeDreapta[w] + lărgimeStânga[w] + 1, nodul cel mai din dreapta al subarborelui cu rădăcina v, care este şi nodul cel mai din dreapta al subarborelui cu rădăcina w se găseşte la distanţa lărgimeDreapta[w] de w, aflându-se în dreapta acestuia, iar w se află la lărgimeStânga[w] + 1 în dreapta lui v.*

Dacă nodul v are şi fiu stâng u, şi fiu drept w, el va fi desenat astfel:

- *este desenat fiul stâng;*
- *este desenat fiul drept;*

- *dacă y[u] $\neq$ y[v], atunci unul dintre cei dou subarbori trebuie translat pe verticală până când rădăcina lui are acelaşi nivel cu rădăcina celuilalt. Cum am văyut mai sus, întotdeauna frunzele de pe ultimul nivel al unui subarbore „terminat” au y = 0. Neputănd avea coordonate cu valori negative, vom transla subarborele „mai scund” pe verticală pentru a ajunge la nivelul celui mai înalt. Înalţimea unui subarbore terminat este chiar y[r], unde r este rădăcina subarborelui;*
- *subarborii aduşi la acelaşi nivel trebuie aşezaţi la distanţă 2 unul de celălalt. Întâi vom suprapune rădăcinile, adica translăm subarborele drept cu x[u] – x[w]. Distanţa dintre rădăcini trebuie să fie distanţa dintre u şi cel mai din dreapta nod al subarborelui stâng plus 2 plus distanţa dintre u şi cel mai din stânga nod al subarborelui drept, adică vom transla subarborele drept cu lărgimeDreaptau] + 2 + lărgimeStânga[w].*
- *v trebuie plasat pe nivelul imediat superior lui u, respectiv w, adica y[v] = y[u] + 1 = y[w] + 1;*
- *v trebuie plasat la jumătatea distanţei pe orizontală dintre u şi w, adică x[v] = (x[u] + x[w]) /2; dacă x[u] + x[w] este impar, se va atribui lui x[v] partea întreagă a rezultatului împărţirii, deoarece sunt acceptate doar coordonate cu valori numere naturale (nodurile pot fi plasate doar în punctele de intersecţie ale grilei;*
- *lărgimeDreapta[v] = lărgimeDreapta[w] + lărgimeStânga[w] + 1 şi lărgimeStânga[v] = lărgimeDreapta[u] + lărgimeStânga[u] + 1 (explicaţii similare cu cazurile anterioare).*

Exemplu:



4. *Într-o sesiune de examene s-au înscris n studenţi 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 sa 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 numarul minim de zile necesare organizării tutror examenelor?*

Soluţie:

Încercarea de a memora într-un graf atât informaţii despre examene cât şi despre toţi studenţii a dus la concluzia că nu se justifică utilizarea unui spaţiu atât de mare şi nici nu au fost găsiţi algoritmi eficienti în această formulă pentru satisfacerea cerinţelor.

Prin urmare, am ajuns la ideea să construim un graf în care reţinem explicit doar informaţii despre disciplinele de examen şi relaţiile dintre acestea (relaţii ce sunt definite mai jos).

*Se construieşte deci graful **G** cu **m** noduri astfel:*

- *fiecare nod **i** corespunde unei discipline de examen;*
- *între nodurile **i** şi **j** există muchie dacă şi numai dacă există un student care doreşte să susţină atât examenul i cât şi examenul j.*

Se observă că, dacă două noduri sunt vecine, atunci examenele corespunzătoare nu pot fi date în aceeaşi zi.

***Obţinerea** grafului G definit mai sus din datele de intrare: se face o parcurgere a listei de studenţi şi pentru fiecare student se parcurge lista de examene pe care doreşte să le susţină. Oricare două examene din această listă sunt unite printr-o muchie (evident, dacă această muchie nu există deja).*

*În continuare, intuitiv, pentru a găsi o posibilă aranjare pe zile a examenelor este nevoie să le împărţim în mulţimi în aşa fel încât între oricare două elemente dintr-o mulţime să nu existe muchie în graful G (acest lucru ar însemna că nu există un elev care doreşte să susţină ambele examene). Numărul de mulţimi reprezintă numărul de zile în care au fost programate examenele, iar numărul de elemente ale fiecărei mulţimi este numărul de examene programate într-o anumită zi. Aceste mulţimi trebuie să formeze o **partiţie** a mulţimii de examene (deci să fie disjuncte şi reunite să dea mulţimea iniţială de examene). În general există mai multe posibile aranjări ale examenelor .*

*Observăm că, respectând ipoteza (un student poate da cel mult un examen pe zi) şi având în vedere construcţia grafului G, **a împărţi pe zile examenele este echivalent cu a colora graful G** definit mai sus, iar **a găsi un grup de examene ce pot fi susţinute în aceeaşi zi este echivalent cu a găsi în graful G un sistem de puncte independente.***

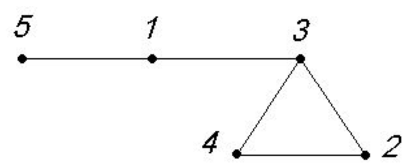
*Pentru a afla aranjarea optimă avem de fapt nevoie de numărul minim de astfel de mulţimi, adică numărul minim de zile şi de maximul dintre cardinalele lor, adică de numărul maxim de examene ce pot fi programate într-o zi. Numărul minim de mulţimi reprezintă numărul minim de culori cu care poate fi colorat graful. Aşadar**, numărul cromatic** $\boldsymbol{\chi(G)}$ este numărul minim de zile în care se pot desfăşura examenele. Numărul maxim de examene ce pot fi programate într-o zi reprezintă maximul dintre cardinalele mulţimilor stabile şi prin urmare, este **numărul de stabilitate** al lui G,* $\boldsymbol{\alpha(G)}$*.*

Exemplu :

Lista de elevi şi disciplinele la care doreşte fiecare elev să susţină examen:

***elev 1**: d1, d3; **elev 2**: d4; **elev 3**: d3, d4; **elev 4**: d2, d3; **elev 5**: d2, d4; **elev 6**: d1, d5.*

Construim graful după metoda de mai sus.



Observăm că numărul cromatic $\boldsymbol{\chi(G)}=3$ *şi numărul de stabilitate al lui G,* $\boldsymbol{\alpha(G)}=2$*. Deci avem minim 3 zile şi maxim 2 examene pe zi. O posibilă aranjare a examenelor ar fi:*

***Ziua 1**: d3, d5;*
***Ziua 2**: d1, d2*
***Ziua 3**: d4.*

GÎRDEA MARTA
URICARU RALUCA
A34
(grupa 4A)

TEMA NR. 2
11 martie 2003

1. *Fie G = (S, T; E) un graf bipartit şi $X \in \{S, T\}$.*

Graful G se numeşte X–lanţ dacă vârfurile mulţimii X pot fi ordonate $X = \{x_1, x_2,..,x_p\}$ (unde $p = |X|$) astfel încât $N_G(x_1) \supseteq N_G(x_2) \supseteq .. \supseteq N_G(x_p)$.

a) Demonstraţi că G este X – lanţ dacă şi numai dacă este $\overline{X}$ – lanţ, unde $\overline{X} = \{S, T\} - \{X\}$.

b) Dacă G (bipartit) este reprezentat cu ajutorul listelor de adiacenţă, are ordinul n şi dimensiunea m descrieţi un algoritm cu timpul O (n + m) care să testeze dacă G este S–lanţ.

Soluţie:

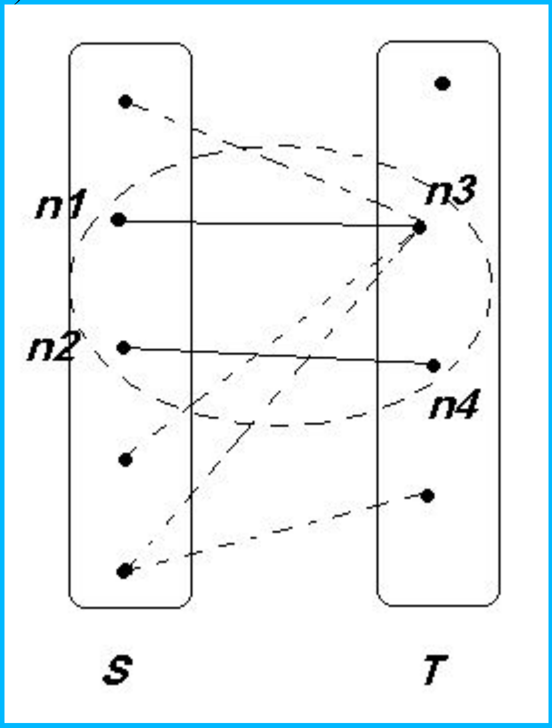
a) *Vom demonstra echivalenţa celor două relaţii demonstrând că cele două relaţii sunt echivalente cu o a treia, şi anume că **G este X-lanţ dacă şi numai dacă G este $2K_2$-free**(unde X este o mulţime dintre S şi T).*

***Implicaţia directă**:*

Dacă G este X- lanţ, atunci este K_2-free.

Reducere la absurd:

Presupunem că G nu este $2K_2$-free. Atunci $2K_2$ este subgraf indus al grafului G. (În schema de mai jos, de observă că graful G' cu nodurile n1, n2, n3 şi n4 este subgraf indus în G).



Aşadar, nu există muchie de la n2 la n3 sau de la n1 la n4 în graful G. Deci n3 nu este inclus în mulţimea vecinilor lui n2 şi n4 nu este inclus în mulţimea vecinilor lui n1 (analog n1 nu este inclus în mulţimea vecinilor lui n4 şi n2 nu este inclus în mulţimea vecinilor lui n3) . Deci:

$$N_G(n1) \not\subset N_G(n2) \text{ şi } N_G(n2) \not\subset N_G(n1)$$

respectiv

$$N_G(n3) \not\subset N_G(n4) \text{ şi } N_G(n4) \not\subset N_G(n3).$$

Prin urmare, dacă $2K_2$ este subgraf indus în G, există două noduri i şi j din X pentru care $N_G(i)$ şi $N_G(j)$ sunt ***incomparabile după relaţia de incluziune*** *(indiferent dacă X este S sau T). În consecinţă, G nu este X-lanţ, ceea ce contrazice ipoteza. Înseamnă că presupunerea de la care am plecat este falsă,* ***deci G este $2K_2$-free.***

***Implicaţia inversă**:*

Dacă G este $2K_2$-free, atunci G este X-lanţ (unde X este S sau T).

Din definiţie rezultă că G este X-lanţ dacă şi numai dacă relaţia de incluziune din enunţ este relaţie de ***ordine totală*** *pe mulţimile de adiacenţă ale nodurilor din X, cu X $\in$ {S, T} (adică oricare două mulţimi sunt comparabile) .*

Faptul că G este X-lanţ este deci echivalent cu faptul că relaţia de incluziune este relaţie de ordine totală peste mulţimea $\{N_G(i) \mid i \in X\}$.

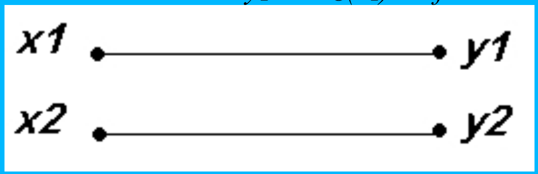
Reducere la absurd:

Presupunem că relaţia de incluziune nu este relaţie de ordine totală peste mulţimea $\{N_G(i) \mid i \in X\}$, adică există două noduri x_1 şi x_2 din X pentru care $N_G(x_1)$ şi $N_G(x_2)$ nu sunt comparabile, adică

$$\exists y_1 \in N_G(x_1) \text{ astfel încât } y_1 \notin N_G(x_2)$$

şi

$$\exists y_2 \in N_G(x_2) \text{ astfel încât } y_2 \notin N_G(x_1).$$



Graful de mai sus este un ***subgraf indus*** *în G (deoarece, conform celor menţionate mai sus, nu există în G muchiile (x_1, y_2) respectiv (x_2, y_1); de asemenea, G fiind bipartit, nu existî muchiile (x_1, x_2) respectiv (x_2, y_1)). Totodată, este un graf $2K_2$. Cum în ipoteză se specifica că graful este $2K_2$-free, înseamnă că am ajuns la un rezultat care contrazice ipoteza. Deci presupunerea iniţială este falsă şi putem spune că avem ordine totală între mulţimile de adiacenţă.*

Conform implicaţiei directe şi cele inverse, am obţinut că un graf este X-lanţ dacă şi numai dacă este $2K_2$-free, unde X este oricare dintre cele două mulţimi S şi T.

Dacă avem G S-lanţ, obţinem conform rezultatului de mai sus că G este $2K_2$-free şi conform aceluiaşi rezultat avem G T-lanţ. Dacă pornim cu G T-lanţ obţinem la fel ca mai sus că G S-lanţ. Deci am demonstrat că G este S-lanţ dacă şi numai dacă este T-lanţ.

q.e.d.

b) *Vom nota p_1=|S| şi p_2=|T| astfel încât $p_1 + p_2 = n$. Mulţimea S este formată din elemente x_i, unde i este un număr între 1 şi p_1 iar mulţimea T, din elemente x_j, unde j este un număr între p_1+1 şi n.*

I. *Iniţial, vom parcurge lista nodurilor din S (în cazul nostru, primele p_1 noduri din listă) şi pentru fiecare nod numărăm nodurile aflate în lista sa de adiacenţă.*

Vom reţine rezultatele într-un vector V de dimensiune p_1 în următoarea formă: fiecare element al lui V este o structură formată din două câmpuri: unul în care reţinem ***indicele*** *nodului şi altul care memorează* ***lungimea listei de adiacenţă*** *a nodului respectiv..*

Această etapă are loc în ***O(m),*** *întrucât se parcurg toate muchiile grafului G o singură dată (avem muchii numai între nodurile din S şi nodurile din T, nu şi între nodurile aceleiaşi mulţimi, deci nu avem posibilitatea de a găasi alte muchii în partea cealaltă a listei de noduri).*

II. *Vom ordona vectorul V după valoarea celui de-al doilea câmp al fiecărui element, adică după lungimea listei de adiacenţă. Întrucât ne aflăm într-un caz particular (aceste valori sunt întregi, cuprinse între 0 şi p_2), putem folosi* ***sortarea prin numărare****, de complexitate* ***$O(p_1+ p_2) = O(n)$****.*

```
procedure SortarePrinNumărare(V)
begin
    for i←0 to p2-1 do
        C[i]←0
        /* folosim vectorul C pentru stocarea temporară a datelor */
        /*C are dimensiunea egală cu numărul de noduri din T, adică p2*/
    for i←0 to p1 -1do
        C[V[i].dim_lista]←C[V[i].dim_lista]+1
        /*pentru fiecare element din V, dacă V[i].dim_lista = j, atunci C[j] este
        incrementat; la acest pas se marchează care elemente din C se găsesc în
        V şi de câte ori*/
    for i←1 to p1-1 do
        C[i]←C[i]+C[i-1]
        /*în C[i] acum memorăm câte elemente mai mici sau egale cu i se găsesc
        în V, deci pe ce poziţie trebuie să se găsească i în vectorul final*/
    for i← p1-1 downto 0 do
        /*elementul i din V se aşează în B pe poziţia indicată în vectorul C*/
        B[C[V[i]. dim_lista] – 1] ← V[i]
        /*o valoare deja procesată nu va mai fi luată în considerare:*/
        C[V[i]. dim_lista] ← C[V[i]. dim_lista]-1
end
```

În B am obţinut rezultatul final, şi anume, nodurile din S ordonate crescător după numărul nodurilor din listele lor de adiacenţă. Totodată, întrucât ***ordonarea prin numărare este stabilă****, dacă două noduri au cardinalele mulţimilor de adiacenţă egale, atunci se află în vectorul B în ordinea în care se aflau în mulţimea S.*

III. *În continuare, trebuie să verificăm dacă mulţimea de adiacenţă a nodului de pe poziţia i din B este inclusă în mulţimea de adiacenţă a nodului de pe poziţia i+1, cu i de la 0 la p_1-1. Această verificare se face într-o structură repetitivă în cadrul căreia avem un fel de interclasare între listele de adiacenţă ale nodurilor de pe poziţiile B[i] şi B[i+1] (se presupune că în listele de adiacenţă nodurile au fost introduse* ***în ordine crescătoare*** *).*

Algoritmul de verificare este:

```
function VerificareIncluziune(G, B)
begin
    for i←0 to p1 - 2 do
        L1 ← m[B[i]]
        L2 ← m[B[i+1]]
    /*m este vectorul de pointeri la listele de adiacenţă corespunzatoare nodurilor */
    /*cât timp nu am epuizat încă lista “mai mică”*/
        while L1≠ NULL do
        /*verificăm dacă nodurile din L2 se găsesc în L1;
        am pus condiţia ca nodurile să fie în lista de adiacenţă în ordinea
        crescătoare a indicelui*/
            while  L1→indice > L2→indice  do
                L2 ← L2→leg
            if L1→ indice = L2→ indice  then
                L1 ← L1→leg
                L2 ← L2→leg
            else
                ” G nu este S-lanţ”
                return 0
    return 1
end
```

*De fiecare dată, în cazul cel mai nefavorabil, în cadrul **for**-ului se face o parcurgere a listei de adiacenţă a nodului de pe poziţia B[i] (în timp **$O(N_G(B[i]))$**) şi a listei de adiacenţă a nodului de pe poziţia B[i+1] (în timp **$O(N_G(B[i+1]))$**). Mai exact, în cazul cel mai nefavorabil, lista de adiacenţă a nodului din B[0] este parcursă o dată, listele de adiacenţă ale nodurilor din B[i], cu i de la 1 la p-2 sunt parcurse de două ori, iar lista de adiacenţă a nodului din B[p-1] este parcursă o dată. Ştim că **lungimea totală a listelor de adiacenţă într-un graf este de ordinul numărului de muchii**.*

În total avem:

$$O(N_G(B[0]))+ O(N_G(B[1])) + O(N_G(B[1]))+ \ldots + O(N_G(B[p_1-2]))+ O(N_G(B[p_1-1])) =$$
$$O(N_G(B[0])) + 2\,(O(N_G(B[1]))+\ldots+ O(N_G(B[p_1-2]))) + O(N_G(B[p_1-1])) < \mathbf{O(2m)}.$$

*Deci putem spune că această parte a algoritmului este de complexitate **O(m)**.*

Aşadar:

*- determinarea lungimii listelor de adiacenţă se execută în **O(m)**;*

*- ordonarea mulţimilor de adiacenţă în ordinea crescătoare a cardinalelor lor are complexitatea O**(n)** ;*

*- verificarea relaţiei de incluziune între oricare două mulţimi $N_G(i)$, $N_G(j)$, pentru care i şi j sunt vecine în vectorul B are complexitatea **O(m)**.*

*Complexitatea algoritmului este în consecinţă O(n+ 2m) = **O(n + m)**.*

2. *Un graf G se numeşte* ***autocomplementar*** *dacă este izomorf cu complementul său:* $G \simeq \overline{G}$.

a)Demonstraţi că un graf autocomplementar este conex şi că ordinul său este multiplu de 4 sau multiplu de 4 plus 1.

b)Demonstraţi că pentru orice graf G există un graf autocomplementar H astfel încât G este subgraf indus în H.

c)Determinaţi toate grafurile autocomplementare cu cel mult 7 vârfuri.

Soluţie:

a)Ordinul lui G este multiplu de 4 sau multiplu de 4 plus 1:

Fie $G: (V(G), E(G))$ *şi* $\overline{G}: (V(\overline{G}), E(\overline{G}))$. *Evident,* $V(G) = V(\overline{G}) = V$.

G şi $\overline{G}$ *sunt izomorfe, deci, prin definiţie, există o funcţie* $\Psi : |E(G)| \to |E(\overline{G})|$ *bijectivă; aşadar numărul de muchii din G este egal cu numărul de muchii din* $\overline{G}$, *adică* $|E(G)| = |E(\overline{G})|$.

Din definiţia lui $\overline{G}$,

$$E(\overline{G}) = P_2(V) - E(G),\quad P_2(V) = \frac{n(n-1)}{2},$$

deci

$$|E(G)| = |E(\overline{G})| = \tfrac{1}{2}\,|P_2(V)| = \frac{n(n-1)}{4},$$

unde n este ordinul lui G.

Cardinalul unei mulţimi este număr natural, deci este necesar ca n(n-1) să fie divizibil cu 4. Cum n şi n-1 sunt numere consecutive şi deci nu pot fi ambele pare, rezultă că:

- *n divizibil cu 4, deci n are forma* ***4k***, $k \in$ ***N**** *(mulţimea vârfurilor unui graf este prin definiţie nevidă, deci nu vom lua în considerare cazul k = 0)*
  sau
- *n-1 divizibil cu 4, deci n are forma* ***4k+1***, $k \in$ ***N***.

G este conex:

Reducere la absurd:

Presupunem că există $G_1: (V(G_1), E(G_1))$ *şi* $G_2: (V(G_2), E(G_2))$ *două subgrafuri ale lui G, cu* $V(G_1) \cap V(G_2) = \Phi$, *şi cu proprietatea că G este reuniunea disjunctă a celor două grafuri. Evident, un astfel de graf nu este conex.*

Presupunem că ordinul lui G_1 *este* ***t*** *iar ordinul lui* G_2 *este* ***n – t*** *(unde n este ordinul lui G). Presupunem* $t \geq n - t$.

Atunci, în graful complementar $\overline{G}$, *fiecare nod din* G_1 *va fi unit printr-o muchie cu fiecare nod din* G_2. *Deci vor exista t noduri cu gradul cel puţin n - t şi n - t noduri cu gradul cel puţin t.*

$G \simeq \overline{G}$ *deci există o funcţie* $\varphi: V(G) \to V(\overline{G})$ *bijectivă astfel încât funcţia* $\Psi : |E(G)| \to |E(\overline{G})|$ *definită prin :* $\forall e = uv \in E(G)$, $\Psi(uv) = \varphi(u)\,\varphi(v)$ *este bine definită şi bijectivă.*

Aşadar, dacă în G există k muchii incidente cu nodul ***u***, *atunci în* $\overline{G}$ *există exact k muchii incidente cu* $\boldsymbol{\varphi(u)}$.

Prin urmare, Ψ fiind bijectivă, este necesar ca şi în G să existe t noduri distincte cu gradul cel puţin n - t şi n - t noduri cu gradul cel puţin t. Dar cel mai mare grad posibil pentru un nod în graful G este t -1, deoarece:

- *în graful G nu există muchii între nodurile din G_1 şi cele din G_2;*
- *am convenit că $t \geq n - t$, cu $t = |G_1|$ şi $n - t = |G_2|$;*
- *pentru un nod dintr-un graf (nu multigraf) cu k noduri pot exista cel mult $k - 1$ muchii incidente, deoarece acesta poate avea cel mult $k - 1$ vecini;*

Dar $t - 1 < t$, deci nu există în G noduri cu gradul cel puţin t. Aşadar, G şi $\overline{G}$ ***nu pot fi izomorfe.***

În concluzie, orice graf autocomplementar este în mod necesar ***conex****.*

b)*Fie G un graf cu $|V(G)| = n$ şi $|E(G)| = m$.*

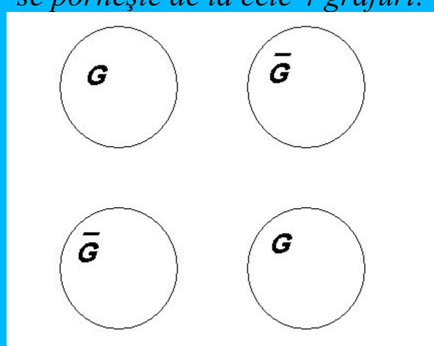
Vom arăta că putem construi întotdeauna un graf H astfel încât G să fie subgraf indus în H şi H să fie autocomplementar.

De la a) rezultă că orice graf autocomplementar are multiplu de 4 sau multiplu de 4 plus 1 noduri. Vom construi un graf H care conţine N = 4n noduri astfel:

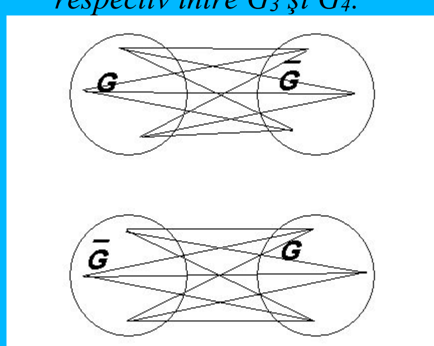
-fie $G_1 = G$, $G_2 = \overline{G}$, $G_3 = \overline{G}$, $G_4 = G$;
-$H_1 = (G_1 + G_2)$ Ů $(G_3 + G_4)$;
-$H_2 = G_1$ Ů $(G_2 + G_3)$ Ů G_4;
-$H = H_1 \cup H_2$.

Paşii construcţiei grafului H:

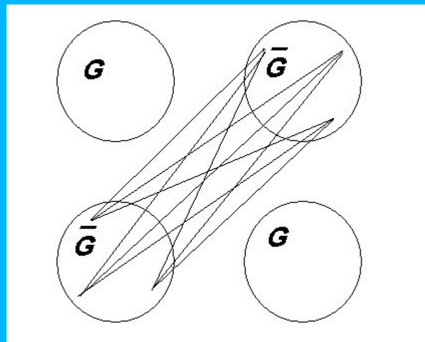
-se porneşte de la cele 4 grafuri:



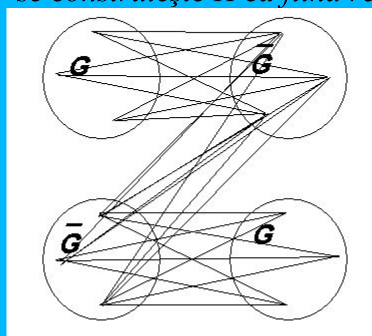
-se construieşte H_1 adăugându-se toate muchiile posibile între G_1 şi G_2, respectiv între G_3 şi G_4:



-se construieşte H_2 adăugându-se toate muchiile posibile între G_2 şi G_3:



-se construieşte H ca fiind reuniunea dintre H_1 şi H_2:



Vom demonstra acum că H respectă condiţiile din cerinţă.

*Din construcţie se observă că într-adevăr **G este subgraf indus în H**, deoarece nu s-au adăugat muchii în interiorul grafurilor G_1, G_2, G_3, G_4, ci numai între aceste grafuri, iar G_1 şi G_4 sunt copii izomorfe ale lui G.*

***Numărul M de muchii ale lui H** este:*

$$M = |E(G_1)| + |E(G_2)| + |E(G_3)| + |E(G_4)| + \sum_{i=1}^{3} \left| \{\{u,v\} \in E(H) \mid u \in V(G_i), v \in V(G_{i+1})\} \right|$$

adică M este suma dintre numărul de muchii al grafului G_i, cu i de la 1 la 4, la care se adună muchiile adăugate ulterior între aceste grafuri.

Ştim că:

$$|E(G_1)| = |E(G_4)| = |E(G)| = m$$

$$|E(G_2)| = |E(G_3)| = |E(\overline{G})| = \frac{n(n-1)}{2} - m$$

Notăm numărul de muchii adăugate între G_i şi G_{i+1} cu $M_{i,i+1}$. Atunci:

$$M_{1,2} = M_{2,3} = M_{3,4} = |V(G)|^2 = n^2$$

Deci:

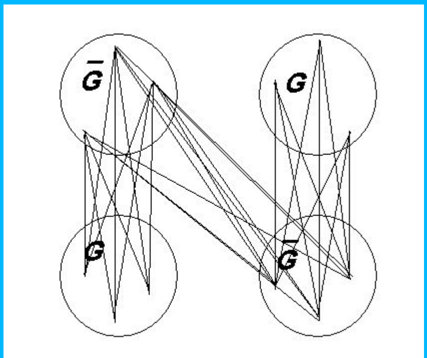
$$M = m + \left(\frac{n(n-1)}{2} - m\right) + m + \left(\frac{n(n-1)}{2} - m\right) + n^2 + n^2 + n^2 = 4n^2 - n = n(4n-1)$$

$$M = \frac{4n(4n-1)}{4} = \frac{N(N-1)}{4}$$

*Noul graf H este **conex**, indiferent dacă G este sau nu conex, fapt care reiese de asemenea din construcţie.*

Aşadar, graful H indeplineşte două condiţii necesare (dar nu şi suficiente) pentru autocomplementaritate.

Pentru a demonstra că H este într-adevăr autocomplementar, com construi complementarul său $\overline{H}$:



$\overline{H}$ se construieşte astfel:

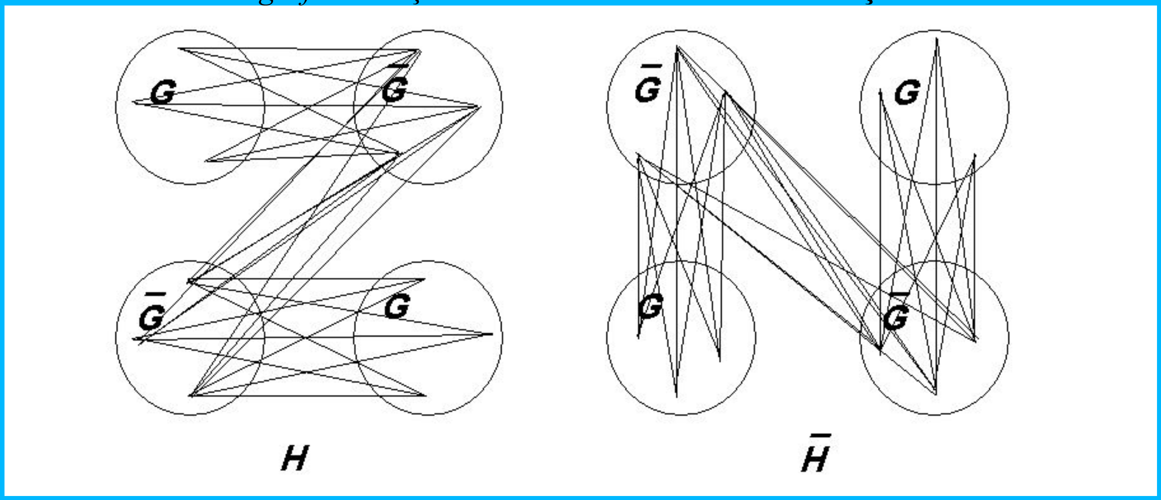
-$V(\overline{H}) = V(H)$

-$E(\overline{H}) = P_2(V(H)) – E(H)$.

Înlocuind muchiile din G_1, respectiv din G_4, cu toate celelalte muchii posibile între nodurile lui G_1, respectiv ale lui G_4 se obţine $\overline{G_1}$, respectiv $\overline{G_4}$, adică două copii izomorfe ale lui $\overline{G}$. Analog, înlocuind muchiile din G_2, respectiv din G_3, cu toate celelalte muchii posibile între nodurile lui G_2, respectiv ale lui G_3 se obţine $\overline{G_2}$, respectiv $\overline{G_3}$, adică două copii izomorfe ale lui G.

Între nodurile lui G_1 şi cele ale lui G_2 nu vom avea nici o muchie, deoarece toate muchiile posibile între aceste noduri apar în E(H). Analog pentru G_2 şi G_3, respectiv pentru G_3 şi G_4. Vor apărea în schimb în $E(\overline{H})$ toate muchiile posibile dintre nodurile subgrafurilor $\overline{G_1}$ şi $\overline{G_4}$, $\overline{G_1}$ şi $\overline{G_3}$, respectiv $\overline{G_2}$ şi $\overline{G_4}$.

Studiind subgrafurile H şi $\overline{H}$ vom constata că ele ***au aceeaşi structură****:*



deci sunt ***izomorfe****. Ca o observaţie, în schema de mai sus, $\overline{H}$ este o rotire cu 90 de grade a lui H.*

Izomorfismul se poate constata şi din studiul ***matricelor de adiacenţă*** *ale grafurilor H şi $\overline{H}$.*

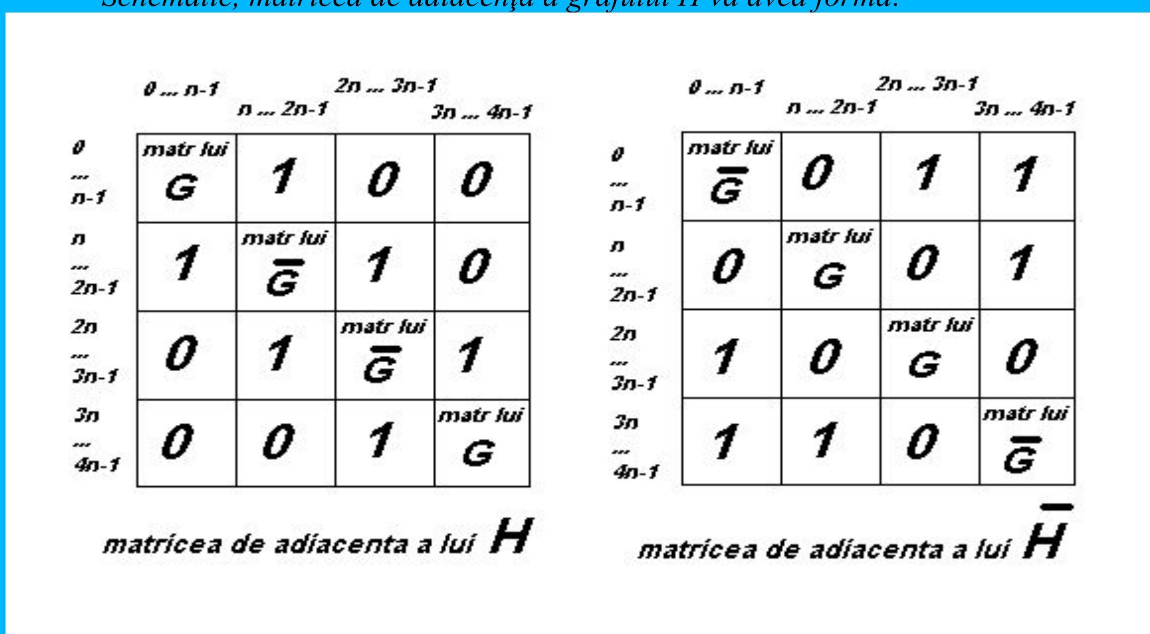
Vom prezenta mai jos ***algoritmul*** *de construcţie a matricei de adiacenţă a grafului H.*

```
procedure ConstruieşteGrafAutocomplementar(G)
begin
        /*dacă graful G are n vârfuri, graful H va avea 4n vârfuri*/
        n ← |V(G)|
        V(H) = {0, 1, …, 4n - 1}
        /*fie AG matricea de adiacenţă a grafului G şi AH matricea de adiacenţă a
        grafului H*/

        /*în V(H), nodurile de la 0 la n-1 corespund nodurilor din G1 (= G)*/
        /*nodurile de la n  la 2n-1 corespund nodurilor din G2 (= G)*/
        /*nodurile de la 2n  la 3n-1 corespund nodurilor din G3 (= G)*/
        /*nodurile de la 3n  la 4n-1 corespund nodurilor din G4 (= G)*/

        /*prin urmare se va copia matricea de adiacenţă a grafului G pe porţiunea
        corespunzătoare intersecţiei coloanelor 0, ... , n-1 cu liniile 0, ..., n-1, respectiv
        pe porţiunea corespunzătoare intersecţiei coloanelor 3n, ... , 4n-1 cu liniile
        3n, ..., 4n-1*/
        for (i ← 0 to n-1) do
                for (j ← 0 to n-1) do
                        AH(i, j) ← AG(i, j)
        for (i ← 3n to  4n-1) do
                for (j ← 3n  to 4n-1) do
                        AH(i, j) ← AG(i mod n, j mod n)
        /*analog, se va copia matricea de adiacenţă a grafului G̅ pe porţiunea
        corespunzătoare intersecţiei coloanelor n, ... , 2n-1 cu liniile n, ..., 2n-1,
        respectiv pe porţiunea corespunzătoare intersecţiei coloanelor 2n, ... , 3n-1 cu
        liniile 2n, ..., 3n-1*/
        /*matricea de adiacenţă a grafului G̅ se obţine din matricea lui G înlocuindu-se
        peste tot 0 cu 1 şi 1 cu 0 (se înlocuiesc muchiile din G cu toate celelalte muchii
        din Kn)*/
        for (i ← n to 2n-1) do
                for (j ← n to 2n-1) do
                        AH(i, j) ←1 - AG(i mod n, j mod n)
        for (i ← 2n to  3n-1) do
                for (j ← 2n  to 3n-1) do
                        AH(i, j) ←1 - AG(i mod n, j mod n)
        /*toate nodurile lui G1 sunt legate de toate nodurile lui G2, deci pe porţiunea
        corespunzătoare intersecţiei coloanelor n, ... , 2n-1 cu liniile 0, ..., n-1, respectiv
        pe porţiunea corespunzătoare intersecţiei coloanelor 0, ... , n-1 cu liniile
        n, ..., 2n-1 toate elementele matricei vor avea valoarea 1*/
        for (i ← 0 to n-1) do
                for (j ← n to 2n-1) do
                        AH(i, j) ←1
                        AH (j, i) ←1
        /*toate nodurile lui G2 sunt legate de toate nodurile lui G3, deci pe porţiunea
        corespunzătoare intersecţiei coloanelor 2n, ... , 3n-1 cu liniile n, ..., 2n-1,
        respectiv pe porţiunea corespunzătoare intersecţiei coloanelor n, ... , 2n-1 cu
        liniile 2n, ..., 3n-1 toate elementele matricei vor avea valoarea 1*/
```

```
for (i ← n to 2n-1) do
        for (j ← 2n to 3n-1) do
                AH(i, j) ←1
                AH(j, i) ←1
/*toate nodurile lui G3 sunt legate de toate nodurile lui G4, deci pe porţiunea corespunzătoare intersecţiei coloanelor 3n, ... , 4n-1 cu liniile 2n, ..., 3n-1, respectiv pe porţiunea corespunzătoare intersecţiei coloanelor 2n, ... , 3n-1 cu liniile 3n, ..., 4n-1 toate elementele matricei vor avea valoarea 1*/
for (i ← 2n to 3n-1) do
        for (j ← 3n to 4n-1) do
                AH(i, j) ←1
                AH(j, i) ←1
/*în rest vom pune peste tot valoarea 0*/
for (i ← 0  to n-1) do
        for (j ← 2n to 4n-1) do
                AH(i, j) ←1
                AH(j, i) ←1
for (i ← n to 2n-1) do
        for (j ← 3n to 4n-1) do
                AH(i, j) ←1
                AH(j, i) ←1
end
```

Complexitatea** algoritmului de mai sus este **O(n²).

Schematic, matricea de adiacenţă a grafului H va avea forma:



Se observă că există o bijecţie φ de la V(H) la V($\overline{H}$) astfel încât să existe o funcţie Ψ: E(H) → E($\overline{H}$) bine definită şi bijectivă. Această funcţie φ poate fi:

$\varphi(i) = i + n$, pentru $0 \le i \le n - 1$;
$i + 2n$, pentru $n \le i \le 2n - 1$;
$i - 2n$, pentru $2n \le i \le 3n - 1$;
$i - n$, pentru $3n \le i \le 4n - 1$.

q.e.d.

Observaţii:

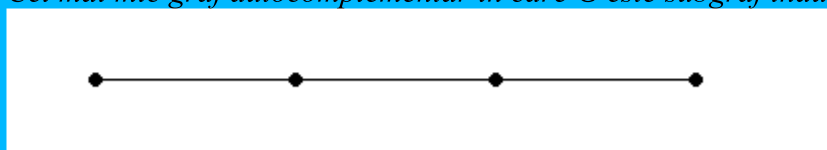
Pot exista grafuri care îndeplinesc cerinţele şi au ordinul mai mic decât 4n. Algoritmul de mai sus nu construieşte în mod obligatoriu cel mai mic graf H autocomplementar cu proprietatea că G este subgraf indus în H.

Chiar graful G ar putea fi autocomplementar şi ar respecta cerinţele de mai sus, însă complexitatea algoritmului de verificare este mult mai mare.

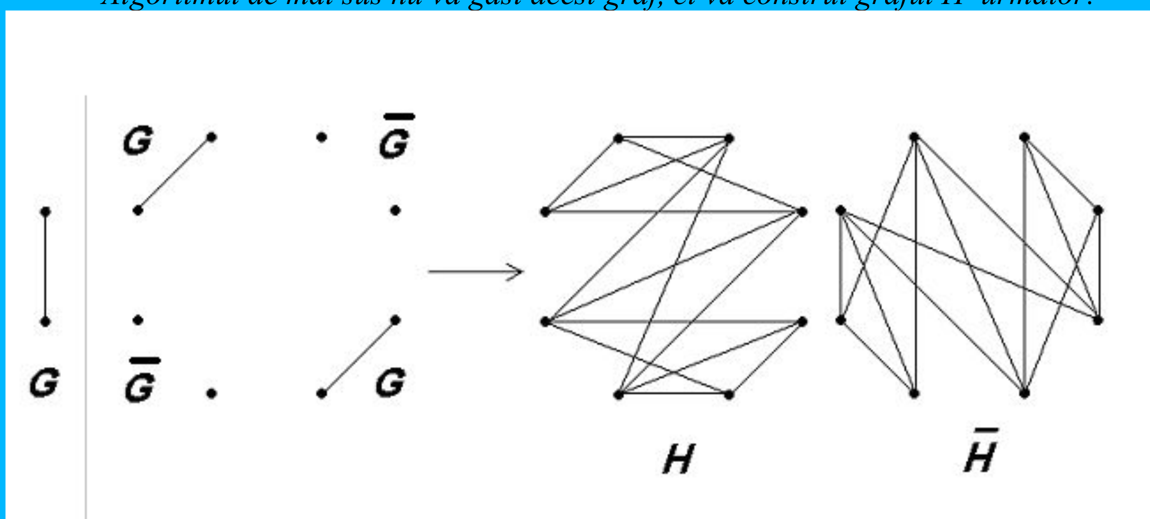
Exemplu:

Fie graful $G = K_2$.

Cel mai mic graf autocomplementar în care G este subgraf indus este P_4*:*



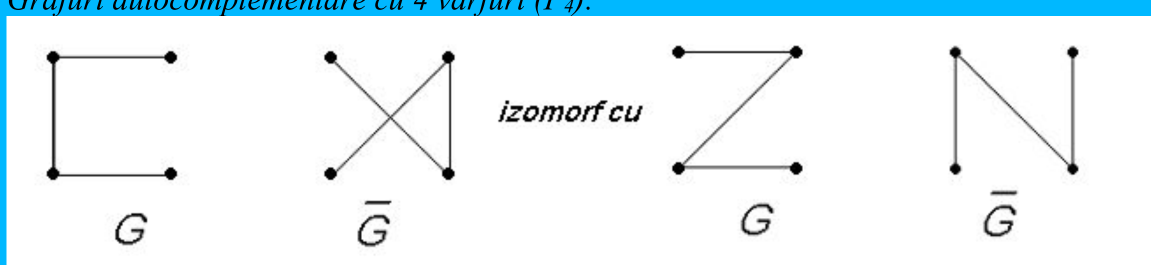
Algoritmul de mai sus nu va găsi acest graf, ci va construi graful H următor:



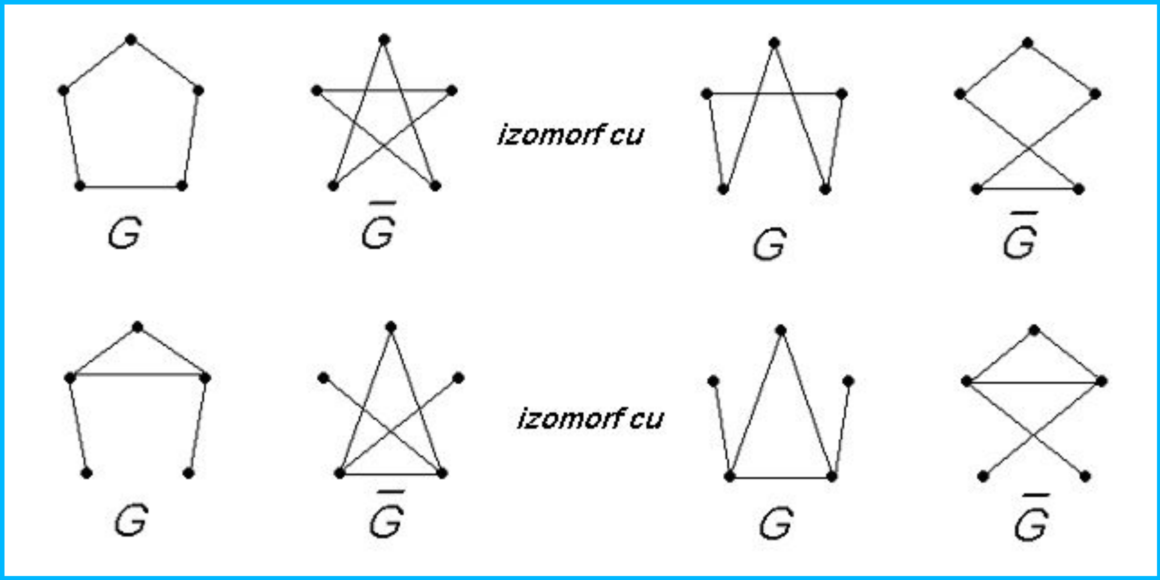
c)*De la a) rezultă că grafurile autocomplementare cu cel mult 7 vârfuri pot avea 1, 4 sau 5 vârfuri şi că trebuie să fie conexe.*

Grafuri autocomplementare cu un vârf: • *G*

Grafuri autocomplementare cu 4 vârfuri (P_4*):*



Grafuri autocomplementare cu 5 vârfuri:



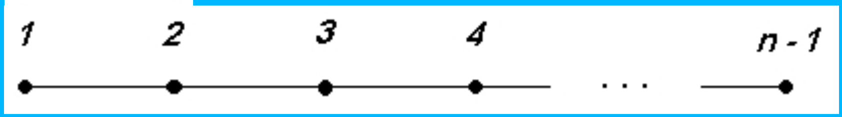
3. *O echipă de doi programatori L(azy) şi T(hinky) primeşte ca sarcină să determine un drum între două noduri care să satisfacă anumite cerinţe, într-un graf G dat, despre care se ştie că este rar:* $|E(G)| = O(|G|)$. *Programatorul L propune ca soluţie generarea (cu backtracking) a tuturor drumurilor dintre cele două noduri şi selectarea celui convenabil, aducând ca argument faptul că într-un astfel de graf nu pot exista prea drumuri între două noduri fixate (sunt puţine muchii şi deci puţine posibilităţi de ramificare ; de exemplu, într–un arbore există exact un drum între orice două noduri fixate). Programatorul T nu este de acord şi dă următorul contraexemplu:se consideră graful* $H = K_2 \times P_{n-1}$ *(n un întreg mare); o pereche de vârfuri de grad 2 adiacente din H se uneşte cu un vârf nou x, iar cealaltă pereche de vârfuri de grad 2 adiacente din H se uneşte cu un vârf nou y; graful obţinut, G , are proprietăţile din problema de rezolvat şi totuşi numărul drumurilor de la x la y în G este prea mare. Ajutaţi-l pe L să înţeleagă contraexemplul, desenându-i graful G arătând că este rar şi estimând numărul drumurilor de la x la y.*

Soluţie:

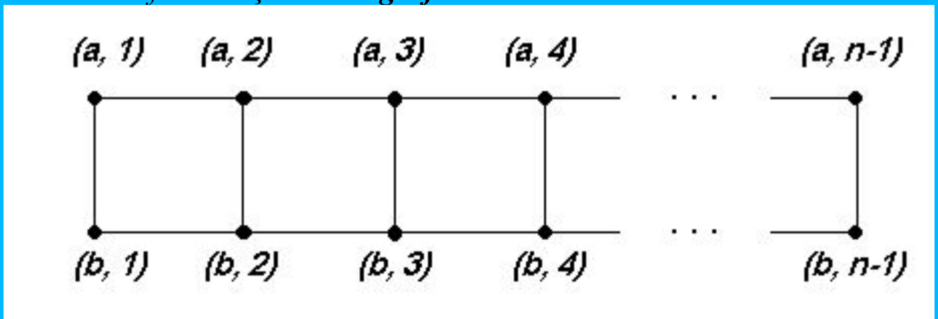
Se consideră graful H ca fiind produsul dintre graful K_2 *şi graful* P_{n-1} $(H = K_2 \times P_{n-1})$.

K_2:

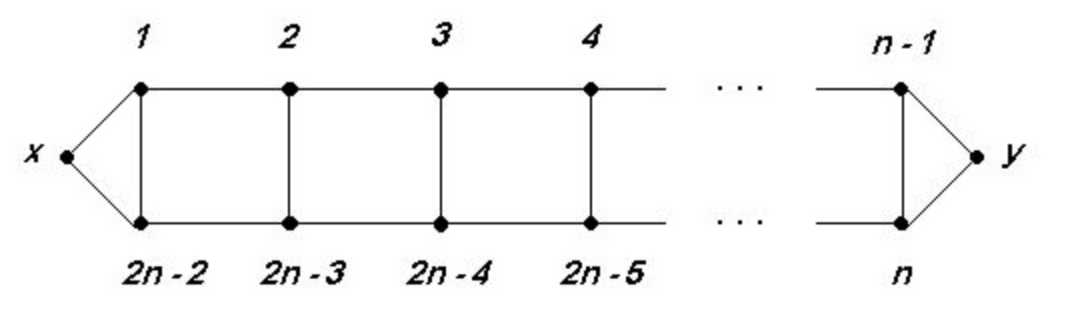
P_{n-1}:



*Graful H obţinut este **graful scară**:*



Pentru a obţine graful G, o pereche de vârfuri de grad 2 adiacente din H se unesc cu un nou vârf x iar o altă pereche de astfel de vârfuri se uneşte cu un nou vârf y. Graful nou obţinut este graful de mai jos (după renotarea nodurilor) :



*Observăm că graful G are 2(n - 1) + 2 = **2n noduri**. Vom calcula numărul de muchii din G (|E(G)|) pentru a putea hotărî dacă este într-adevăr graf rar aşa cum susţine programatorul Thinky. Facem următoarele observaţii:*

- *porţiunea superioară a grafului G, alcatuită din nodurile de la 1 la n-1 cuprinde un total de **n-2 muchii** (acestea formând de fapt chiar graful P_{n-1}). Acelaşi lucru îl putem spune şi despre porţiunea inferioară a grafului, formată din nodurile de la n la 2n-2.*
- *între nodurile de la 1 la n-1 şi cele de la n la 2n-2 sunt exact **n-1 muchii** (fiecare nod din primul grup este legat de câte un nod din al doilea grup prin exact o muchie; mai exact, avem exact câte o muchie între nodurile **i** şi **2n – 1 – i**, cu i de la 1 la n - 1).*
- *la aceste muchii se mai adaugă şi cele **4 muchii** obţinute prin adăugarea la H a nodurilor x şi y (muchiile de la 2n-2 şi 1 la x, muchiile de la n-1 şi n la y).*

Aşadar, în total se obţin

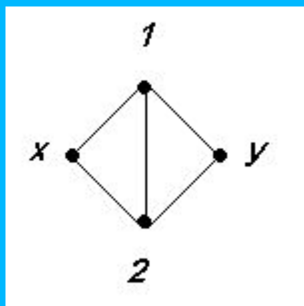
*2(n-2) + (n-1) + 4 =**3n - 1 muchii**.*

*Întrucât 3n+1 este de ordinul lui n la fel ca şi 2n (numărul de noduri) putem spune că **numărul de muchii este de ordinul numărului de vârfuri**, deci graful G este într-adevăr un **graf rar**.*

*Vom calcula acum **numărul drumurilor dintre x şi y**. **Vom demonstra prin inducţie că pentru graful de mai sus există 2^n astfel de drumuri**.*

Etapa I:

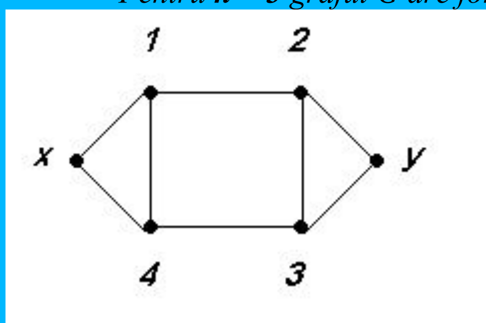
*Pentru **n = 2** graful G are forma următoare:*



*Există **4 (= 2^2)** drumuri în acest graf între x şi y. Acestea sunt:*

-$x \rightarrow 1 \rightarrow y$;
-$x \rightarrow 1 \rightarrow 2 \rightarrow y$;
-$x \rightarrow 2 \rightarrow 1 \rightarrow y$;
-$x \rightarrow 2 \rightarrow y$.

*Pentru **n = 3** graful G are forma:*



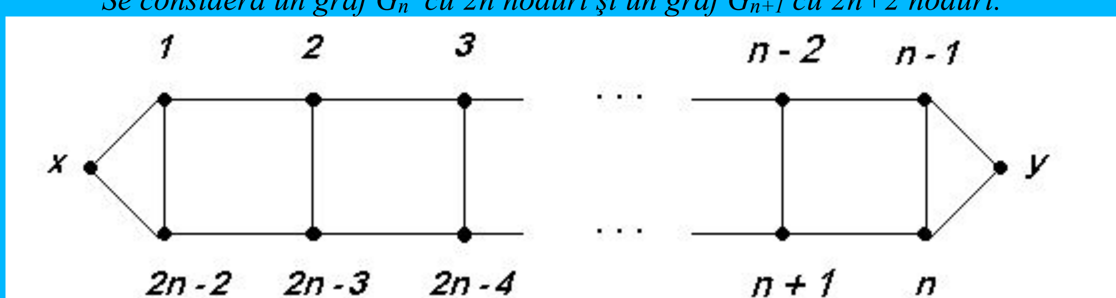
*Există **8 (= 2^3)** drumuri în acest graf între x şi y. Acestea sunt:*

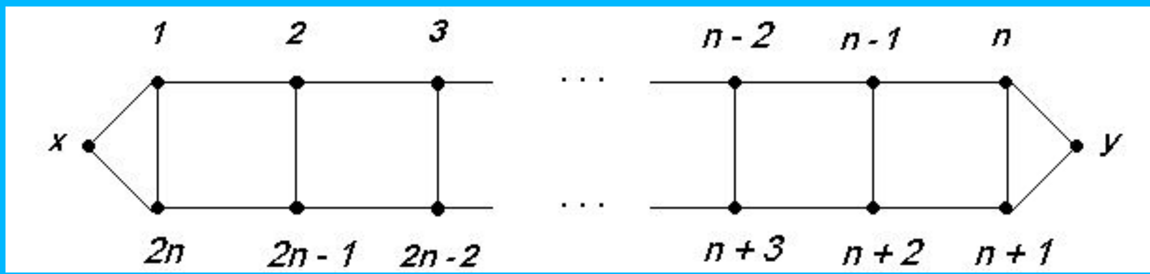
-$x \rightarrow 1 \rightarrow 2 \rightarrow y$;
-$x \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow y$;
-$x \rightarrow 1 \rightarrow 4 \rightarrow 3 \rightarrow y$;
-$x \rightarrow 1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow y$;
-$x \rightarrow 4 \rightarrow 3 \rightarrow y$;
-$x \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow y$;
-$x \rightarrow 4 \rightarrow 1 \rightarrow 2 \rightarrow y$;
-$x \rightarrow 4 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow y$.

Etapa a II-a:

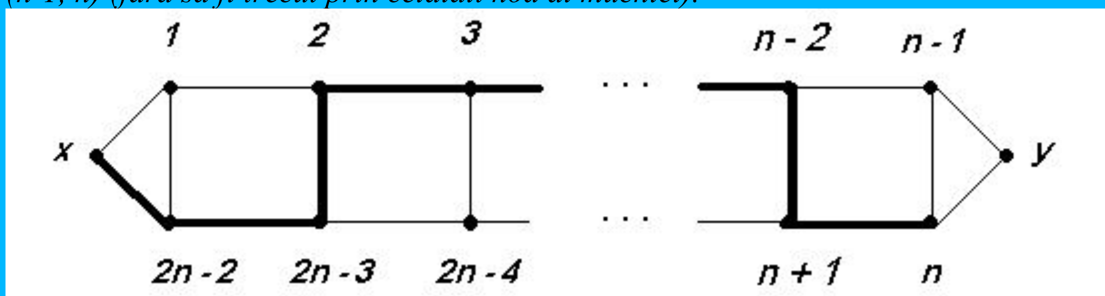
Presupunem afirmaţia adevărată pentru $k \leq n$ şi demonstrăm că este adevărată pentru $k = n+1$.

Se consideră un graf G_n cu 2n noduri şi un graf G_{n+1} cu 2n+2 noduri.

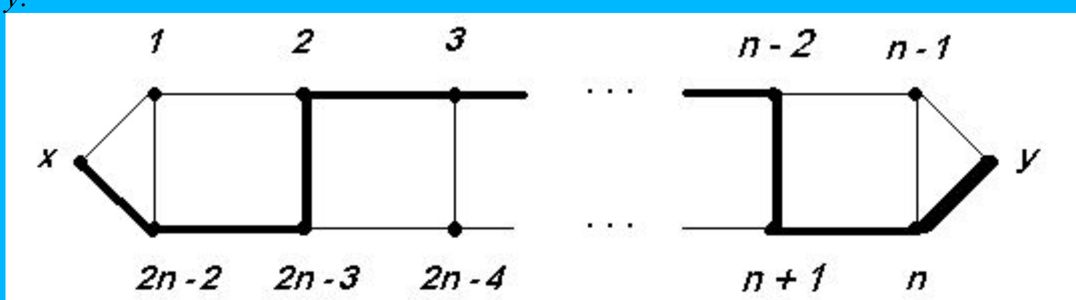


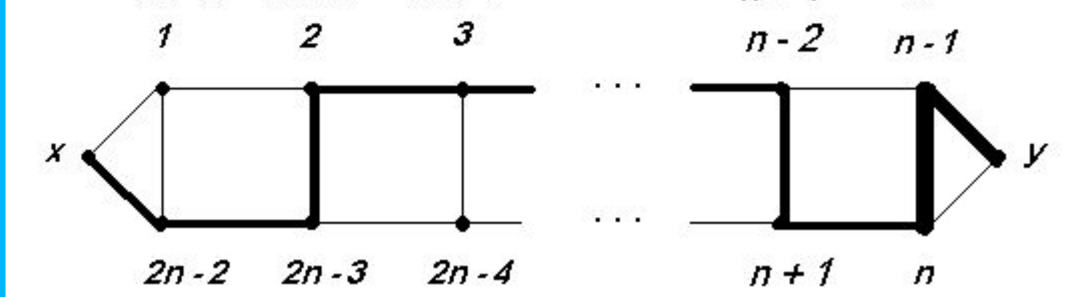


Fie în graful G_n un drum arbitrar care ajunge pe unul din nodurile de pe muchia (n-1, n) (fără să fi trecut prin celălalt nod al muchiei).



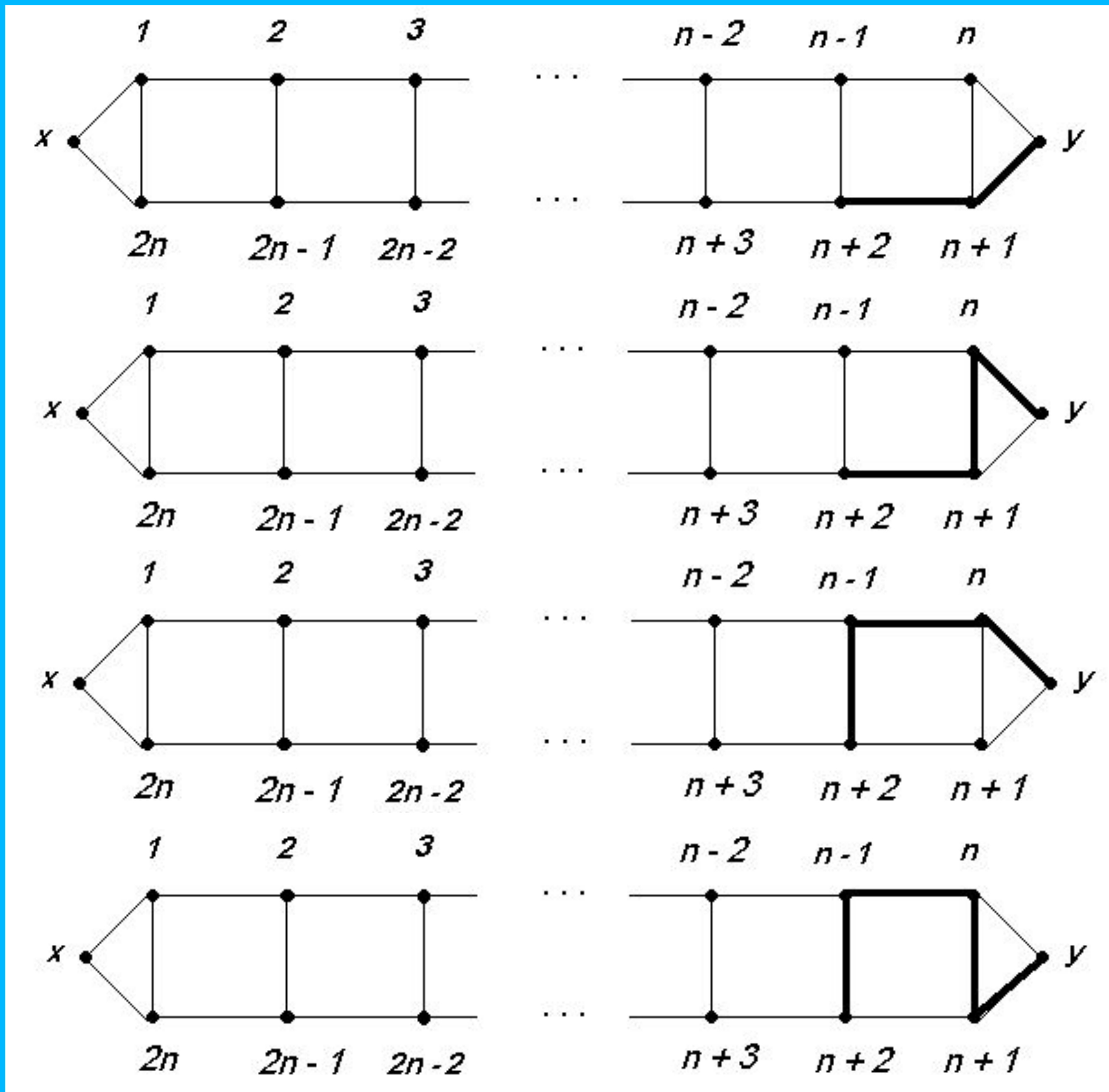
Se observă că există două posibilităţi de a continua acest drum pentru a ajunge la y:





Din ipoteza inductivă rezultă că în graful G_n există 2^n drumuri de la x la y. Cum fiecare astfel de drum, o dată ajuns într-o extremitate a muchiei (n-1, n) fără să fi trecut prin cealaltă extremitate, are două posibilităţi de a se continua, înseamnă că ***există 2^{n-1} drumuri de la x la o extremitate a muchiei (n-1, n) care nu parcurg această muchie****.*

Privind acum graful G_{n+1}, observăm că, plecând dintr-o extremitate a muchiei (n-1, n+2), există 4 posibilităţi de a continua drumul spre y:



Dar, aşa cum am văzut mai sus, toate drumurile posibile de la x la o extremitate a muchiei (n-1, n+2) sunt în număr de 2^{n-1}, şi pentru fiecare astfel de drum avem 4 posibilităţi de a îl continua pentru a ajunge la y.

Aşadar, ***numărul total de drumuri de la x la y în graful G_{n+1} este $2^{n-1} * 4 = 2^{n+1}$.***

q.e.d.

Am arătat deci că $\forall n \in$ **N,** *într-un* ***graf rar*** *G de forma de mai sus cu* ***2n noduri*** *există două noduri x şi y între care* ***numărul de drumuri este 2^n****. Un algoritm backtracking de căutare a celui mai scurt drum între x şi y ar fi deci ineficient.*

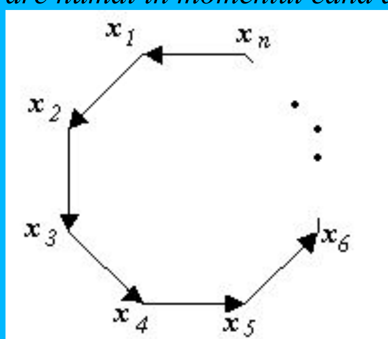
4. *Presupunem că un turneu (digraf cu proprietatea că orice două vârfuri sunt unite printr-un arc) are un circuit C de lungime n>3. Arătaţi că pentru orice vârf x al lui C se pot determina în timpul O(n), încă două vârfuri ale lui C, y şi z, astfel incât (x, y, z) este un circuit de lungime 3.*

Soluţie:

*Presupunem că avem un circuit de lungime **n** într-un graf G şi separăm aceste vârfuri şi arcele care le leagă de restul grafului. Notăm graful astfel obţinut cu H.*

*Graful H este tot **turneu,** deoarece se păstrează proprietatea oricare două noduri din H sunt unite prin exact un arc.*

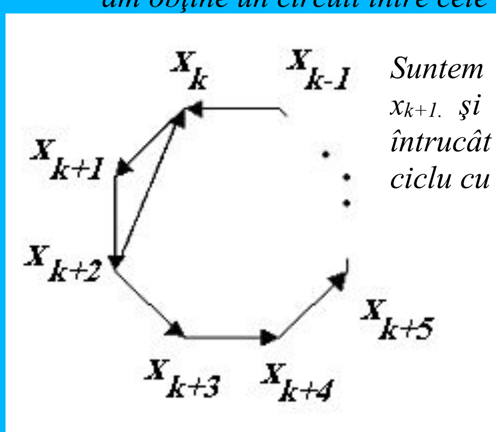
Notăm vârfurile care alcătuiesc circuitul cu x_k, unde k este un număr între 1 şi n. Considerăm aceste noduri ca fiind aşezate într-o listă circulară, astfel încât pentru orice vârf x_k putem găsi imediat succesorul $x_{k'}$ (unde k' = (k + 1) mod n). În continuare vom desena acest circuit, fără a desena şi toate arcele care leagă aceste noduri. Desenăm un arc numai în momentul când acesta este important în tratarea algoritmului.



Din circuitul C se alege un vârf x_k (k între 1 şi n) pentru care trebuie să găsim alte două vârfuri cu care să formeze un circuit de dimensiune 3. Prin vârfurile x_{k+1} şi x_{k+2} inţelegem vârfurile succesoare nodului x_k, din ciclul C.

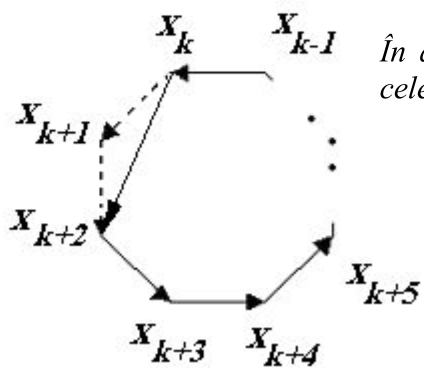
Problema se rezolvă astfel:

- *se consideră cele două noduri de mai sus, succesoare ale nodului x_k . Ştim că avem arc de la x_k la x_{k+1} şi de la x_{k+1} la x_{k+2}. . Dacă am avea arc şi de la x_{k+2} la x_k, am obţine un circuit între cele trei noduri.*



Suntem în cazul în care obţinem circuit între nodurile x_k , x_{k+1}. şi x_{k+2} . În această situaţie algoritmul se opreşte întrucât am găsit cele două noduri y şi z care să formeze un ciclu cu x (= x_k).

- *când nu există acest arc, înseamnă că avem arc de la x_k la x_{k+2}. (întrucât graful iniţial este turneu, deci între oricare două noduri există exact un arc; dacă nu există arc într-o direcţie, cu siguranţă există arc în cealaltă direcţie) ceea ce duce la obţinerea unui **ciclu format din n-1 noduri** (se exclude nodul x_{k+1} iar succesorul lui x_k în listă devine x_{k+2}). În continuare se aplică din nou pasul anterior, de această dată având ca succesori ai lui x_k pe x_{k+2} şi x_{k+3}.*



În acest caz se lucrează cu nodurile x_k , x_{k+2}, x_{k+3} şi cu celelalte noduri din ciclul iniţial, fără x_{k+1}.

- *algoritmul continuă în aceasta manieră, reducându-se de fiecare data numărul de noduri din ciclul iniţial. În cel mai rău caz se reduc n-3 noduri şi cu siguranţă cele 3 noduri obţinute în final sunt nodurile x, y şi z.*

*De fapt, algoritmul se reduce la fiecare pas la verificarea existenţei unui arc între două noduri, operatie care se face în **O(1)**. Dacă există arcul convenabil (arc de la nodul x la succesorul succesorului lui x în ciclul de la pasul curent) algoritmul se opreşte. Altfel, algoritmul continuă şi în cel mai rău caz se fac **n-3 verificări** de existenţă a arcelor, fiecare în O(1). Deci aceste operatii se fac în **O(n)**.*

După ultima verificare, în cazul cel mai nefavorabil, am ajuns la un triunghi format din x şi primii doi predecesori ai sai din ciclul iniţial. Deoarece la pasul anterior, în aceasta situaţie, am obţinut că nu există arc să pornească de la x_{k-2} şi să ajungă în x_k, acum avem arc între cele două noduri în sens invers care impreună cu cele două arce care au mai rămas formează un ciclu.

Deci algoritmul verifică orientarea arcului de la x_k la x_{k+i} cu i de la 2 la –2 (vârfurile sunt aranjate în ordine ciclică) şi la fiecare pas reduce problema de la un ciclu cu k noduri printre care trebuie să cautăm y-ul şi z-ul la altul cu k-1 noduri, până când obţinem ciclul cu 3 noduri în cazul cel mai nefavorabil, adică exact x, y, z.

*În concluzie, se fac **cel mult n-3 consultări ale matricei de adiacentă**, fiecare în timp constant **O(1)** deci problema se rezolvă în timp **O(n)**.*

***Algoritmul** prorpriu-zis de rezolvare a problemei este:*

```
function CautăCircuitDe3(C, x)
begin
        /*C este lista circulară a nodurilor din G ce formează circuitul*/
        /*x este un nod oarecare din acest circuit*/
        /*A este matriea de adiacenţă a grafului G*/

        while ( A(x→succesor→succesor, x) ≠ 1) do
                (x→succesor) ← ( x→succesor→succesor)
        return x, x→succesor, x→succesor→succesor
end
```

GÎRDEA MARTA
URICARU RALUCA
A34
echipa 21

ALGORITMICA GRAFURILOR

TEMA NR. 3
18 martie 2003

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, cosiderăm una oarecare şi cu ajutorul ei construim matricea de incidenţă vârf – arc Q $\in$ {0, 1, -1}^{nXm} 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 matrice diagonală şi precizaţi semnificaţia combinatorie a elementelor ei.

Soluţie:

Fie G' digraful obţinut printr-o orientare oarecare a muchiilor lui G.

Matricea Q a grafului G' are n linii, corespunzătoare vârfurilor, şi m coloane, corespunzătoare arcelor.

*Din definiţia matricei Q se observă că pe fiecare coloană **i** există exact un element cu valoarea -1, pe linia corespunzătoare extremităţii iniţiale a arcului i, şi exact un element cu valoarea 1, pe linia corespunzătoare extremităţii finale a arcului i.*

Fie R = QQ^T.

La înmulţirea matricei Q cu Q^T se obţine o matrice pătratică R de dimensiune n X n, pentru care

$$R(i, j) = \sum_{k=0}^{n-1} Q(i,k)Q^T(k,j).$$

Pentru i $\neq$ j:

Din definiţie se deduce că $Q(i,k)Q^T(k,j) \neq 0$ dacă şi numai dacă atât i, cât şi j sunt extremităţi ale arcului k.

*G fiind graf simplu (nu multigraf) există cel mult o muchie între două vârfuri i şi j, deci cel mult un element al sumei de mai sus poate fi diferit de 0. Mai mult, valoarea acestui element este -1, deoarece unul dintre nodurile i şi j este extremitate iniţială (**Q(l,k) = -1**) iar celălalt este extremitate finală (**Q(l', k) = 1**) a arcului k.*

Pentru i = j:

$$R(i, i) = \sum_{k=0}^{n-1} Q(i,k)Q^T(k,i) = \sum_{k=0}^{n-1} Q(i,k)^2.$$

*Suma de mai sus va avea un număr de termeni nenuli egal cu numărul de elemente nenule de pe linia i. Din definiţie, Q(i, k) $\neq$ 0 dacă şi numai dacă i este extremitate a arcului k. Aşadar, **R(i, i) este egal cu numărul de arce incidente cu nodul j**.*

Aşadar:

$$R(i,j) = \begin{cases} -1, & \text{dacă } i \neq j \text{ şi există arc între i şi j } (A(i,j) = 1); \\ |N_G(i)|, & \text{dacă } i = j; \\ 0, & \text{în rest.} \end{cases}$$

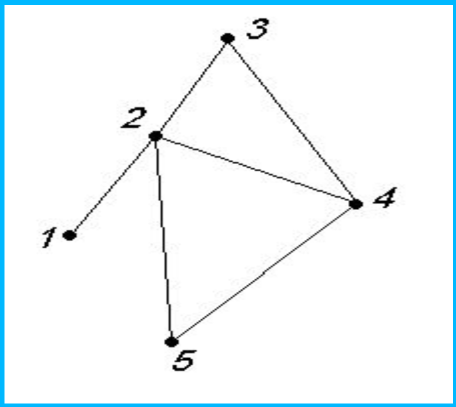
Fie M = A + QQ^T (= A + R).

$$M(i,j) = A(i,j) + R(i,j) = \begin{cases} 1 + (-1) = \mathbf{0}, & \text{dacă } i \neq j \text{ şi există arc între } i \text{ şi } j\ (A(i,j) = 1); \\ 0 + |N_G(i)| = |\mathbf{N_G(i)}|, & \text{dacă } i = j; \\ 0 + 0 = \mathbf{0}, & \text{în rest.} \end{cases}$$

*Aşadar, $M(i, j) = 0$ pentru $i \neq j$. M este matrice diagonală. Elementele de pe diagonala au următoarea semnificaţie: $M(i, i) = |N_G(i)|$ (elementul de la intersecţia liniei i cu coloana i este **gradul nodului i**).*

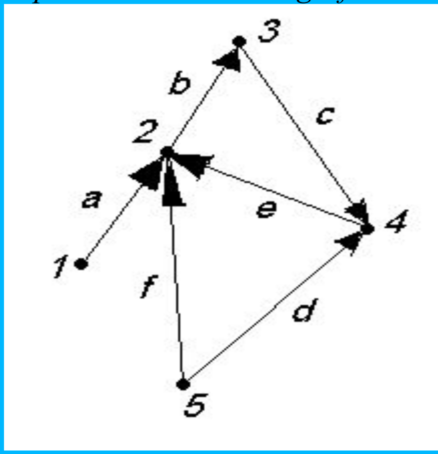
Exemplu:

Fie graful G cu matricea de adiacenţă A:



	1	**2**	**3**	**4**	**5**
1	0	1	0	0	0
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	0	1	0	1	0

O posibilă orientare a grafului G este:



Matricea Q:

	a	**b**	**c**	**d**	**e**	**f**
1	-1	0	0	0	0	0
2	1	-1	0	0	1	1
3	0	1	-1	0	0	0
4	0	0	1	1	-1	0
5	0	0	0	-1	0	-1

$R = QQ^T$. Matricea R:

	1	**2**	**3**	**4**	**5**
1	1	-1	0	0	0
2	-1	4	-1	-1	-1
3	0	-1	2	-1	0
4	0	-1	-1	3	-1
5	0	-1	0	-1	2

$M = A + R$. Matricea M:

	1	**2**	**3**	**4**	**5**
1	1	0	0	0	0
2	0	4	0	0	0
3	0	0	2	0	0
4	0	0	0	3	0
5	0	0	0	0	2

2. *Fie G un graf oarecare şi notăm cu b(G) graful obţinut din G prin inserarea câte unui nod pe fiecare muchie.*
Demonstraţi că b(G) este graf bipartit.
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 se reduce polinomial la testarea izomorfismului a două grafuri bipartite.

Soluţie:

\# ***Arătăm că b(G) este graf bipartit.***

Fie G un graf de ordin n şi dimensiune m. Prin inserarea câte unui nod pe fiecare muchie a lui G se obţine graful b(G) cu ***n + m noduri*** *şi* ***2m muchii****.*

La construirea lui b(G), se introduce câte un nod între două noduri adiacente din G astfel:

Fie ***v*** *şi* ***w*** *cele 2 noduri adiacente şi fie* ***u*** *nodul introdus între ele; se obţine muchie de la* ***u*** *la* ***v*** *şi de la* ***u*** *la* ***w*** *şi se elimină muchia dintre* ***v*** *şi* ***w****. Nodul nou introdus,* ***u****, nu va fi adiacent cu nici un alt nod din graful iniţial şi va avea întotdeauna* $\boldsymbol{N_G(u)=2}$*. Graful obţinut prin inserarea unui astfel de nod între oricare două noduri adiacente are următoarele proprietăţi:*

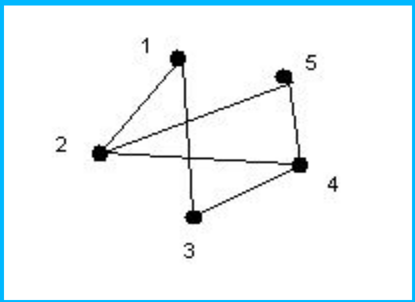
- *nodurile nou introduse sunt adiacente numai cu cele 2 noduri între care au fost introduse şi deci nu sunt adiacente între ele;*
- *nodurile grafului iniţial G sunt adiacente numai cu noile noduri, întrucât în locul oricărei muchii între două noduri din G avem acum două noi muchii, de această dată între nodurile din G şi nodurile nou introduse.*

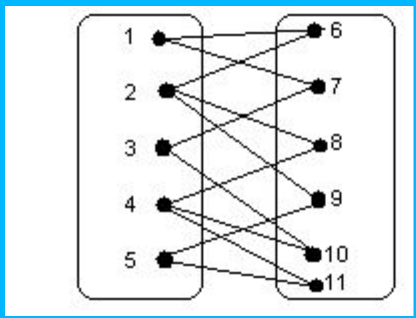
Aşadar, V(b(G)) poate fi împărţită în două submulţimi:

- *S = V(G) (nodurile iniţiale), cu |S| = n;*
- *T = V(b(G)) - V(G) (nodurile nou introduse), cu |T| = m.*

Aşa cum am arătat mai sus, între nodurile din X , cu $X \in \{S, T\}$ *nu esistă muchii în b(G), deci* ***b(G) este graf bipartit****.*

Exemplu:





S ***T***

\# ***Arătăm că două grafuri G şi H sunt izomorfe dacă şi numai dacă b(G) este izomorf cu b(H).***

Implicaţia directă.

$G \cong H$.

Atunci:

$\exists\, \varphi : V(G) \to V(H)$ *bijectivă a. î.* $\exists\, \Psi : E(G) \to E(H)$ *bine definită şi bijectivă a.î.* $\forall e = uv \in E(G)$, $\Psi(uv) = \varphi(u)\ \varphi(v)$.

Construim b(G) şi b(H) ca mai sus.

Putem extinde φ la $\boldsymbol{\varphi e : V(b(G)) \to V(b(H))}$ *astfel:*

$\varphi e(u) = \boldsymbol{\varphi(u)}$, *dacă* $u \in V(G)$;

$\quad\quad$ $\boldsymbol{w}$, *dacă* $u \in V(b(G)) - V(G)$, *unde vecinii lui u sunt* v_1 *şi* v_2, *iar vecinii lui w sunt* $\varphi(v_1)$ *şi* $\varphi(v_2)$.

Se observă că

$$\varphi e(V(G)) = V(H)$$

şi

$$\varphi e(V(b(G))) - V(G)) = V(b(H)) - V(H).$$

Restricţia funcţiei φe *pe mulţimea V(G) este injectivă (deoarece* φ *este injectivă). Dacă notăm cu* $\boldsymbol{f1 : V(G) \to V(H)}$ *o funcţie pentru care*

$$\boldsymbol{f1(u) = \varphi e(u),\ \forall u \in V(G)},$$

această funcţie ***f1 este bijectivă*** *(fiind egală chiar cu* φ*).*

Trebuie să arătăm acum că $\boldsymbol{f2 : V(b(G)) - V(G) \to V(b(H)) - V(H)}$, *cu*

$$\boldsymbol{f2(u) = \varphi e(u),\ \forall u \in V(b(G)) - V(G)}$$

este bijectivă.

Din definiţia lui f2, respectiv a lui φe, *se deduce că, dacă nodul u a fost introdus pe muchia cu extremităţile v1 şi v2, atunci, funcţia f2 îi aociază acel nod din b(H) care a fost introdus pe muchia* $\varphi(v_1)\,\varphi(v_2)$. *Cu alte cuvinte, definiţia lui f2 este echivalentă cu definiţia lui* Ψ *din izomorfismul grafurilor G şi H. Deoarece* Ψ *este bijectivă, şi* ***f2 este bijectivă****.*

Deoarece domeniile, respectiv codomeniile celor două funcţii f1 şi f2 sunt mulţimi disjuncte, rezultă că şi ***funcţia*** $\boldsymbol{\varphi e}$ *pentru care:*

$\varphi e(u) = \boldsymbol{f1(u)}$, *dacă* $u \in V(G)$;

$\quad\quad$ $\boldsymbol{f2(u)}$, *dacă* $u \in V(b(G)) - V(G)$

este funcţie bijectivă.

Arătăm că $\boldsymbol{\exists\, \Psi e : E(b(G)) \to E(b(H))}$ *bine definită şi bijectivă cu proprietăţile cerute.*

$\forall e = uv \in E(b(G))$, $\Psi e(uv) = \varphi e(u)\ \varphi e(v)$.

- *Arătăm că* ***Ψe este bine definită****:*

Din definiţia grafurilor b(G) şi b(H) , rezultă că, $\forall e = uv \in E(b(G))$, $u \in V(G)$ *şi* $v \in V(b(G)) - V(G)$ *sau invers (adică, în mod obligatoriu, una din extremităţi este din V(G) iar cealaltă este un nod introdus ulterior).*

Fie $u \in V(G)$ *şi* $v \in V(b(G)) - V(G)$. *Atunci*

$$\exists w \in V(G) \text{ a.î. } uw \in E(G)$$

şi v este nodul introdus pe muchia uw.

$G \cong H \Rightarrow \exists u', w' \in V(G)$ *a.î.*

$$u' = \varphi(u)\ (= \varphi e(u))$$

$$w' = \varphi(w)\ (= \varphi e(w))$$

$$u'w' \in E(H)$$

Din definiţia lui $\varphi e \Rightarrow \exists v' \in V(b(H)) - V(H)$ *a.î.* $\varphi e(v) = v'$. *(v' are ca noduri adiacente în b(H) pe u' şi w').*

Deci ***există în b(H) muchia*** $\boldsymbol{\varphi e(u)\ \varphi e(v)}$***, oricare*** $\boldsymbol{uv \in E(b(G))}$***.***

- *Arătăm că* ***Ψe este injectivă****:*

$(\forall e1, e2 \in E(b(G)),\ \Psi e(e1) = \Psi e(e2) \Rightarrow e1 = e2)$.

Fie e1 = uv, e2 = xy două muchii din E(b(G)) a.î. $\Psi e(uv) = \Psi e(xy)$.

Atunci:

$$\varphi e(u)\ \varphi e(v) = \varphi e(x)\ \varphi e(y) \Rightarrow$$
$$\varphi e(u) = \varphi e(x) \text{ şi } \varphi e(v) = \varphi e(y)$$
$$\text{sau}$$
$$\varphi e(u) = \varphi e(y) \text{ şi } \varphi e(v) = \varphi e(x)$$

φe este funcţie injectivă $\Rightarrow$ u = x şi v = y sau u = y şi v = x $\Rightarrow$ ***uv = xy*** $\Rightarrow$ ***Ψe este injectivă.***

- *Arătăm că* ***Ψe este surjectivă***:

$(\forall \in E(b(H))\ \exists e' \in E(b(G)$ a.î. $e = \Psi e(e')\ )$

Ştim că Ψe este bine definită şi injectivă, deci cardinalul mulţimii $\Psi e(E(b(G)))$ (imaginea lui E(b(G)) prin funcţia Ψe) este egal cu cardinalul lui E(b(G)):

$$|\ \Psi e(E(b(G)))\ | = |E(b(G))\ |.$$

Dar $\Psi e(E(b(G))) \subseteq |E(b(H)))|$.

De asemenea, din construcţia grafurilor b(G) şi b(H) $\Rightarrow |E(b(G))\ | = |E(b(H))| = 2m$.

$\Rightarrow \Psi e(E(b(G))) = E(b(H)) \Rightarrow$ ***Ψe este surjectivă.***

$\Rightarrow$ ***Ψe este bijectivă.***

Am demonstrat că, dacă G $\cong$ H, atunci b(G) $\cong$ b(H).

Implicaţia inversă.

b(G) $\cong$ b(H).

Atunci:

$\exists\ \varphi : V(b(G)) \to V(b(H))$ bijectivă a.î. funcţia $\Psi : E(b(G)) \to E(b(H))$ definită prin :

$$\forall e = uv \in E(b(G))\ ,\ \Psi(uv) = \varphi(u)\ \varphi(v)$$

este bine definită şi bijectivă.

Fie ***N*** *numărul de noduri din b(G), respectiv b(H), iar* ***M*** *numărul de muchii din b(G), respectiv b(H).*

Din modul de construcţie a grafurilor b(G) şi b(H) avem:

- *numărul nodurilor lui G = N – M / 2 = numărul nodurilor lui H;*
- *numărul muchiilor lui G = M / 2 = numărul muchiilor lui H.*

Pentru a demonstra izomorfismul dintre G şi H trebuie să găsim două funcţii $\varphi r : V(G) \to V(H)$ bijectivă şi $\Psi r : E(G) \to E(H)$ bine definită şi bijectivă astfel încât :

$$\forall e = uv \in E(G)\ ,\ \Psi r(uv) = \varphi r(u)\ \varphi r(v).$$

Alegem ***φ r*** *ca fiind restricţia funcţiei φ pe mulţimea V(G).*

$$\varphi r(u) = \varphi(u)\ ,\ \forall u \in V(G).$$

Ştim că oricare muchie din b(G), respectiv b(H) are o extremitate din V(G), respectiv V(H) şi o extremitate din mulţimea nodurilor introduse la construcţie.

Deoarece b(G) $\cong$ b(H), fiecărei muchii e_1 din E(b(G)) i se asociază o muchie e_2 din E(b(H)) prin funcţia Ψ, astfel încât extremităţii din V(G) a muchiei e_1 îi corespunde extremitatea din V(H) a muchiei e_2.

Aşadar, imaginea lui V(G) prin funcţia φr va fi inclusă în V(H).

φr este ***injectivă***, *deoarece φ este injectivă (iar injectivitatea se păstrază la restricţiile funcţiilor).*

Vom arăta că $\varphi r : V(G) \to V(H)$ este surjectivă.

φr este ***injectivă*** *$\Rightarrow |\ V(G)\ | = |\varphi r\ (V(G))\ |$.*

Dar $\varphi r\,(V(G)) \subseteq V(H)$, şi, aşa cum am văzut mai sus, $|\,V(G)\,| = |\,V(H)\,|$.

*Aşadar, $\varphi r\,(V(G)) = V(H)$. Deci **φ r este surjectivă**.*

$\Rightarrow$ φ r este bijectivă.

- *Arătăm că funcţia **Ψr este bine definită**.*

Muchiile din G se obţin intuitiv din muchiile lui b(G), unind două muchii care au un vârf comun ce nu aparţine lui V(G). Deci două muchii din b(G) se transformă în mod unic într-o muchie din G.

$$\forall e = uv \in E(G)\,,\ \Psi r(uv) = \varphi r(u)\ \varphi r(v).$$

$u, v \in V(G) \Rightarrow \exists\, w \in V(b(G))$ a.î. $uw\,,\ wv \in E(b(G))$.

$b(G) \cong b(H) \Rightarrow \exists\, w', u', v' \in V(b(G))$ a.î.:

$$w' = \varphi(w)$$
$$u' = \varphi(u)\ (= \varphi r\,(u))$$
$$v' = \varphi(w)\ (= \varphi r\,(v))$$
$$\Psi(uw) = \varphi(u)\ \varphi(w) = u'w'$$
$$\Psi(vw) = \varphi(v)\ \varphi(w) = v'w'$$

$$\left.\begin{array}{l} \Rightarrow u'w',\ v'w' \in E(b(H)); \\ \quad w' \notin V(H) \\ \quad u',\ v' \in V(H) \end{array}\right\} \Rightarrow u'v' \in E(H) \Rightarrow \varphi r(u)\ \varphi r(v) \in E(H)\ \ \forall e = uv \in E(G)$$

$\Rightarrow$ Ψr este bine definită.

- *Arătăm că **Ψr este injectivă**.*

$(\forall e1,\ e2 \in E(G),\ \Psi r(e1) = \Psi r(e2) \Rightarrow e1 = e2)$.

Fie $e1 = uv$, $e2 = xy$ două muchii din E(G) a.î. $\Psi r(uv) = \Psi r(xy)$.

Atunci:

$$\varphi r(u)\ \varphi r(v) = \varphi r(x)\ \varphi r(y) \Rightarrow$$
$$\varphi r(u) = \varphi r(x)\ \text{ şi }\ \varphi r\,(v) = \varphi r(y)$$
$$\text{sau}$$
$$\varphi r(u) = \varphi r(y)\ \text{şi}\ \varphi r\,(v) = \varphi r(x)$$

*φr este funcţie injectivă $\Rightarrow u = x$ şi $v = y$ sau $u = y$ şi $v = x \Rightarrow$ **uv = xy $\Rightarrow$ Ψr este injectivă.***

- *Arătăm că **Ψr este surjectivă**:*

$(\forall \in E(H)\ \exists\, e' \in E(G)$ a.î. $e = \Psi r(e')\,)$

Ştim că Ψr este bine definită şi injectivă, deci cardinalul mulţimii Ψr(E(G)) (imaginea lui E(G) prin funcţia Ψr) este egal cu cardinalul lui E(G):

$$|\,\Psi r(E(G))\,| = |E(G)\,|.$$

Dar $\Psi r(E\,(G)) \subseteq |E(H)|$.

De asemenea, din construcţia grafurilor b(G) şi b(H) $\Rightarrow |E(G)\,| = |E(H)\,| = M / 2$.

*$\Rightarrow \Psi r(E(G)) = E(H) \Rightarrow$ **Ψr este surjectivă.***

$\Rightarrow$ Ψr este bijectivă.

Am demonstrat că, dacă b(G) $\cong$ b(H), atunci G $\cong$ H.

Din cele două implicaţii am obţinut că G $\cong$ H dacă şi numai dacă b(G) $\cong$ b(H).

Arătăm că testarea izomorfismului a două grafuri se reduce polinomial la testarea izomorfismului a două grafuri bipartite.

Aşa cum am demonstrat mai sus, pentru a testa izomorfismul a două grafuri este suficient să demonstrăm că grafurile bipartite construite de la aceste grafuri sunt izomorfe. Trebuie să arătăm că dintr-un graf G putem obtine în timp polinomial graful b(G).

Reprezentăm graful G folosind matrice de adiacenţă. Pentru a construi graful b(G) se parcurg următorii paşi:

- *se parcurge matricea de adiacenţă a grafului G, numai în jumătatea sa superioară şi se numără muchiile, obţinându-se m (în $O(n^2/2)$);*
- *se creează matricea de adiacenţă a grafului b(G), cu n+m linii şi coloane şi se umple cu zerouri;*
- *se parcurge din nou matricea lui G şi unde găsim muchie între i şi j , se pune muchie între i şi k şi j între j şi k, unde k este un număr de la n la m + n - 1, care creşte cu o unitate după această operaţie (O(1) pentru fiecare muchie din G).*

Algoritmul este:

```
procedure Construieşte_b(G)(G)
begin
    /*numărarea muchiilor*/
    n ← |G|
    m ← 0
    for i ← 0 to n-1 do
        for j ← i+1 to n-1 do
            if ( A(i,j) = 1 ) then
                m++
    /*crearea matricii cu n+m linii şi coloane şi zerorizarea*/
    V(b(G)) ← {0, 1, ...n + m - 1}
    /*introducerea muchiilor între nodurile lui b(G)*/
    /*primele n noduri, de la 0 la n – 1,  corespund nodurilor din G*/
    /*nodurile de la n la n + m – 1, sunt nodurile adăugate ulterior, care vor fi distribuite pe fiecare muchie din G*/
    k ← n
    for i ← 0 to n-1 do
        for j = i+1 to n-1 do
            if ( A(i, j) = 1 ) then
                B(i, k) ← B(k, i) ← B(j, k) ← B(k, j) ← 1
                k++
    end
```

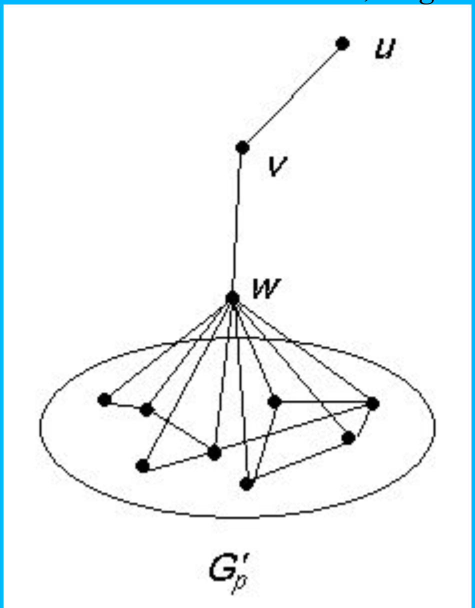
Algoritmul are complexitate ***$O(n^2)$***. *Deci testarea izomorfismului a două grafuri se poate reduce polinomial la testarea izomorfismului grafurilor bipartite obţinute din ele.*

3. *Graful **paianjen** cu n vârfuri este graful care se obţine unind din 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 ale matricei de adiacenţă. (o probă este un acces la un element oarecare al matricei, fără a-l memora explicit pentru utilizări ulterioare.)*

Soluţie:

*Un graf paianjen G_p are proprietatea că există 3 noduri în $V(G_p)$, notate **u, v** şi **w**, astfel încât:*

- ***nodul u** are un singur vecin, pe v;*
- ***nodul v** are exact doi vecini, pe u şi pe w;*
- ***nodul w** are n – 2 vecini; singurul nod din G care nu este vecin cu w este u.*



Notăm cu G_p' subgraful indus de mulţimea $V(G_p)$ – {u, v, w}. Putem observa câteva proprietăţi ale nodurilor din subgraful G_p'.

*Fie un nod **x** din G_p'. Atunci, aşa cum se vede din desenul de mai sus, avem:*

*(am notat prin **Vecini(i)** mulţimea vecinilor nodului i în graful G_p şi prin **Vecini(Vecini(i))** mulţimea tuturor vecinilor vecinilor lui i)*

- ***w** $\in$ **Vecini(x)**;*
- ***v** $\notin$ **Vecini(x)**;*
- ***v** $\in$ **Vecini(Vecini(x))** ;*
- ***u** $\notin$ **Vecini(x)**;*
- ***u** $\notin$ **Vecini(Vecini(x))** ;*

*Pentru a putea decide dacă un graf G este graf paianjen, vom încerca să găsim, pornind dintr-un nod oarecare din graf, nodul **u**.*

*Pornim dintr-un nod oarecare **x** din V(G). Decidem mai întâi dacă x poate fi **u**, **v** sau **w**. Dacă nu este, mergem la pasul următor.*

*Ştim că, dacă există un nod **w** în graful G, atunci acesta este vecin al lui **x** şi că orice nod, în afară de **u**, se leagă de **w**. Deci vom putea elimina dintre nodurile din lista vecinilor lui **x** acele noduri **y** pentru care există un nevecin **z** dintre nodurile care nu sunt vecine cu **x**, deoarece:*

- *dacă nodul nevecin **z** este chiar **u**, atunci nu este vecin cu nici un nod din lista vecinilor lui **x,** deci va fi depistat de algoritm (deoarece lista vecinior lui **x** rămâne goală);*

- *dacă nodul nevecin **z** nu este **u,** atunci nodul eliminat **y** nu poate fi **w,** deoarece **w** este vecin cu toate nodurile cu excepţia lui **u**.*

*Aşadar, dacă graful este paianjen, **w** va fi eliminat din listă numai de către **u** (care va elimina de fapt toate nodurile vecine cu **x**). Aşadar lista de vecini ai lui **x** va deveni goală doar atunci când este găsit nodul **u** (dacă aceste există).*

*De asemenea, ştim că **u** nu este vecin cu **x** şi cu niciunul din vecinii lui **x**. Deci putem elimina dintre nodurile candidate acele noduri care nu au această proprietate.*

*Un **algoritm de verificare dacă G este graf paianjen** este:*

```
function EstePaianjen(G)
begin
        n ← |G|
        /*dacă G nu are cel puţin 4 noduri, atunci, conform definiţiei, nu poate fi graf paianjen*/
        if (n < 4)
        then
                return false

        /*putem alege un nod oarecare x din V(G), de exemplu 0*/

        /*ETAPA I */
        /*creez două liste:
     -  lista vecinilor lui x → listaV;
     -  lista nodurilor care nu sunt vecine cu x → listaNonV; */

        for  i ← 1 to n – 1 do
                if (A(0, i) = 1)
                then AdaugăLaListă(i, listaV)
                else AdaugăLaListă(i, listaNonV)

        /*ETAPA A II-A*/
        /*Verificăm dacă nodul x ales poate fi u, v sau w*/

        switch (lungime(listaV))
        case 0:
                /* graful G nu este conex, deci nu poate fi graf paianjen*/
                 return false
        case 1:
                /* nodul x ar putea fi u*/
                if (EsteU(x))
                then return true
                break
        case 2:
                /* nodul x ar putea fi v*/
                if (EsteV(x))
                then return true
                break
        case n - 2:
```

```
            /* nodul x ar putea fi w*/
            if (EsteW(x))
            then return true
            else
                  /* într-un graf paianjen, unicul nod de grad n-2 este w. Dacă s-a găsit un nod de
                  grad n-2 care nu satisface toate proprietăţile lui w (adică unicul nod care nu îi
                  este vecin are gradul 1 şi vecinul său are grad 2 şi este vecinşi cu w) atunci graful
                  nu poate fi paianjen, deoarece toate celelalte noduri au grad cel mult n-3 */
                  return false
            break
      case n -1:
            /*există un nod vecin cu toate celelalte, deci nu poate exista nodul u în acest graf*/
            return false
      default:
            break


      /*ETAPA A III-A*/
      /*Îl căutăm pe u în listaNonV. Acele noduri care cu siguranţă nu pot fi u vor fi eliminate din
listă pentru a se evita efectuarea unor operaţii inutile.
      Ne vom baza pe proprietatea că într-un graf paianjen u este unicul nod nevecin cu w, deci este
suficient să găsim un nod de gradul 1 nevecin cu un nod de gradul n-2 pentru a putea decide dacă
graful este paianjen, adică un nod care nu este vecin cu nici unul din vecinii lui x.*/

      posibilU ← -1
      for all i in listaNonV do
            while (listaV ≠ NULL) do
                  j ← (listaV→first)
                  if (A(j, i) = 1)
                  then
                        /*dacă nodul i curent din listaV are un vecin în listaNonV, atunci nu
                        poate fi u, deci va fi eliminat, şi se trece la nodul următor, deci la pasul
                        următor din bucla for*/
                        EliminăDinListă(i, listaNonV)
                        break
                  else
                        /*se elimină acele noduri vecine cu x care nu sunt vecine cu i. Dacă există
                        un nod w, atunci acesta va fi eliminat doar dacă i = u. */
                        EliminăDinListă(j, listaV)
            /*dacă pentru un nod i din listaNonV nu s-a găsit nici un nod vecin cu el în listaNonV
            (adică dacă s-a ieşit din bucla while prin nesatisfacerea condiţiei listaV ≠ NULL),
            atunci acesta ar putea fi nodul u căutat, deoarece, dacă w există în listă, acel nod i îl
            elimină; dacă nu este acesta, cu siguranţă graful nu poate fi paianjen, deoarece ar exista
            cel puţin două noduri care nu sunt vecine cu nici un vecin al lui x, deci nu ar exista nici
            un vecin al lui x care să aibă exact n - 2 vecini (în consecinţă nu ar exista un nod w în
            graf)*/
            if (listaV = NULL)
            then
```

```
                posibilU ← i
                break

        /*ETAPA A IV-A*/
        /*Concluziile trase din rezultatul de la etapa anterioară*/
        /*Dacă s-a ieşit din bucla for pentru că s-a găsit un posibil nod u se face verificarea; altfel,
        graful nu poate fi graf paianjen*/
        if (posibilU ≠ -1)
        then
            /*număr vecinii lui posibilU; dacă nu este doar unul, graful nu este paianjen*/
            numărVecini ← 0
            for l ← 0 to n – 1 do
                if ( A(l, posibilU) = 1)
                then
                    numărVecini++
                    if (numărVecini > 1)
                    then
                        return false
            return EsteU(posibilU)
        else
            return false
end

function EsteU(i)
begin
        /*caută vecinii lui i*/
        for j ← 0 to n-1 do
            if (A(i, j) = 1)
            then break
        /*verifică dacă j, vecinul lui i are 2 vecini*/
        numărVecini ← 0
        for k ← 0 to n-1 do
            if (A(k, j) = 1)
            then
                numărVecini++
                if (k ≠ i)
                then vecinDiferitDe_i ← k

        if (numărVecini = 2)
        then
            /*verificăm dacă celălalt vecin al lui j are exact n – 2 vecini şi dacă nodul care nu este
            vecin cu acesta este chiar i*/
            numărVecini ← 0
            for m ← 0 to n – 1 do
                if (A(m, vecinDiferitDe_i) = 1)
                then
                    numărVecini++
```

```
                        else
                                if (m ≠ vecinDiferitDe_i )
                                then nodNevecin ← m
                if (numărVecini = n - 2)
                then
                        if (nodNevecin = i)
                        then
                                return true
                        else
                                return false
                else return false
        else
                return false
end

function EsteV(i)
begin
        /*caută vecinii lui i*/
        vecin1 ← -1
        vecin2 ← -1
        for j ← 0 to n-1 do
                if (A(i, j) = 1)
                then
                        if (vecin1 = -1)
                        then
                                vecin1 ← j
                        else
                                vecin2 ← j
                                break
        /*numără vecinii acestora, memorând, în aceeaşi parcurgere, pentru fiecare ultimul nod cu care
        nu este vecin*/
        numărVecini1 ← 0
        numărVecini2 ← 0
        nevecin1 ← -1
        nevecin2 ← -1
        for k ← 0 to n-1 do
                if (A(k, vecin1) = 1)
                then
                        numărVecini1++
                else
                        if (k ≠ vecin1)
                        then
                                nevecin1 ← k
                if (A(k, vecin2) = 1)
                then
                        numărVecini2++
                else
```

```
                    if (k ≠ vecin2)
                    then
                              nevecin2 ← k
          if (numărVecini1 = 1 and numărVecini2 = n - 2)
          then
                    /*verificăm dacă vecin1 este u şi vecin2 este w, adică dacă vecin1 este unicul nod nevecin
                    cu vecin2*/
                    if (nevecin2 = vecin1)
                    then
                              return true
                    else
                              return false
          else
                    if (numărVecini2 = 1 and numărVecini1 = n - 2)
                    then
                    /*verificăm dacă vecin2 este u şi vecin1 este w, adică dacă vecin1 este unicul nod nevecin
                    cu vecin1*/
                              if (nevecin1 = vecin2)
                              then
                                        return true
                              else
                                        retrun false
          else return false
end

function EsteW(i)
begin
          /*caută nodul care nu este vecin cu i*/
          for j ← 0 to n-1 do
                    if (A(i, j) = 0 and j ≠ i)
                    then break
          /*verifică dacă j are exact 1 vecin; dacă da, îl memorează*/
          numărVecini ← 0
          for k ← 0 to n-1 do
                    if (A(k, j) = 1)
                    then
                              numărVecini++
                              vecin ← k
          if (numărVecini = 1)
          then
                    /*verificăm dacă vecinul lui j are exact 2 vecini şi dacă celălalt vecin este i*/
                    numărVecini ← 0
                    for m ← 0 to n – 1 do
                              if (A(m, vecin) = 1)
                              then
                                        if (m ≠ j and m ≠ i)
                                        then
```

```
                              return false
            return true
        else
            return false
end
```

Numărul accesărilor matricei de adiacenţă a grafului G:

Etapa I:

Pentru crearea listelor listaV şi listaNonV se parcurge o singură dată linia x a matricei A. Vor fi n – 1 = ***O(n)*** *accesări.*

Etapa aII-a:

Pentru verificarea dacă nodul ales este u, v sau w se vor cerceta cel mult trei dintre liniile matricei A, deci se vor efectua 3n = ***O(n)*** *accesări.*

Etapa a III-a:

Fie ***t*** *dimensiunea listei listaV; atunci listaNonV va avea dimensiunea* ***n – 1 – t****.*

În această etapă se încearcă detectarea în graf a nodului ***u****.. Dacă există,* ***u*** *se va afla în listaNonV, iar* ***w*** *în listaV.*

Ştim că, dacă G este graf paianjen, nodul ***u*** *nu trebuie să fie legat nici de* ***x****, nici de vreunul dintre vecinii lui* ***x****, deci nici de* ***w****.*

Îl căută pe ***u*** *în listaNonV. Verificăm pentru fiecare nod din listaNonV dacă are vecini în listaV. Dacă* ***i*** *din listaNonV are un vecin din listaV, atunci cu siguranţă nu poate fi* ***u****, deci va fi eliminat din listă.*

Aşadar, pentru fiecare element din listaNonV, fiecare accesare a matricei cu răspunsul 1 are drept consecinţă eliminarea nodului respectiv din listă. Vom avea ***cel mult |listaNonV| = n – t – 1 accesări ale matricei cu răspunsul 1.***

Ştim că un nod ***i*** *poate fi* ***u*** *numai dacă va elimina toate nodurile vecine cu* ***x*** *din listă. (dacă graful este paianjen, toate celelalte noduri în afară de* ***u*** *sunt vecine cu unul din vecinii lui x, adică cu* ***w****). Deci, pentru fiecare i din listaNonV, vom elimina acele noduri care nu sunt vecine cu el din listaV, ştiind că, dacă există* ***w****, acesta va fi eliminat doar de u).*

Aşadar, pentru fiecare element din listaV se vor putea da cel mult cel mult un răspuns ***0*** *la interogarea matricei A (la un astfel de răspuns va fi eliminat). Deci vor fi* ***cel mult |listaV| = t accesări ale matricei la care răspunsul este 0****.*

Cum elementele matricei nu poit fi decât 0 sau 1, înseamnă că se vor efectua în această etapă cel mult t + n – t – 1 = n – 1 probe ale matricei A. Deci numărul probelor matricei A efectuate în această etapă este ***O(n).***

Etapa a IV-a:

Numărarea vecinilor unui nod ***i*** *se poate face prin parcurgerea liniei* ***i*** *din matricea de adiacenţă a grafului, deci prin* ***n = O(n)*** *accesări. Aşa cum am văzut mai sus, verificarea dacă un nod este nodul* ***u*** *se face efectuând* ***O(n)*** *accesări ale matricei de adiacenţă.*

În concluzie*, putem decide dacă graful G este graf paianjen după O(n) + O(n) + O(n) + O(n) =* ***O(n)*** *probe ale matricei de adiacenţă a acestuia.*

4. *Asociem unui arbore 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 decendentul stâng al lui v este w, atunci se introduc în P_{3n} arcele v_1w_2 şi w_3v_2; dacă decendentul drept al lui v 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 3 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 în-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.

Soluţie:

Fie T arborele pentru care se efectuează o parcuregre ca mai sus.

Intuitiv, putem spune că şirul obţinut prin parcurgerea drumului P_{3n} de la extremitatea iniţială la extremitatea finală are următoarea formă:

- *primul nume din şir este numele rădăcinii;*
- *în cazul în care arborele are subarbore stâng, urmează numele nodurilor din subarborele stâng, în forma precizată de aceşti paşi;*
- *urmează din nou numele rădăcinii arborelui;*
- *în continuare sunt aşezate nodurile din subarborele drept în aceeaşi ordine, în cazul în care acesta există;*
- *şirul este încheiat de numele nodului rădacină.*

*Un **algoritm** de construcţie a şirului ar arăta astfel:*

procedure *Construieşte(r)*
begin
    Introducere_în_şir(r_1)
    if *(r→stânga $\neq$ NULL)*
    then
        Construieşte(r→stânga)
    Introducere_în_şir(r_2)
    if *(r→dreapta $\neq$ NULL)*
    then
        Construieşte(r→dreapta)
    Introducere_în_şir(r_3)
end

*Deci acest şir de noduri se defineşte într-o manieră **recursivă**, întrucât definirea arborelui T se face utilizând definirea pentru subarborele stâng şi definirea pentru subarborele drept. Din acest motiv putem utiliza **metoda inducţiei complete** după numărul de noduri ale arborelui, pentru a demonstra că într-adevăr obţinem cele 3 parcurgeri.*

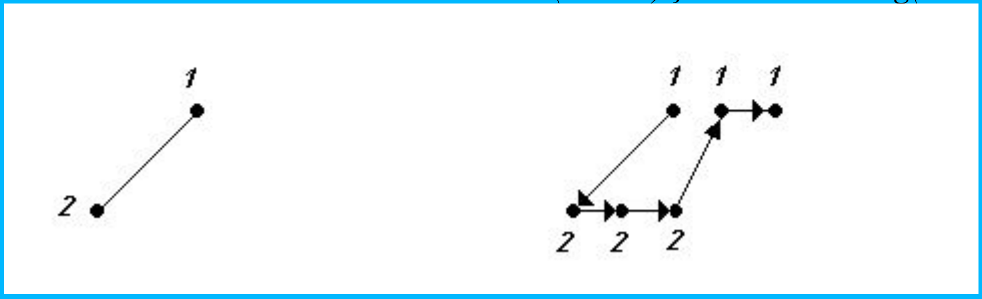
Pasul de bază:

n = 1

*Pentru arborele cu un singur nod (pe care îl vom nota **1**), şirul obţinut este 1 1 1; se observă că, într-adevăr, considerând doar prima apariţie vom obţine parcurgerea **pre-ordine**, considerând doar a doua apariţie vom obţine parcurgerea **in-ordine**, iar considerând ultima apariţie vom obţine parcurgerea **post-ordine** a arborelui cu n singur nod.*

n = 2.

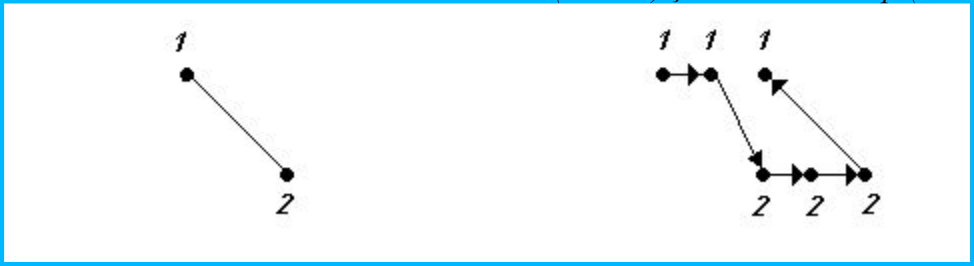
Fie T un arbore cu două noduri: rădăcină (notat 1) şi subarbore stâng(notat 2).



Şirul obţinut este: 1 2 2 2 1 1. Observăm că:

- *dacă selectăm numai **primele** apariţii ale numelor nodurilor obţinem 1 2, care reprezintă parcurgerea **pre-order** a arborelui iniţial*
- *dacă am selecta numai **a doua** apariţie a nodurilor în şir am obţine 2 1, care reprezintă parcurgerea **in-oder** a arborelui.*
- *dacă am selecta doar **a treia** apariţie a nodurilor am obţine 2 1, şi anume parcurgerea în **post-order** a arborelui.*
-

Fie T un arbore cu două noduri: rădăcină (notat 1) şi subarbore drept(notat 2).

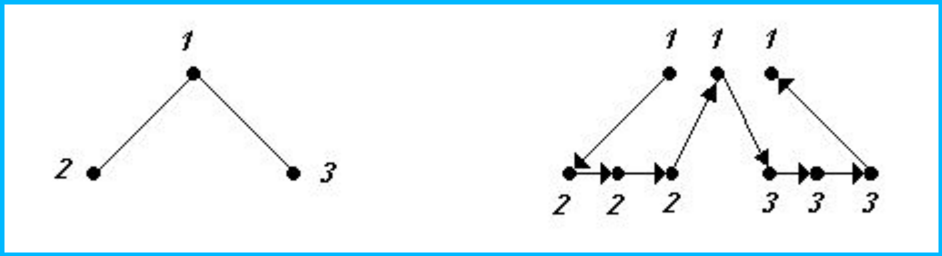


Şirul obţinut este: 1 1 2 2 2 1. Observăm că:

- *dacă selectăm numai **primele** apariţii ale numelor nodurilor obţinem 1 2, care reprezintă parcurgerea **pre-order** a arborelui iniţial*
- *dacă am selecta numai **a doua** apariţie a nodurilor în şir am obţine 2 1, care reprezintă parcurgerea **in-oder** a arborelui.*
- *dacă am selecta doar **a treia** apariţie a nodurilor am obţine 2 1, şi anume parcurgerea în **post-order** a arborelui.*

n = 3.

*Fie un arbore T un arbore cu **3** noduri, astfel încât unul dintre ele să fie rădăcină şi celelalte să fie fiii săi.*



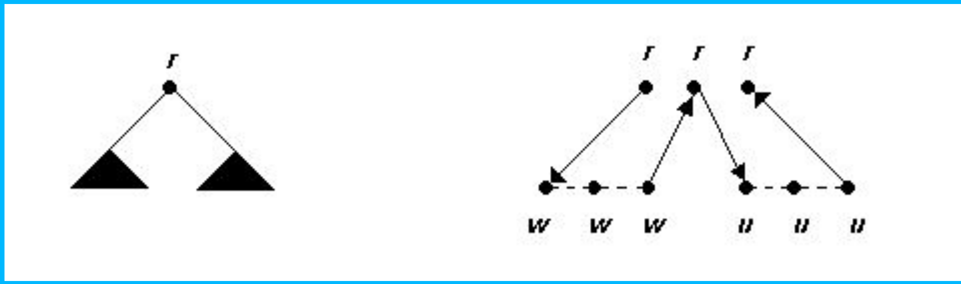
Şirul obţinut după regulile date este de forma : 1 2 2 2 1 3 3 3 1. Observăm că:

- *dacă selectăm numai **primele** apariţii ale numelor nodurilor obţinem 1 2 3, care reprezintă parcurgerea **pre-order** a arborelui iniţial*
- *dacă am selecta numai **a doua** apariţie a nodurilor în şir am obţine 2 1 3, care reprezintă parcurgerea **in-oder** a arborelui.*
- *dacă am selecta doar **a treia** apariţie a nodurilor am obţine 2 3 1, şi anume parcurgerea în **post-order** a arborelui.*

Totodată, în toate cele trei cazuri tratate, şirul obţinut este de forma specificată mai sus, şi anume: rădăcină, şirul subarboreui stâng, rădăcină, şirul subarborelui drept, rădăcină.

Pasul inductiv:

*Se presupune că şirul obţinut este de forma de mai sus şi că se obţin aceste parcurgeri pentru un arbore cu **k noduri, $\forall k \leq n$**. Demonstrăm că avem aceeaşi formă şi pentru un şir obţinut dintr-un arbore cu **n+1 noduri** şi că obţinem aceleaşi parcurgeri .*



*Fie T un arbore cu **n+1 noduri**. Atunci subarborii fii ai rădăcinii au cel mult **n noduri**, de unde deducem, folosind ipoteza inductivă, că şirurile obţinute din ei suntde forma cerută. În continuare, trebuie să demonstrăm că sirul obţinut după regula de mai sus din acest arbore cu n+1 noduri respectă condiţia.*

- *vom porni cu rădăcina **r**, deci primul nod din drum va fi chiar nodul rădăcină.*
- *pentru **r** se verifică dacă are subarbore stâng. În cazul în care avem subarborele stâng, se introduce arc de la rădăcină la descendentul stâng notat **w** şi se urmează acelaşi procedeu pentru acest nod (descendentul stâng), printr-o parcurgere în adâncime a arborelui; cum şirul obţinut, după regulile precizate, din subarborele stâng al arborelui iniţial are cel mult n noduri putem spune că respectă presupunerea făcută, deci se încheie cu nodul rădăcină al acestui subarbore, w. În continuare, se introduce arc de la noua intrare a lui **w** în şir la noua intrare a lui **r** în şir (a doua). Dacă nu avem subarborele stâng, introducem arc de la prima intrare a lui **r** în şir la a doua. În ambele cazuri, şirul obţinut până la acest pas se încheie cu **r**.*
- *verificăm dacă avem subarbore drept pentru nodul **r**. Dacă avem descendent drept al lui **r**, notat **u**, introducem arc de la **r** la prima intrare a lui **u** în şir şi în continuare se aplică recursiv algoritmul de construcţie al şirului pentru subarborele drept, la fel ca la pasul anterior. Şi acest şir se încheie cu ultima apariţie a lui **u** şi conform regulilor, se introduce*

*arc de la **u** la ultima apariţie a lui **r** în şirul final. Dacă nu ar exista subarborele drept, s-ar introduce arc direct de la a doua intrare a lui **r** în şir, la a treia.*

Deci şirul obţinut dintr-un arbore cu n+1 noduri este de aceeaşi formă cu cea a şirurilor obţinute din arbori cu mai puţine noduri, şi anume : rădăcină, şir subarbore stâng, rădăcină, şir subarbore drept, rădăcină.

*Dacă extragem din acest şir numai **primele** apariţii ale nodurilor obţinem:*

- *rădăcina, adică nodul **r**;*
- *primele apariţii ale nodurilor din şirul obţinut din subarborele stâng al lui **r**, dacă există (care reprezintă parcurgerea **pre-order** a subarborelui, aşa cum am presupus în ipoteza inductivă);*
- *primele apariţii ale nodurilor din şirul obţinut din subarborele drept al lui **r**, dacă există (adică parcurgerea **pre-order** a subarborelui);*

*Aceasta este de fapt definiţia recursivă a parcurgerii **pre-order** pentru un arbore binar şi în concluzie, şirul obţinut din primele apariţii ale nodurilor pentru un arbore cu n+1 noduri reprezintă **parcurgerea pre-order a acestui arbore**.*

*Dacă extragem din acest şir numai **a doua** apariţie a fiecărui nod, obţinem:*

- *a doua apariţie a fiecărui nod din şirul obţinut din subarborele stâng al lui **r**, dacă există (care reprezintă parcurgerea **in-order** a subarborelui, aşa cum am presupus în ipoteza inductivă);*
- *rădăcina, adică nodul **r**;*
- *a doua apariţie a fiecărui nod din şirul obţinut din subarborele drept al lui **r**, dacă există (adică parcurgerea **in-order** a subarborelui);*

*Aceasta este de fapt definiţia recursivă a parcurgerii **in-order** pentru un arbore binar şi în concluzie, şirul obţinut din primele apariţii ale nodurilor pentru un arbore cu n+1 noduri reprezintă **parcurgerea in-order a acestui arbore**.*

*Dacă extragem din acest şir numai **ultimele** apariţii ale nodurilor obţinem:*

- *ultimele apariţii ale nodurilor din şirul obţinut din subarborele stâng al lui **r**, dacă există (care reprezintă parcurgerea **post-order** a subarborelui, aşa cum am presupus în ipoteza inductivă);*
- *ultimele apariţii ale nodurilor din şirul obţinut din subarborele drept al lui **r**, dacă există (adică parcurgerea **post-order** a subarborelui);*
- *rădăcina, adică nodul **r**;*

*Aceasta este de fapt definiţia recursivă a parcurgerii **post-order** pentru un arbore binar şi în concluzie, şirul obţinut din primele apariţii ale nodurilor pentru un arbore cu n+1 noduri reprezintă **parcurgerea post-order a acestui arbore**.*

Am obţinut aşadar proprietatea adevărată pentru un arbore cu n + 1 noduri.

Deci proprietatea este adevărată pentru orice arbore, adică şirul are forma intuită şi se obţin acele parcurgeri.

q. e. d.

GÎRDEA MARTA
URICARU RALUCA
A34
echipa 21

TEMA NR. 4
25 martie 2003

1. *Prezentaţi (pe cel mult o pagină) o problemă interesantă din domeniul IT care să necesite rezolvarea eficientă a unei probleme de drum minim într-un digraf adociat problemei iniţiale.*

Soluţie:

Problema găsirii drumului de cost minim într-un graf apare foarte des în realitate, una dintre cele mai importante si mai des folosite implementari a acesteia fiind ***problema rutării pachetelor de date în reţeaua Internet****.*

În reţeaua World Wide Web există un munăr mare de noduri, numite ***routere****, prin care pot trece pachetele de date în drumul lor de la un calculator la altul. Pentru a evita supraîncărcarea infrastructurii, trimiterea fiecărui pachet de date trebuie făcută pe o singură cale, şi trebuie să fie cât mai rapidă şi mai eficientă. Mai exact, drumul parcurs de un pachet de date de la sursă la destinaţie trebuie să fie de cost minim.*

Fiind o reţea complexă, cu multe noduri interconectate, există o multitudine de drumuri între oricare două noduri, găsirea celui mai eficient drum necesitând un algoritm complex, ce trebuie să functioneze cu o bază mare de date şi trebuie să ţină cont de mai mulţi ***factori****, cum ar fi:*

- ***numărul de noduri*** *(routere) prin care trece un pachet, întrucât fiecare router încetineşte transmiterea datelor (la sosirea unui pachet într-un router, acesta proceseaza informaţiile suplimentare ataşate pachetului: tipul pachetului, adresa destinatarului, etc.);*

- ***viteza*** *de transmitere a datelor între două noduri;*

- ***rata de pierdere*** *pe un anumit traseu (procentul pachetelor pierdute dintr-o serie de pachete);*

- ***devierea*** *de la distanţa minimă.*

Unii dintre aceşti factori sunt de fapt ***costuri*** *ale muchiilor.*

Ţinând cont de dimensiunea impresionantă a reţelei Internet şi de numărul mare de pachete transmise în fiecare secundă, orice îmbunătăţire a algoritmilor poate aduce un spor de performanţă relativ mare.

Ca metode de rutare implementate, două se disting în mod deosebit:

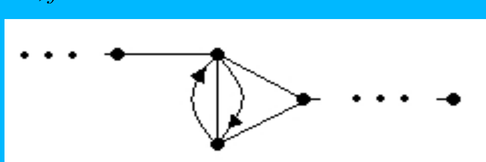
- rutarea cu traseu prestabilit*, atunci când un singur router (primul) stabileşte traseul pachetului, incluzând traseul astfel stabilit în pachetul de date, şi trimite pachetul spre primul nod din listă; fiecare nod, la primirea pachetului, se şterge din listă şi trimite pachetul mai departe; se utilizează deci un algoritm de calculare a drumului de cost minim dintre două noduri terminale într-un graf (reţeaua internet), calcularea se face o singură dată.*

- ***rutarea punct cu punc****t, când pachetul este trimis fără nici o altă informaţie (evident, cu excepţia destinatarului, a expeditorului şi a altor informaţii despre pachetul de date) către primul nod prin care se poate ajunge la destinatar. Se utilizează un algoritm de calculare a nodului următor, astfel încât drumul parcurs de un pachet de date să fie minim (din punct de vedere local). Nu este propriu-zis un algoritm de drum de cost minim, ci o combinaţie cu un algoritm greedy, drumul de cost minim fiind calculat în momentul stabilirii bazei de date (tabela de rutare) şi nu în momentul rutării pachetelor de date.*

Ambele metode sunt imperfecte.

Prima metodă este vulnerabilă schimbărilor în structura reţelei, întrucât aceasta este reţinută într-o bază de date, updatată doar la anumite intervale de timp, verificarea permanentă a fiecarui nod fiind nepractică. Din acest motiv, este posibil ca la un moment dat să se încerce transmiterea unui pachet printr-un drum ce conţine noduri inexistente la momentul respectiv. De asemenea, deoarece traseul stabilit este inclus în pachetul de date, creşterea dimensiunii pachetului de date este proporţională cu lungimea drumului de parcurs.

Metoda rutarii punct cu punct poate genera trimiteri ciclice, atunci când un pachet este trimis în mod repetat între o serie de noduri, fiecare nod considerând ca drumul cel mai bun este prin nodul următor, ca în figură :



Această situaţie nu poate să apara în cazul rutării cu traseu prestabilit. Dar dimensiunea pachetelor de date rămâne mică, bazele de date cu structura reţelei (tabela de rutare) fiind mai mici, fiecarui router fiindu-i necesare informaţii doar despre vecinii imediaţi, schimbările se pot observa mai repede, dar fiecare router încetineşte trimiterea pachetului de date

Ca o combinaţie, se poate utiliza o variantă a metodei rutării cu traseu prestabilit, fiecare nod putând să modifice acest traseu, în momentul în care găseşte o neconcordanţă (un nod temporar scos din funcţiune sau o linie întreruptă). Routerul care găseşte neconcordanţa are responsabilitatea de a calcula un nou drum de cost minim prin nodurile disponibile şi de a modifica traseul din pachet.

2. *Fie G = (V, E) un graf,* $\mathbf{s} \in V$ *un vârf oarecare al lui G iar* **t** *un alt vârf accesibil în G printr-un drum din s. O mulţime* **A** *de muchii se numeşte* ***st-inevitabilă*** *dacă există* $\mathbf{S} \subset \mathbf{V}$ *a.î.* $\mathbf{s} \in S$, $t \notin S$ *şi* $A = \{e \in E \mid e = uv,\ u \in S, v \notin S\}$. *Arătaţi că numărul maxim de mulţimi st-inevitabile disjuncte două câte două este egal cu distanţa de la s la t şi că se poate determina o familie de astfel de mulţimi cu ajutorul unui BFS al lui G din s.*

Soluţie:

Notaţii : d = lungimea drumului minim între s şi t

n = numărul maxim de mulţimi st-inevitabile disjuncte două câte două

Demonstrăm că n=d prin demonstrarea unei duble inegalităţi:

I Demonstrăm $\mathbf{n \le d}$

Fie $D_S(t)$ *un drum minim de la* **s** *la* **t**. *Arătăm că orice mulţime* ***st-inevitabilă*** *conţine cel puţin o muchie din* $D_S(t)$.

Reducere la absurd:

Presupunem că există o mulţime ***A st-inevitabilă*** *care nu conţine nici o muchie din drumul* $D_S(t)$.

A** fiind **st-inevitabilă,** există două mulţimi de noduri din V,* **S** *şi* **T,** *cu* $\mathbf{s \in S}$ *şi* $\mathbf{T = V - S}$, ***cu $\mathbf{t \in T}$, *astfel încât A este mulţimea muchiilor din G care au o extremitare în S şi o extremitate în T. Prin definiţie, acestă mulţime A este* ***st-inevitabilă***.

Fie $V(D_S(t)) = \{s = v_1, v_2, ..., v_d, v_{d+1} = t\}$ *mulţimea nodurilor de pe drumul* $D_S(t)$.

Demonstrăm prin ***inducţie*** *că, în acest caz, toate nodurile din* $V(D_S(t))$ *se vor afla în mulţimea S, inclusiv* ***t****, ceea ce contrazice ipoteza.*

Pasul I:

Ştim din ipoteză că $\mathbf{s = v_1} \in S$. *De asemenea, muchia* $\mathbf{v_1v_2 \notin A}$, *deoarece am stabilit că nici o muchie din drumul* $D_S(t)$ *nu aparţine mulţimii A. Deci* $\mathbf{v_1}$ *şi* $\mathbf{v_2}$ *se găsesc în aceeaşi mulţime. Dar* $\mathbf{v_1} \in S \Rightarrow \mathbf{v_2} \in S$.

Pasul II:

Presupunem că $v_k \in S$. *Arătăm că* $v_{k+1} \in S$.

Muchia $\mathbf{v_kv_{k+1} \notin A}$, *deoarece am stabilit că nici o muchie din drumul* $D_S(t)$ *nu aparţine mulţimii A. Deci* $\mathbf{v_k}$ *şi* $\mathbf{v_{k+1}}$ *se găsesc în aceeaşi mulţime. Dar* $\mathbf{v_k} \in S \Rightarrow \mathbf{v_{k+1}} \in S$.

Aşadar, $\forall j \le d+1$, $\mathbf{v_j} \in S$. *Dar* $\mathbf{v_{d+1} = t \Rightarrow t} \in S$, *ceea ce contrazice ipoteza.*

Presupunerea făcută este deci falsă $\Rightarrow$ *orice mulţime* ***st-inevitabilă*** *conţine cel puţin o muchie din* $D_S(t)$.

Dacă două mulţimi ***st-inevitabile*** *conţin aceeaşi muchie* $\mathbf{v_iv_{i+1}}$, *atunci ele nu sunt disjuncte. Deci numărul maxim de mulitmi* ***st-inevitabile disjuncte*** *este mai mic sau egal cu numărul de muchii din* $D_S(t)$.

$\Rightarrow \mathbf{n \le d}$**.**

II Vom arăta că pentru orice graf G există o familie de mulţimi st-inevitabile de cardinal d.

Definim familiile de mulţimi S_k *şi* A_k *astfel:*

- $S_1 = \{\mathbf{s}\}$
- $A_1 = \{sy \mid y \in N_G(s)\}$
- $S_{i+1} = S_i \cup (\cup_{y \in S_i} N_G(y))$
- $A_{i+1} = \{xy \mid x \in S_{i+1}, y \in N_G(x) - S_{i+1}\}$

Observaţia 1: *Din definirea mulţimilor S_k, se observă că S_k conţine toate nodurile aflate la distanţa cel mult k-1 de nodul* ***s****, în mulţimea $S_k - S_{k-1}$ aflându-se toate nodurile situate exact la distanţa k-1 de* ***s****.*

Demonstrăm că mulţimile A_k sunt disjuncte.

Reducere la absurd:

Presupunem că există o muchie $xy \in A_i$ şi $xy \in A_j$, $i \neq j$. Din definiţia mulţimii A_i şi faptul că $xy \in A_i$, rezultă că ori x, ori $y \in S_i$. Vom considera cazul când $x \in S_i$, celălalt caz tratându-se la fel.

$xy \in A_i \Rightarrow y \in N_G(x) - S_i$, $x \in S_i$ $\Rightarrow$ $y \in S_{i+1} - S_i$. *(din definiţia lui S_{i+1})*

$xy \in A_j \Rightarrow y \in N_G(x) - S_j$, $x \in S_j$ $\Rightarrow$ $y \in S_{j+1} - S_j$. *(din definiţia lui S_{j+1})*

Se observă din definiţia mulţimilor S_k:

$S_1 \subseteq S_2 \subseteq \ldots \subseteq S_m$.

Presupunem $S_{j+1} \subseteq S_{i+1}$, cazul celălalt tratându-se similar.

$S_{j+1} \subseteq S_{i+1}$ *şi* $i \neq j \Rightarrow S_{j+1} \subseteq S_i$.

Dar $y \in S_{j+1}$, deci $y \in S_i$.

$y \in S_{i+1} - S_i$

S_i şi $S_{i+1} - S_i$ sunt submulţii disjuncte ale lui S_{i+1} $\Rightarrow$ *contradicţie.*

$\Rightarrow$ *mulţimile A_k definite ca mai sus sunt* ***disjuncte.***

Demonstrăm că mulţimile A_k sunt ***st-inevitabile*** *pentru* $\mathbf{k \leq d}$.

Presupunem că $\exists\ i \leq$ ***d*** *astfel încât mulţimea A_i nu este* ***st-inevitabilă*** *(adică, eliminând din graful G muchiile din mulţimea A_i, nodul* ***t*** *rămâne accesibil din nodul* ***s****).*

Din definiţia mulţimii $A_i \Rightarrow$ eliminând din graful G muchiile din A_i, orice nod din $S_{i+1} - S_i$ devine inaccesibil din orice nod din S_i. În consecinţă, orice nod din mulţimile S_{i+j}, $j \geq 1$ devine inaccesibil din orice nod din mulţimea S_i.

Dar mulţimile S_k se află în relaţia $S_1 \subseteq S_2 \subseteq \ldots \subseteq S_m$.

$\mathbf{s} \in S_1 \Rightarrow \mathbf{s} \in S_i$.

$i \leq \mathbf{d} \Rightarrow \exists\ j \geq 1$ *a.î.* $\mathbf{t} \in S_{i+j} - S_i$.

$\Rightarrow$ ***t*** *este inaccesibil din* ***s*** $\Rightarrow A_i$ *este* ***st-inevitabilă*** $\forall i \leq d$.

Am obţinut deci ***d*** *mulţimi disjuncte şi* ***st-inevitabile****, ceea ce arată că* ***numărul maxim de mulţimi st-inevitabile disjuncte este cel puţin d .***

Deoarece $\mathbf{n \geq d}$ ***şi*** $\mathbf{n \leq d}$***, se obţine*** $\mathbf{n=d}$.

III Determinarea unei familii de astfel de mulţimi cu ajutorul unui BFS al lui G din s.

În urma unei parcurgeri ***BFS*** *a grafului pornind din nodul* ***s*** *obţinem un arbore de lăţime, arbore în care avem câte un drum minim de la s la fiecare dintre nodurile grafului iniţial. Printre acestea se află şi nodul* ***t****.*

*Vom arăta că acest arbore are următoarea proprietate: între nodurile aflate în arbore pe un nivel k şi cele aflate pe nivelurile diferite de k+1 şi k-1 nu există muchii în graful G (****avem muchii numai între nodurile de pe niveluri consecutive şi între cele de pe acelaşi nivel****).*

Presupunem prin ***reducere la absurd*** *că am avea muchie între un nod u de pe un nivel k şi un nod v de pe un nivel k+ j, j >1. Înseamnă că nodurile u şi v sunt adiacente şi, după principiul parcurgerii BFS, ar fi trebuit ca nodul v să fie descoperit de nodul u (asta în cazul în care v nu ar fi fost descoperit anterior, de alt nod adiacent cu el). Deci nodul v ar trebui să fie situat, cel mult pe nivelul k+1. Aşadar presupunerea este* *falsă şi deci v nu poate fi situat pe un nivel mai jos de $k+1$. Ne aflăm în aceeaşi situaţie şi pentru presupunerea că v s-ar afla pe nivelul $k - j$, $j > 1$.*

*Notăm cu **N** numărul de niveluri din arborele de lăţime obţinut prin parcurgerea BFS din s. Facem următoarele alegeri:*

- *r = nivelul pe care se află t, $1 \le r \le N$;*
- *mulţimea $S_i = \{s\} \cup \{ u \mid u$ se află pe nivelul k, $2 \le k \le i\}$, $1 \le i < r$;*
- *$T_i = V - S_i$ (nodurile de pe nivelurile mai jos de i), $1 \le i < r$;*
- *$A_i = \{e \mid e=uv, u \in S_i, v \in T_i$, u se află pe nivelul i şi v se află pe nivelul $i+1\}$, $1 \le i < r$.*

*Vom demonstra că mulţimile A_i cu i de la 1 la r-1 formează o familie de mulţimi **st-inevitabile** disjuncte două câte două (cu număr maxim de asemenea mulţimi).*

***Observaţia 2**: Numărul de mulţimi A_i este r-1. Într-o parcurgere BFS, drumul obţinut în arbore de la **s** la **t** este un drum de lungime minimă. Am convenit că t se află pe nivelul r, deci lungimea drumului de la s la t este $r - 1$, adică este egală cu numărul mulţimilor A_i construite.*

Arătăm că mulţimile A_i sunt st-inevitabile.

Reducere la absurd:

*Presupunem că o astfel de mulţime A_i nu este **st-inevitabilă.** Atunci există un drum de la s la t care nu are nici o muchie în mulţimea A_i, adică, dacă am elimina din graful G muchiile din A_i, nodul t ar rămâne în continuare accesibil din s.*

Am arătat mai sus că nodurile de pe nivelul i+1 sunt adiacente doar cu nodurile de pe nivelurile i, i+1 şi i+2. Eliminând muchiile din A_i, adică toate muchiile dintre nodurile de pe nivelul i şi nodurile de pe nivelul i+1, nici un nod de pe nivelurile i+j, cu $j \ge 1$ nu va mai fi accesibil din nici un nod de pe nivelurile k, $1 \le k \le i$.

Dar s se află pe nivelul 1 ($1 < i$), t se află pe nivelul r ($r \ge i + 1$), deci prin eliminarea muchiilor din A_i t devine inaccesibil din s.

*Deci mulţimile A_i definite ca mai sus sunt **st-inevitabile.***

Arătăm că mulţimile A_i sunt disjuncte două câte două.

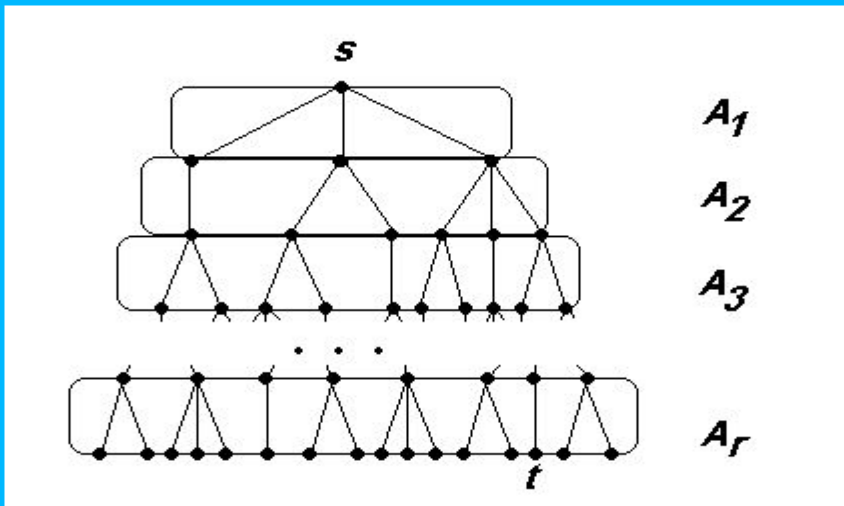
Mulţimile A_i au următoarea formă:

- *A_1 conţine muchiile între nivelul 1 şi nivelul 2*
- *A_2 conţine muchiile între nivelul 2 şi nivelul 3*
- *…*
- *A_{r-1} conţine muchiile între nivelul r-1 şi nivelul r*

Evident, aceste mulţimi sunt disjuncte două câte două întrucât nodurile din V apar o singură dată în arborele BFS.

Am arătat că familia de mulţimi A_i este o familie de mulţimi st-inevitabile şi disjuncte două câte două iar numărul de mulţimi A_i este numărul maxim de mulţimi cu această proprietate.

*Deci printr-**o parcurgere BFS** din nodul s obţinem o familie maximală de astfel de mulţimi, fiecare mulţime conţinând muchiile dintre două niveluri consecutive.*



*Un **algoritm** care să obţină o astfel de familie de mulţimi printr-o parcurgere BFS este:*

```
procedure construieşteAi(G, s, t)
begin
    //introducem s în coada BFS
    adaugă_BFS(s)
    // iniţializăm cu 1 contorul i, care ne spune la ce mulţime A am ajuns şi pe ce nivel ne aflăm.
    // iniţializăm cu 1 variabila n1 în care vom reţine numărul de noduri de pe nivelul curent
    // iniţializăm cu 0 variabila n2 în care vom reţine numărul de noduri de pe nivelul următor
    i← 1
    n1←1
    n2←0
    /*vizităm primul nod din coadă şi introducem toţi vecinii săi nevizitaţi în coada BFS, incrementând
    n2 la fiecare nod descoperit, apoi îl extragem din coadă şi decrementăm n1*/
    /*când am extras exact atâtea noduri câte se află pe nivelul curent, adică n1=0, incrementăm i şi
    n1←n2 iar n2←0*/
    while ( t este nevizitat and BFS != NULL ) do
            u ←(BFS→first)
            while (listaVeciniNevizitaţi_u ≠ NULL) do
                    v ←(listaVeciniNevizitaţi_u →first)
                    adaugă_BFS(v)
                    // introducem în Ai toate muchiile descoperite prin acest procedeu
                    AdaugăLaMulţimeaDeMuchii(uv, Ai)
                    elimină (v, listaVeciniNevizitaţi_u)
                    n2++
            /*la fiecare nod descoperit şi introdus în coada BFS, incrementăm n2 şi introducem muchia
            în Ai*/
            extrage_BFS(BFS→first) //de câte ori extragem un nod decrementăm n1
            n1--
            if ( n1 = 0 ) then
                    i++
                    n1 ← n2
                    n2 ← 0
end
```

3. *Fie $G = (V, E)$ un graf conex şi **v** un vârf al său cu proprietatea că $N_G(v) \neq V - \{v\}$. Dacă pentru $A \subset V$ notăm cu $N_G(A) = \cup_{a \in A} N_G(a) - A$, se observă că există mulţimi de vârfuri A care satisfac proprietăţile: $v \in A$, $[A]_G$ este conex, $N = N_G(A) \neq \Phi$ şi $R = V - (A \cup N) \neq \Phi$ (de exemplu, $A = \{v\}$).*

a) *Demonstraţi că, dacă se consideră o mulţime A maximală (în raport cu incluziunea) satisfăcând proprietăţile enunţate, atunci orice vârf din R este adiacent cu orice vârf din N.*

b) *Dacă, în plus, graful G este $\{C_k\}_{k\geq 4}$ – free, atunci mulţimea N de la punctul a) are proprietatea că este clică în graful G.*

c) *Deduceţi că singurele grafuri $\{C_k\}_{k\geq 4}$ – free, regulate şi conexe sunt grafurile complete.*

Soluţie:

a) *Fie $G = (V, E)$ un graf conex, **v** un vârf al său cu proprietatea că $N_G(v) \neq V - \{v\}$ şi $A \subset V$ o mulţime cu proprietăţile:*

- *$v \in A$;*
- *$[A]_G$ este conex;*
- *$N = N_G(A) \neq \Phi$;*
- *$R = V - (A \cup N) \neq \Phi$;*
- *A maximală cu aceste proprietăţi în raport cu incluziunea.*

Reducere la absurd:

Presupunem că nu orice vârf din R este adiacent cu orice vârf din N, adică

$$\exists\, \mathbf{u} \in \mathbf{R},\ \exists\, \mathbf{w} \in \mathbf{N},\ \text{a.î. } \mathbf{uw} \notin \mathbf{E(G)}.$$

Construim $\mathbf{A' = A \cup \{w\}}$.

$v \in A$ şi $A \subset A' \Rightarrow \mathbf{v \in A'}$

$$\left.\begin{array}{l} w \in N \Rightarrow \exists\, w' \in A \text{ a.î. } ww' \in E(G). \\ [A]_G \text{ este conex} \end{array}\right\} \Rightarrow \mathbf{[A']_G \text{ este conex}}$$

$\mathbf{N_G}(A') = ((\mathbf{N_G}(A) - \{w\}) \cup \mathbf{N_G}(w)) - A$.

Dacă $\mathbf{N_G}(A') = \Phi \Rightarrow (\mathbf{N_G}(A) - \{w\}) \cup \mathbf{N_G}(w) \subseteq A \Rightarrow$

$\Rightarrow \mathbf{N_G}(A) - \{w\} \subseteq A$ şi $\mathbf{N_G}(w) \subseteq A$.

$$\left.\begin{array}{l} \mathbf{N_G}(A) - \{w\} \subseteq A \\ \mathbf{N_G}(A) \cap A = \Phi \text{ (din definiţia lui } \mathbf{N_G}(A)) \end{array}\right\} \Rightarrow \mathbf{N_G}(A) = \{w\}.$$

Totodată, dacă $\mathbf{N_G}(A') = \Phi$, atunci $\forall i$ cu $iw \in E(G) \Rightarrow i \in A'$ (adică singurii vecini ai lui w se găsesc în A').

Dar $A' \cap R = \Phi$ (din definiţia lui R)

$$\left.\begin{array}{l} \Rightarrow \forall j \in R\ \nexists\, jw \in E(G) \Rightarrow \forall j \in R\ \forall i \in N\ \nexists\, ji \in E(G) \text{ şi} \\ \quad N \cap R = \Phi \text{ (din definiţia lui R)} \Rightarrow \forall j \in R\ \forall k \in A\ \nexists\, jk \in E(G) \end{array}\right\} \Rightarrow G \text{ nu este conex, ceea ce contrazice ipoteza.}$$

$\Rightarrow \mathbf{N_G(A') \neq \Phi}$

$R' = V - (A' \cup N_G(A'))$.

$\mathbf{R' \neq \Phi}$, deoarece R' conţine cel puţin nodul u :

$$\left.\begin{array}{l} u \notin \mathbf{N_G}(w) \text{ (conform presupunerii făcute)} \\ \end{array}\right\} \Rightarrow u \in V - (A' \cup \mathbf{N_G}(A')) \Rightarrow$$

*$u \notin$ **N şi** $u \notin A$ (din ipoteză, deoarece $u \in R$ şi $R = V - (A \cup N)$*

$\Rightarrow u \in R' \Rightarrow R' = V - (A' \cup N_G(A')) \neq \Phi$.

*Aşadar, am arătat că, dacă nu orice vârf din R este adiacent cu orice vârf din N, atunci **există o mulţime A' $\supseteq$ A cu proprietăţile cerute**. Aceasta ar însemna că A nu este maximală, ceea ce ar contrazice ipoteza.*

*În concluzie, **presupunerea făcută este falsă**, adică, **dacă A cu proprietăţile de mai sus este maximală în raport cu incluziunea, atunci orice vârf din R este adiacent cu orice vârf din N**.*

b) *Fie $G = (V, E)$ un graf $\{C_k\}_{k\geq 4}$ – free pentru care sunt satisfăcute proprietăţile din enunţ.*

Reducere la absurd:

Presupunem că N nu este clică în G.

Atunci $/N/ \geq 2$ (deoarece, dacă $/N/ < 2$, adică $/N/ = 1$, N ar fi clică în G, întrucât graful cu un nod este graf complet). În plus,

$$\exists\, u, w \in N \text{ a.î. } uw \notin E(G).$$

De la a) $\Rightarrow \forall x \in R$, $ux \in E(G)$ şi $wx \in E(G)$.

Din definiţia lui N (= $\cup_{a \in A} N_G(v) - A$)$\Rightarrow \exists\, u', w' \in A$ a.î. $uu' \in E(G)$ şi $ww' \in E(G)$.

u' şi w' nu sunt în mod obligatoriu distincte. Avem următoarele cazuri:

1. *u' = w'. Atunci graful din Fig 3.1 este **subgraf indus în G**, deoarece:*

- *conform presupunerii făcute, nu există muchie între u şi w;*
- *$x \in R$ şi $u' \in A$, $N \cap R = \Phi \Rightarrow$ nu există muchia xu'.*

Totodată, graful de mai jos este graful C_4. Aceasta contrazice ipoteza conform căreia G este $\{C_k\}_{k\geq 4}$ – free.

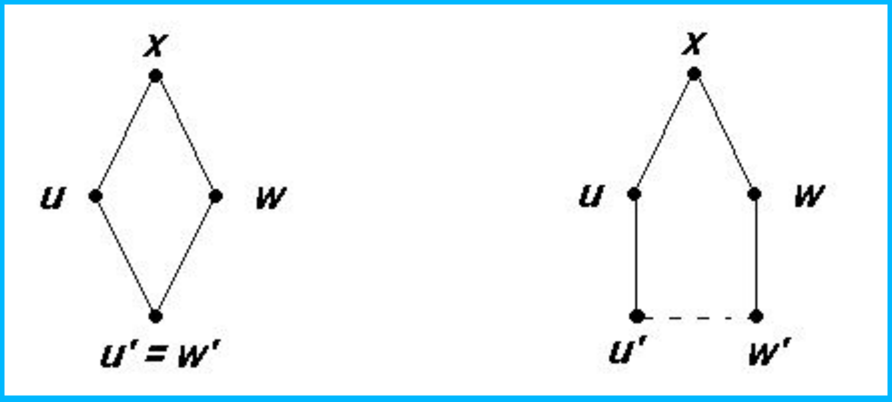


Fig. 3.1 *Fig 3.2*

2. *u' $\neq$ w' şi nu există un vecin comun al nodurilor u şi v în A (dacă ar exista, această situaţie s-ar reduce la cazul anterior). Ştim că $[A]_G$ este conex, deci există în $[A]_G$ un drum de lungime $L \geq 1$ între u' şi w'. Se vor alege acele noduri u' şi w' pentru care drumul de la u' la w' în $[A]_G$ este minim. Atunci graful din Fig. 3.2 este **subgraf indus în G**, deoarece:*

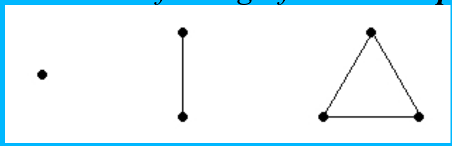
- *conform presupunerii făcute, nu există muchie între u şi w;*
- *$x \in R$ şi $u', w' \in A$, $N \cap R = \Phi \Rightarrow$ nu există muchia xu', respectiv muchia xw'; totodată, toate nodurile u_i de pe drumul dintre u' şi w' sunt din A, deci nu există muchie nici între u_i şi x (dacă $\exists\, u_i$, adică dacă $L > 1$) .*
- *s-au ales acele noduri u' şi w' vecine cu u, respectiv w, pentru care drumul de la u' la w' în $[A]_G$ este minim, deci nu există pe drumul de la u' la w' alte noduri vecine cu u sau cu w (dacă ar exista, drumul nu ar fi minim).*

Dar graful de mai sus este graful C_{L+4}. Aceasta contrazice ipoteza conform căreia G este $\{C_k\}_{k\geq 4}$ – free.

*Am văzut că în ambele cazuri s-a ajuns la o contradicţie, deci presupunerea făcută este **falsă.** Aşadar, **dacă graful G este $\{C_k\}_{k\geq 4}$ – free, atunci mulţimea N de la punctul a) are proprietatea că este clică în graful G.***

c) *Fie G un graf $\{C_k\}_{k\geq 4}$ – free, regulat ($\delta(G) = \Delta(G)$) şi conex.*

*Dacă un astfel de graf G are **mai puţin de 4 noduri**, atunci este complet:*



Vom studia cazul grafurilor cu cel puţin 4 noduri.

Reducere la absurd:

Presupunem că G nu este complet.

Atunci, G fiind regulat, avem $d(u) = t\ (= \delta(G) = \Delta(G)) < n - 1,\ \forall u \in V(G)$.

Fie $v \in V(G)$ oarecare; avem $d(v) = t < n - 1$. Aşadar $N_G(v) \neq V - \{v\}$, deci există mulţimea A cu $v \in A$ şi având proprietăţile de mai sus.

*O mulţime A maximală se poate construi astfel (printr-un algoritm **greedy**):*

- ***$A \leftarrow \{v\}$; $N \leftarrow N_G(v)$; $R \leftarrow V - (A \cup N)$;***
- ***cât timp $\exists w \in (V - A)$ a.î. :***
    - ***$[A']_G$ este conex, unde $A' = A \cup \{w\}$;***
    - ***$N' = \cup_{u \in A'} N_G(u) - A' \neq \Phi$;***
    - ***$R' = V - (A' \cup N') \neq \Phi$;***

  execută $A \leftarrow A \cup \{w\}$.

G este $\{C_k\}_{k\geq 4}$ – free $\Rightarrow$ N definită ca mai sus este clică în G (conform punctului b).

A este maximală $\Rightarrow$ fiecare nod din N este adiacent cu fiecare nod din R (conform punctului a)).

Fie $a = |A|$, $b = |N|$, $c = |R|$. Evident, $a + b + c = |R| = n$.

Fie un nod $u \in N$ şi un alt nod $w \in R$.

$$\left.\begin{array}{l} u \in N \Rightarrow u \text{ are cel puţin un vecin în } A. \\ N \text{ este clică} \Rightarrow u \text{ are exact } b - 1 \text{ vecini în } N. \\ A \text{ este maximală} \Rightarrow u \text{ are exact } c \text{ vecini în } R. \end{array}\right\} \Rightarrow \mathbf{d(u) \geq b + c}.$$

$$\left.\begin{array}{l} w \in R \Rightarrow w \text{ nu are nici un vecin în } A. \\ A \text{ este maximală} \Rightarrow w \text{ are exact } b \text{ vecini în } N. \\ w \text{ cel mult } c - 1 \text{ vecini în } R. \end{array}\right\} \Rightarrow \mathbf{d(w) \leq b + c - 1}.$$

$\Rightarrow d(u) \neq d(w) \Rightarrow$ graful G nu poate fi regulat.

*Aşadar, presupunerea făcută este falsă. **Singurele grafuri $\{C_k\}_{k\geq 4}$ – free, regulate şi conexe sunt grafurile complete.***

4. *Arătaţi că se poate utiliza o parcurgere DFS pentru a determina un circuit par într-un graf 3-regulat oarecare.*

Soluţie:

*Fie G un graf 3-regulat şi T arborele de adâncime obţinut prin parcurgerea DFS a grafului G dintr-un nod oarecare **v**. Fie **u** primul nod găsit în G pentru care nu mai există vecini nevizitaţi. Aşadar, dintre cei 3 vecini ai lui **u**, u_1, u_2 şi u_3, unul este părintele lui **u** în T, iar ceilalţi doi au fost vizitaţi înainte.*

***u** fiind primul nod după care nu se mai poate înainta, arborele construit până la acest pas este de fapt un lanţ, aşa cum se observă în Fig. 4:*

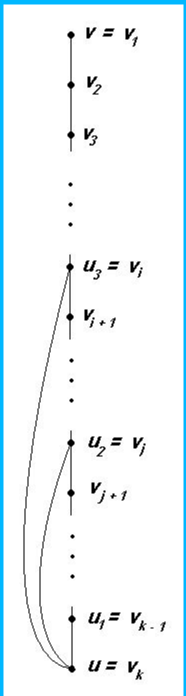


Fig. 4

*Ştim că există muchie de la u la u_2, aşadar avem un circuit C_1 format din u_2, v_{j+1}, ... , v_{k-1}, **u**.*

*Ştim că există muchie de la u la u_3, aşadar avem un circuit C_2 format din u_3, v_{i+1}, ... , v_{k-1}, **u**.*

*De asemenea, există circuit C_3 format din nodurile u_3, v_{i+1}, ... , u_2, **u**.*

Dacă C_1 este circuit par, atunci s-a găsit circuitul căutat.
Altfel, dacă C_2 este circuit par, atunci s-a găsit circuitul căutat.
Dacă nici unul dintre cele două circuite nu este par, atunci:

- *lungimea drumului de la u_2 la u este pară (C_1 este circuit impar, deci eliminându-se muchia u_2u se obţine un drum de lungime L_1 pară);*
- *lungimea drumului de la u_3 la u este pară (C_2 este circuit impar, deci eliminându-se muchia u_3u se obţine un drum de lungime L_2 pară);*

Circuitul C_3 este format din muchiile din drumul de la u_3 la u_2, la care se adaugă muchiile u_2u şi uu_3. Lungimea L_3 a drumului de la u_3 la u_2 este diferenţa dintre L_2 şi L_1, care este număr par, fiind diferenţă de numere impare.

Deci circuitul C_3 are lungimea $L_3 + 2$, care este număr par. S-a găsit deci şi în acest caz un circuit de lungime pară.

*Un **algoritm** de căutare a unui circuit par într-un graf 3-regulat este:*

```
procedure CautăCircuitParDFS (G)
begin
    for all i in V(G) do
        vizitat(i) ← 0
    v ← AlegeVârf(G)
    vizitat(v) ← 1
    /*fiecare vârf vizitat este introdus într-o stivă*/
    Push(v, StivaDFS)
    /*atâta timp cât pentru ultimul nod vizitat mai există vecini nevizitaţi se continuă parcurgerea DFS*/
    while (NumărVeciniNevizitaţi(Top(StivaDFS)) ≠ 0) do
        w ← AlegeVecinNevizitat(Top(StivaDFS))
        vizitat(w) ← 1
        Push(w, StivaDFS)
```

```
/*primul nod care nu satisface condiţia din w h i l e este nodul u de mai sus*/
u ← Top(StivaDFS)
Pop(StivaDFS)
/*construim primul circuit C1*/
C ← Adaugă(u)
u1 ← Top(StivaDFS)
Pop(StivaDFS)
C ← Adaugă(u1)
w ← Top(StivaDFS)
/*adăugăm la C nodurile din stivă până la u2 inclusiv (u2 este vecin cu u, deci C devine lista
nodurilor unui circuit (lista nodurilor circuitului C1)*/
while (w nu este vecin cu u) do
        C ← Adaugă(w)
        Pop(StivaDFS)
        w ← Top(StivaDFS)
u2 ← w;
C ← Adaugă(u2)
/*dacă C1 nu este circuit par, construim C2*/
if (Lungime(C) este număr par)
then
        return C
else
        Pop(StivaDFS)
        w ← Top(StivaDFS)
        /*adăugăm la C nodurile din stivă până la u3 inclusiv (u3 este vecin cu u, deci C devine lista
        nodurilor unui circuit (lista nodurilor circuitului C2)*/
        while (w nu este vecin cu u) do
                C ← Adaugă(w)
                Pop(StivaDFS)
                w ← Top(StivaDFS)
        u3 ← w;
        C ← Adaugă(u3)
        /*dacă nici C2 nu este circuit par, construim C3, care cu siguranţă va fi circuit par, aşa cum
        am arătat mai sus*/
        if (Lungime(C) este număr par)
        then
                return C
        else
                Elimimă din lista C nodurile dintre u şi u2 (fără u şi u2)
                return C
end.
```

GÎRDEA MARTA
URICARU RALUCA
A34
echipa 21

ALGORITMICA GRAFURILOR

TEMA NR. 5
1 aprilie 2003

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

Soluţie:

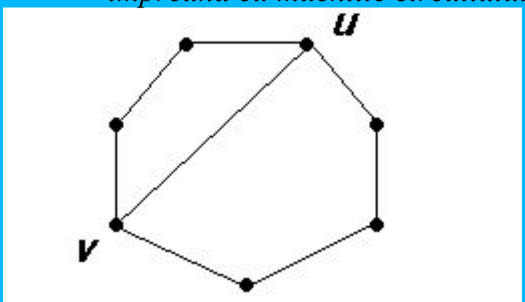
“⇒”
Fie G un graf bipartit. Arătăm că $\forall H$ *subgraf indus al lui G satisface proprietatea* $\mathbf{2\alpha(H) \geq |H|}$.
G bipartit $\Leftrightarrow \exists S \subseteq V$, $T = V - S$ *două mulţimi stabile.*
Fie $A \subseteq V$ *oarecare şi* $H = [A]_G$.

Se disting următoarele două cazuri:
1. $A \subseteq X$ $(X \in \{S, T\})$
X *este mulţime stabilă* $\Rightarrow A$ *este mulţime stabilă* $\Rightarrow H = [A]_G = N_{|A|} \Rightarrow \alpha(H) = |A| = |H|$
$\Rightarrow 2\alpha(H) \geq |H|$.
2. $\exists M \subseteq A$ *a.î.* $M \subseteq S$ *şi* $A - M \neq \Phi$.
$M \subseteq S$ *şi* $(A - M) \subseteq T \Rightarrow M$ *şi* $A - M$ *sunt mulţimi stabile în G* $\Rightarrow$
$\Rightarrow M$ *şi* $A - M$ *sunt mulţimi stabile în* $H = [A]_G$.
$\Rightarrow H$ *este graf bipartit.*
$\alpha(H) \geq \max(|M|, |A - M|)$.
$\Rightarrow \alpha(H) \geq |M|$ *şi* $\alpha(H) \geq |A - M| \Rightarrow 2\alpha(H) \geq |M| + |A - M| \Rightarrow 2\alpha(H) \geq |A| \Rightarrow 2\alpha(H) \geq |H|$.

“⇐”
Fie G un graf pentru care $2\alpha(H) \geq |H|$, $\forall H$ *subgraf indus al lui G. Arătăm că G este bipartit.*

Reducere la absurd:
Presupunem că G nu este bipartit.
Ştim că G este bipatrit $\Leftrightarrow$ *G nu are circuite impare.*
Presupunerea făcută implică deci faptul că există circuite impare în G.

Fie C un astfel de circuit impar în graful **G** *şi V(C) mulţimea nodurilor din acest circuit. Fie* $|V(C)| = k$, *k impar. Avem următoarele două cazuri:*
1. *C este subgraf indus în G.*
2. *C nu este subgraf indus în G. Atunci există* $u, v \in V(C)$ *a.î.* $uv \in E(G)$. *Această muchie* ***uv*** *formează*
împreună cu muchiile circuitului C alte două circuite C^1 *şi* C^2.



Fie $|V(C^1)| = a$ *şi* $|V(C^2)| = b$. *Întrucât toate muchiile din* C^1 *şi* C^2 *apar în C, cu excepţia lui* ***uv***, *care este muchie comună, avem:*

$$a + b = k + 2.$$

Dar k este impar, deci ***a + b este impar***, *aşadar* ***cel puţin unul din circuitele*** $\mathbf{C^1}$ ***şi*** $\mathbf{C^2}$ ***este impar***.

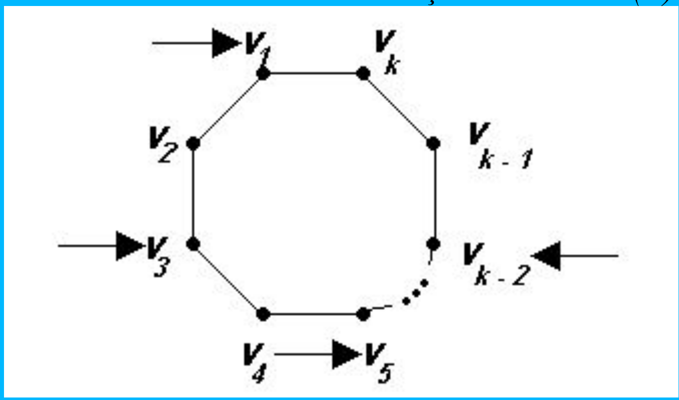
Cel mai mic circuit impar care poate fi descoperit astfel este C_3 *(circuitul de dimensiune 3), care, fiind în acelaşi timp graf complet, este cu siguranţă subgraf indus.*

Aşadar, dacă G are circuite impare, vom putea găsi întotdeauna un astfel de circuit C care să fie subgraf indus in G.

Dar $\alpha(C) = (|V(C)| - 1)/2$ *pentru orice circuit impar.*

(Demonstraţie: Fie C un circuit, $V(C) = \{v_1, v_2, \ldots, v_k\}$*, k impar.*

Se încearcă construirea unei mulţimi stabile în V(C).



Alegându-se, de exemplu, nodurile cu indice impar $(v_1, v_3, \ldots)$ *se observă că nici unul dintre nodurile* v_{k-1}*,* v_k *nu poate fi introdus în mulţimea S stabilă din cauza adiacenţei lor cu câte un nod care se afla deja în mulţime* $(v_{k-2}$*, respectiv* $v_1)$*.*

Am construit deci o mulţime stabilă S maximală cu $|S| = (|V(C)| - 1)/2$*.*

$\Rightarrow \alpha(C) \geq (|V(C)| - 1)/2$.

Presupunem că există o mulţime $S' \subseteq V(C)$ *stabilă cu* $|S'| > (|V(C)| - 1)/2$*.*

Fie $|S'| = t$*.*

C este circuit, deci pentru orice $i \in V(C)$ *avem* $d(i) = 2$ *(fiecare nod are exact 2 vecini în acest circuit). S' fiind mulţime stabilă, fiecare nod i din S' are exact 2 vecini în V – S. Aceasta înseamnă că între mulţimile S' şi V – S' avem* **2t** *muchii. Dar circuitul C are exact* $|V(C)| = k$ *muchii,* ***k impar*** $\Rightarrow 2t \leq k - 1. \Rightarrow t \leq (k-1)/2$*, adică* $|S'| \leq (|V(C)| - 1)/2$*, ceea ce contrazice ipoteza conform căreia* $|S'| > (|V(C)| - 1)/2$*.*

Aşadar, nu există nici o mulţime stabilă în V(C) cu cardinalul strict mai mare decât $(|V(C)| - 1)/2 \Rightarrow \alpha(C) \leq (|V(C)| - 1)/2$*.*

$\Rightarrow \alpha(C) = (|V(C)| - 1)/2$*.)*

Aşadar, $\alpha(C) = (|V(C)| - 1)/2 \Rightarrow 2\alpha(C) = (|V(C)| - 1) \Rightarrow 2\alpha(C) < |C|$*, ceea ce contrazice ipoteza conform căreia* $2\alpha(H) \geq |H|$*,* $\forall H$ *subgraf indus al lui G.*

Presupunerea făcută este deci falsă $\Rightarrow$ ***G nu poate avea circuite impare*** $\Rightarrow$ ***G este graf bipartit.***

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

Soluţie:

Fie G = (S, V – S; E) un graf bipartit. Vom nota T = V – S.

Presupunem $|S| = c$*, respectiv* $|T| = n - c$*.*

Din definiţia grafurilor bipartite,

$$\forall v \in X, \text{ cu } X \in \{S, T\}, N_G(v) \cap X = \Phi.$$

Aşadar, toţi vecinii unui nod din S se găsesc în T, respectiv toţi vecinii unui nod din T se găsesc în S. Fiecare nod din S poate avea deci cel mult $|T| = n - c$ *vecini, respectiv fiecare nod din T poate avea deci cel mult* $|S| = c$ *vecini. Aşadar, lungimea totală a listelor de adiacenţă este:*

$$L = c(n - c) + (n - c)c = \mathbf{2c(n - c)}.$$

Ştim că lungimea totală a listlor de adiacenţă este de două ori numărul de muchii $\Rightarrow$ ***m = c(n – c)***.

Studiind variaţia expresiei ***c(n – c)*** *pentru c întreg din intervalul (0, n) se observă că se obţine valoarea maximă pentru* ***c = [n/2]***, *acestă valoare maximă fiind* $\mathbf{n^2/4}$.

Aşadar,

$$m \le n^2/4 \Leftrightarrow \mathbf{4m \le n^2}.$$

Algoritm de decizie dacă un graf G este complementarul unui graf bipartit

Observaţia 1: *Dacă graful G este complementarul unui graf bipartit, atunci, conform celor demonstrate mai sus,*

$$4m' \le n^2$$

unde $m' = |E(\overline{G})|$, *iar n este ordinul grafului G. Dar*

$$|E(G)| + |E(\overline{G})| = |\mathcal{P}_2(V)| \text{ şi } |\mathcal{P}_2(V)| = n(n-1)/2$$

$$\Rightarrow 4(n(n-1)/2 - |E(G)|) \le n^2$$

$$\Rightarrow 4|E(G)| \ge n^2 - 2n \Leftrightarrow \mathbf{4m \ge n^2-2n}.$$

Aşadar, o ***condiţie necesară*** *(dar nu şi suficientă) pentru ca G să fie complementarul unui graf bipartit este ca* $\mathbf{4m \ge n^2-2n}$, *unde* $m = |E(G)|$.

Observaţia 2: *Un graf H este bipartit* $\Leftrightarrow$ *H acceptă o 2-colorare.*

Vom încerca să decidem dacă graful G este complementarul unui graf bipartit construind $\overline{G}$ *şi verificând dacă acesta acceptă o 2-colorare.*

Pentru ca algoritmul de mai jos să aibă complexitatea cerută este necesar ca graful G să fie reprezentat prin ***liste de adiacenţă***.

```
function EsteComplementarDeGrafBipartit(G)
begin
        n ← |V(G)|
        m ← |E(G)|
        if (4m < n²-2n) then return FALSE
        H ← Complementar(G)
        return EsteBipartit(H)
end

function Complementar(G)
begin
        /*construim graful H cu n noduri*/
        V(H) ← V(G)
        /*în lista de adiacenţă a nodului i în H se plaseaza acele noduri care nu se găsesc în lista de
        adiacenţă a lui i în G*/
        /*Considerăm listele de adiacenţă sortate crescător*/
        for i ← 0 to n – 1 do
                p ← listaDeAdiacenţăG(i)
                for j ← 0 to n – 1 do
                        if (p→indexNod = j)
                        then  p ← (p→next)
                        else
                                if (j ≠ i)
                                then Adaugă(j, listaDeAdiacenţăH(i))
        return H
end
```

```
function EsteBipartit(H)
begin
    /*se va încerca realizarea unei 2-colorări a grafului*/
    /*cele două culori folosite sunt 0 şi 1*/
    /*iniţial toate nodurile sunt necolorate*/
    for i ← 0 to n – 1 do
        culoare(i) ← -1
    /*se realizează o parcurgere DFS a fiecărei componente conexe a grafului H */
    for i ← 0 to n – 1 do
        /*dacă nodul nu a fost deja colorat*/
        if (culoare(i) = -1)
        then
            /*dacă eşuează 2-colorarea uneia dintre componentele conexe, atunci graful cu
            siguranţă nu este bipartit*/
            if (DoiColorareDFS(i, 0) = FALSE)
            then return FALSE
    return TRUE
end

function DoiColorareDFS(v, culoare)
begin
    culoare(v) ← culoare
    for fiecare w din lista de adiacenţă a lui v do
        /*dacă w nu este deja colorat, se va continua parcurgeea DFS şi colorarea nodurilor*/
        if (culoare(w) = -1)
        then DoiColorareDFS(w, 1 - culoare)
        else
            /*dacă nodul a fost deja colorat, se verifică dacă cele două noduri vecine, v şi w,
            au culori diferite; dacă nu, atunci graful nu permite 2-colorare, deci nu este
            bipartit*/
            if (culoare(w) = culoare(v))
            then return FALSE
    return TRUE
end
```

Complexitatea algoritmului:

- ***verificarea condiţiei necesare ca G să fie complementarul unui graf bipartit, $4m \geq n^2$,*** *se face prin parcurgerea în totalitate a listelor de adiacenţă ale nodurilor din graful G şi numărarea nodurilor din aceste liste; se ştie că lungimea totală a listelor de adiacenţă este 2m, iar parcurgerea lor integrală se face în timpul* ***$O(n + m)$.*** *Dacă nu este îndeplinită această condiţie, algoritmul se termină. Altfel se trece la pasul următor.*
- ***construcţia grafului complementar*** *se efectuează în timpul* ***$O(n^2)$****, pentru fiecare vârf verificându-se dacă toate celelalte vârfuri se află sau nu în lista de adiacenţă. Dar, dacă s-a ajuns la acest pas, înseamnă că:*

  $$n^2-2n \leq 4m,$$

  adică $$n^2(1- 2/n) \leq 4m.$$

  $$\Rightarrow n^2 = O(m).$$

  Deci se poate spune că această etapă se execută în timpul ***$O(m)$****.*
- ***parcurgerea DFS pentru colorarea grafului H*** *se execută în timpul* ***$O(n + m')$,*** *unde m' este numărul de muchii din graful H (complementarul lui G). Dar $m' \leq n^2/4$ (condiţie verificată la pasul 1) $\Rightarrow m' = O(n^2)$. În plus, aşa cum am arătat la pasul 2, $n^2 = O(m)$. Rezultă deci că* ***$m' = O(m)$****. Aşadar, acest pas se execută în timpul* ***$O(n + m)$.***

Aşadar, complexitatea algoritmului de mai sus este $O(n + m) + O(m) + O(n + m) =$ ***$O(n + m)$****.*

3. *Arătaţi că orice graf G cu* ***m*** *muchii are un graf parţial H bipartit cu cel puţin* ***m/2*** *muchii.*

Soluţie:

Pentru a demonstra că orice graf G cu ***m*** *muchii are un graf parţial H bipartit cu cel puţin* ***m/2*** *muchii vom arăta că putem construi un astfel de graf parţial bipartit H = (S, T; E(H)) pentru orice graf G = (V, E).*

Vom construi o primă variantă a mulţimilor S şi T ale grafului H printr-un algoritm ***greedy****:*

- *se alege un nod* ***v*** *arbitrar din G, care este introdus în mulţimea* ***S****;*
- *se alege un nou nod* ***w*** *; dacă acesta este adiacent cu* ***v****, atunci introducem nodul w în* ***T****, altfel îl introducem în* ***S****;*
- *pentru fiecare nod pe care îl alegem din G, verificăm unde are mai mulţi vecini: în S sau în T (construite până la pasul curent). Îl introducem în mulţimea unde are mai puţini vecini, astfel încât să obţinem mai multe muchii între nodurile din S şi cele din T.*

În graful parţial H păstrăm numai muchiile din G cu o extremitate în S şi una în T. Evident, graful astfel construit este ***bipartit****. Dar nu putem spune nimic despre numărul de muchii ale grafului H obţinut. Este posibil ca acest graf să indeplinească condiţia cerută. În caz contrar, numărul de muchii din G care apar în H este mai aproape de m/2 decât în majoritatea cazurilor în care s-ar fi făcut o partiţionare arbitrară a mulţimii, evitându-se efectuarea unui număr mare de operaţii la pasul 2.*

Pentru a obţine un graf parţial H bipartit cu cel puţin m/2 muchii procedăm astfel:

- *dacă* ***numărul de muchii ale lui H obţinut până la acest pas este cel puţin m/2****, atunci graful H construit la pasul anterior are proprietatea cerută, deci am arătat că există un graf parţial al lui G care să respecte cerinţele şi algoritmul se opreşte;*
- ***altfel****, intrăm într-o buclă repetitivă, având drept condiţie de oprire inegalitatea : $|E_H| \geq m/2$; verificăm la fiecare pas dacă există noduri în $X \in \{S, T\}$ (mulţimile S şi T obţinute la pasul curent) care au mai mulţi vecini în X, decât în V-X. În acest caz, putem obţine prin mutarea fiecărui astfel de nod în V-X, cel puţin o muchie în plus. Păstrăm graful H bipartit, scoţând din H muchiile dintre nodul mutat şi nodurile din V-X, şi introducând muchiile între nodul respectiv şi nodurile din X.*

Un ***algoritm*** *prin care se construieşte acest graf parţial este:*

```
function ConstruieşteH(G)
begin
        H ← construieşteS_T_Greedy (G)
        S_T_Rearanjare (G, S, T, H)
        return H
end

function construieşteS_T_Greedy (G)
begin
      V(H) ← V(G)
      S ← Φ
      T ← Φ
      muchii ← 0  //iniţializăm  numărul de muchii cu 0
      /* fiecare nod din V va fi plasat în acea mulţime în care are maipuţini vecini (se vor considera
      mulţimile S şi T construite până la pasul curent)*/
      for  all i in V do
            if ( |N_G(i) ∩ S | > |N_G(i) ∩ T |  )
            then
```

```
                //dacă i are mai mulţi vecini în S, îl introducem în T
                adaugă (T, i)
                muchii ← muchii + |N_G(i) ⋂ S |
        else
                //altfel îl adăugăm pe i la S
                adaugă  (S, i)
                muchii ←  muchii + |N_G(i) ⋂ T |
        //adăugăm la E(H) muchiile dintre i şi nodurile din mulţimea în care nu se află acesta
        E(H) ← E(H) ⋃ {iv | iv ∈ E(G), v∈ V-X, cu X ∈ {S, T} }
    return H
end

procedure S_T_Rearanjare (G, H, S, T)
begin
    /*la fiecare pas căutăm acele noduri i din mulţimea X (X ∈ {S, T}) pentru care numărul de
    vecini (în G) din X este mai mare decât numărul de vecini din V–X; dacă găsim un astfel de nod,
    acesta este mutat în cealaltă mulţime, şi se reactualizează numărul de muchii*/
    while ( muchii < m/2 ) do
    for  X in {S, T}  do
        for  all i in X  do
            if ( |N_G(i) ⋂ X | > |N_G(i) ⋂ (V − X) |  )
            then
                //dacă i are mai mulţi vecini în X, îl introducem în V-X
                extrage (X, i)
                adaugă (V-X, i)
                /*se elimină muchiile dintre i şi nodurile din V-X şi se introduc muchiile
                dintre i şi nodurile din X; se actualizează numărul de muchii*/
                muchii ← muchii – |N_G(i) ⋂ (V − X) | + |N_G(i) ⋂ X |
                //actualizăm şi mulţimea E(H)
                E(H) ←(E(H) – ⋃ {iv | iv ∈ E(G), v∈ V-X }) ∪ ⋃ {iw | iw ∈ E(G), w∈X }
end
```

Algoritmul de mai sus se opreşte, iar la oprire graful construit are proprietatea cerută, deoarece:

- *dacă a fost efectuată o mutare dintr-o submulţime în alta, înseamnă că s-a găsit cel puţin un nod care are mai mulţi vecini în submulţimea în care a fost repartizat anterior decât în cealaltă submulţime; deci, prin mutarea lui, se câştigă cel puţin o muchie;*
- *numărul de muchii din G este evident finit, numărul de muchii din H este majorat de numărul de muchii din G, iar numărul de muchii obţinut la fiecare pas din transformarea lui H este strict crescător, aşa cum am arătat mai sus, deci numărul de paşi este finit;*
- *dacă nu este efectuată nici o schimbare, atunci:*
  $\nexists v$ *din X, cu* $X \in \{S, T\}$, *a.î.* $|N_G(v) \cap X| > |N_G(v) \cap (V - X)| \Rightarrow$
  $|N_G(v) \cap (V - X)| \geq |N_G(v) \cap X| \Rightarrow$
  $\Rightarrow$ *pentru fiecare nod din V, cel puţin jumătate din muchiile adiacente cu el în G apar şi în H*
  $\Rightarrow$ $\mathbf{E(H) \geq E(G)/2}$. *(aceasta arată că atunci când nu se mai pot efectua mutări scopul a fost atins).*

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 – P₂(V – S)***. *Deduceţi că* ***α(G) ≥ (|G| - 1) / Δ(G)*** *pentru orice graf conex G.*

Soluţie:

Vom arăta că se poate construi o mulţime S stabilă cu proprietatea din cerinţă.

Folosim ***parcurgerea BFS*** *dintr-un nod oarecare al grafului G, bazându-ne pe faptul că în arborele de lăţime obţinut nu există muchii decât între nodurile de pe nivelurile consecutive şi ar putea exista muchii în graful iniţial între nodurile de pe acelaşi nivel.*

Pasul 1

Iniţial, introducem în mulţimea ***S***, *care trebuie să devină în final stabilă,* ***nodurile de pe nivelurile impare ale arborelui de lăţime***. *În cealaltă mulţime rămân nodurile de pe nivelurile pare. Păstrăm exact muchiile din arbore (adică muchiile dintre niveluri). Întrucât arborele de lăţime este graf conex înseamnă că şi graful obţinut din acest arbore prin procedeul de mai sus rămâne conex. Deci deocamdată nu există muchii între nodurile din S şi nici între nodurile lui V-S.*

Însă mulţimea ***S nu este obligatoriu stabilă la acest pas***.

Pasul 2

Este necesar să verificăm dacă ***există muchii între nodurile din S care se aflau pe acelaşi nivel în arbore***. *Pentru fiecare nod v din S se verifică dacă există muchie între nodul v şi un alt nod din S. Dacă găsim o astfel de muchie:*

- *extragem nodul v din S şi îl introducem în V - S;*
- *pentru fiecare nod w din V - S, adiacent cu v, verificăm dacă mai are vecini în S după eliminarea nodului v; există posibilitatea să distrugem conexitatea grafului (întrucât toţi vecinii lui w s-ar putea afla în V-S şi atunci w ar rămâne izolat). Dacă nu mai are vecini în S, mutăm nodul w în S, în locul lui v. w va avea cel puţin un vecin în V - S, şi anume pe v.*

Observaţie: *Excludem din graful iniţial toate muchiile între nodurile mulţimii V-S.*

Mulţimea ***S*** *obţinută este* ***stabilă*** *deoarece:*

- *am eliminat din S toate nodurile care erau adiacente cu noduri din S;*
- *am adăugat un nod din V - S la S numai în cazul în care acesta nu mai era adiacent cu nici un nod din S, deci nu avea vecini decât în V - S.*

Graful ***H*** *obţinut în această manieră este* ***conex*** *întrucât:*

- *un* ***nod v din S*** *care era* ***rădăcina*** *arborelui de lăţime sigur este legat de cel puţin un nod din*
  V-S, şi anume de nodurile de pe nivelul 2 din arbore care sunt fiii lui v;
- *un* ***nod v din S*** *care se afla în arbore pe unul din* ***nivelurile*** *de forma* ***2k+1*** *(cu 0<k≤ $[\frac{N-1}{2}]$, unde N este numărul total de niveluri ale arborelui) este sigur legat de un nod w de pe nivelul 2k, aflat în mulţimea V-S (adică părintele lui); w nu putea fi mutat în S deoarece el avea un vecin în S, şi anume v.*
- *un* ***nod v din S*** *care se afla pe unul dintre* ***nivelurile*** *de forma* ***2k*** *(cu 0< k≤ $[\frac{N-1}{2}]$) are sigur un vecin situat pe nivelul 2k-1 şi care se află în mulţimea V-S; nodul v de pe nivel par nu a putut ajunge în mulţimea S decât printr-un schimb cu un nod de pe nivelul 2k-1 adiacent cu el, care a ajuns în V-S (după cum se observă din construcţia mulţimii S);*
- *un* ***nod w din V-S*** *care se afla pe unul dintre* ***nivelurile*** *de forma* ***2k*** *(cu 0< k≤ $[\frac{N-1}{2}]$) este sigur legat de un nod v de pe nivelul 2k-1 (nodul de care a fost descoperit în BFS).*

Acest nod v poate fi situat în S şi atunci există muchie între cele două (deci w nu este izolat) sau se poate afla în V-S; dacă v a fost mutat în V-S fără a-l muta şi pe w înseamnă, conform algoritmului, că w avea un alt vecin în S (deci nici în acest caz w nu rămâne izolat).

- *un **nod w din V-S** care se afla în arbore pe unul din **nivelurile** de forma **2k+1** (cu $0<k\leq [\frac{N-1}{2}]$) este sigur legat de un nod de pe acelaşi nivel cu el, aflat în S (din această cauză a fost mutat în această mulţime).*

*Deci **nu există noduri izolate în graful H nici în S, nici în V - S**, întotdeauna existând un nod adiacent cu el din cealaltă mulţime, aşa cum am arătat mai sus; aşadar **graful H obţinut este conex**.*

*Un **algoritm** care construieşte mulţimea S este:*

```
procedure construieşteS(G)
begin
        construieşteS_BFS (G)
        rearanjareS (G)
end

procedure construieşteS_BFS (G)
begin
      //alegem un nod s aleator, nod de start pentru BFS
      //introducem s în coada BFS
      adaugă(BFS, s)
      // iniţializăm cu 1 contorul i, care ne spune pe ce nivel ne aflăm.
      // iniţializăm cu 1 variabila n1 în care vom reţine numărul de noduri de pe nivelul curent
      // iniţializăm cu 0 variabila n2 în care vom reţine numărul de noduri de pe nivelul următor
      i← 1
      n1←1
      n2←0
      /*vizităm primul nod din coadă şi introducem toţi vecinii săi nevizitaţi în coada BFS,
      incrementând n2 la fiecare nod descoperit, apoi îl extragem din coadă şi decrementăm n1*/
      /*când am extras exact atâtea noduri câte se află pe nivelul curent, adică n1=0, incrementăm i şi
      n1←n2 iar n2←0*/
      while ( t este nevizitat and BFS != NULL ) do
                u ← (BFS→first)
                while (listaVeciniNevizitaţi_u ≠ NULL) do
                        v ← (listaVeciniNevizitaţi_u → first)
                        adaugă(BFS, v)
                        elimină (listaVeciniNevizitaţi_u, v)
                        n2++
                /*la fiecare nod v descoperit şi introdus în coada BFS, incrementăm n2 şi introducem v
                în S dacă i este număr impar sau în V-S dacă este par*/
                if ( i este impar ) then
                        adaugă (S, v)
                else
                        adaugă (V-S, v)
                extrage_BFS(BFS→first) //de câte ori extragem un nod decrementăm n1
                n1--
                if ( n1 = 0 ) then
                        i++
                        n1 ← n2
```

```
                    n2 ← 0
end

procedure rearanjareS (G)
begin
    //pentru fiecare nod din S verificăm dacă mai are vecini în S
    for all i in S do
          if ( NG(i) ⋂ S ≠ ϕ ) then
                //dacă i are vecini în S, îl scoatem din S şi îl mutăm în V-S
                extrage (S, i)
                adaugă(V-S, i)
                /*extragem din V-S vecinii lui i care nu mai au noduri adiacente în S şi îi
                introducem în S*/
                for all j in NG(i) ⋂ (V-S) do
                      if (NG(i) ⋂ S = ϕ ) then
                            extrage (V-S, j)
                            adaugă (S, j)
end
```

Demonstrăm că $\alpha(G) \geq (|G| - 1) / \Delta(G)$ pentru orice graf conex G.

Aşa cum am arătat mai sus, există o mulţime stabilă S astfel încât graful bipartit ***$H = (S, V - S; E')$*** *este conex, unde* ***$E' = E - P_2(V - S)$.***

Vom studia valoarea lui ***$|E'|$.***

Notaţie: $m = |E'|$.

Demonstrăm prin ***inducţie după numărul de noduri*** *că orice graf conex G are cel puţin $|G| - 1$ muchii.*

I. *Pentru **n = 1**: $m = 0 \Rightarrow m = n - 1 \Rightarrow m \geq n - 1$.*

   *Pentru **n = 2**: $m = 1 \Rightarrow m = n - 1 \Rightarrow m \geq n - 1$.*

II. *Presupunem afirmaţia adevărată pentru orice graf G cu $|G| = k$ şi o demonstrăm pentru grafurile G cu $|G| = k + 1$:*

*Fie G un graf conex cu **k** noduri. Prin adăugarea unui nod **v** se obţine un nou graf G'. Pentru a păstra conexitatea, este necesar ca **v** să aibă cel puţin un vecin printre nodurile lui G, deci se va adăuga cel puţin o muchie. Aşadar*

$$|E(G')| - |E(G)| \geq 1 \Rightarrow |E(G')| \geq |E(G)| + 1. \textbf{ (1)}$$

*Dar G are **k** noduri, deci din ipoteza inductivă $\Rightarrow |E(G)| \geq k - 1$.* ***(2)***

În plus, $|G'| = k + 1$.

Din (1) şi (2) $\Rightarrow |E(G')| \geq k \Rightarrow |E(G')| \geq |G'| - 1$.

Am demonstrat că orice graf G conex are cel puţin $|G| - 1$ muchii. Proprietatea se păstrează şi pentru grafurile bipartire.

$\Rightarrow \boldsymbol{m \geq |G| - 1}$ ***(3)***

Presupunem $S = \{v_1, v_2, \dots, v_k\}$, unde $k = |S|$.

$H = (S, V - S; E')$ este graf bipartit şi S este mulţime stabilă în G $\Rightarrow$

$\Rightarrow \boldsymbol{m} = |E'| = d(v_1) + d(v_2) + \dots + d(v_k)$.

Dar, $\forall v \in V, d(v) \leq \Delta(G) \Rightarrow$

$\Rightarrow \boldsymbol{m} \leq \underbrace{\Delta(G) + \Delta(G) + \dots + \Delta(G)}_{\text{de } k \text{ ori}} \Rightarrow \boldsymbol{m} \leq k\,\Delta(G)$.

S este mulţime stabilă în G $\Rightarrow k = |S| \leq \alpha(G)$

$\Rightarrow \boldsymbol{m} \leq \alpha(G)\ \Delta(G)$ ***(4)***

Din (3) şi (4) $\Rightarrow \boldsymbol{|G| - 1 \leq m \leq \alpha(G)\ \Delta(G) \Rightarrow \alpha(G)\ \Delta(G) \geq |G| - 1 \Rightarrow}$
$\Rightarrow \boldsymbol{\alpha(G) \geq (|G| - 1) / \Delta(G)}$.

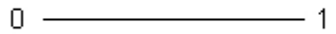
GÎRDEA MARTA **ALGORITMICA GRAFURILOR**
URICARU RALUCA
A34
echipa 21

TEMA NR. 6
8 aprilie 2003

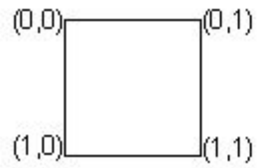
1. *Pentru $d \in N^*$ se consideră graful $G_d = K_2 * K_2 * \ldots * K_2$.*
Să se determine ordinul, dimensiunea şi diametrul lui G_d.
Să se arate că G_d este bipartit şi să se determine $\alpha(G_d)$.

Soluţie:

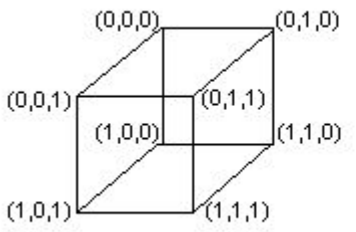
Pentru a intui modul de construire a grafurilor de forma dată vom asocia fiecărui nod coordonate, pornind de la graful $K_2 = G_1$.



În continuare, pentru a obţine G_d, cu $d > 1$, folosim definiţia produsului cartezian a două grafuri. Fie $G=(V, E)$ şi $H=(V', E')$ două grafuri. Notăm cu $G \times H$ graful obţinut după următoarele reguli:
$V(G \times H) = V \times V'$
$E(G\times H) = \{ (v, v')(w, w') \mid v=w$ şi $v'w' \in E'$ sau $v'=w'$ şi $vw \in E \}$
Graful $G_2 = K_2 * K_2$ va arăta astfel (pătratul):



Graful $G_3 = K_2 * K_2 * K_2$ arată astfel (cubul):



1. <u>*Vom arăta prin **inducţie** că **numărul de vârfuri ale grafurilor G_d este 2^d, $d \geq 1$.***</u>

Pas I. *Evident, numărul de vărfuri ale lui $G_1=K_2$ este 2^1 şi deci 2.*
Pas II. *Presupunem că $|V(G_d)| = 2^d$, $d >1$ şi demonstrăm că $|V(G_{d+1})| = 2^{d+1}$.*

*Din modul în care se construiesc aceste grafuri, observăm că nodurile pentru graful G_{d+1} se obţin din nodurile grafului G_d: la d-uplele care formează reprezentarea nodurilor lui G_d adăugăm 0 şi 1 (prin produsul cartezian dintre multimea de d-uple şi perechea (0,1)). Obţinem reprezentarea nodurilor din G_{d+1} şi de fapt elementele **codului Gray** pe d+1 poziţii. Deci pentru fiecare nod din G_d obţinem două d+1-uple (unul prin adăugarea lui 0 şi unul prin adăugarea lui 1). În concluzie, $|V(G_{d+1})| = |V(G_d)| * 2=2^d * 2= 2^{d+1}$.*

*Am demonstrat că **numărul de vârfuri este 2^d**, pentru grafurile **G_d**.*

2. *Pentru a calcula <u>**numărul de muchii**</u> ne folosim de definiţia produsului cartezian a două grafuri.*

*Iniţial, arătăm că **în graful G_d există muchie numai între acele d-uple care diferă printr-un singur element**. Vom demonstra această proprietate prin **inducţie**.*

***I.** Pentru **pasul de bază**, verificăm proprietatea pentru G_2, întrucât pentru G_1 este evident adevărată. Observăm că avem muchie între (0,0) şi (0,1), între (0,1) şi (1,0), între (1,0) şi (1,1), şi între (0,1) şi (1,1). Evident, aceste muchii respectă relaţia de mai sus.*

II. Presupunem proprietatea adevărată pentru G_d, cu d>2 şi demonstrăm pentru G_{d+1}.

Fie două d+1-uple de forma coordonatelor nodurilor: $(x_1, x_2,..., x_d, y_1)$ şi $(x_1', x_2', ..., x_d', y_2)$, unde x_i, x_i', y_1, y_2 au valoarea 0 sau 1. Conform definiţiei produsului cartezian, există muchie între cele două noduri dacă:

- *$(x_1, x_2,..., x_d) = (x_1', x_2', ..., x_d')$ şi y_1, y_2 sunt numere diferite (adică unul este 0 şi unul este 1, pentru a avea muchie în graful K_2;*

sau

- *$y_1 = y_2$ şi între $(x_1, x_2,..., x_d)$ şi $(x_1', x_2', ..., x_d')$ exista muchie în graful G_d.*

*În **prima situaţie**, este evident că d+1-uplele diferă printr-un singur element (şi anume y_1, respectiv y_2). În cea de**-a doua situaţie** avem $y_1 = y_2$ şi ştim că exista muchie între nodurile determinate de d-uple în graful G_d. Conform **ipotezei inductive**, avem că **în graful G_d există muchie numai între acele d-uple care diferă***

***printr-un singur element.** Deci d-uplele $(x_1, x_2,..., x_d)$ şi $(x_1', x_2', ..., x_d')$ diferă printr-un singur element iar $y_1 = y_2$. Aşadar, cele două d+1-uple diferă printr-un singur element, în cazul în care între acestea există muchie în graful G_{d+1}.*

*Am demonstrat deci că **în graful G_d există muchie numai între acele d-uple care diferă printr-un singur element.***

*Pentru a afla numărul de muchii, este necesar să verificăm pentru fiecare nod din G_d cu cine este legat, şi conform proprietăţii demonstrate mai sus, trebuie de fapt să determinăm toate nodurile ale căror coordonate diferă de cele ale nodului ales printr-un singur element. **Pentru un d-uplu** (corespunzător coordonatelor unui nod din G_d) avem **alte d d-uple care să difere de acesta printr-un singur element** (în locul fiecărui element din d-uplu se poate pune opusul său)**.***

Deci, pentru fiecare nod, obţinem d noduri adiacente cu el (adică gradul nodului este d, deci G_d este graf d-regulat).

*Ştiind de la punctul anterior că numărul de noduri este 2^d, şi ştiind că numărul de muchii este jumătate din suma gradelor tuturor nodurilor, se obţin $(d * 2^d)/2$ muchii, adică **$d * 2^{d-1}$** muchii în graful G_d.*

***3.** Pentru a determina **<u>diametrul grafului G_d</u>**, arătăm că **drumul minim dintre două noduri distincte ale grafului este numărul de elemente distincte din d-uplele corespunzătoare acestor noduri** (evident, acesta este un număr între 1 şi d).*

Fie $v=(x_1, x_2,..., x_d)$ şi $w=(x_1', x_2', ..., x_d')$, două d-uple care reprezintă coordonatele a două noduri din G_d. Presupunem că acestea diferă prin k elemente, $1 \le k \le d$.

Fie a = lungimea drumului minim de la v la w. Demonstrăm prin dublă inegalitate că a = k.

I. $a \ge k$:

Demonstrăm că orice drum de la v la w are lungimea $\ge k$. Implicit, drumul minim dintre cele două noduri va avea aceeaşi proprietate.

Reducere la absurd:

Presupunem că există un drum de la v la w de lungime l ($l < k$). Fie v, u_1,, u_2, ..., u_{l-1}, w nodurile de pe drumul de la v la w.

După cum am arătat mai sus, nu avem muchie decât între nodurile ale căror d-uple diferă printr-un singur element. Deci d-uplul corespunzător lui u_1 diferă de v printr-un singur element.

Există muchie între v şi u_1,, deci d-uplul lui v diferă de al lui u_1 prin exact un element.

Există muchie între u_i şi u_{i+1},, deci d-uplul lui u_i diferă de al lui u_{i+1} prin exact un element, pentru i de la 1 la l - 2.

Există muchie între u_l şi w,, deci d-uplul lui u_l diferă de al lui w prin exact un element.

*În consecinţă,d- uplul lui v diferă de al lui w prin **cel mult** l elemente.*

***Observaţie:** am menţionat mai sus că d-uplele a două noduri diferă prin cel mult p elemente şi nu prin exact p elemente deoarece:*

- *fiecare muchie transformă un singur 0 sau 1din d-uplul unei extremităţi într-un 1 sau 0 din d-uplul celeilalte exremităţi.*
- *este posibil ca după parcurgerea unui număr de muchii, un element din d-uplu să revină la valoarea iniţială.*

Exemplu:

Fie un drum care trece prin următoarele noduri:

… → 00100 → 00110 → 01110 → 01100 → …

Mersul de mai sus are toate nodurile şi toate muchiile distincte, deci este drum. D-uplul primului nod diferă printr-un singur element de d-uplul ultimului, deşi drumul dintre ele are lungime 3. Evident, un astfel de drum nu este un drum minim.

Am ajuns la o contradicţie întrucât ştiam că d-uplul lui w diferă prin k elemente faţă de al lui v şi $l < k$.

*Deci **presupunerea este falsă** şi am obţinut că **orice drum dintre v şi w are lungimea cel puţin k**; în consecinţă, drumul minim are aceeaşi proprietate $\Rightarrow$ **$a \geq k$.***

II. $a \leq k$.

Ideea este că pentru a putea ajunge de la v la w trebuie să schimbăm pe rând câte un element al lui v din cele k distincte, până ajungem la forma lui w. Avem nevoie de k schimbări şi deci de k muchii prin care să trecem.

Dacă am avea mai multe schimbări de valoare pentru acelaşi element (fiecărei astfel de schimbare îi corespunde o muchie, deci am avea implicit mai mult de k muchii), ar apărea una din următoarele situaţii:

- *la pasul i_r am schimbat elementul de pe poziţia j (din cele k ce trebuie schimbate) şi obţinem valoarea pe care dorim să o obţinem pe această poziţie*
- *la pasul i_s ($s > r$) schimbăm din nou elementul de pe poziţia j (am revenit la valoarea lui iniţială)*
- *la pasul i_t ($t > s$) schimbăm elementul de pe poziţia j şi ajungem din nou la valoarea dorită pe această poziţie.*

Această situaţie este redundantă, ea putând fi evitată, nemaitrecând prin paşii i_s şi i_t.

Sau

- *la pasul i_r am schimbat elementul de pe poziţia j (dintre elementele care nu trebuiau schimbate)*
- *la pasul i_s ($s > r$) schimbăm din nou elementul de pe poziţia j (şi revenim la valoarea pe care trebuie să o avem în final)*

Şi această situaţie este redundantă, putând fi ocolită dacă nu se mai trece prin paşii i_r şi i_s, care nu sunt absolut necesari.

În final am obţinut că drumul minim dintre două noduri ale grafului G_d este numărul de elemente distincte din d-uplele corespunzătoare acestor noduri.

Evident, numărul maxim de elemente care pot fi distincte între cele două noduri este d şi deci ***maximul lungimilor drumurilor minime din graf este d****.*

4. *Pentru a arăta că graful* ***G_d este bipartit*** *folosim următoarea proprietate:*
Un graf este bipartit dacă şi numai dacă nu admite circuite impare*.*
Alegem un circuit C şi fie $V(C) = \{u_1, u_2, \ldots, u_k\}$.
Muchiile acestui circuit sunt: u_1u_2, u_2u_3, …, $u_{k-1}u_k$, u_ku_1.
Este necesar ca u_1 şi u_k să difere printr-un singur element. Ideea este că în momentul în care trecem de la un nod u_i la nodul u_{i+1} şi schimbăm unul din elementele din u_1, este necesar ca la un pas ulterior să readucem acel element la valoarea sa iniţială. Deci ***pentru fiecare muchie (care schimbă elementul de pe poziţia i) trebuie să avem şi "complementara" ei*** *(care aduce la valoarea iniţială acest element). Trebuie să* ***avem număr par de muchii şi deci număr par de noduri****.*
În concluzie, nu putem avea circuite impare şi deci ***graful este bipartit****.*

5. *Vom demonstra prin dublă inegalitate că* ***$\alpha(G_d) = 2^{d-1}$****.*

I. $\alpha(G_d) \geq 2^{d-1}$.
Vom arăta că se poate construi o mulţime stabilă maximală S în G_d cu $|S| = 2^{d-1}$.

Folosind definirea codului Gray şi modul în care se formează muchiile în graful G_d, construim o mulţime S stabilă maximală astfel:
- *pornim de la un nod oarecare v;*
- *alegem un nod neadiacent cu v, cel mai apropiat de v (adică la două muchii distanţă); acest nod îl găsim folosind codul Gray: este al doilea element din şirul codului Gray, după v. Evident, acest nod nu este adiacent cu v, deoarece are două elemente distincte faţă de v şi deci drumul minim între ele este 2 (aşa cum am arătat la punctul 3);*
- *în continuare alegem nodurile din şirul codului Gray din 2 în 2 (vedem acest şir ca pe o listă circulară), până când ajungem din nou la nodul v.*

Lungimea acestui şir Gray este 2^d, iar noi alegem jumătate din elementele şirului, deci 2^{d-1} elemente.

Vom arăta că în momentul în care alegem un element din şir aflat la un element distanţă faţă de cel ales la pasul anterior, acesta nu este adiacent cu nici un element care se află deja în mulţimea S*.*

Reducere la absurd:
Presupunem că nodul v ales la pasul curent, urmând algoritmul de mai sus, este adiacent cu un nod aflat deja în mulţimea S; fie acel nod w. Deci reprezentările lui v şi w diferă printr-un singur element din d-uplu.
Între v şi w în şirul determinat de codul Gray avem număr impar de elemente *(pentru fiecare nod aflat printre acestea şi care a fost inclus în S, mai avem câte două elemente care îl încadrează, deci dacă între v şi w mai sunt s elemente incluse în S, în şirul codului Gray vom avea 2s+1 elemente între v şi w).*
Elementele de la v la w din şir determinat de codul Gray formează un lanţ *(fiecare diferă de cel anterior lui şi de cel de după el printr-un singur element din d-uplu, şi deci sunt legate prin muchii în G_d)* ***şi deci formează un circuit întrucât am presupus că avem muchie de la v la w****.*
Dar acest ***circuit are număr impar de elemente****, astfel încât am ajuns la o* ***contradicţie*** *deoarece la* ***punctul 4*** *am arătat că graful G_d este bipartit şi deci nu admite circuite impare.*
Deci presupunerea făcută este falsă.

Arătând acest lucru, am obţinut că ***mulţimea S construită ca mai sus este stabilă şi evident maximală*** *(dacă am mai alege un element care se află în şir între cele deja alese, el ar fi adiacent cu două noduri din mulţime, cu nodul de după el din şir şi cu nodul de înainte).*

Deci $\boldsymbol{\alpha(G_d) \geq 2^{d-1}}$.

Observaţie: *Acelaşi lucru se poate demonstra ţinând cont de problema 1 din tema 5, adică: un graf G este bipartit dacă şi numai dacă orice subgraf indus H al lui G satisface proprietatea* $\boldsymbol{2\alpha(H) \geq |H|}$.

Am arătat că G_d *este bipartit şi* $|G_d| = 2^d$. *Evident,* G_d *este subgraf indus în* G_d. *Aşadar* $\boldsymbol{2\alpha(G_d) = |G_d|}$, *adică* $\boldsymbol{\alpha(G_d) \geq 2^{d-1}}$.

II. $\boldsymbol{\alpha(G_d) \leq 2^{d-1}}$.

Reducere la absurd:

Presupunem că există o mulţime stabilă $S' \subseteq V(G_d)$ *astfel încât* $|S'| > 2^{d-1}$. *Fie* $|S'| = s$.

Aşa cum am arătat mai sus, graful G_d *este d-regulat, adică gradul oricărui vârf v din* $V(G_d)$ *este d. Implicit, pentru fiecare vârf v din S' există exact d muchii incidente cu acesta. S' fiind mulţime stabilă, toţi vecinii nodurilor din S' se găsesc în* $V(G_d) - S'$, *adică între mulţimile S' şi* $V(G_d) - S'$ *sunt exact* ***d*s muchii***. *Dar* $\boldsymbol{s \geq 2^{d-1}}$, *deci pentru numărul m de muchii dintre mulţimile S' şi* $V(G_d) - S'$ *avem relaţia:*

$$\boldsymbol{m \geq d * 2^{d-1}}.$$

Dar, aşa cum am arătat mai sus, $\boldsymbol{E(G_d) = d * 2^{d-1}}$.

Se obţine deci $\boldsymbol{m \geq E(G_d)}$, *ceea ce este o* ***contradicţie***.

Presupunerea făcută este deci falsă $\Rightarrow \boldsymbol{\alpha(G_d) \leq 2^{d-1}}$.

Întrucât $\boldsymbol{\alpha(G_d) \geq 2^{d-1}}$ ***şi*** $\boldsymbol{\alpha(G_d) \leq 2^{d-1} \Rightarrow \alpha(G_d) = 2^{d-1}}$.

2. *Un graf cu cel puţin 3 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 >2. Demonstraţi că un graf conex G = (V, E), cu cel puţin 3 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.*

Soluţie:

"$\Rightarrow$":

Vom presupune că ***graful G, cu*** $\boldsymbol{|V(G)| = n, n > 2}$, ***şi*** $\boldsymbol{G \neq K_n}$, ***este confidenţial conex***.

Trebuie să demonstrăm relaţiile 1 şi 2.

1. ***Demonstrăm că*** $\boldsymbol{\forall v}$, ***mulţimea*** $\boldsymbol{\overline{N}(v) \neq \phi}$ ***şi induce un graf conex***.

1.1 Demonstrăm că $\boldsymbol{\forall v \in V(G)}$, ***mulţimea*** $\boldsymbol{\overline{N}(v) \neq \phi}$.

Reducere la absurd:

Presupunem că $\exists$ ***v astfel încât*** $\boldsymbol{\overline{N}(v) = \phi}$.

Această presupunere ne conduce la faptul că nodul v are ***n-1*** *vecini. Întrucât graful G nu este complet,*

$\exists$ ***a,b*** *două noduri din V(G)* ***neadiacente*** *(şi evident* ***diferite de v*** *deoarece, altfel, ar exista muchie între cele două noduri conform alegerii lui v).*

Din definiţia grafurilor confidenţial conexe şi din faptul că nodurile a şi b alese anterior nu sunt adiacente, obţinem că ***există un drum de la a la b*** *care conţine* ***vârfuri interne****, acestea fiind* ***diferite de v şi de vecinii lui v****.*

Am obţinut că pe unul dintre drumurile de la a la b, $\exists\, w \in V(G)$, $w \neq v$ şi $w \notin N(v)$. Dar v era adiacent cu toate nodurile lui G deci am ajuns la o contradicţie.

Aşadar presupunerea de la care am plecat este falsă.

Am demonstrat că $\forall\, v \in V(G)$, ***mulţimea*** $\overline{N}(v) \neq \phi$.

1.2 Demonstrăm că $\forall\, v \in V(G)$, mulţimea $\overline{N}(v)$ induce un graf conex.

Demonstrăm că pentru $\forall\, a, b \in \overline{N}(v)$ ***există un drum de la a la b care are toate nodurile interne în $\overline{N}(v)$*** *(aceasta în cazul în care nodurile a şi b nu sunt adiacente; altfel, relaţia este automat adevărată).*

Din definiţia grafurilor confidenţial conexe, ştim că $\forall\, a, b \in V(G)$, există un drum de la a la b ale cărui noduri interne să fie diferite de v şi neadiacente cu acesta. Evident, nodurile aflate pe acest drum se află în $\overline{N}(v)$, şi deci aparţin grafului indus de $\overline{N}(v)$.

Am demonstrat de fapt că $\forall\, a \in \overline{N}(v)$, acesta este adiacent cu toate celelalte noduri din $\overline{N}(v)$ prin drumuri care se păstrează în $\overline{N}(v)$. Deci mulţimea ***$\overline{N}(v)$ induce un graf conex, $\forall\, v \in V(G)$.***

2. *Demonstrăm că* ***$\forall\, ab \in E(G)$, $a, b \in V(G)$, avem muchia ab inclusă într-un circuit C_4 indus în graful G, sau se află în mijlocul unui P_4 indus în graf.***

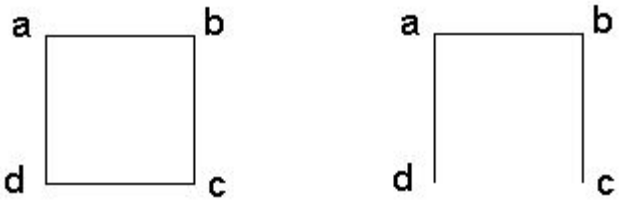
Această proprietate se reduce la următoarea relaţie:

- ***$\forall\, a \in V(G)$, $d_G(a) \geq 2$;***
- ***$\forall\, ab \in E(G)$, cu $a, b \in V(G)$, avem $N(a) - N(b) \neq \phi$.***

Demonstrăm echivalenţa celor două relaţii prin dublă implicaţie.

"$\Rightarrow$" : *Această implicaţie este evidentă.*

Avem următoarele două situaţii:



Întrucât graful G este conex, putem spune că orice nod al grafului are un nod adiacent cu el, cu care să formeze o muchie. Această muchie se află în unul din cele două cazuri, după cum am presupus la început, deci evident nodurile care formează muchia respectivă au gradul mai mare sau egal cu 2.

De asemenea, observăm că în ambele cazuri:

- ***$N(a) - N(b) \supseteq \{d\}$***
- ***$N(b) - N(a) \supseteq \{c\}$***

"$\Leftarrow$": *Considerăm o muchie* ***$ab \in E(G)$, cu $a, b \in V(G)$.***

Ştim că

- ***$N(a) - N(b) \neq \phi \Rightarrow \exists\, d \in V(G)$ astfel încât $d \in N(a)$ şi $d \notin N(b)$***
- ***$N(b) - N(a) \neq \phi \Rightarrow \exists\, c \in V(G)$ astfel încât $c \in N(b)$ şi $c \notin N(a)$***

Observăm că avem drum de la d la c, care cuprinde nodurile : d, a, b, c. Deoarece ***$d \notin N(b)$ şi $c \notin N(a)$*** *putem spune că singurele muchii între nodurile a, b, c, d sunt:* ***ac, ad, bc*** *şi* ***cd în cazul***

în care c şi d sunt adiacente**. Deci graful indus de cele 4 vârfuri conţine numai aceste muchii şi, **în cazul în care $\exists$ muchia cd, graful indus este un ciclu de lungime 4, altfel este un P_4.

*Am arătat că cele două relaţii sunt echivalente deci demonstraţia noastră de la **punctul 2** se reduce la a arăta că pentru un graf confidenţial conex sunt îndeplinite relaţiile:*

- $\forall\, a \in V(G)$, $d_G(a) \geq 2$;
- $\forall\, ab \in E(G)$, cu $a, b \in V(G)$, avem $N(a) - N(b) \neq \phi$.

Vom demonstra iniţial că $\forall\, a \in V(G)$, $d_G(a) \geq 2$.

*Ştim că graful nostru este conex, deci $\forall\, a \in V(G)$, **avem** $d_G(a) \geq 1$.*

Reducere la absurd:

*Presupunem că $\exists\, a \in V(G)$ **pentru care** $d_G(a) = 1$.*

*Fie **c nodul adiacent cu a**. Deoarece graful G are ordinul mai mare decât 2 putem spune că $\exists\, b \in V(G)$, astfel încât $b \notin N(a)$ şi $b \in N(c)$ (deoarece graful este conex, c nu poate avea gradu1).*

Conform definiţiei grafurilor confidenţial conexe, ar trebui să avem un drum de la a la b, care să aibă toate nodurile interne diferite de c şi de vecinii lui c. Însă, singurul drum de la a la b trece prin c, deoarece c este singurul nod adiacent cu a.

*Am ajuns la o contradicţie, deci **relaţia** $\forall\, a \in V(G)$, $d_G(a) \geq 2$ **este adevărată.***

*La acest pas, **demonstrăm că în ipoteza că graful G este confidenţial conex, cu ordinul mai mare decât 2 şi nu este complet, avem adevărată relaţia:** $\forall\, ab \in E(G)$, **cu** $a, b \in V(G)$, **avem** $N(a) - N(b) \neq \phi$.*

Considerăm o muchie $ab \in E(G)$, cu $a, b \in V(G)$.

Reducere la absurd:

Presupunem că $N(a) - N(b) = \phi$, ceea ce înseamnă de fapt că toţi vecinii lui a sunt şi vecini ai lui b.

*Alegem un nod c din V(G), astfel încât c să nu fie adiacent cu a (există un astfel de nod deoarece am arătat la **punctul 1** că $\forall\, a \in V(G)$, mulţimea $\overline{N}(a) \neq \phi$).*

*Din definiţia grafurilor confidenţial conexe, rezultă că **există drum de la a la c care să nu-l conţină pe b, sau noduri adiacente cu acesta**. Întrucât între a şi c nu există muchie (conform alegerii lui c), pentru a ajunge de la a la c trebuie să trecem mai întâi din a printr-un nod intermediar, vecin cu a, pe care îl putem nota cu v. Deci $v \in N(a)$ şi, conform definiţiei, $v \notin N(b)$.*

Dar din presupunerea iniţială obţinem tocmai că $v \in N(b)$. Am ajuns deci la o contradicţie, aşa că presupunerea de la care am plecat este falsă.

*Am demonstrat că pentru un graf confidenţial conex avem adevarate cele două relaţii şi deci avem adevărată şi relaţia de la care am plecat: $\forall\, ab \in E(G)$, $a, b \in V(G)$, **avem muchia ab inclusă într-un circuit C_4 indus în graful G, sau se află în mijlocul unui P_4 indus în graf.***

*Conform **punctelor 1 şi 2, implicaţia directă este adevărată.***

"$\Leftarrow$": Presupunem relaţiile 1 şi 2 adevărate şi demonstrăm că graful este confidenţial conex.

*Alegem trei noduri: a, b, c ale grafului G. Trebuie să demonstrăm că **există un drum de la a la b care să nu conţină printre punctele sale interne (dacă există) nodul c sau vecini ai nodului c**.*

*În cazul în care există muchie de la a la b, avem un drum de la a la b fără noduri interne şi deci **proprietatea este respectată.***

Când nodurile a şi b sunt neadiacente, avem următoarele trei situaţii:

1. *Cazul în care $a, b \notin N(c)$.*

Deci a şi b sunt noduri ale grafului indus de nevecinii lui c. Conform ***relaţiei 1 din ipoteză****, care spune că graful indus de nevecinii unui nod oarecare al grafului G este conex, putem spune că* ***între nodurile a şi b există un drum care rămâne în graful indus****. Deci există un drum de la a la b care are toate nodurile interne nevecini ai lui c, şi diferite de acesta.*

2. *Când unul dintre nodurile a şi b este vecin al lui c iar celălalt nod aparţine mulţimii nevecinilor lui c (considerăm* $\mathbf{a \in N(c)}$ ***şi*** $\mathbf{b \notin N(c)}$*; celălalt caz este similar).*
În acest caz ***folosim relaţia 2 din ipoteză, în forma ei echivalentă****. Ştim că* $\forall$ $\mathbf{a \in V(G)}$, $\mathbf{d_G(a) \geq 2}$ *deci a admite un alt nod adiacent cu el în afară de c. Totodată, pentru* $\forall$ $\mathbf{ab \in E(G)}$***, cu*** $\mathbf{a, b \in V(G)}$***, avem*** $\mathbf{N(a) - N(b) \neq \phi}$***.*** *Folosim această relaţie pentru muchia* ***ac*** *şi obţinem* $\mathbf{N(a) - N(c) \neq \phi}$***.*** *Deci a are noduri adiacente care nu sunt adiacente cu c. Fie* $\mathbf{d \in N(a)}$*, un astfel de nod (*$d \notin \mathbf{N(c)}$*,* $\mathbf{d \neq b}$ *deoarece* $\mathbf{b \notin N(a)}$*). Conform* ***cazului 1*** *al demonstraţiei, putem spune că* ***există un drum de la d la b care să aibă toate nodurile interne diferite de c şi de vecinii lui c.*** *La acest drum adăugăm* ***muchia ad*** *şi obţinem un drum de la a la b care respectă proprietatea întrucât drumul de la d la b respectă această proprietate şi nodul d este neadiacent cu b.*

3. *Cazul în care a şi b sunt vecini ai lui c (*$\mathbf{a, b \in N(c)}$ *).*
Folosim relaţia 2 din ipoteză, în forma ei echivalentă*. Ştim că* $\forall$ $\mathbf{a \in V(G)}$, $\mathbf{d_G(a) \geq 2}$ *deci a admite un alt nod adiacent cu el în afară de c. Totodată, pentru* $\forall$ $\mathbf{ab \in E(G)}$***, cu*** $\mathbf{a, b \in V(G)}$***, avem*** $\mathbf{N(a) - N(b) \neq \phi}$***.*** *Folosim această relaţie pentru muchia* ***ac*** *şi obţinem* $\mathbf{N(a) - N(c) \neq \phi}$***.*** *Deci a are noduri adiacente care nu sunt adiacente cu c. Fie* $\mathbf{d \in N(a)}$*, un astfel de nod (*$d \notin \mathbf{N(c)}$*). Analog, putem spune că există un nod* $\mathbf{e \in N(b)}$ ***astfel încât*** $e \notin \mathbf{N(c)}$*.*
Dacă ***d=e****, drumul de la a la b căutat este* ***a, d, b,*** *care respectă proprietatea întrucât nodul său intern, şi anume d, nu este adiacent cu c.*
Altfel*, conform* ***cazului 1*** *al demonstraţiei,* ***există un drum de la d la e care să aibă toate nodurile interne diferite de c şi de vecinii lui c.*** *Pentru a obţine un drum de la a la b trebuie să adăugăm* ***muchiile ad şi be****. Acesta este un* ***drum care respectă proprietatea*** *întrucât* ***d*** *şi* ***e*** *nu sunt adiacente cu c, din modul în care au fost alese, iar restul nodurilor sunt şi ele* ***nevecini ai lui c****, deoarece fac parte din drumul de la* ***d*** *la* ***e*** *care respecta proprietatea.*

Am demonstrat că indiferent de alegerea nodurilor a, b şi c obţinem un drum de la a la b care să aibă nodurile interne (dacă acestea există) nevecini ai lui c şi diferite de c.

În urma demonstrării celor două implicaţii, putem spune că un graf conex cu cel puţin 3 vârfuri şi care nu este complet ***este confidenţial conex dacă şi numai dacă îndeplineşte cele două relaţii.***

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 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 adauga arcele* $\overline{v_i}w_i$ *şi* $\overline{w_i}v_i$ *(folosind, evident, convenţia referitoare la dubla negare). Demonstraţi că există o atribuire a valorilor de adevăr 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. Argumentaţi complexitatea timp de O(n+m) pentru testarea proprietăţii de mai sus.*

Soluţie:

„ ⇒” *Arătăm că, dacă există o atribuire a valorilor de adevăr şi fals pentru variabilele booleene astfel încât fiecare clauză din C să fie adevărată, atunci, pentru orice i $\in \{1, ..., n\}$, $\overline{x_i}$ şi x_i sunt în componente tare conexe diferite.*

Reducere la absurd:

*Presupunem că există i $\in \{1, ..., n\}$ astfel încât $\overline{x_i}$ şi x_i să se găsească în aceeaşi componentă tare conexă. Aceasta înseamnă că **există drum d_1 în G de la** $\overline{x_i}$ **la** x_i, şi **există drum d_2 în G de la** x_i **la** $\overline{x_i}$. (din definiţia componentelor tare conexe într-un digraf).*

$\boldsymbol{d_1}$: $\overline{x_i}$, $\overline{x_i}\,u_1$, u_1, u_1u_2, u_2, ... , u_{k-1}, $u_{k-1}u_k$, u_k, u_kx_i, x_i.

$\boldsymbol{d_2}$: x_i, x_iv_1, v_1, v_1v_2, v_2, ... , v_{j-1}, $v_{j-1}v_j$, v_j, $v_j\overline{x_i}$, $\overline{x_i}$.

Din modul de construcţie a grafului G rezultă:

$uv \in E \Leftrightarrow (\overline{u} \vee v) \in C$ *(sau* $(v \vee \overline{u}) \in C$*, ţinând cont de comutativitatea lui* $\vee$*)*

*Aşadar, arcelor din drumul **d_1** le corespund următoarele clauze din C:*

$$C_{h1} = x_i \vee u_1;$$
$$C_{h2} = \overline{u_1} \vee u_2;$$
$$...$$
$$C_{hk-1} = \overline{u_{k-1}} \vee u_k;$$
$$C_{hk} = \overline{u_k} \vee x_i.$$

*respectiv arcelor din drumul **d_2** le corespund următoarele clauze din C:*

$$C_{p1} = \overline{x_i} \vee v_1;$$
$$C_{p2} = \overline{v_1} \vee v_2;$$
$$...$$
$$C_{pj-1} = \overline{v_{j-1}} \vee v_j;$$
$$C_{pj} = \overline{v_j} \vee \overline{x_i}.$$

*Am pornit de la ipoteza că există o atribuire a valorilor de adevăr şi fals pentru variabilele booleene astfel încât **toate clauzele din C să fie adevărate**.*

- ***<u>dacă x_i = true</u>*** **⇒** $\overline{x_i}$ = ***false*** *(din principiul noncontradicţiei).*
  *Vom demonstra prin **inducţie** că v_s = true, $\forall s \in \{1, ..., j\}$.*
  I. C_{p1} = ***true*** *şi* $\overline{x_i}$ = ***false*** $\Rightarrow$ v_1 = ***true***.
  II. *Presupunem v_s = **true** $\forall s \in \{1, ..., m\}$. Demonstrăm că v_{m+1} = true.*
  C_{pm} = ***true*** **⇒** $\overline{v_m} \vee v_{m+1}$ = ***true.***
  În plus, $\overline{v_m}$ =***false*** *(deoarece v_m = true)* $\Rightarrow$ v_{m+1} = ***true.***
  *Aşadar, **v_j = true**, deci $\overline{v_j}$ = false, şi cum şi $\overline{x_i}$ = false, se obţine $\overline{v_j} \vee \overline{x_i}$ = false, adică C_{pj} = **false**.*

- ***<u>dacă x_i = false</u>:***
  *Vom demonstra prin **inducţie** că u_s = true, $\forall s \in \{1, ..., k\}$.*

I. C_{h1} = **true** *şi* x_i = **false** $\Rightarrow u_1$ = **true**.

II. *Presupunem* u_s = **true** $\forall s \in \{1, \ldots, m\}$. *Demonstrăm că* u_{m+1} = *true.*

C_{hm} = **true** $\Rightarrow$ $\overline{u_m} \vee u_{m+1}$ = **true.**

În plus, $\overline{u_m}$ =**false** *(deoarece* u_m = *true)* $\Rightarrow$ u_{m+1} = **true.**

Aşadar, u_k = *true, deci* $\overline{u_k}$ = *false, şi cum şi* x_i = *false, se obţine* $\overline{u_k} \vee x_i$ = *false, adică* C_{hk} = **false**.

Se observă că, indiferent de valoarea atribuită lui x_i*, dacă* x_i *şi* $\overline{x_i}$ *se află în aceeaşi componentă tare conexă* ($\forall i \in \{1, \ldots, n\}$)*, atunci* ***există cel puţin o clauză falsă în C****, ceea ce contrazice ipoteza.*

Aşadar, presupunerea făcută este falsă $\Rightarrow$ ***pentru orice i*** $\in \{1, \ldots, n\}$, $\overline{x_i}$ ***şi*** x_i ***sunt în componente tare conexe diferite.***

„$\Leftarrow$” *Arătăm că, dacă, pentru orice i* $\in \{1, \ldots, n\}$, $\overline{x_i}$ *şi* x_i *sunt în componente tare conexe diferite, atunci există o atribuire a valorilor de adevăr şi fals pentru variabilele booleene astfel încât fiecare clauză din C să fie adevărată.*

Observaţie:

$a \vee b$ *este echivalent cu* $\left(\overline{a} \Rightarrow b\right) \wedge \left(\overline{b} \Rightarrow a\right)$.

Demonstraţie: $\left(\overline{a} \Rightarrow b\right) \wedge \left(\overline{b} \Rightarrow a\right) \leftrightarrow \left(\overline{\overline{a}} \vee b\right) \wedge \left(\overline{\overline{b}} \vee a\right) \leftrightarrow \left(a \vee b\right) \wedge \left(b \vee a\right) \leftrightarrow a \vee b$.

Aşadar, fiecărui arc uv îi corespunde implicaţia $u \Rightarrow v$*. Evident,*

$$(u \Rightarrow v = \textbf{true}) \leftrightarrow (\overline{v} \Rightarrow \overline{u} = \textbf{true}) \leftrightarrow (\overline{u} \vee v = \textbf{true}).$$

Deci, pentru ca o clauză $C = a \vee b$ *să fie adevărată, este necesar şi suficient ca implicaţiile reprezentate de arcele corespunzătoare lui C în G să fie adevărate.*

În consecinţă, ***pentru ca mulţimea de clauze C să fie satisfiabilă,*** *este necesar şi suficient* ***să existe o asignare astfel încât implicaţiile corespunzătoare tuturor arcelor să fie adevărate****.*

Dacă toate arcele din G sunt de forma:

- *sursa 0 şi destinaţia 0 sau 1;*
- *sursa 1 şi destinaţia 1.*

atunci pentru această asignare C este satisfiabilă.

Vom demonstra implicaţia de mai sus arătând că, în condiţiile date, se poate construi o asignare pentru care mulţimea de clauze C este satisfiabilă.

function *ConstruieşteAsignare(G)*
begin
  for *(fiecare i din V)*
    asignare(i) ← -1   ***//iniţial toate nodurile au valoare nedefinită***
  while *(există noduri cu valoare nedefinită)*
    /*Se alege un nod a dintre nodurile nevizitate din mulţimea V, corespunzător unui literal x_i ***sau*** $\overline{x_i}$ ***. Conform ipotezei,*** x_i ***şi*** $\overline{x_i}$ ***nu se găsesc în aceaşi componentă tare conexă. Deci nu există drum de la*** a ***la*** $\overline{a}$***, sau nu există drum de la*** $\overline{a}$ ***la*** a***. Se alege acel nod dintre cele două din care nu este accesibil celălalt. Dacă ambele au această proprietate, se poate face oricare alegere.*/***

```
            a = AlegeNodNevizitat(V)
            if (ā este accesibil din a)
            then u ← ā
            /*dacă ā este accesibil din a, atunci conform ipotezei, a sigur nu este accesibil din ā */
            else u ← a
            asignare(u) ← 1
            asignare(ū) ← 0
            ParcurgereŞiMarcareDFS(u)
        return asignare
end

procedure ParcurgereŞiMarcareDFS(u)
begin
        for fiecare w din lista de adiacenţă exterioară a lui u do
            if (asignare(w) = -1)
            then
                    asignare(w) ← 1
                    asignare(w̄) ← 0
                    ParcurgereŞiMarcareDFS(w)
end
```

Corectitudinea algoritmului

Deoarece în graful G $\overline{x_i}$ şi x_i sunt în componente tare conexe diferite (conform ipotezei), cel puţin unul dintre aceste două noduri nu este accesibil din celălalt.

*Fie **u** un nod cu această proprietate.*

→ *Vom arăta că **w este accesibil din u dacă şi numai dacă $\overline{w}$ nu este accesibil din u**.*

Reducere la absurd:

Fie w un nod accesibil din u. Presupunem că şi $\overline{w}$ este accesibil din u.

Aceasta înseamnă că există un drum de la u la w, format din arcele uv_1, v_1v_2, ..., v_kw, şi există un drum de la u la $\overline{w}$, format din arcele uu_1, u_1u_2, ..., $u_p\overline{w}$.

Dar, din modul de construcţie a grafului G reiese că există în G şi arcele $\overline{w}\,\overline{v_k}$, ..., $\overline{v_2}\,\overline{v_1}$, $\overline{v_1}\,\overline{u}$, (adică există un drum de la $\overline{w}$ la $\overline{u}$), respectiv arcele $w\overline{u_p}$, ..., $\overline{u_2}\,\overline{u_1}$, $\overline{u_1}\,\overline{u}$ (adică există un drum de la w la $\overline{u}$). Aşadar, există un drum de la u la $\overline{u}$ format din arcele uv_1, v_1v_2, ..., v_kw, $w\overline{u_p}$, ..., $\overline{u_2}\,\overline{u_1}$, $\overline{u_1}\,\overline{u}$, ceea ce contrazice ipoteza.

Presupunerea făcută este deci falsă ⇒ $\overline{w}$ nu este accesibil din u.

Implicaţia inversă se demonstrează similar.

→ *Conform observaţiei de mai sus, C este satisfiabilă dacă şi numai dacă implicaţiile reprezentate de toate arcele grafului G sunt adevărate.*

*În consecinţă, dacă unui literal (reprezentat de nodul x) i se atribuie valoarea **true**, atunci este necesar ca toate nodurile accesibile din x să reprezinte literali cu valoarea **true.** Dacă literalul reprezentat de nodul x are valoarea **false**, indiferent de valoarea booleană a literalilor reprezentaţi de nodurile din lista de adiacenţă exterioră a lui x, implicaţiile corespunzătoare sunt adevărate.*

Pentru a "construi" o asignare astfel încât C să fie satisfiabilă se parcurg următorii paşi:

1. *Se alege un nod u din V cu proprietatea că $\overline{u}$ nu este accesibil din u. Literalului corespunzător lui u i se atribuie valoarea* ***true****. (Evident, literalului corespunzător lui $\overline{u}$ i se atribuie valoarea* ***false****).*
2. *Printr-o parcurgere DFS pornind din nodul u, se atribuie literalilor corespunzători tuturor nodurilor w accesibile din u (şi cărora nu li s-a atribuit încă o valoare) valoarea* ***true*** *(totodată, literalilor corespunzători nodurilor $\overline{w}$ li se atribuie valoarea* ***false****).*
3. *Atâta timp cât mai există noduri nevizitate, se alege dintre acestea un nou nod v cu aceeaşi proprietate ca şi u la pasul 1 şi se vor repeta paşii 1 şi 2 pentru nodul v.*

La ***pasul 1*** *vom alege dintre x_i şi $\overline{x_i}$ acel nod u din care nu este accesibil celălalt şi i se atribuie valoarea* ***true****. Acest lucru este permis de următoarele aspecte:*

- *funcţia implicaţie este tranzitivă; aşadar, orice drum de la* ***a*** *la* ***b*** *în G este echivalent cu implicaţia* ***a*** $\Rightarrow$ ***b****.*
- *neexistând drum în G de la u la $\overline{u}$, nu vom avea nici implicaţia u $\Rightarrow$ $\overline{u}$, care este falsă.*
- *este posibil să existe drum de la $\overline{u}$ la u; dacă acesta există, atunci, din tranzitivitatea implicaţiei, vom avea $\overline{u}$ $\Rightarrow$ u, care este adevărată, deoarece $\overline{u}$ este fals.*

Prin urmare, în urma asignărilor de la pasul 1 nu se pot obţine implicaţii false.

La ***pasul 2****, tuturor nodurilor w accesibile din u li se atribuie valoarea* ***true****. Aşadar, toate arcele parcurse prin DFS la acest pas vor avea ambele extremităţi* ***true****, implicaţiile corespunzătoare fiind, în consecinţă, adevărate. Aşa cum am arătat mai sus, dacă w este acesibil din u, atunci $\overline{w}$ nu este accesibil din u, deci nu se vor face asignări contradictorii.*

În paralel, nodurilor $\overline{w}$ li se atribuie valoarea ***false****. Tuturor arcelor care au ambele extremităţi astfel de noduri $\overline{w}$ le corespund implicaţiile 0 $\Rightarrow$ 0, care sunt adevărate. De asemenea, tuturor drumurilor cu destinaţia $\overline{u}$ le corespund implicaţiile 0 $\Rightarrow$ 0.*

Dacă w este accesibil din u (i se atribuie valoarea ***true****) atunci nu contează din ce alte noduri mai este accesibil w, întrucât* ***true***$\Rightarrow$ ***true*** *şi* ***false***$\Rightarrow$ ***true****.*

Aşadar, în urma asignărilor de la pasul 2 se obţin numai implicaţii adevărate.

Un nod ***v*** *ales la* ***pasul 3*** *se poate afla în unul din următoarele cazuri:*

- ***∃ un nod v' deja vizitat, cu valoarea true, a.î. v' este accesibil din v****; fie w acel descendent al lui v care este părintele lui v'. Deoarece v' este ales la pasul 3, atât el cât şi toţi descendenţii lui nevizitaţi încă vor primi valoarea* ***true****. Deci arcul wv' va avea sursa* ***true*** *şi destinaţia* ***true****, implicaţia* ***w*** $\Rightarrow$ ***v'*** *fiind* ***adevărată****.*

  Totodată, arcul $\overline{v'}\ \overline{w}$ va avea sursa ***false*** *şi destinaţia* ***false****, deci implicaţia $\overline{v'} \Rightarrow \overline{w}$ este* ***adevărată.***

  (cazul ca v' să fie accesibil din v este imposibil, deoarece acesta ar fi fost descoperit prin parcurgerea DFS din v, deci nu ar fi putut rămâne între nodurile nevizitate).
- *v este accesibil dintr-un nod $\overline{v'}$ căruia i s-a atribuit anterior valoarea* ***false****. Din construcţia grafului G, constatăm că acest lucru este echivalent cu faptul că v' este accesibil din $\overline{v}$. Deoarece există drumuri de la $\overline{v'}$ la v şi de la $\overline{v}$ la v', din tranzitivitatea funcţiei implicaţie vom obţine $\overline{v'}$ $\Rightarrow$ v, respectiv $\overline{v}$ $\Rightarrow$ v'. Lui v i se atribuie* ***true****, iar $\overline{v'}$ era deja* ***false****, deci evaluarea acestor implicaţii este: 0 $\Rightarrow$ 1, respectiv 0 $\Rightarrow$ 1, ambele fiind deci* ***adevărate*** *în această asignare.*

(cazul ca $\overline{v'}$ să fie accesibil din v este imposibil, deoarece acesta ar insemna că $\overline{v}$ este accesibil din v', deci $\overline{v}$ fi fost descoperit prin parcurgerea DFS din v', adică nici v, nici $\overline{v}$ nu ar fi putut rămâne între nodurile nevizitate)

- ***nu există drumuri între v şi nici un alt nod vizitat anterior****. În acest caz, asignările făcute până acum nu influenţează asignările care se fac la acest pas. Implicaţiile corespunzătoare arcelor parcurse prin DFS vor fi* ***adevărate****, conform celor menţionate la paşii 1 şi 2.*
- ***Nu*** *este posibil ca dintr-un nod* ***v*** *ales la pasul 3 să ajungem într-un nod* ***v'*** *marcat deja cu* ***false****, deoarece:*
  - *dacă există drum din v în în v', atunci există drum din $\overline{v'}$ în $\overline{v}$*
  - *v' este marcat cu* ***false****, atunci $\overline{v'}$ este marcat cu 1*
  - *$\overline{v}$ este accesibil din $\overline{v'}$, atunci el a fost marcat la pasul 2, deci $\overline{v}$ este marcat cu* ***true*** *$\Rightarrow$ v este marcat cu* ***false****, ceea ce contrazice faptul că v nu este vizitat.*

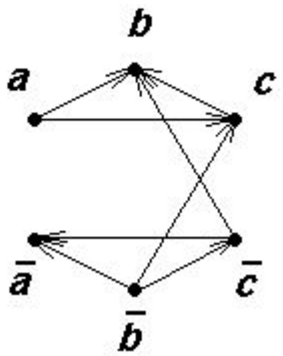
Aşadar, noile mulţimi nu intră în conflict cu mulţimile deja stabilite.

Am arătat că algoritmul de mai sus construieşte o asignare astfel încât implicaţiile corespunzătoare tuturor arcelor din G să fie adevărate. Aşadar, conform observaţiei de mai sus, pentru asignarea construită de algoritm, ***C este satisfiabilă****.*

Exemplu *de funcţionare a algoritmului de mai sus:*

Fie C = $\{\overline{a} \vee b, b \vee c, c \vee \overline{a}, \overline{c} \vee b\}$.

Graful G asociat acestei mulţimi de clauze este:



Se observă că, în G, u şi $\overline{u}$ se găsesc în componente tare conexe diferite, $\forall u \in \{a, b, c, \overline{a}, \overline{b}, \overline{c}\}$. În plus, nu există drum de la a la $\overline{a}$.

Aşadar, a = ***true****, $\overline{a}$ =****false****.*

Se face apoi parcurgerea DFS din nodul a, atribuindu-se tuturor nodurilor accesibile din a valoarea ***true****, adică: b =* ***true****, $\overline{b}$ =****false****, c =* ***true****, $\overline{c}$ =****false.***

Pentru această asignare avem:

$\overline{a} \vee b$ = false $\vee$ true = ***true****.*

$b \vee c$ = true $\vee$ true = ***true****.*

$c \vee \overline{a}$ = true $\vee$ false = ***true****.*

$\overline{c} \vee b$ = false $\vee$ true = ***true****.*

Aşadar, C este satisfiabilă pentru această asignare.

Demonstrăm că se poate testa dacă C este satisfiabilă în timpul O(m + n).

Am arătat că C este satisfiabilă dacă şi numai dacă în graful G $\overline{x_i}$ şi x_i sunt în componente tare conexe diferite. Este suficient deci să testăm dacă există în G $\overline{x_i}$ şi x_i în aceeaşi componentă tare conexă.

Împărţirea grafului în componente tare conexe se face în timpul O(n + m), întrucât sunt necesare două parcurgeri DFS ale grafului. Verificarea dacă $\overline{x_i}$ şi x_i sunt în aceeeaşi componentă tare conexă se face în timpul O(n), deoarece fiecare nod este interogat o singură dată.

Aşadar, complexiatea unui algoritm de verificare a proprietăţii de mai sus este

*O(m + n) + O(n) = **O(m + n)**.*

***Algoritmul** care poate face verificarea proprietăţii este:*

***procedure** CompTareConexe(G)*
begin
DFSComponenteTareConexe(G) ***//timp O(m+n)***
Calculează G^T
// $G^T = (V, E^T)$, unde $uv \in E^T \Leftrightarrow vu \in E$
//timp O(m + n)
DFSComponenteTareConexe(G^T) ***//timp O(m+n)***
/*în care, în bucla for principală, vârfurile sunt vizitate în ordinea descrescătoate a timpilor finali de vizitare*/
/*fiecare arbore T calculat la acest pas este o componentă tare conexă*/
***return all** T*
/*T calculaţi la pasul anterior*/
end

***procedure** DFSComponenteTareConexe(G)*
begin
for all** v **in** V **do
culoare(v) ← 0
părinte(v) ← 0 //in părinte se memorează „pădurea” DFS
k ← 0
timp ← 0
for all** v **in** V **do
***if** (culoare (v) = 0)*
***then** DFSComponenteTareConexeRECURSIV(v)*
end

***procedure** DFSComponenteTareConexeRECURSIV(v)*
begin
timp ← timp + 1
timpVizită(i) ← timp
culoare(v)
for** (fiecare w din lista de adiacenţă exterioara a lui v) **do
***if** (culoare(w) = 0)*
then
părinte(w) ← v
DFSComponenteTareConexeRECURSIV(w)
timp ← timp + 1

```
        timpFinal←timp
end

function VerificareComponenteConexe(G)
begin
        for (all v in V) do
                if (v şi v̄ sunt în acelaşi T)
                        return FALSE
        return TRUE
end.
```

GÎRDEA MARTA
URICARU RALUCA
A34
echipa 21

TEMA NR. 7
15 aprilie 2003

1. ***Gossip Problem****. Într-un grup de m "doamne", fiecare cunoaşte o parte dintr-o bârfă pe care celelalte nu o cunosc. Ele comunică prin telefon şi orice aplel telefonic între orice două doamne are ca efect faptul că fiecare dintre 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.

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 elemente din şirul considerat.

Soluţie:

b) Modelarea problemei în limbajul teoriei grafurilor

Problemei de mai sus i se asociază un graf ***G = (V, E)*** *construit astfel:*

- *V = {0, 1, …, n-1}, adică fiecărei doamne îi corespunde câte un nod;*
- *iniţial G = N_n; (adică se porneşte cu graful nul, E = Φ);*
- *un telefon între doamnele i şi j se reprezintă prin adăugarea muchiei* ***ij*** *în E; fiecare muchie este etichatată cu un număr, începând cu* ***1****, în ordinea în care se dau telefoanele.*
- *pentru ca informaţia pe care o cunoşte iniţial doamna i să fie cunoscută de doamna j,* ***este necesar şi suficient*** *să existe în graful G un* ***drum „crescător"*** *D de la i la j (prin* ***drum crescător*** *se înţelege un drum D cu V(D) = {i, v_1, …, v_{m-1}, j} şi E(D) = {e_1, e_2, …, e_m}, căruia îi corespunde secvenţa*

  $$\mathbf{D: i, e_1, v_1, e_2, v_2, \ldots, e_{m-1}, v_{m-1}, e_m, j.}$$

  astfel încât

  $$label(e_1) < label(e_2) < \ldots < label(e_m)$$

  unde ***label*** *este eticheta asociată fiecărei muchii aşa cum am arătat mai sus (se poate considera că telefoanele nu se dau simultan, deci nu putem avea două muchii cu aceeaşi etichetă, prin urmare relaţia de ordine este strictă).*

  Argumentaţie:

  „ ⇐" *Dacă există un drum crescător de la i la j, atunci doamna j cunoaşte informaţiile deţinute iniţial de doamna i.*

  Fie D_{ij} un drum crescător de la i la j în graful G.

  Cazul 1: *D_{ij} este reprezentat de o singură muchie (deoarece orice mulţime cu un element poate fi considerată lanţ, drumul format dintr-o singură muchie este un drum crescător). Dar muchia* ***e = ij*** *există în E dacă şi numai dacă s-a produs convorbirea telefonică între doamna i şi doamna j (din modul de construcţie a grafului G), deci informaţia de la doamna* ***i*** *s-a propagat la doamna* ***j****.*

Cazul 2: *$V(D_{ij}) = \{i, v_1, \dots, v_{m-1}, j\}$ şi $E(D_{ij}) = \{e_1, e_2, \dots, e_m\}$, cu $m > 1$ şi*

$$D: v_0 = i, e_1, v_1, e_2, v_2, \dots, e_{m-1}, v_{m-1}, e_m, j = v_m$$

$$label(e_1) < label(e_2) < \dots < label(e_m).$$

Aşadar, la momentul producerii convorbirii dintre v_i şi v_{i+1}, v_i deţinea deja informaţia lui v_0, $\forall i \in \{0, \dots, m-1\}$. Aşadar, în urma acestui lanţ de convorbiri, informaţia de la $v_0 = i$ se propagă prin toate nodurile din acest lanţ până la $v_m = j$.

„ ⇒” *Dacă doamna j cunoaşte informaţiile deţinute iniţial de doamna i, atunci există un drum crescător de la i la j.*

Doamna i cunoaşte informaţiile deţinute iniţial de doamna j, deci a existat un şir de telefoane între doamnele i şi k_1, k_1 şi k_2, ... , k_{n-1} şi k_n, k_n şi j, date în această ordine, astfel încât informaţiile deţinute iniţial de doamna i să ajungă la doamna j. Acestor apeluri le corespund în graf muchiile ik_1, k_1k_2, ... , $k_{n-1}k_n$, k_nj, etichetate astfel încât

$$label(ik_1) < label(k_1k_2) < \dots < label(k_{n-1}k_n) \dots < label(k_nj)$$

(reiese din modul de construcţie a grafului). Aşadar există un drum D crescător de la i la j, căruia îi corespunde următoarea secvenţă:

$$D: i, ik_1, k_1, k_1k_2, k_2, \dots, k_{n-1}k_n, k_n, k_nj, j.$$

Observaţie: Afirmaţia este valabilă şi pentru cazul când $n = 0$, adică $V(D) = \{i, j\}$ şi $E(D) = \{ij\}$, deoarece drumul format dintr-o singură muchie poate fi considerat drum crescător.

Pentru a rezolva „problema bârfei” trebuie deci să se construiască un graf $G = (V, E)$, cu muchii etichetate, de dimensiune minimă, astfel încât, $\forall i \in V$, $\forall j \in V - \{i\}$, există un drum „crescător” de la i la j.

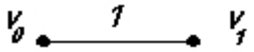
Observaţie: *Este necesar ca un astfel de graf să fie* ***conex***.

a) Demonstrăm că dimensiunea minimă a unui astfel de graf este:

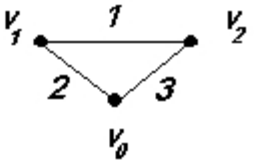
- ***1 pentru $|G| = 2$;***
- ***3 pentru $|G| = 3$;***
- ***$2|G| - 4$ pentru $|G| \geq 4$.***

I. Arătăm că se pot construi grafuri cu proprietăţile cerute care să aibă dimensiunile precizate mai sus.

Pentru $|G_2| = 2$: $G_2 = K_2$.

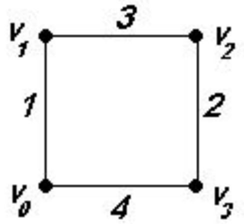


Pentru $|G_3| = 3$: $G_3 = K_3$.

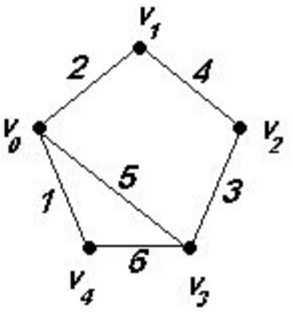


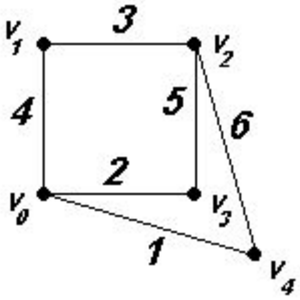
Pentru $|G| \geq 4$ se demonstrează prin ***inducţie*** *după numărul de noduri că se poate construi un graf G cu proprietăţile de mai sus şi cu dimensiunea $2|G| - 4$.*

Pasul I: *$|G_4| = 4$: $G_4 = C_4$; $|E(G_4)| = 4 = 2*4 - 4$.*



$|G_5| = 5$: $\qquad |E(G_5)| = 6 = 2*5 - 4$.





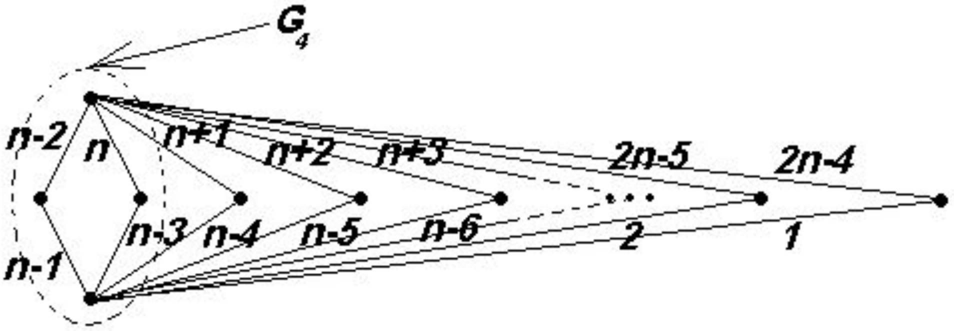
Pasul II: *Presupunem afirmaţia adevărată pentru* $|G| \leq n$. *Demonstrăm pentru* $|G| = n + 1$.
Graful G_{n+1} *se poate construi din graful* G_n *astfel:*

- *la mulţimea vârfurilor din* G_n *se adaugă un nou nod, fie acesta n. Astfel,* $V(G_{n+1}) = \{0, 1, ..., n\text{-}1, n\}$.
- *prima muchie inserată în* $E(G_{n+1})$ *la construirea grafului* G_{n+1} *este muchia* $\boldsymbol{(n, i_0)}$, *cu* $\boldsymbol{i_0}$ *oarecare din* $\{0, 1, ..., n\text{-}1\}$, *care va fi etichetată cu* ***1****;*
- *se introduc apoi în* $E(G_{n+1})$ *muchiile dintre nodurile 0, 1, ..., n-1, în ordinea în care au fost introduse în* G_n, *etichetate* ***de la 2 la 2n – 3*** *(deoarece am presupus la pasul inductiv că* G_n *are dimensiunea 2n – 4);*
- *din modul de construcţie până la acest pas a lui* G_{n+1}, *reiese că subgraful indus în* G_{n+1} *de mulţimea {0, 1, ..., n-1 } este chiar* G_n, *deci are proprietatea că,* ***∀i ∈ {0, 1, …, n-1 }, ∀j ∈ {0, 1, …, n-1 } – {i}, există un drum „crescător" de la i la j.***
  În particular, $\forall j \in \{0, 1, ..., n\text{-}1\} - \{i_0\}$, *există un drum crescător de la* i_0 *la j. Întrucât unica legătură a nodului* ***n*** *cu nodurile din acest subgraf indus este muchia* $\boldsymbol{(n, i_0)}$, *înseamnă că orice drum între n şi un nod j din {0, 1, ..., n-1} începe cu muchia* $\boldsymbol{(n, i_0)}$ *şi se continuă cu drumul corespunzător de la* i_0 *la j. Deoarece muchia* $\boldsymbol{(n, i_0)}$ *are cea mai mică etichetă, orice drum crescător pornind din* i_0 *în faţa căruia se adaugă această muchie rămâne drum crescător, conform definiţiei. Aşadar,* ***există cel puţin un drum crescător de la n la j, ∀j ∈ {0, 1, …, n-1}****.*
- *în final se adaugă muchia* $\boldsymbol{(n - 1, j_0)}$, *cu* $\boldsymbol{j_0}$ *oarecare din {0, 1, ..., n-1}, care va fi etichetată cu* ***2n – 2****. Conform celor arătate la pasul anterior, există un drum crescător de la j la* j_0, $\forall j \in \{0, 1, ..., n\text{-}1\} - \{j_0\}$.
  Dacă la un astfel de drum se adaugă, la sfârşit, muchia $\boldsymbol{(j_0, n)}$, *drumul astfel obţinut între* ***j*** *şi* ***n*** *rămâne crescător, deoarece eticheta lui* $\boldsymbol{(j_0, n)}$ *este mai mare decât eticheta lui* e_i, $\forall e_i \in E(G_{n+1}) - \{(j_0, n)\}$. *Aşadar,* ***∀j ∈ {0, 1, …, n-1}, există cel puţin un drum crescător de la j la n.***

Se obţine prin acest mod de construcţie un graf cu proprietatea că $\forall i \in V$, $\forall j \in V - \{i\}$, *există un drum „crescător" de la i la j. În plus, dimensiunea acestui graf este:*

$$1 + (2n - 4) + 1 = 2n - 2 = \mathbf{2(n + 1) - 4 = 2|G_{n+1}| - 4}.$$

O posibilă structură a grafurilor G_n construite prin paşii de mai sus pornindu-se de la graful G_4 este:



Evident, aceasta nu este unica schemă posibilă.

Aşa cum se vede şi în exemplul pentru G_5 şi aşa cum reiese din paşii descrişi mai sus, nodul adăugat pentru construcţia lui G_n din G_{n-1} poate fi legat prin două muchii de oricare două noduri (nu neapărat distincte, fiind permisă şi obţinerea unui multigraf) dintre cele din G_{n-1}. Condiţia impusă de algoritmul descris mai sus este ca una dintre aceste muchii să aibă eticheta minimă, cealaltă să aibă eticheta maximă, iar etichetele muchiilor dintre celelalte noduri să fie în relaţia în care erau în graful G_{n-1}.

Pot exista şi alte modalităţi de construire a unui graf cu proprietăţile cerute.

***Algoritmul** care construieşte recursiv G_n urmând paşii explicaţi mai sus este:*

```
procedure ConstruieşteSchema(n)
begin
    if (n < 2)
        mesaj: "Nu se poate construi o schemă pentru acest număr."
        return
    /*iniţial, G este graful nul cu n noduri*/
    V(G) ←{0, 1, …, n-1}
    E(G) ← Φ
    /*se trartează cazurile particulare: n = 2 şi n = 3*/
    switch(n)
        case 2:
            E(G) ←{(0, 1)}
            label((0, 1)) ←1
            break
        case 3:
            E(G) = {(0, 1), (1, 2), (2, 0)}
            label((0, 1)) ←1
            label((1, 2)) ←2
            label((2, 0)) ←3
            break
        default:
            G ←ConstruieşteSchemaRecursiv (n, 1)
            break
end

function ConstruieşteSchemaRecursiv(n, eticheta)
```

```
begin
    if (n = 4)
        E(G) ← {(0, 1), (1, 2), (2, 3), (3, 0)}
        label((0, 1)) ← eticheta
        label((1, 2)) ← eticheta + 2
        label((2, 3)) ← eticheta + 1
        label((3, 0)) ← eticheta + 3
    else
        E(G) ← E(G) ∪ {(n - 1, 0), (n - 1, 2)}
        label((n - 1, 0)) ← eticheta
        G ← ConstruieşteSchemaRecursiv (n – 1, eticheta + 1)
        label((n - 1, 2)) ← 2*n – 4 – eticheta + 1
    return G
end
```

II. Arătăm că dimensiunile precizate mai sus sunt dimensiunile minime posibile pentru astfel de grafuri.

Pentru $|G_2| = 2$: *Dimensiunea lui* G_2 *construit la* ***I*** *este 1. Dacă dimensiunea lui* G_2 *ar fi mai mică decât 1 (adică 0), graful* G_2 *nu ar fi* ***conex****, deci nu ar exista nici un drum între cele două noduri ale sale. Implicit, nu există drumuri crescătoare. Aşadar,* ***dimensiunea minimă*** *pentru un graf* G_2 *este* ***1****.*

Pentru $|G_3| = 3$: *Dimensiunea lui* G_3 *construit la* ***I*** *este 3.*

Presupunem că există un graf G_3' *cu* $|G_3'|=3$ *şi dimensiunea mai mică decât 3. Dacă dimensiunea lui* G_3' *ar fi 0 sau 1, graful nu este conex, deci există în acest graf noduri între care nu există drumuri, deci nici drumuri crescătoare.*

Dacă dimensiunea grafului este 2, atunci acest graf este $\boldsymbol{P_3}$*. Într-un astfel de graf există exact un drum între oricare două noduri. Fie i şi j cele două noduri de grad 1 din acest graf. Deoarece în graful* G_3' *două muchii nu pot avea aceeaşi etichetă, dacă drumul de la i la j este crescător, atunci nu există drum crescător de la j la i, şi reciproc. Aşadar, Graful* G_3' *trebuie să aibă* ***dimensiunea 3****.*

Pentru a demonstra optimalitatea prntru G_n *cu* $|G_n| \geq 4$ *se foloseşte următoarea propoziţie:*

Propoziţia 1: Dacă G_n ***cu*** $n \geq 4$ ***are dimensiunea minimă posibilă, atunci*** $\exists w \in V(G_n)$ ***a.î.*** $d(w) \leq 3$***.***

(Demonstraţie: Dacă toate nodurile ar avea gradul cel puţin 4, atunci $\sum_{i \in V} d(i) \geq 4n$ *. În plus, dimensiunea oricărui graf este jumătate din suma gradelor tuturor vârfurilor. Se obţine deci că* $|E(G_n)| \geq 2n$*. Dar am arătat că se poate construi* G_n *de dimensiune* $2n - 4$*, deci dimensiunea unui graf optimal este mai mică sau egală cu* $2n - 4$*, adică* $2(n+1) \leq 2n - 4 \rightarrow$ *contradicţie).*

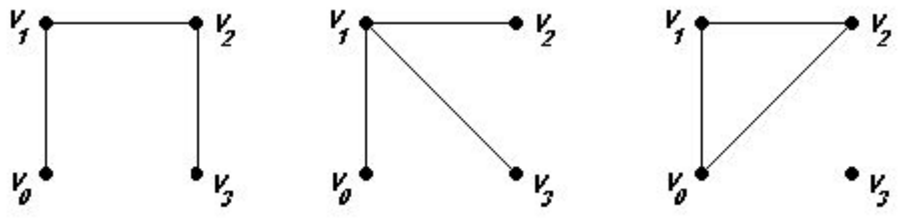
Pentru $|G_n| \geq 4$ *se demonstrează prin* ***inducţie*** *după* $|G_n|$ *că un graf* G_n *cu proprietăţile de mai sus are dimensiunea cel puţin* $2|G_n| - 4$*.*

Pasul I:

Pentru $|G_4| = 4$:

Dacă dimensiunea acestui graf ar fi 0, 1, sau 2, graful nu ar fi conex, deci, conform celor arătate mai sus, nu ar putea avea proprietatea cerută.

Dacă dimensiunea este 3, graful în cauză poate fi unul din grafurile din figura de mai jos:



I *II* *III*

*Se observă că graful **III** nu este conex, deci există cel puţin două noduri între care nu este nici un drum în acest graf. Implicit, între aceste două noduri nu există nici drumuri crescătoare, deci un graf cu această formă nu are proprietatea cerută.*

*Dacă studiem grafurile **I** şi **II** remarcăm faptul că aceste grafuri sunt de fapt **arbori**, în astfel de grafuri existând **un singur drum** între oricare două noduri.*

Fie două noduri i şi j neadiacente într-un astfel de graf (există, deoarece grafurile nu sunt complete). Atunci unicul drum dintre ele are lungimea cel puţin 2.

Indiferent de modul de etichatare a muchiilor, deoarece în graf două muchii nu pot avea aceeaşi etichetă (din modul de construcţie), dacă drumul de la i la j este crescător, atunci nu există drum crescător de la j la i, şi reciproc. Aşadar, graful G_4 nu poate avea nici această formă.

*Am arătat deci că graful G_4 trebuie să aibă **dimensiunea cel puţin 4**.*

***Pasul II:** Presupunem afirmaţia adevărată pentru $|G| \leq n$. Demonstrăm pentru $|G| = n + 1$.*

*Conform **propoziţiei 1** (pagina 4), există cel puţin un nod w în $V(G_{n+1})$ cu gradul mai mic sau egal cu 3. Se disting următoarele cazuri posibile:*

***Cazul 1: d(w) = 0**. Atunci nodul w este izolat, deci graful G nu este conex, neavând, în consecinţă, nici proprietăţile cerute. Aşadar acest caz este **practic imposibil**.*

***Cazul 2**: **d(w) = 1**. Fie v unicul vecin al lui w.*

Ştim că există drum crescător de la w la i şi de la i la w, $\forall i \in V(G_{n+1}) - \{w\}$.

Deoarece w are gradul 1, avem:

***(1)** Toate drumurile crescătoare de la w la i încep cu muchia wv, deci **etichetele tuturor muchiilor de pe aceste drumuri sunt mai mari decât label(wv)**.*

***(2)** Toate drumurile crescătoare de la i la w se termină cu muchia wv, deci **etichetele tuturor muchiilor de pe aceste drumuri sunt mai mici decât label(wv)**.*

*Din **(1)** şi **(2)** rezultă că, $\forall e \in E(G_{n+1}) - \{wv\}$, dacă **e** este pe un **drum crescător de la i la w**, atunci **e** nu poate fi pe nici un **drum crescător de la w la j**, cu i şi j arbitrare diferite de w şi v. (Demonstraţie:*

*Dacă e este pe un drum crescător de la i la w, atunci label(e) < label(wv), conform **(2).***

*Dacă e este pe un drum crescător de la w la j, atunci label(e) > label(wv), conform **(1).***

*$\Rightarrow$ **contradicţie**).*

*Există drum crescător de la w la i, $\forall i \in V(G_{n+1}) - \{w\}$, deci graful parţial care conţine doar muchiile de pe aceste drumuri este conex. În consecinţă are cel puţin $|G_{n+1}| - 1 = n$ **muchii**.*

*Există drum crescător de la i la w, $\forall i \in V(G_{n+1}) - \{w\}$, deci graful parţial care conţine doar muchiile de pe aceste drumuri este de asemenea conex, având cel puţin $|G_{n+1}| - 1 = n$ **muchii**.*

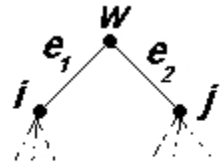
Intersecţia mulţimilor de muchii ale celor două grafuri are cardinalul 1, conţinând doar muchia wv, după cum am arătat mai sus.

Prin urmare, graful G_{n+1} care are un nod de grad 1 conţine cel puţin

$$n + n - 1 = 2n - 1 = 2\,|G_{n+1}| - 3 \quad \textbf{muchii}.$$

Dar am arătat că se poate construi un graf cu ***2** $|G_{n+1}| - 4$* ***muchii**, deci* ***această structură nu este optimală.***

Cazul 3: d(w) = 2. *Se disting următoarele două subcazuri:*



a) *b)*

Cazul 3a: *w are un unic vecin i de care este legat cu două muchii (G poate fi multigraf).*

Atunci, ***oricare ar fi u şi v diferite de w şi i, drumul crescător de la u la v nu trece prin w*** *(deoarece un* ***drum*** *este un mers cu toate nodurile distincte; dacă un mers trece prin w, trebuie să treacă de două ori prin i, deci nu este drum).*

Aşadar, subgraful G' indus de $V(G_{n+1}) - \{w\}$ păstrează proprietatea că $\forall u, v \in V(G') = V(G_{n+1}) - \{w\}$, există în G' drum crescător de la u la v. Dar $|G'| = $ ***n**, deci, conform ipoteyei inductive, dimensiunea lui G' este cel puţin $2n - 4$. În consecinţă, dimensiunea lui G_{n+1} este* ***cel puţin***

$$\boldsymbol{2n - 4 + 2 = 2n - 2 = 2(n + 1) - 4 = 2|G_{n+1}| - 4.}$$

Cazul 3b: *w are doi vecini i şi j, iar etichetele muchiilor wi şi wj se găsesc în relaţia*

$$\boldsymbol{e_1 = label(wi) < label(wj) = e_2.}$$

Există drum crescător de la w la u, $\forall u \in V(G_{n+1})$. Prin urmare, graful parţial care conţine doar muchiile de pe aceste drumuri este conex, deci are cel puţin ***$|G_{n+1}| - 1 = n$ muchii.***

Există drum crescător de la u la w, $\forall u \in V(G_{n+1})$. Deoarece unicii vecini ai lui w sunt i şi j, toate aceste drumuri trec prin i sau prin j.

Toate muchiile de pe drumul crescător de la u la w au etichetele mai mici fie decât e_1, fie decât e_2.

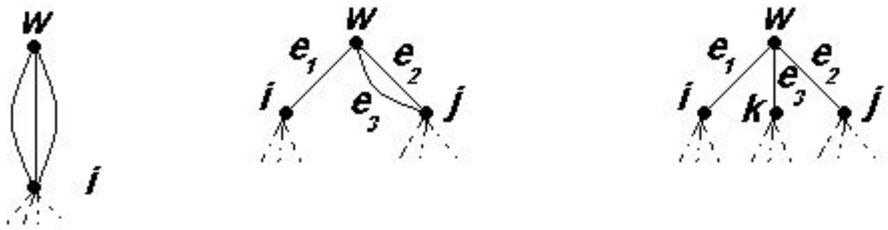
- *Dacă drumul crescător de la u la w trece prin i, atunci etichetele tuturor muchiilor de pe acest drum sunt mai mici decât e_1. Dar $e_1 < e_2$, deci aceste etichete sunt mai mici şi decât e_2. Prin urmare, nici una din aceste muchii nu ar putea face parte dintr-un drum crescător de la w la v, $\forall v \in V(G_{n+1})$. Toate aceste muchii sunt deci distincte de cele n menţionate mai sus.*
- *Dacă drumul crescător de la u la w trece prin j, atunci etichetele tuturor muchiilor de pe acest drum sunt mai mici decât e_2. Aşadar, aceste muchii nu ar putea face parte dintr-un drum crescător din w prin j. Unele ar putea face parte dintr-un drum crescător din w prin i. Dar, pentru ca graful parţial menţionat mai sus să fie minimal, trebuie să fie arbore, deci să nu admită cicluri. Aşadar, pentru a crea legătura dintre o muchie dintr-un drum crescător din w prin i, şi j, trebuie adăugată cel puţin o muchie distinctă de cele n din graful parţial.*

În concluzie, este necesară cel puţin câte o muchie pentru fiecare din cele $n - 2$ noduri diferite de w, i şi j. În plus, aceste muchii sunt distincte între ele două câte două, având cel puţin câte o extremitate diferită.

Aşadar, şi în acest caz, graful trebuie să aibă ***cel puţin***

$$\boldsymbol{n + n - 2 = 2n - 2 = 2(n + 1) - 4 = 2|G_{n+1}| - 4 \text{ muchii.}}$$

Cazul 4: d(w) = 3. *Se disting următoarele trei subcazuri:*



a) b) c)

Cazul 4a: *w are un unic vecin i de care este legat prin 3 muchii. Fie e_1, e_2, e_3 etichetele celor 3 muchii, astfel încât $e_1 < e_2 < e_3$. Acest graf nu este minimal, întrucât se poate renunţa la muchia cu eticheta e_2 cu păstrarea proprietăţilor:*

- *toate drumurile crescătoare spre sau de la w trec prin i, acesta fiind unicul lui vecin;*
- *aşa cum am arătat la cazul 3a, oricare ar fi u şi v diferite de w şi i, drumul crescător de la u la v nu trece prin w;*
- *dacă un drum crescător spre w trece prin e_2, atunci $e_2 <$ label(e), oricare ar fi o muchie e de pe acest drum; dar $e_1 < e_2$, deci $e_1 <$ label(e), adică acest drum ar putea trece prin e_1;*
- *dacă un drum crescător de la w trece prin e_2, atunci $e_2 >$ label(e), oricare ar fi o muchie e de pe acest drum; dar $e_3 > e_2$, deci $e_3 >$ label(e), adică acest drum ar putea trece prin e_3.*

Prin urmare, ***o astfel de structură nu este minimală, reducându-se la cazul 3a*** *prezentat mai sus.*

Cazul 4b: *w are 2 vecini i şi j, fiind legat de j prin două muchii.*

- *Dacă există u şi v astfel încât drumul crescător de la u la v să treacă prin w, atunci acest drum conţine muchia e_1 şi una din muchiile e_2 şi e_3, fie aceasta e.*
    - *Dacă trece întâi prin e_1 şi apoi prin e, toate muchiile de până la i au eticheta mai mică decât e_1, respectiv toate muchiile de după j au eticheta mai mare decât e. În plus, $e_1 <$ e, deci e este mai mare sau egala cu valoarea „din mijloc" dintre e_1, e_2 şi e_3. Deci, în orice astfel de drum crescător, putem înlocui cele două muchii cu o muchie între i şi j care are ca etichetă valoarea medie dintre cele 3.*
    - *Dacă trece întâi prin e şi apoi prin e_1, toate muchiile de până la i au eticheta mai mare decât e_1, respectiv toate muchiile de după j au eticheta mai mică decât e. În plus, e $< e_1$, deci e_1 este mai mică sau egala cu valoarea „din mijloc" dintre e_1, e_2 şi e_3. Deci, în orice astfel de drum crescător, putem înlocui cele două muchii cu o muchie între i şi j care are ca etichetă valoarea medie dintre cele 3.*

  Adăugând ca mai sus o muchie între i şi j şi eliminând complet nodul w, se obţine un graf de ordin $\boldsymbol{|G_{n+1}| - 1}$ *care păstrează proprietatea că între oricare două noduri există un drum crescător. Dar acest graf are dimensiunea cel puţin* $\boldsymbol{2(|G_{n+1}| - 1) - 4}$ *(din ipoteza inductivă). Deoarece a fost obţinut din* $\boldsymbol{G_{n+1}}$ *prin eliminarea a 3 muchii şi introducerea uneia noi, înseamnă că* $\boldsymbol{G_{n+1}}$ *are dimensiunea* ***cel puţin***

  $$\boldsymbol{(2(|G_{n+1}| - 1) - 4) - 1 + 3 = 2|G_{n+1}| - 4.}$$

- *Dacă nu există u şi v astfel încât drumul crescător de la u la v să treacă prin w, atunci, prin eliminarea lui u şi a muchiilor care îl leagă de i şi j, se obţine un graf de ordin* $\boldsymbol{|G_{n+1}| - 1}$ *care păstrează proprietatea că între oricare două noduri există un drum crescător. Dar acest graf are dimensiunea cel puţin* $\boldsymbol{2(|G_{n+1}| - 1) - 4}$ *(din* *ipoteza inductivă). Deoarece a fost obţinut din* G_{n+1} *prin eliminarea a 3 muchii, înseamnă că* G_{n+1} *are dimensiunea* ***cel puţin***

$$(2(|G_{n+1}| - 1) - 4) + 3 = 2|G_{n+1}| - 4.$$

***(această structură nu este deci optimală**, întrucât am arătat că se pot construi grafuri de dimensiune 2* $|G_{n+1}| - 4$*).*

Cazul 4c: *w are 3 vecini, i, j şi k. iar etichetele muchiilor wi, wj şi wk se găsesc în relaţia*

$$e_1 = label(wi) < e_2 = label(wj) < e_3 = label(wk).$$

- *Există drum crescător de la w la u,* $\forall u \in V(G_{n+1})$*. Prin urmare, graful parţial care conţine doar muchiile de pe aceste drumuri este conex, deci are cel puţin* $|G_{n+1}| - 1 = n$ ***muchii.***
- *Există drum crescător de la u la w,* $\forall u \in V(G_{n+1})$*. Deoarece unicii vecini ai lui w sunt i, k şi j, toate aceste drumuri trec prin i, prin k sau prin j.*
  *Aşa cum am arătat la **cazul 3b**, pentru ca să existe drum crescător de la oricare nod din graf la w, este necesară cel puţin câte o muchie pentru fiecare din cele* $n - 3$ *noduri diferite de w, i şi j. În plus, aceste muchii sunt distincte între ele două câte două, având cel puţin câte o extremitate diferită, fiind distincte şi de cele n menţionate mai sus.*
- *Considerând relaţia menţionată între* e_1, e_2 *şi* e_3*, există deja drum crescător de la i la j de la i la k şi de la j la k prin w. Pentru ca să existe drum crescător de la k la j sau la i, fie este necesară o muchie între k şi aceste noduri (care nu făcea parte nici dintre muchiile din drumurile crescătoare din w, nici din cele crescătoare spre w), fie un drum crescător care trece printr-un alt nod.*
  Dar orice drum crescător din k spre un nod u care să aibă muchii deja menţionate la punctele anterioare este în una din următoarele situaţii:
    - *este constituit dintr-o singură muchie, ca parte a unui drum descrescător de la u la w, deci* $label(uk) > e_3$*;*
    - *este parte a unui drum crescător din w prin k, deci toate muchiile au etichetele mai mari decât* e_3*.*

  Respectiv, orice drum crescător de la u la i sau la j care să aibă muchii deja menţionate la punctele anterioare este în una din următoarele situaţii:
    - *este constituit dintr-o singură muchie, ca parte a unui drum crescător de la u la w, deci* $label(uj) < e_2$ *sau* $label(ui) < e_1$*;*
    - *este parte a unui drum crescător spre w prin i sau j, deci toate muchiile au etichetele mai mici decât* e_2*, respectiv* e_1*.*

  În plus, $e_1 < e_3$ *şi* $e_2 < e_3$*, deci la concatenarea a două drumuri crescătoare ca cele de mai sus, unul de la k la u şi unul de la u la j sau la i, nu se obţine un drum crescător de la k la i sau la j, deoarece toate muchiile dintre k şi u au etichetele mai mari decât toate muchiile dintre u şi i (sau j).*
  *Prin urmare, este necesară cel puţin **o muchie** distinctă de toate cele menţionate la primele două puncte din acest subcaz, pentru obţinerea drumului crescător de la k la i (dacă toate muchiile de pe acest drum au etichetele mai mici decât* e_1*, atunci avem şi drum de la k la j prin i şi w).*
  *Aşadar, şi în acest caz, graful trebuie să aibă **cel puţin***

$$n + n - 3 + 1 = 2n - 2 = 2(n + 1) - 4 = 2|G_{n+1}| - 4 \text{ muchii.}$$

Conform propoziţiei 1 (pagina 4), un graf G cu proprietăţile cerute, de dimensiune cel mult $2|G| - 4$*, se află în cel puţin unul din cazurile studiate mai sus (exceptând cazul 1, care contrazicea ipoteza). Pentru fiecare astfel de caz am arătat că numărul minim de muchii necesare pentru* *satisfacerea proprietăţilor este* ***2|G| – 4***. *Întrucât am arătat anterior că există grafuri cu această dimensiune, se obţine faptul că un graf G care are proprietăţile cerute şi are dimensiunea* ***2|G| – 4*** *este o variantă optimală de rezolvare a problemei.*

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

$$\frac{a(C^*)}{b(C^*)} = \min\{\frac{a(C)}{b(C)}, C \text{ circuit în } D\}.$$

Soluţie:

Fie ***p***: *E(D) →* $\boldsymbol{R}_+$ *funcţia de cost ataşată grafului de mai sus, definită prin :*

$$p(e) = \frac{a(e)}{b(e)},\ \forall e \in E(D).$$

Extensia funcţiei p la drumuri în digraf, şi implicit la circuite, se defineşte astfel:

$$p(Dr) = \sum_{e \in E(Dr)} \frac{a(e)}{b(e)},\ \forall Dr \text{ drum (inchis sau deschis) în digraful } D.$$

Observaţie: *Evident, costul p al oricărui drum în digraf este limitat superior de* $\boldsymbol{\max\{\frac{a(e)}{b(e)},\ e \in E(D)\}}$, *respectiv limitat inferior de* $\boldsymbol{\min\{\frac{a(e)}{b(e)},\ e \in E(D)\}}$. *(Ambele limite sunt nenegative, fapt care reiese din definiţia funcţiilor a şi b.)*

(Demonstraţia se face folosind proporţii derivate).

În aceste condiţii, este suficient să căutăm costul minim al unui circuit din digraful D în intervalul (m, M), unde m = $\boldsymbol{\min\{\frac{a(e)}{b(e)},\ e \in E(D)\}}$ *şi M =* $\boldsymbol{\max\{\frac{a(e)}{b(e)},\ e \in E(D)\}}$.

Se defineşte o nouă funcţie c: E(D) → $\boldsymbol{R}$, *cu parametrul λ, unde*

$$c(e) = a(e) - \lambda b(e).$$

iar λ este o valoare din intervalul (m, M) precizat mai sus.

Pentru eficienţa căutării valorii λ în acest interval se porneşte cu valoarea $\frac{m+M}{2}$ *pentru λ şi se verifică dacă există circuite care să aibă pentru acest parametru costul* ***c negativ***.

Se disting următoarele 3 cazuri posibile:

Cazul 1. *Există cel puţin un circuit negativ*

Dacă c(C) < 0 ⇒ a(e) - λb(e) < 0.

Deoarece b(e) > 0 (din definiţia funcţiei b), se obţine că

$$\frac{a(C)}{b(C)} - \boldsymbol{\lambda < 0 \Rightarrow} \frac{a(C)}{b(C)} < \boldsymbol{\lambda}.$$

Prin urmare, dacă s-a găsit un circuit C cu costul c negativ, atunci acel circuit C are valoarea pentru $\frac{a(C)}{b(C)}$ *mai mică decât λ. Se actualizează λ cu* $\frac{a(C)}{b(C)}$*, acesta fiind costul p minim al unui circuit din D până la acest pas.*

Prin urmare, λ ales în acest moment este prea mare. Reactualizăm marginile intervalului în care se caută λ:

$m \leftarrow m$

$M \leftarrow \lambda$

$\lambda \leftarrow \frac{m+M}{2}$.

Se continuă cu verificarea circuitelor, pentru noul λ.

Cazul 2. *Nu s-au găsit circuite de costuri negative şi nici circuite cu costul 0.*

Atunci λ ales este prea mic. Se reactualizează λ astfel:

$m \leftarrow \lambda$

$M \leftarrow M$

$\lambda \leftarrow \frac{m+M}{2}$.

Se continuă cu verificarea circuitelor, pentru noul λ.

Cazul 3. *Nu s-au găsit circuite negative dar există circuit cu costul 0. Atunci λ curent este costul circuitului minim şi trebuie descoperit circuitul, cunoscându-se un nod din acest circuit.*

Algoritmul se opreşte atunci când s-a descoperit un λ pentru care există un circuit C cu* $\frac{a(C^*)}{b(C^*)} = \boldsymbol{\lambda}$ *şi oricare ar fi alt circuit C în D, c(C) ≥ 0, adică* $\frac{a(C)}{b(C)} \geq \lambda$*. Algoritmul se opreşte întotdeauna deoarece costul circuitului minim se află în intervalul considerat iniţial. λ poate fi găsit întotdeauna prin împărţiri la 2, chiar dacă în realitate λ nu are număr finit de zecimale, deoarece calculatorul realizează trunchierea.*

Pentru detectarea circuitelor de cost negativ în digraful D se utilizează algoritmul lui Floyd-Warshall.

procedure *CautăCircuitCostMinim(D)*
begin

$m \leftarrow \boldsymbol{min}\{\frac{a(e)}{b(e)},\ \boldsymbol{e \in E(D)}\}$

$M \leftarrow \boldsymbol{max}\{\frac{a(e)}{b(e)},\ \boldsymbol{e \in E(D)}\}$.

$\lambda \leftarrow \frac{m+M}{2}$

*/*găsire minimului şi a maximului se realizează printr-o parcurgere a vectorilor în care reţinem valorile funcţiilor a şi b pentru fiecare muchie; lungimea acestor vectori este exact |E(D)|*/*

găsit ← false

while *(not găsit)* ***do***

rez ← CautăCircuitNegativ(D)

```
        switch (rez)
            case -1:
                m ← λ
                λ ← (m + M) / 2
                break
            case 1:
                M ← λ
                λ ← (m + M) / 2
                break
            case 0:
                CautăCircuit(D, s, λ)
                găsit ← true
                break
end
```

Se apelează funcţia ***CautăCircuit*** *de log |**E(D)**| ori (deoarece avem căutare binară), unde* ***E(D)*** *este de ordinul lui* n^2*. Deci se face un număr de apelări ale funcţiei* ***CautăCircuit*** *de ordin* $\log n^2 = 2\log n$.

```
function CautăCircuitNegativ(D)
begin
    /*algoritmul utilizat este algoritmul Floyd – Warshall; acest algoritm lungimea drumului
minim dintre oricare 2 noduri*/
    /*W este matricea in care iniţial sunt costurile c ale muchiilor;
      dacă ij ∈ E(D), W(i, j) = c(i, j), unde c(e) = a(e) - λb(e); altfel, c(ij) = +∞.
    După terminarea algoritmului, W(i, i) = costul minim al unui circuit ce conţine i*/
    for i ← 0 to n – 1 do
        for j ← 0 to n – 1 do
            if (ij ∈ E(D)) then W(i, j) = a(ij) - λb(ij)
            else W(i, j) = +∞
    for k ← 0 to n – 1 do
        for i ← 0 to n – 1 do
            for j ← 0 to n – 1 do
                if (W(i, j) > W(i, k) + W(k, j))
                then W(i, j) ← W(i, k) + W(k, j)

    /*se parcurge diagonala pentru detectarea costului minim*/
    cMinim ← W(0, 0)
    s ← 0
    for i ← 1 to n – 1 do
        if (W(i, i) < cMinim) then
            cMinim ← W(i, i)
            s ← i
    /*se returnează -1 dacă există circuite negative, 0 dacă nu există circuite negative dar
există circuite de cost 0, respectiv 1 dacă toate costurile sunt pozitive*/
    if (cMinim < 0)
```

then return -1
else
if (cMinim = 0)
then return 0
else return 1
end

Funcţia ***CautăCircuitNegativ*** *se execută în timpul* $\boldsymbol{O(n^3)}$ *deoarece ştim că algoritmul lui Floyd-Warshall are complexitatea* $O(n^3)$.

procedure *CautăCircuit(D, s, λ)*
begin
/*se va folosi algoritmul lui Dijkstra adaptat pentru acest caz; adaptarea algoritmului constă îm faptul că, printre nodurile spre care se caută drumuri de cost minim se va afla întotdeauna nodul s (deoarece căutăm un circuit care conţine s). În plus, se vor lua în consideraţie doar costurile circuitelor.*/
/*Dacă în funcţia CautăCircuitNegativ(D) am găsit un nod i pentru care W(i, i) = 0, atunci ∀j din acest circuit, W(j, j) = 0. În consecinţă, o altă modificare adusă algoritmului este aceea că se vor introduce în V' (mulţimea din care alegem nodurile) doar aceste noduri j.*/

$V' \leftarrow \{j \mid W(j, j) = 0\}$
$\text{înainte}(s) \leftarrow 0$
for $i \in V'$ ***do***
$u_i \leftarrow a(si) - \lambda b(si)$
// dacă nu există muchia si, u_i ***va fi*** $+\infty$
$\boldsymbol{S \leftarrow \{\}}$
while $(s \notin S)$
$x \leftarrow \min(u_i)$
$S \leftarrow S \cup x$
$V' \leftarrow V' - x$
for $i \in V'$ ***do***
if $(u_i > u_x + a(xi) - \lambda b(xi))$
then
$u_i \leftarrow a(xi) - \lambda b(xi)$
$\text{predecesor}(i) \leftarrow x$
// în predecesor, pornind din s se poate găsi circuitul de cost minim în ordine inversă.
end

Funcţia ***CautăCircuit*** *are complexitatea* $\boldsymbol{O(n^2)}$ *întrucât este adaptarea algoritmului lui* ***Dijkstra*** *şi se cunoaşte că acest algoritm are complexitatea* $O(n^2)$.

În final, obţinem complexitatea algoritmului $O(n^3\log n + n^2) = \boldsymbol{O(n^3\log n)}$.

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

Soluţie:

Construim un graf G cu următoarele proprietăţi:

- *Nodurile grafului G sunt mulţimile* A_i, *cu* $1 \le i \le n$;

- *Fiind date mulţimile A_i şi A_j, avem muchie între cele două noduri reprezentate de mulţimile respective numai în cazul în care $A_i \Delta A_j = \{x\}$ (adică $A_i = A_j \cup \{x\}$ sau $A_j = A_i \cup \{x\}$);*
- *Fiecare muchie este etichetată: eticheta este reprezentată de elementul care formează diferenţa simetrică a celor două mulţimi ce se află în cele două noduri.*

Observaţie: *În cazul în care avem două sau mai multe perechi de mulţimi pentru care mulţimile obţinute prin diferenţa simetrică sunt egale, introducem muchie numai între nodurile pentru primele mulţmi pentru care am obţinut acel rezultat (celelalte noduri nu vor mai fi legate).*

Presupunem că mulţimea obţinută prin ***reuniunea celor n mulţimi este o mulţme A*** *care are* ***cardinalul m.***

Dacă ***m<n*** *înseamnă că există elemente din S care nu au fost incluse în nici una dintre mulţmile A_i. Deci, dacă am considera x ca fiind unul dintre aceste elemente am obţine relaţia căutată întrucât din diferenţele A_i - {x}, A_2 - {x},..., A_n - {x} rezultă tot mulţimile A_1, A_2, ..., A_n care erau distincte din ipoteză.*

Dacă ***m=n,*** *notăm* ***numărul de muchii cu M*** *şi urmărim să demonstrăm că M<n. Este evident că în momentul în care* ***nu*** *toate elementele din mulţimea S etichetează muchii din graful G, obţinem un element care să respecte proprietatea. Alegem* ***x ca fiind unul dintre acele elemente care nu etichetează nici o muchie*** *(deci x* ***nu*** *este unul dintre elementele pe care dacă le scoatem dintr-o mulţime obţinem altă mulţime dintre cele n). Aşadar putem spune că dacă scoatem x din toate cele n mulţimi, obţinem tot mulţimi distincte.*

Vom demonstra acum că într-adevăr M<n *(este evident că M nu poate fi mai mare decât m, doarece am specificat că muchiile au etichete distincte).*

Reducere la absurd:

Presupunem că există n muchii în graful G.

Întrucât graful G are n noduri putem spune că admite cicluri (un graf de ordin n care nu admite cicluri şi este maximal cu această proprietate are dimensiunea n – 1 şi este arbore; oricum s-ar adăuga o muchie la un astfel de graf se formează cel puţin un circuit).

Fie C unul dintre ciclurile din graful G şi fie A_{i1}, A_{i2}, ..., A_{ik} mulţimile din nodurile care formează ciclul de lungime k, legate în această ordine. Presupunem că o mulţime A_{ij} "este legată" de o mulţime A_{ij+1}; vom nota eticheta muchiei dintre cele două noduri cu a_{ij}. Conform modului de introducere a muchiilor între două noduri, putem spune că avem următoarele relaţii între mulţimile de mai sus:

- $| card(A_{i1}) - card(A_{i2}) | = 1$
- $| card(A_{i2}) - card(A_{i3}) | = 1$
- …
- $| card(A_{ik-1}) - card(A_{ik}) | = 1$
- $| card(A_{ik}) - card(A_{i1}) | = 1$

Aceste relaţii se reduc la faptul că două mulţimi consecutive din lanţ pot avea cel mult un element în plus sau unul în minus una faţă de alta.

Din relaţiile de mai sus putem trage următoarele concluzii:

- $| card(A_{i1}) - card(A_{i2}) | = 1$
- $| card(A_{i1}) - card(A_{i3}) | = 0$ *sau* $| card(A_{i1}) - card(A_{i3}) | = 2$
- $| card(A_{i1}) - card(A_{i4}) | = 1$ *sau* $| card(A_{i1}) - card(A_{i3}) | = 3$
- $| card(A_{i1}) - card(A_{i5}) | = 0, 2,$ *sau* 4
- …

- *| card(A_{i1}) – card(A_{ik}) | = 0, 2, …sau k-1 pentru k număr impar*
  = 1, 3, …sau k-1 pentru k număr par.

*Relaţiile de mai sus sunt evidente întrucât la fiecare pas pe care îl parcurgem în ciclu putem avea cel mult un element în plus sau în minus faţă de elementul anterior. Putem demonstra acest fapt prin **inducţie**.*

***Pasul de bază** este :*

| card(A_{i1}) – card(A_{i2}) | = 1

| card(A_{i1}) – card(A_{i3}) | = 0 sau | card(A_{i1}) – card(A_{i3}) | = 2

*La **pasul inductiv** presupunem că avem adevărată relaţia pentru k mulţimi şi demonstrăm pentru a k+1-a mulţime.*

Ştim că avem muchie între A_{ik} şi A_{ik+1}. Deci diferenţa simetrica a celor două mulţimi este formată dintr-un element şi după cum am spus şi mai sus avem | card(A_{ik}) – card(A_{ik+1}) | = 1. În cazul în care k este număr par, avem | card(A_{i1}) – card(A_{ik}) | = 1, 3, …sau k-1. Din cele două relaţii obţinem că | card(A_{i1}) – card(A_{ik+1}) | = 0, 2, … k-2 sau k, adică exact ce trebuia să arătăm. Avem aceeaşi situaţie şi în cazul în care k este număr impar.

Am demonstrat deci că

| card(A_{i1}) – card(A_{ik}) | = 0, 2, …sau k-1 pentru k număr impar
= 1, 3, …sau k-1 pentru k număr par.

*În cazul în care **| card(A_{i1}) – card(A_{ik}) | ≠ 1** nu putem avea muchie între cele două noduri şi deci obţinem **contradicţie** faţă de presupunerea iniţială.*

*Trebuie să mai arătăm că şi pentru **| card(A_{i1}) – card(A_{ik}) | = 1** obţinem contradicţie.*

*Considerăm cazul în care **card(A_{ik})=card(A_{i1})+1**. Deci $A_{ik} = A_{i1} \cup \{x\}$ (şi din modul în care am construit muchiile). Dar ca să fi putut ajunge la această situaţie este necesar ca anterior să fi introdus într-o mulţime, printr-o **muchie etichetată cu x**, elementul x (pentru că mulţimea iniţială A_{i1} nu conţinea acest element şi acesta este singurul mod de a obţine elemente noi în mulţimile de după A_{i1}). **Muchia dintre A_{ik} şi A_{i1} este etichetată cu x** (deoarece x este elementul care formează diferenţa simetrică a celor două mulţimi). Am ajuns la o **contradicţie** întrucât noi am considerat muchii cu etichete distincte şi am obţinut că **există două muchii etichetate cu x**. Analog se tratează şi cazul în care **card(A_{i1})=card(A_{ik})+1**.*

*Presupunerea de la care am plecat este falsă, deci nu pot exista cicluri în graful G şi deci graful G **nu poate avea n muchii**.*

*Putem spune aşadar că **întotdeauna va exista cel puţin un element care să nu eticheteze nici o muchie** şi deci **unul dintre aceste elemente va fi elementul x căutat**.*

Am demonstrat că există un element x în mulţimea S astfel încât $A_1 - \{x\}$, $A_2 - \{x\}$,…, $A_n - \{x\}$ să fie de asemenea distincte.

4. *Fie G un graf şi $C:E(G)\rightarrow R_+$ o funcţie de capacitate a muchiilor. Oricărui drum din graf cu măcar o muchie i se asociază **locul îngust** ca fiind muchia sa de capacitate minimă. Daţi un algoritm eficient care să determine, pentru 2 vârfuri s şi t distincte ale grafului, drumul cu locul îngust cel mai mare (dintre toate drumurile de la s la t în G).*

Soluţie:

Iniţial verificăm dacă s şi t se află in aceeaşi componentă conexă în graful G, adică dacă există drum de la s la t.

*În cazul în care există un astfel de drum vom ordona muchiile, în funcţie de capacităţile asociate, în ordine descrescătoare. Ideea se bazează pe **algoritmul union-find**. Pornim cu toate **nodurile aflate în arbori diferiţi**, deci fiecare nod va fi rădăcina unui arbore.*

Într-un vector cu n componente (unde n este numărul de noduri din arbore) reţinem pentru fiecare nod părintele său in arborele din care face parte la pasul curent. Dacă un nod este rădăcina unui astfel de arbore, el nu are părinte, deci elementul corespunzător lui din vector va avea valoarea -1. Iniţial vectorul conţine evident numai valori de –1 (astfel sugerăm că toate nodurile sunt rădăcini).

***La fiecare pas** vom considera o muchie **(x,y)** din şirul de muchii ordonate descrescător. Vom modifica părinţii nodurilor din arborele lui y astfel încât y să devină rădăcina acelui arbore. Apoi, **rădăcina lui y o vom schimba în x**. Astfel, **am legat arborele lui y de arborele lui x**, păstrând şi muchiile aşa cum sunt în graful iniţial. **Verificăm dacă s şi t se află deja în acelaşi arbore**. Dacă nu, continuăm algoritmul în aceeeaşi manieră ca mai sus.*

***În cazul în care x şi y au fost aduşi deja în aceeaşi componentă**, putem spune că am format un drum de la s la t, şi nu orice drum, ci drumul cu locul îngust maxim.*

***Observaţie**: întotdeauna **s rămâne rădăcină** în arborele în care se află la un moment dat.*

*Pentru a **reface drumul de la s la t**, pornim din t şi mergem în sus în arbore, din părinte în părinte, până ajungem în s. Şirul de noduri obţinut este drumul de la t la s căutat.*

Avem nevoie de următoarele funcţii:

- *Funcţia **union** prin care legăm un arbore de un alt arbore folosind regulile enumerate mai sus;*
- *Funcţia **find** prin care verificăm din care arbore face parte un anume nod;*
- *Funcţia de ordonare descrescătoare a muchiilor după capacităţile asociate.*

Pentru început extragem toate muchiile din graful G şi le punem într-un vector de structuri de tip muchie în care reţinem prima şi a doua extremitate a fiecărei muchii.

```
function ExtrageMuchii(a)
begin
        /*parcurgem matricea de adiacenţă, numai în jumătatea ei superioară, şi mărim numărul
        de muchii descoperite până la momentul curent, introducând şi muchia nou descoperită*/
        m ← 0 //numărul de muchii din graful G
        for i ← 0 to n – 1 do
                for j ← i+1 to n – 1 do
                        if  ( a(i, j) = 1 ) then
        /*Muchii este vectorul de elemente de tip muchie */
                                Muchii[m].extremitate1=i
                                Muchii[m].extremitate2=j
                                m = m+1
end
```

*Funcţia are **complexitatea** $O((n-1)*n/2) = O(n^2)$.*

*În continuare vom utiliza un algoritm de sortare, de preferat **quicksort**, care în cele mai multe cazuri funcţionează cel mai bine, dintre algoritmii de sortare cunoscuţi (nu putem utiliza în acest caz algoritmi de sortare în timp liniar, întrucât nu ne permite tipul valorilor care trebuie ordonate). Complexitatea acestui algoritm este de $O(n^2)$ în cazul cel mai defavorabil, însă timpul mediu de execuţie este $O(nlgn)$.*

```
function QuickSort(Muchii, p, r)
begin
        if (p<r) then
```

```
                q ← Partiţie(Muchii, p, r)
                QuickSort(Muchii, p, q)
                QuickSort(Muchii, q+1, r)
end

function Partiţie(Muchii, p, r)
begin
        x ← f(Muchii[p].extremitate1, Muchii[p].extremitate2)
        i ←p-1
        j ← r+1
        while (true) do
                repeat
                        j ← j-1
                until  C(Muchii[j].extremitate1, Muchii[j].extremitate2) ≥ x
                repeat
                  i ← i+1
                until  C(Muchii[i].extremitate1, Muchii[i].extremitate2) ≤ x
                if (i < j) then
                        interschimbă ( Muchii[i], Muchii[j] )
                else return j
end
```

Se iniţializează vectorul în care reţinem pentru fiecare nod părinţii lui cu –1. Numim acel vector ***Părinte*** *(are dimensiunea n). Algoritmul care răspunde la cerinţă este:*

```
procedure LocIngust ()
begin
        /*putem verifica iniţial dacă t este accesibil din s, printr-o parcurgere bfs sau dfs; în cazul
        în care s şi t nu se află în aceeaşi componentă conexă nu are rost să mai continuăm*/
        if (VerificăAccesibilBFS(s, t) = false) then return
        ExtrageMuchii(a)
        /*în vectorul Muchii avem toate muchiile din graful G */
        QuickSort(Muchii, 0, m-1)
        /*în vectorul Muchii avem toate muchiile din graful G ordonate descrescător după
        capacităţi*/
        i ← 0
        while (s≠ find(t)) do
         //cât timp t nu se află în arborele lui s, deci nu am găsit drum de la s la t
                if (find(Muchii[i].extremitate2) ≠ find(Muchii[i].extremitate1) then
                /*în cazul în care cele două extremităţi se află deja în aceeaşi componentă nu
                facem nici o modificare*/
                        if (Muchii[i].extremitate1=s) then
                                union (s, Muchii[i].extremitate2)
                        else
                         if (Muchii[i].extremitate2=s) then
                                union (s, Muchii[i].extremitate1)
                        else
                /*prin expresiile condiţionale de mai sus ne asigurăm că s rămâne mereu în postura
                de rădăcină */
```

```
                    union (Muchii[i].extremitate1, Muchii[i].extremitate2)
               i ← i+1
          //apelăm  în continuare o funcţie care obţine drumul de la t la s căutat
          Drum (s, t)
end
```

În continuare vom trata funcţiile ***union*** *şi* ***find****.*

```
function find (x)
begin
          temp ← x
          //înaintăm până când ajungem în rădăcină
          while ( Părinte[temp] > 0 ) do

               temp ← Părinte[temp]
          return  temp
end;
```

Complexitatea *funcţiei este* ***O(n)*** *deoarece parcurgem un număr de vârfuri de ordinul lui n****.***

```
function union (x, y)
begin
          temp1 ← Părinte[y]
          temp2 ← y
          while  ( temp1> 0 ) do
//inversăm toate legăturile astfel încât y să devină rădăcina subarborelui în care se află
                p ← Părinte [temp1]
               Părinte[temp1] ← temp2
               temp2  ← temp1
               temp1 ← p
          Părinte[y] ← x
//legăm subarborele cu y drept rădăcină direct de x
end
```

Complexitatea *funcţiei este* ***O(n)****, la fel ca pentru funcţia* ***find.***

```
function Drum (s, t)
begin
          temp ← t
//afişăm nodurile din drumul de la t la s căutat
          while ( temp ≠ s ) do
               afişează temp
               temp ←Părinte[temp]
          afişează s
end.
```

Complexitatea *funcţiei este* ***O(n)****.*

Complexitatea algoritmului este ***O((n-1)*n/2) + O(n^2) + O(n*m)*** *(se apelează funcţiile union şi find de m = numărul de muchii ori). La aceasta se adaugă complexitatea parcurgerii pe care o facem pentru a verifica dacă există drum de la s la t şi care se face în* ***O(m+n)****. Deci* ***complexitatea*** *finală a algoritmului este* ***O((n+1)*(n+m))****.*

Corectitudinea algoritmului *se bazează pe modul în care am realizat unificarea arborilor şi care păstrează muchiile din realitate, spre deosebire de algoritmul de* ***union-find*** *clasic, unde nu conta modul în care se realizau legăturile. Evident ceea ce obţinem în final este un drum de la s la t deoarece în momentul în care am găsit un element din componenta lui s care este legat de un element din componenta lui t ataşăm cele două componente. Ceea ce am obţinut în final este un arbore care îi conţine pe s şi pe t, precum şi muchiile din graful iniţial, până la momentul când am legat cele două componente şi ne-am oprit din adăugarea de muchii.*

Drumul obţinut este şi drumul cu muchiile cele mai lungi şi deci locul îngust al acestui drum este locul ingust cel mai mare dintre locurile înguste ale drumurilor de la s la t; am obţinut această relaţie întrucât am ales muchiile din şirul muchiilor ordonate descrescător.

Vom demonstra prin ***inducţie structurală*** *că în momentul în care ataşăm un arbore unui alt arbore, obţinem un nou arbore în care avem drum de la rădăcină la orice nod al său, drum preluat chiar din graful iniţial.*

Pentru ***pasul de bază*** *ne referim la situaţia iniţială în vectorul* ***Părinte****, când fiecare nod din graful iniţial era rădăcina unui arbore. Când am ataşat un arbore de un alt arbore, am legat de fapt două noduri între care exista muchie în graful iniţial. Deci noul arbore obţinut, format numai din două noduri, respectă proprietatea pe care dorim să o demonstrăm: avem drum de la rădăcină la celălalt nod, drum preluat din graful iniţial.*

La ***pasul inductiv*** *presupunem că avem deja k arbori care respectă proprietatea şi demonstrăm că dacă legăm doi arbori dintre aceştia se obţine un nou arbore care respectă proprietatea.*

Presupunem că ***am gasit o muchie între un nod x dintr-un arbore A_1 şi un nod y dintr-un arbore A_2.*** *Iniţial, algoritmul transformă arborele A_2 astfel încât* ***y să devină rădăcină****. Transformarea s-a făcut păstrând muchiile din graful iniţial, însă s-au modificat părinţii unor noduri din arbore (pentru* ***nodurile aflate în subarborele în care se află şi y, până la nivelul lui y****). Apoi,* ***părintele lui y devine x****.*

Conform ipotezei inductive, ştim că în A_1 ***exista drum de la rădăcină la x, preluat din graful iniţial****.*

De asemenea, ştim ***că în A_2 există drum între orice două noduri****,* ***drum care există şi în graful iniţial*** *(fie a, b două noduri din A_2; din ipoteza inductivă ştiam că există drum de la rădăcină la a şi la b, drumuri din graful iniţial; dacă “lipim” cele două drumuri în rădăcină, obţinem un drum de la a la b cu muchii din graful iniţial). Deci în A_2 avem* ***drum din graful G de la y la orice nod****.*

Din cele ***două relaţii*** *şi din faptul că* ***între x şi y exista muchie****, obţinem că* ***există drum din G de la rădăcina lui A_1, care devine rădăcina arborelui nou format, la orice nod din A_2****. Şi cum ipoteza inductivă spune că avem drum în A_1, preluat din graful iniţial, de la rădăcină la orice nod al său, putem concluziona că pentru arborele nou format se respectă proprietatea.*

În urma demonstraţiei prin inducţie structurală putem spune că ***între s, care era radăcina unui arbore (convenţie făcută în program), şi t, aflat în alt arbore, obţinem drum din graful iniţial, după legarea celor doi arbori*** *.*

Vom demonstra că drumul obţinut este într-adevăr drumul căutat, adică cel cu locul îngust cel mai mare, prin ***reducere la absurd****.*

Presupunem că există un alt drum D_2 în G al cărui loc îngust este mai mare decât locul îngust al drumului descoperit prin acest algoritm, D_1. Fie m_1 muchia care reprezintă locul îngust al lui D_1 şi m_2 muchia care reprezintă locul îngust al lui D_1. Am presupus că $\boldsymbol{m_2 > m_1}$.

Dacă ***m_2 > m_1,*** *înseamnă că m_2 se afla înaintea lui m_1 în şirul de muchii, ordonat descrescător după capacitate. Înseamnă că m_2 a fost descoperită înaintea lui m_1. Cum m_2 era locul îngust al lui D_2, celelalte muchii de pe drumul D_2 se află înaintea sa în şir, deci au fost deja decoperite. Aşadar, întrucât m_1 nu era încă descoperită când am descoperit şi ultima muchie din drumul D_2, putem spune că drumul D_1 a fost descoperit după drumul D_2. Am ajuns la o* ***contradicţie****, deoarece am presupus că* ***D_1 era primul drum descoperit între s şi t*** *(când descoperim un drum algoritmul se termină).*

Am demonstrat că drumul descoperit este şi drumul căutat*, pentru că dacă ar exista un drum cu locul îngust mai mare, acesta ar fi fost descoperit înainte şi algoritmul s-ar fi oprit.*

Deci algoritmul determină într-adevăr drumul de la s la t din graful iniţial, cu locul îngust cel mai mare.

GÎRDEA MARTA
URICARU RALUCA
A34
echipa 21

TEMA NR. 8
22 aprilie 2003

1. *Fie G un graf conex şi o funcţie de cost c:E(G)→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.

b) Deduceţi că, dacă funcţia de cost c este injectivă, atunci G are un unic arbore parţial de cost minim.

c) Sunt adevărate reciprocele afirmaţiilor a şi b?

Soluţie:

a) Fiind conex, graful G acceptă cel puţin un arbore parţial de cost minim. Vom utiliza teorema generală de construcţie a arborilor parţiali de cost minim, particularizată pentru cazul în care funcţia de cost are proprietatea că orice tăietură are o unică muchie de cost minim.

Se fac următoarele notaţii:

- $T^K = (T_1^k, \ldots, T_{n-k}^k)$, *unde* T_i^k, *cu* $1 \le i \le n-k$, *sunt arbori (subgrafuri ale lui G);*
- $V(T_i^k)$ *= partiţie a lui V(G);*
- e^* *este muchia de cost minim printre toate cu o extremitate pe* T_s^k *ales şi cealaltă în* $V-V(T_s^k)$.

Vom demonstra următoarea teoremă:

Fie G=(V,E) cu V = {1, …, n} şi o funcţie de cost c: E(G)→R astfel încât orice A, tăietură, are o unică muchie de cost minim. Dacă G este conex ($T_G \neq \phi$)*, atunci există şi este unic* T^* *(arbore parţial de cost minim) şi pentru oricare k între 0 şi n-1,* $E(T^K) = \bigcup_{j=1}^{n-k} E(T_j^k) \subseteq E(T^*)$ *(pentru k = n - 1 obţinem soluţia problemei)*

Demonstraţie:

Vom demonstra teorema prin ***inducţie după k****, adică după pasul la care am ajuns cu construcţia arborelui.*

Pasul de bază*:*

Pentru k=0 avem G conex, deci există T^* ($T_G \neq \phi$ *şi finită).*

$E(T^*) = \phi \subset E(T^*)$*, deci relaţia este adevărată.*

Pasul inductiv:

Presupunem relaţia adevărată pentru k < n - 1 adică: există şi este unic T^**astfel încât* $E(T^k) \subseteq E(T^*)$.

Trebuie să demonstrăm că există şi este unic T^**astfel încât* $E(T^{k+1}) \subseteq E(T^*)$.

$E(T^{k+1}) = E(T^k) \cup \{e^*\}$*, unde* e^* *este muchia considerată mai sus. Avem două posibilităţi:*

1. *dacă* ***muchia*** e^* ***este din*** $E(T^*)$*, unde* T^* *este arborele parţial de cost minim de la pasul k, atunci la acest pas nu modificăm arborele de la pasul anterior şi deci teorema are loc.*
2. *dacă* ***muchia*** e^* ***nu este din*** $E(T^*)$*, unde* T^* *este arborele parţial de cost minim de la pasul k, atunci* $T^* + e^*$ *are exact un circuit C care are toate muchiile, în afară de* e^**, în* $E(T^*)$. *Întrucât* T^* *este arbore, putem spune că există deja o muchie* e_1 *cu o extremitate pe* T_s^k *şi cealaltă în afară, la fel ca* e^*. *Deci, pentru a obţine un alt arbore parţial de cost minim,*

este necesar să înlocuim e_1 cu e^* în T^*; însă înlocuirea se poate face numai în cazul în care $c(e^*) \leq c(e_1)$. Dar T^* era arbore parţial de cost minim aşadar $c(T^*)$ era minim; nu putem obţine un arbore parţial de cost mai mic decât T^* dar am putea obţine un alt arbore parţial, de cost egal cu $c(T^*)$. Acest lucru s-ar întâmpla dacă am găsi o muchie e^* astfel încât $c(e^*)= c(e_1)$.

*$(V(T_s^k), V(G)-V(T_s^k))$ determină o tăietură iar e_1 este muchia de cost minim din această tăietură. Ştim că funcţia de cost are o singură muchie de cost minim într-o tăietură, aşa că nu există o altă muchie de cost egal cu costul lui e_1. Deci nu găsim o muchie e^* cu proprietatea de mai sus, aşa că **această situaţie nu este posibilă**.*

Aşadar, muchia e^* este obligatoriu din $E(T^*)$ de la pasul anterior, deci T^* nu se poate modifica; el rămâne la fel la fiecare pas.

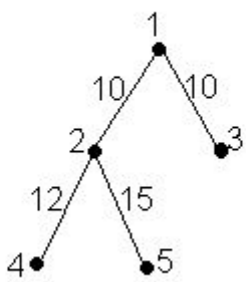
*Am demonstrat că în ipoteza de la care am pornit obţinem **un unic arbore parţial de cost minim**.*

b) Dacă funcţia c este injectivă, înseamnă că pentru oricare două muchii e_1, e_2 din $E(G)$, diferite, avem $c(e_1) \neq c(e_2)$. Deci pentru orice mulţime de tăieturi obţinem că are o unică muchie de cost minim, întrucât orice altă muchie diferită de ea trebuie să aibă cost diferit, şi evident mai mare întrucât muchia considerată era de cost minim.

Ne aflăm acum în condiţiile de la punctul a) şi conform teoremei demonstrate mai sus, obţinem că există un unic arbore parţial de cost minim.

c) Reciprocele afirmaţiilor de la punctele a) şi b) nu sunt adevarate. Acest lucru îl putem arăta cel mai bine pe un graf conex G pe care îl considerăm încă de la început arbore. Este evident că acest graf are un unic arbore parţial de cost minim, şi anume chiar graful G, deoarece nu putem scoate nici o muchie din el pentru că nu ar mai fi conex (este minimal cu această proprietate). Vom arăta că nu este obligatoriu ca funcţia de cost să aibă una din proprietăţile de mai sus: să fie injectivă sau să accepte o unică muchie de cost minim în orice tăietură.

Vom considera un exemplu, sugestiv pentru aceste două cazuri:



După cum putem observa, tăietura determinată de mulţimea S={1} şi V(G)-S={2, 3, 4, 5} are două muchii de cost minim = 10, şi anume: muchia (1,2) şi muchia (1,3). Cu toate acestea, aşa cum am precizat şi anterior, graful nostru adimite un unic arbore parţial de cost minim.

Pentru punctul b), subliniem faptul că funcţia c asociată grafului G nu este injectivă: există două muchii diferite, (1,2) şi (1,3), care au costuri egale: 10.

Deci, reciprocele afirmaţiilor de la punctele a) şi b) nu sunt adevărate, întrucât există situaţii în care există un unic arbore parţial de cost minim pentru un graf şi totuşi nici una din proprietăţile funcţiei de cost de mai sus nu este îndeplinită.

2. *Considerăm 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 elemente 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.

b) Demonstraţi că vectorii caracteristici ai mulţimilor muchiilor circuitelor grafului G generează un subspaţiu vectorial U al lui GF^m ortogonal pe X.

c) Arătaţi că $dim(X) \geq n-1$

d) Arătaţi că $dim(U) \geq m-n+1$

e) Deduceţi că $dim(X) = n-1$ şi că $dim(U) = m-n+1$.

Soluţie:

*a) Vom începe prin a defini **spaţiul vectorial** şi vom specifica ce înseamnă subspaţiu vectorial.*

*Fie K un corp. Se numeşte **spaţiu vectorial** (peste corpul K) un grup abelian (V,+) pe care este dată o lege de compoziţie externă cu operatori în K,*

$\boldsymbol{K \times V \to V, (\alpha, u) \to \alpha u}$, *care verifică axiomele:*

$S_1)$ $(\alpha + \beta)u = \alpha u + \beta u$,

$S_2)$ $\alpha (u + v) = \alpha u + \alpha v$,

$S_3)$ $\alpha (\beta u) = (\alpha \beta)u$,

$S_4)$ $1 u = u$,

Oricare ar fi α, $\beta \in K$ şi $u, v \in V$.

*În cazul nostru, $(GF^m, +_{(mod2)})$ este grupul abelian iar $(GF, +_{(mod2)}, *)$ este corpul K din definiţie. Pentru a demonstra că X este un subspaţiu vectorial al lui GF^m , **iniţial, trebuie să demonstrăm că X este subgrup al lui GF^m**.*

*$0_m \in$ **X** din ipoteză.*

*Este evident că **X are acelaşi element neutru ca şi GF^m**, şi anume vectorul 0_m.*

*Trebuie să **demonstrăm că din oricare două elemente din X, compuse prin operaţia de adunare modulo 2 pe vectori, se obţine un element tot din X.***

De fapt, trebuie să demonstrăm că din două tăieturi A_1 şi A_2, reprezentate prin vectori caracteristici, se obţine o nouă tăietură A_3, $\boldsymbol{A_3=A_1+A_2}$. În momentul în care adunăm cei doi vectori, folosind adunarea modulo 2, putem avea următoarele cazuri, cu semnificaţiile proprii:

***0+0 = 0** – dacă o muchie **i**, cu i între 0 şi m-1, nu aparţine nici lui A_1 şi nici lui A_2, atunci ea nu aparţine nici lui A_3 (în vectorul caracteristic al lui A_3 vom avea 0 pe poziţia i)*

***0+1 = 1+ 0 = 1** – dacă o muchie **i** aparţine măcar uneia dintre tăieturi, atunci aparţine şi lui A_3; deci va fi 1 pe poziţia i în vectorul lui A_3.*

***1+1 = 0** – dacă o muchie aparţine ambelor tăieturi, ea nu aparţine lui A_3; vom avea 0 pe poziţia i în vectorul A_3.*

*Aceste trei situaţii se pot traduce în limbajul mulţimilor prin **diferenţa simetrică**, şi anume $\boldsymbol{A_3 = A_1 \Delta A_2}$. În A_3 avem numai acele muchii care nu se află şi în A_1 şi în A_2.*

*Trebuie să arătăm că **noua mulţime de muchii A_3 este tot tăietură**. Pentru aceasta, **trebuie să găsim o mulţime S_3**, corespunzătoare lui A_3, astfel încât muchiile din A_3 să aibă o extremitate în S_3 şi cealaltă în $V(G)$-S_3. Vom demonstra că mulţimea S_3 corespunzătoare este $\boldsymbol{S_1 \Delta S_2}$, unde S_1, S_2 sunt mulţimile corespunzătoare lui A_1, respectiv A_2; arătăm că toate muchiile cu o extremitate în $\boldsymbol{S_1 \Delta S_2}$ şi cealaltă în afară se află în A_3.*

$\boldsymbol{S_1 \Delta S_2}$*=$\{v \mid v \in S_1$ şi $v \notin S_2$ sau $v \in S_2$ şi $v \notin S_1\}$*

*Demonstrăm prin **dublă inegalitate**:*

1) $\boldsymbol{A_3 \subseteq \{uv \in E(G) \mid u \in S_1 \Delta S_2 \text{ şi } v \in V(G)-(S_1 \Delta S_2)\}}$

Considerăm o muchie ***uv din*** A_3. *Ştim că* $uv \in A_1 \Delta A_2$. *Presupunem că* $uv \in A_1$. *Deci* ***u se află în*** S_1 ***şi v se află în*** $V(G)-S_1$***, sau invers***. *Deoarece* $uv \notin A_2$ *înseamnă că* ***u şi v se află amândouă în*** S_2 ***sau în*** $V(G)-S_2$.
Dacă u şi v $\in S_2$, *avem* $u \notin S_1 \Delta S_2$ *şi* $v \in S_1 \Delta S_2$. *Deci avem muchia uv inclusă în mulţimea* $\{uv \in E(G) \mid u \in S_1 \Delta S_2 \text{ şi } v \in V(G)-(S_1 \Delta S_2)\}$.
Dacă u şi v $\in V(G)-S_2$, *avem* $v \notin S_1 \Delta S_2$ *şi* $u \in S_1 \Delta S_2$. *Deci avem muchia uv inclusă în mulţimea* $\{uv \in E(G) \mid u \in S_1 \Delta S_2 \text{ şi } v \in V(G)-(S_1 \Delta S_2)\}$.
Analog*, obţinem şi pentru* ***v în*** S_1 ***şi u în*** $V(G)-S_1$.

2) $\{uv \in E(G) \mid u \in S_1 \Delta S_2 \text{ şi } v \in V(G)-(S_1 \Delta S_2)\} \subseteq A_3$

Considerăm o muchie ***uv din*** $\{uv \in E(G) \mid u \in S_1 \Delta S_2 \text{ şi } v \in V(G)-(S_1 \Delta S_2)\}$.
Presupunem că $u \in S_1$ ***şi*** $u \notin S_2$. *Din faptul că* $v \in V(G)-(S_1 \Delta S_2)$ *putem spune că v fie nu aparţine nici lui* S_1, *nici lui* S_2, *fie aparţine lui* $S_1 \cap S_2$.
Dacă $v \notin S_1$ ***şi*** $v \notin S_2$, *înseamnă că* $v \in V(G)-S_1$ *şi* $v \in V(G)-S_2$. *Conform presupunerii anterioare, obţinem că* $uv \in A_1$ ***şi*** $uv \notin A_2$***, deci*** $uv \in A_3$.
Dacă $v \in S_1 \cap S_2$, *înseamnă că* $v \in S_1$ *şi* $v \in S_2$. *Conform presupunerii anterioare, obţinem că* $uv \in A_2$ ***şi*** $uv \notin A_1$***, deci*** $uv \in A_3$.
Analog, demonstrăm şi în presupunerea că $u \in S_2$ ***şi*** $u \notin S_1$.

Conform cazurilor 1 şi 2, obţinem că $A_3 = \{uv \in E(G) \mid u \in S_1 \Delta S_2 \text{ şi } v \in V(G) - (S_1 \Delta S_2)\}$. ***Deci*** A_3 ***este tăietură, cu*** $S_3 = S_1 \Delta S_2$.

Întrucât ***simetricul x' al unui element*** x^A ***din spaţiul X este tot*** x^A ($x^A + x^A = 0$) *care aparţine lui X şi* ***compunerea a oricare două elemente din X este tot un element din X, putem spune că X este subgrup al lui*** GF^m.

În continuare, verificăm axiomele:

S_1) $(\alpha + \beta)u = \alpha u + \beta u,\ \alpha, \beta \in \{0,1\},\ u \in X$

$\alpha = 0$ şi $\beta = 1 : (\alpha + \beta)u = u$

$\alpha u + \beta u = 0_m + u = u$

$\alpha = 0$ şi $\beta = 0 : (\alpha + \beta)u = 0_m \in X$

$\alpha u + \beta u = 0_m + 0_m = 0_m$

$\alpha = 1$ şi $\beta = 1 : (\alpha + \beta)u = 0_m u = 0_m \in X$

$\alpha u + \beta u = u + u = 0_m$

S_2) $\alpha (u + v) = \alpha u + \alpha v,\ \alpha \in \{0,1\},\ u, v \in X$

$\alpha = 0 :\ \alpha (u + v) = 0_m \in X$

$\alpha u + \alpha v = 0_m + 0_m = 0_m$

$\alpha = 1 :\ \alpha (u + v) = u + v \in X$ *aşa cum am demonstrat anterior*

$\alpha u + \alpha v = u + v$

S_3) $\alpha (\beta u) = (\alpha \beta)u,\ \alpha, \beta \in \{0,1\},\ u \in X$

$\alpha = 0$ şi $\beta = 1 :\ \alpha (\beta u) = 0\, u = 0_m = (\alpha \beta)u$

$\alpha = 0$ şi $\beta = 0 :\ \alpha (\beta u) = 0\, 0_m = 0_m = (\alpha \beta)u$

$\alpha = 1$ şi $\beta = 1 :\ \alpha (\beta u) = 1\, u = u = (1*1)\, u = (\alpha \beta)u$

$\alpha = 1$ şi $\beta = 0$: $\alpha(\beta u) = 1\ 0_m = 0_m = 0\ u = (\alpha\ \beta)u$

S_4) $1\ u = u$, $u \in X$ evident

Întrucât X este subgrup al grupului GF^m şi respectă axiomele, putem spune că X este subspaţiu vectorial al lui GF^m.

b) $U = <\{x^C \mid C$ circuit în $G\}>$

Mulţimea tuturor vectorilor caracteristici asociaţi circuitelor din graful G este inclusă în U. Notăm un astfel de vector cu x^C, unde $x^C[i] = 1$ dacă muchia i se află în circuitul C şi 0 altfel.

Pentru a demonstra că subspaţiul vectorial generat de vectorii caracteristici asociaţi mulţimilor muchiilor circuitelor grafului G este ortogonal pe X, trebuie să considerăm vectorul caracteristic al unui circuit (din U), să îl înmulţim cu vectorul caracteristic al unei tăieturi (din X) şi să demonstrăm că obţinem rezultatul 0.

Fie C un circuit şi x^C vectorul caracteristic asociat şi fie A o tăietură. Prin înmulţirea vectorului x^C cu x^A înţelegem produsul scalar al celor doi vectori, şi anume:

$$x^C \times x^A = \left(\sum_{i=0}^{m-1} x^C(i)x^A(i)\right) \ (\text{mod } 2).$$

*De fapt $\sum_{i=0}^{m-1} x^C(i)x^A(i)$ **reprezintă numărul de muchii pe care le au în comun circuitul C şi tăietura A**, întrucât $x^C(i)x^A(i)$ este 1 numai în momentul în care amândoi sunt 1, deci numai dacă muchia I se află şi în circuit şi în tăietură. **Deoarece suma obţinută este modulo 2, putem spune că de fapt, produsul scalar este 0 dacă circuitul şi tăietura au un număr par de muchii în comun şi 1 dacă numărul de muchii în comun este impar.***

*Deci **demonstraţia noastră se reduce la a demonstra că orice circuit şi tăietură au în comun număr par de muchii** (numai în acest caz produsul scalar este 0).*

Fie C un circuit şi A o tăietură. Fie k numărul de muchii comune lui A şi C. Fie S şi $V - S$ mulţimile corespunzătoare tăieturii A.

Dacă $k = 0$, k este par, ceea ce trebuia demonstrat.

Dacă $k > 0$:

Există un nod $v \in V(C) \cap S$ (evident, deoarece, dacă acest circuit are muchii comune cu o tăietură, atunci, în mod necesar, el are noduri în ambele submulţimi determinate de bipartiţia corespunzătoare tăiaturii).

Circuitul C este un drum închis de la v la v. Fiecare muchie din cele k comune tăieturii A şi circuitului C este o trecere din S în $V - S$ sau invers.

Presupunem că pornim din nodul v şi parcurgem muchiile acestui circuit în ordinea în care ele apar (rezultatul este acelaşi indiferent de direcţia în care se porneşte).

Întrucât capătul „final" al acestui drum închis se găseşte în S, pentru fiecare muchie te trecere de la S la $V - S$ trebuie să existe o altă muchie de revenire în S. În consecinţă, k trebuie să fie par.

*Aşadar, **orice circuit şi tăietură au în comun număr par de muchii** şi după cum am explicat mai sus, putem spune că U este ortogonal pe X.*

c) Vom încerca să construim incremental (în maniera construcţiei arborelui parţial de cost minim) tăieturi, prin adăugarea la fiecare pas a unui nod; vom demonstra că prin acest mod de construcţie se obţin n - 1 tăieturi liniar independente, adică nici o tăietură nu poate fi obţinută dintr-o combinaţie liniară de alte tăieturi. Prin urmare, există cel puţin n - 1 tăieturi liniar independente.

Se porneşte iniţial cu un singur nod aflat în mulţimea S a primei tăieturi. La fiecare pas se adaugă un nou nod în S-ul de la pasul anterior.

Facem convenţia ca nodul ales la pasul ***i*** *să fie notat cu* $\boldsymbol{u_i}$ *iar mulţimea S obţinută la acest pas să fie* $\boldsymbol{S_i}$*. Tăietura determinată de mulţimea* S_i *va fi* A_i*.*

1. Vom demonstra că tăieturile obţinute în această manieră sunt ***distincte****.*

Reducere la absurd:

Presupunem că există S_i, S_j *cu* $j > i$, *a.î.* $A_i = A_j$.

Din modul de construcţie a mulţimilor S $\Rightarrow S_i \subset S_j$ *(incluziune strictă), adică*

$$S_j = S_i \cup \{u_1, \ldots, u_{j-i}\}.$$

Mulţimea $\{u_1, \ldots, u_{j-i}\}$ *este evident nevidă, deoarece am presupus* $j > i$, *având exact* $j - i$ *noduri.*

Vom studia următoarele două cazuri(care acoperă toate posibilităţile):

Cazul 1: $\exists w \in S_i$, $\exists u_p \in S_j - S_i$ *a.î.* $wu_p \in E(G)$. *Atunci:*

- $u_p \in S_j - S_i \Rightarrow u_p \notin S_i \Rightarrow \boldsymbol{u_p \in V(G) - S_i}$; *în plus,* $w \in S_i$; $\Rightarrow u_p$ *şi w sunt în submulţimi diferite ale bipartiţiei corespunzătoare tăieturii* $A_i \Rightarrow \boldsymbol{wu_p \in A_i}$;
- $u_p \in S_j - S_i \Rightarrow \boldsymbol{u_p \in S_j}$; *în plus,* $w \in S_i$ *şi* $S_i \subset S_j$ *(aşa cum am arătat mai sus)* $\Rightarrow \boldsymbol{w \in S_j} \Rightarrow$ u_p *şi w sunt în aceeaşi submulţime dintre cele două corespunzătoare tăieturii* $A_j \Rightarrow \boldsymbol{wu_p \notin A_j}$.

Am arătat deci că, în acest caz, $wu_p \in A_i$ *şi* $wu_p \notin A_j \Rightarrow A_i - A_j \neq \Phi \Rightarrow \boldsymbol{A_i \neq A_j}$.

Cazul 2: $\forall w \in S_i$, $\forall u_p \in S_j - S_i$ *a.î.* $wu_p \notin E(G)$.

Atunci, întrucât G este conex, $\exists u_p \in S_j - S_i$ *şi* $\exists s \in V(G) - S_j$, *a.î.* $su_p \in E(G)$. *(Am precizat că* $s \in V(G) - S_j$, *deoarece:*

- *am presupus în ipoteza acestui caz că nici un nod din* $S_j - S_i$ *nu este adiacent cu nici un nod din* S_i;
- *dacă pentru orice* $u_p \in S_j - S_i$ *nu ar exista nici un alt nod din* $V(G) - S_j$ *cu care să fie adiacent, nodurile din* $S_j - S_i$ *nu ar fi accesibile din nici un alt nod din graf care nu aparţine acestei mulţimi, deci graful nu ar fi conex.).*

- $u_p \in S_j - S_i \Rightarrow u_p \in S_j$; *în plus,* $s \in V(G) - S_j \Rightarrow u_p$ *şi s sunt în submulţimi diferite ale bipartiţiei corespunzătoare tăieturii* $A_j \Rightarrow \boldsymbol{su_p \in A_j}$;
- $u_p \in S_j - S_i \Rightarrow u_p \notin S_i \Rightarrow \boldsymbol{u_p \in V(G) - S_i}$; *în plus,* $s \in V(G) - S_j$ *şi* $S_i \subset S_j$ *(aşa cum am arătat mai sus)* $\Rightarrow$ $\boldsymbol{s \in V(G) - S_i} \Rightarrow u_p$ *şi s sunt în aceeaşi submulţime dintre cele două corespunzătoare tăieturii* $A_i \Rightarrow \boldsymbol{su_p \notin A_i}$.

Am arătat deci că, şi în acest caz, $su_p \in A_j$ *şi* $su_p \notin A_i \Rightarrow A_j - A_i \neq \Phi \Rightarrow \boldsymbol{A_j \neq A_i}$.

2. Vom demonstra că tăieturile obţinute în această manieră sunt ***liniar independente*** *şi deci ele aparţin bazei spaţiului vectorial X.*

Demonstraţia o vom face prin ***inducţie****:*

Pasul de bază:

Se alege un nod oarecare pe care am convenit să-l notăm cu u_0. *Mulţimea* S_0 *este formată numai din elementul* u_0. A_0 *este formată numai din acele muchii care au o extremitate în* u_0.

$A_0 = \{ u_0v \mid v \in V(G), u_0v \in E(G) \}$

Întrucât graful G este conex, u_0 *are cel puţin un nod adiacent cu el în mulţimea* $V - \{u_0\}$. *Deci* A_0 *este mulţime nevidă (are cel puţin o muchie) şi vectorul caracteristic asociat* x^{A_0} *este diferit de vectorul nul* 0_m.

Aşadar $\alpha_0 x^{A}{}_0 = 0_m$ *,* $\alpha_0 \in \{0,1\} \Leftrightarrow \alpha_0 = 0$*. Deci mulţime de tăieturi* $\{A_0\}$ *este liniar independentă.*

Pasul inductiv:

Presupunem că toate tăieturile până la A_k *inclusiv, cu* $k<n-1$ *sunt liniar independente. Vom alege nodul* u_{k+1} *din* $V - S_k$*.*

$S_{k+1} = S_k \cup \{ u_{k+1}\}$.

Tăietura A_{k+1} *va cuprinde toate muchiile din* A_k*, mai puţin eventualele muchii cu o extremitate în* u_{k+1}*, la care se adaugă eventualele muchii cu o extremitate în* u_{k+1} *şi cealaltă în* $V(G)-S_{k+1}$*. Deci* $A_{k+1} = A_k - \{ u_{k+1}v |\ v \in S_k,\ u_{k+1}v \in E(G) \} \cup \{ u_{k+1}v |\ v \in V(G)-S_{k+1},\ u_{k+1}v \in E(G) \}$*.*

Noua tăietură este diferită de toate cele de înainte prin cel puţin una din următoarele două situaţii (sau chiar amândouă):

- *dacă* u_{k+1} *are vecini în mulţimea* $V(G)-S_{k+1}$ *atunci se formează muchii care nu au existat până acum în nici o altă tăietură, deoarece până la acest pas* u_{k+1} *şi aceşti vecini ai săi s-au aflat în aceeaşi mulţime:* $V(G)-S_i$*, cu i între 0 şi k.*
- *dacă* u_{k+1} *are vecini în mulţimea* S_k *atunci* A_{k+1} *diferă de toate celelalte tăieturi de înainte deoarece această tăietură nu mai conţine muchiile dintre* u_{k+1} *şi aceşti vecini.*

Demonstraţie: *Deoarece graful este conex, există drum între oricare două noduri.*

Demonstrăm *că mulţimea de tăieturi* $\{A_0, A_1, \ldots A_k, A_{k+1}\}$ *rămâne liniar independentă.*

Avem $\sum_{i=0}^{k} \alpha_i\ x^{A}{}_i = \alpha_{k+1} x^{A}{}_{k+1},\ \alpha_i \in \{0,1\}$*.*

Cazul 1

$\alpha_{k+1} = 0 \Rightarrow \sum_{i=0}^{k} \alpha_i\ x^{A}{}_i = 0_m \Leftrightarrow \alpha_i = 0,\ \forall\ i=0,\ldots,\ k$ *conform ipotezei inductive că tăieturile până la pasul k inclusiv sunt liniar independente.*

Cazul 2

$\alpha_{k+1} = 1 \Rightarrow \sum_{i=0}^{k} \alpha_i\ x^{A}{}_i + x^{A}{}_{k+1} = 0_m.$

De la punctul ***a)*** *ştim că suma a două tăieturi reprezentate prin vectori caracteristici este de fapt diferenţa simetrică a celor două tăieturi. Acest rezultat se păstrează şi pentru suma dintre mai multe tăieturi. Întrucât diferenţa simetrică este asociativă, putem scrie că:*

$$\sum_{i=0}^{k} \alpha_i\ x^{A}{}_i + x^{A}{}_{k+1} \Leftrightarrow \underset{\alpha_i = 1}{\Delta} S_i \text{ , cu i de la 0 la k+1}$$

Dar $\underset{\alpha_i = 1}{\Delta} S_i$ *este nevidă întrucât conţine nodul* u_{k+1} *(acest nod nu apare decât în mulţimea* S_{k+1} *şi deci rămâne în diferenţa simetrică. Deci şi* $\sum_{i=0}^{k} \alpha_i\ x^{A}{}_i + x^{A}{}_{k+1}$ *este diferită de vectorul nul.*

Deci, în ambele cazuri, vectorii caracteristici asociaţi tăieturilor până la pasul k+1 inclusiv sunt liniar independenţi. Aşadar aceste tăieturi sunt liniar independente şi deci aparţin bazei spaţiului vectorial X. Întrucât se pot face n-1 adăugări de noduri, până când în mulţimea V(G)-S rămâne un singur nod, putem spune că ***avem n-1 astfel de tăieturi liniar independente*** *care aparţin bazei lui X şi deci* $\mathbf{dim(X) \geq n-1}$*.*

d) Considerăm un arbore parţial al grafului G. Ştim că un astfel de arbore are exact n-1 muchii, este conex şi deci între oricare două noduri există un unic drum.

***Dacă adăugăm câte o muchie la acest arbore, din cele care nu sunt incluse deja, obţinem la fiecare pas cel puţin un nou ciclu**. Presupunem că adăugăm muchia uv, care nu fusese deja adăugată. Ştim că **între u şi v există un drum în arbore format numai din muchiile iniţiale ale acestuia**; prin adăugarea acestei muchii, se închide respectivul drum şi se formează un ciclu care conţine numai muchii iniţiale, mai puţin muchia uv. Se pot adăuga **m-n+1** astfel de muchii. **Toate circuitele de această formă (cu o singură muchie în afară de cele iniţiale) sunt liniar independente**, adică nu se pot obţine prin combinaţii liniare ale altor circuite, întrucât fiecare circuit are o muchie care nu poate fi obţinută din celelalte circuite (muchia adăugată care nu aparţinea iniţial arborelui).*

*Deci aceste circuite aparţin bazei spaţiului vectorial U şi obţinem că **dim(U) $\geq$ m-n+1**.*

*e) Întrucât X şi U sunt ortogonale (deci nu au elemente în comun) şi sunt subspaţii ale spaţiului vectorial GF^m, putem spune că **dim(GF^m) $\geq$ dim(U) + dim(X)**. **Dimensiunea lui GF^m este m** întrucât baza acestui spaţiu vectorial este formată numai din acei vectori caracteristici care au 1 pe o singură poziţie (toţi ceilalţi vectori pot fi obţinuţi din aceştia).*

*Deci **m $\geq$ dim(U) + dim(X)** **(1)**.*

Dar am demonstrat mai sus că:

dim(U) $\geq$ m-n+1

dim(X) $\geq$ n-1

Aşadar dim(U) + dim(X) $\geq$ m (2).

*Din **(1)** şi **(2)** obţinem că **dim(U) + dim(X) = m.***

*Şi cum dim(U) $\geq$ m-n+1 şi dim(X) $\geq$ n-1 obţinem **dim(X) = n-1 şi dim(U) = m-n+1.***

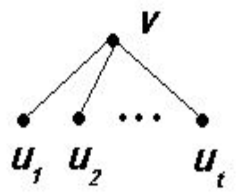
3. *Arătaţi că orice arbore cu gradul maxim $t > 0$ are cel puţin t vârfuri pendante.*

Soluţie:

Fie T un arbore cu $n = |T|$ şi $\Delta(T) = t$.

*Atunci există un nod **$v \in V(T)$** astfel încât **$d(v) = \Delta(T) = t$**.*

Fie $u_1, u_2, \dots, u_t$ vecinii lui v.



Printr-o parcurgere BFS a arborelui T pornind din nodul v se obţine un arbore de lăţime, notat T', care este, de fapt, arborele T în care considerăm nodul v ca fiind rădăcină.

T' şi T au, evident, acelaşi număr de noduri, aceleaşi muchii, deci, implicit, acelaşi număr de noduri pendante (mai mult, nodul u este pendant în T' dacă şi numai dacă este pendant în T). Este suficient deci să studiem numărul de noduri pendante din T'.

În fiecare din subarborii din T' cu rădăcinile $u_1, u_2, \dots, u_t$ există cel puţin un nod de grad 1. (Demonstraţie: Reducere la absurd:

Presupunem că $\forall u \in T_{uk}$, $d(u) \geq 2$, unde T_{uk} este subarborele din T' cu rădăcina u_k.

Deoarece numărul de muchii din orice graf este jumătate din suma gradelor nodurilor, se obţine:

$$|E(T_{uk})| = \frac{1}{2}\sum_{u\in V(T_{uk})} d(u) \geq \frac{1}{2}(2|V(T_{uk})|) \geq |V(T_{uk})|$$

Dar T_{uk} este arbore, şi în orice arbore $T = (V, E)$ avem $|E| = |V| - 1$, deci

$$|V(T_{uk})| - 1 \geq |V(T_{uk})| \rightarrow \textbf{contradicţie})$$.

Aşadar, fiecare din cei t subarbori are cel puţin un nod pendant. Acestea sunt distincte, deoarece subarborii sunt disjunţi.

*În cosdsecinţă, T' are cel puţin t noduri pendante $\Rightarrow$ **T are cel puţin t noduri pendante**.*

4. *Fie $T = (V, E)$ un arbore cu rădăcina r (un vârf oarecare) şi cu $parent(v)$ 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 astfel încât $w\ parent(v) \in M$.*

a) Demonstraţi că orice cuplaj propriu este de cardinal maxim.

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

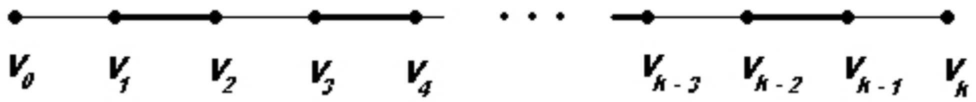
Soluţie:

a) *Fie M un cuplaj propriu al lui T.*

Conform ***Teoremei lui Berge****, M este cuplaj de cardinal maxim $\Leftrightarrow$ M nu admite drumuri de creştere. Este suficient deci să arătăm că M nu admite drumuri de creştere.*

Reducere la absurd:

Presupunem că există în T un drum de creştere ***P*** *relativ la M.*



Conform definiţiei drumului de creştere, v_0 şi v_k sunt vârfuri expuse. Din definiţia cuplajelor proprii, orice vârf expus ***u*** *are un frate* ***w*** *astfel încât $w\ parent(v) \in M$.*

În consecinţă, v_0 are un frate w_0 a.î. $w_0\ parent(v_0) \in M$, respectiv v_k are un frate w_k a.î. $w_k parent(v_k) \in M$.

T este arbore $\Rightarrow$ T este conex $\Rightarrow$ există drum între oricare două noduri.

În plus, într-un arbore aceste drumuri sunt unice.

Fie $\mathbf{v_i}$ *acel nod de pe drumul de creştere P considerat mai sus, cu proprietatea că drumul D_i de la r (rădăcina arborelui T) la v_i are*

$$lungime(D_i) = \mathbf{min}\{lungime(D) \mid D \text{ drum de la } r \text{ la un nod } v_j \text{ de pe } P\}.$$

$\rightarrow$ *v_i este* ***unic*** *cu această proprietate.*

*(**Demonstraţie**: Presupunem că $\exists\, v_j \neq v_p \in V(P)$ a.î.*

$$lungime(D_j) = lungime(D_p) = \mathbf{min}\{lungime(D) \mid D \text{ drum de la } r \text{ la un nod } v_j \text{ de pe } P\}.$$

Există drum de la r la v_j care trece prin v_p. Acesta se obţine prin concatenarea drumului D_p cu drumul P_{jp}, unde P_{jp} este acea porţiune din P aflată între nodurile v_j şi v_p.

$v_j \neq v_p \Rightarrow lungime(P_{jp}) > 0$.

Deci lungimea drumului de la r la v_j prin v_p este strict mai mare decât $lungime(D_p)$, deci şi decât $lungime(D_j)$. În consecinţă, drumul de la r la v_j prin v_p este diferit de D_j, adică există două noduri în arborele T între care sunt mai multe drumuri (mai mult de un drum) $\Rightarrow$ ***contradicţie****).*

$\rightarrow$ *Într-un arbore, numărul nivelului pe care se găseşte un nod u este dat de distanţa dintre acest nod u şi rădăcina arborelui.*

Distanţa de la r la v_j este mai mare decât distanţa de la r la v_i, $\forall\, v_j \neq v_i \in V(P) \Rightarrow$ nodurile v_j se găsesc pe nivelurie inferioare nivelului lui v_i în T.

De asemenea, în orice arbore, oricare două noduri adiacente sunt în relaţia părinte - fiu. În plus, întotdeauna părintele este unic şi se găseşte pe nivelul imediat superior nuvelului fiului.

Presupunem că $\mathbf{v_i}$ *se află în T pe nivelul* ***h****. Vom demonstra prin* ***inducţie*** *că:*

- *v_{i+j} se găseşte pe nivelul **h+j**, avându-l pe $v_{i+(j-1)}$ ca părinte ($j = \overline{1, k-i}$);*
- *v_{i-j} se găseşte pe nivelul **h+j**, avându-l pe $v_{i-(j-1)}$ ca părinte ($j = \overline{1, i}$)**.***

Pasul I:

- *dacă $i > 0$, v_{i-1} este adiacent cu v_i; distanţa(r, v_{i-1}) > distanţa(r, v_i) (din modul de alegere a lui v_i) $\Rightarrow$ v_{i-1} este fiu al lui v_i (nu poate fi părinte, deoarece se găseşte pe un nivel inferior) şi se află în arbore pe nivelul h+1 (distanţa(r, v_{i-1}) = distanţa(r, v_i) + distanţa(v_i, v_{i-1}));*
- *dacă $i < k$, v_{i+1} este adiacent cu v_i; distanţa(r, v_{i+1}) > distanţa(r, v_i) (din modul de alegere a lui v_i) $\Rightarrow$ v_{i+1} este fiu al lui v_i (nu poate fi părinte) şi se află în arbore pe nivelul h+1.*

Pasul II:

Presupunem afirmaţiile adevărate pentru $j1 \le m1$, respectiv $j2 \le m2$ ($1 \le j1 \le m1 < k - i$, $1 \le j2 \le m2 < i$).

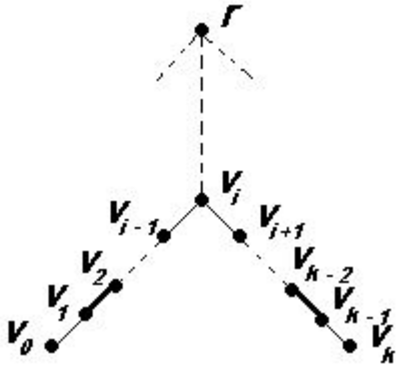
- *Demonstrăm pentru **j1 = m1 + 1**:*
  Din ipoteza inductivă $\Rightarrow$
    - *v_{i+m1} se află pe nivelul h + m1;*
    - *parent(v_{i+m1}) = v_{i+m1-1} .*

  v_{i+m1+1} este adiacent cu v_{i+m1} (deci sunt în relaţia părinte – fiu) şi v_{i+m1} are un unic părinte (pe v_{i+m1-1}) $\Rightarrow$
  *$\Rightarrow$ **v_{i+m1} = parent(v_{i+m1+1})** $\Rightarrow$*
  $\Rightarrow$ v_{i+m1+1} se află pe nivelul imediat inferior nivelului lui v_{m1}, şi anume pe
  *(h + m1) + 1 = **h + (m1 + 1)**.*

- *Demonstrăm pentru **j2 = m2 + 1**:*
  Din ipoteza inductivă $\Rightarrow$
    - *v_{i-m2} se află pe nivelul h + m2;*
    - *parent(v_{i-m2}) = v_{i-m2-1} .*

  $v_{i-(m2+1)}$ este adiacent cu v_{i-m2} (deci sunt în relaţia părinte – fiu) şi v_{i-m2} are un unic părinte (pe $v_{i-(m2-1)}$) $\Rightarrow$
  *$\Rightarrow$ **v_{i-m2} = parent($v_{i-(m2-1)}$)** $\Rightarrow$*
  $\Rightarrow$ $v_{i-(m2+1)}$ se află pe nivelul imediat inferior nivelului lui v_{i-m2}, şi anume pe
  *(h + m2) + 1 = **h + (m2 + 1)**.*

Aşadar, schematic, poziţia nodurilor şi a muchiilor de pe drumul P în arborele T este:



Evident, dacă $v_i = v_0$ sau $v_i = v_k$, una dintre cele două ramuri care pornesc de la v_i lipseşte.

Vom presupune că $\mathbf{v_i} \neq \mathbf{v_k}$. *Aceasta corespunde cazurilor:*

- $v_i \neq v_0$ *şi* $v_i \neq v_k$;
- $v_i = v_0$ *(în acest caz, evident,* $v_i \neq v_k$, *deoarece orice drum de creştere are lungimea nenulă, deci* $v_0 \neq v_k$*).*

Cazul când $v_i = v_k$ *se tratează similar cu cel tratat mai jos, înlocuind peste tot orice indice j (*$j = \overline{0,k}$*) cu* $k - j$.

$v_{k-1} = parent(v_k)$ *şi* v_k *este nod expus* $\Rightarrow$ *are un frate* w_k *a.î.* $w_k parent(v_k) \in M$.

$parent(w_k) = v_{k-1} \Rightarrow w_k v_{k-1} \in M$.

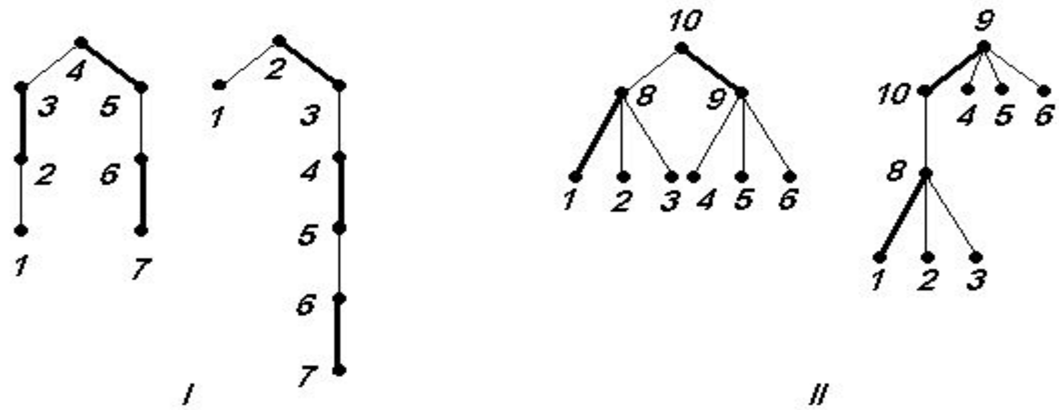
Dar $v_{k-2} v_{k-1} \in M$ *şi* $w_k \neq v_{k-2}$, *(deoarece* $v_{k-2} = parent(v_{k-1})$*), deci nu poate fi şi fiu al acestuia)* $\Rightarrow$

$\Rightarrow d_M(v_{k-1}) \geq 2 \Rightarrow$ ***contradicţie*** *cu definiţia cumplajului, conform căreia, dacă M este cuplaj în G, atunci,* $\forall u \in V(G)$, $d_M(v) \leq 1$.

În consecinţă, presupunerea făcută este ***falsă*** $\Rightarrow$ *M nu admite drumuri de creştere* $\Rightarrow$ ***M este cuplaj de cardinal maxim,*** *conform teoremei lui Berge.*

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

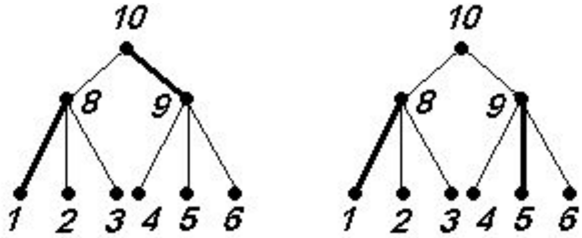
Pentru a arăta aceasta este suficient să arătăm că există un algoritm care construieşte, pentru orice arbore cu ***n*** *vârfuri, dat prin listele de adiacenţă, un cuplaj propriu în timpul O(n).*



Observaţie: *Aşa cum se vede în cele două exemple de mai sus, pentru un arbore T, un cuplaj M este sau nu propriu în funcţie de nodul considerat rădăcină.*

În exemplul I, dacă rădăcina este 4, atunci există un nod expus (acest nod este 1) care nu are nici un frate, deci pentru care nu există w fiu al lui 2 astfel încât 2w $\in$ *M (M = {32, 45, 67}). Această problemă dispare dacă vom considera nodul 2 ca rădăcină a arborelui.*

De asemenea, dacă studiem arborele din exemplul II, vom constata că, indiferent de cuplajul maxim considerat, dacă rădăcina arborelui este 10, respectivul cuplaj nu este propriu, întrucât fie rădăcina, fie o parte din nodurile frunze sunt noduri expuse fără a avea un frate care împreună cu părintele lor comun să formeze o muchie din cuplaj.



O condiţie necesară şi suficientă ca un cuplaj M să fie prorpiu pentru un arbore este ca orice nod expus să aibă un frate care, împreună cu părintele lor comun, să formeze o muchie din cuplajul M.

O altă condiţie necesară pentru ca un cuplaj M să fie propriu este aceea ca rădăcina să nu fie nod expus. Rădăcina nu are fraţi, deci, dacă ar fi nod expus într-un cuplaj M, atunci respectivul cuplaj nu ar putea fi propriu pentru arborele respectiv cu rădăcina considerată. În plus, se precizează că rădăcina arborelui pentru care se construieşte cuplajul propriu este un nod oarecare v, aşadar nodul rădăcină nu prezintă importanţă.

În plus, cuplajele proprii sunt întotdeauna cuplaje de cardinal maxim, deci se folosesc pentru a putea construi pentru arbori cuplaje de cardinal maxim în timp liniar (în condiţiile în care, pentru grafuri oarecare, un cuplaj maximal se construieşte în $O(n^3)$). Un cuplaj de cardinal maxim îşi păstrează proprietatea indiferent de nodul rădăcină a arborelui, aşadar nodul pe care îl cosiderăm rădăcină în momentul construcţiei cuplajului propriu nu afectează rezultatul final.

*O posibilitate de construcţie a unui astfel de cuplaj este parcurgerea **DFS** a arborelui T şi selectarea acelor unui număr de muchii care să îndeplinească condiţia de mai sus.*

*Trebuie evitată acea situaţie când pentru un nod **frunză** u expus nu se găseşte nici un alt nod frate care împreună cu părintele să formeze o muchie din cuplaj. Această situaţie poate apărea atunci când, în urma parcurgerii, a fost selectată muchia parent(u) parent(parent(u)), iar toţi fiii lui parent(u) sunt frunze.*

De aceea, selectarea muchiilor pentru cuplaj nu se va face la prima parcurgere a fiecărei muchii, ci la întoarcerea în fiecare nod, după ce muchiile dintre nivelele inferioare au fost deja parcurse şi s-a stabilit dacă sunt sau nu selectate. Cu alte cuvinte, această selecţie se face „de jos în sus”, tocmai pentru a se evita situaţia menţionată mai sus.

***Algoritmul** care construieşte un cuplaj propriu este:*

***function** ConstruieşteCuplajPropriuDFS(T)*
begin

/*iniţial:

- ***mulţimea de muchii M (care va memora în final cuplajul propriu) este vidă;***
- ***toate nodurile sunt nevizitate;***
- ***nici un nod nu este extremitate a unei muchii din cuplaj, deoarece M ← Φ deci nu este nod saturat.***

***/*

M ← Φ
for all** u **in** V(T) **do
*vizitat (v) ←**false***
*saturat (v) ←**false***
/*se selectează un nod oarecare din arbore, din care se porneşte parcurgerea */
v ←AlegeNodDinArbore(T)
rădăvină ←v

```
        vizitat (v) ← true
        for all u in listaDeAdiacenţă(rădăcină) do
                /*se efectuează parcurgerea DFS şi selectarea muchiilor pentru fiecare subarbore
                care are ca rădăcină un nod de pe nivelul 2 (adică un fiu al rădăcinii lui T)*/
                ConstruieşteRecursiv(u)
                if (not saturat (u) and not saturat(rădăcină))
                then
                        /*dacă este posibil, se adaugă muchia dintre rădăcină şi un fiu al său în
                        cuplaj*/
                        M ← M ∪ {u rădăcină}
                        saturat(rădăcină) ← true
                        saturat(u) ← true
        if ( saturat(rădăcină) = false )
        then
                /*este posibil ca, în urma selectării tuturor muchiilor, rădăcina să rămână nod
                nesaturat; aceasta se întâmplă atunci când nu se poate niciodată adăuga o muchie
                dintre primele 2 nivele în cuplaj, adică atunci când toate nodurile de pe al doilea
                nivel sunt saturate*/
                /*aşa cum am arătat mai sus, un cuplaj este sau nu propriu pentru un arbore în
                funcţie de nodul ales rădăcină*/
                /*dacă, pentru cuplajul construit, vom alege ca rădăcină un fiu al actualei rădăcini,
                vechea rădăcină ca avea un frate care împreună cu părintele să formeze o muchie
                din cuplaj, deoarece nodul acela era saturat, iar toate celelalte noduri îşi păstrează
                proprietăţile*/
                /*în consecinţă, M devine cuplaj propriu pentru arborele cu noua rădăcină*/
                v ← un fiu oarecare al rădăcinii
                rădăcină ← v
        return M
end

function ConstruieşteRecursiv(u)
begin
        vizitat(u) ← true
        for all w in listaDeAdiacenţă(u) do
                /*dacă se întâlneşte un nod deja vizitat, acela nu poate fi decât părintele, întrucât T
                este arbore*/
                if (vizitat(w) = true)
                then părinte ← w
                else
                        ConstruieşteRecursiv(w)
        if (not saturat(părinte) and not saturat(u))
        then
                /*dacă este posibil, se adaugă muchia dintre u şi părintele său în cuplaj*/
                M ← M ∪ { u părinte}
                saturat(părinte) ← true
                saturat(u) ← true
end
```

Corectitudinea algoritmului:

Vom arăta că, într-adevăr, M construit de algoritm este cuplaj propriu, orice nod expus să aibă un frate care, împreună cu părintele lor comun, să formeze o muchie din cuplajul M.

*Fie **u** un **nod expus** (nici o muchie din cuplaj nu este incidentă cu el).*

***Caz 1:** u nu este rădăcina (adică nodul din care s-a pornit parcurgerea DFS). Atunci există parent(u) şi, cu siguranţă, muchia **u parent(u)** nu este din cuplaj. Aceasta înseamnă că, în funcţia **ConstruieşteRecursiv(u)**, condiţia din ultimul **if** era falsă, adică*

*(**not** saturat(părinte) **and not** saturat(u)) = **false**.*

*Ştim deja că (**not** saturat(u)) = **true** $\Rightarrow$ **not** saturat(părinte) = **false***

*$\Rightarrow$ saturat(părinte) = **true** $\Rightarrow$ părinte este deja extremitatea unei muchii din cuplaj.*

Deoarece selecţia muchiilor se face „de jos în sus”, la acest pas nu ar fi putut fi selectată încă o muchie dintre parent(u) şi parent(parent(u)). Aşadar, întrucât parent(u) este deja extremitatea unei muchii din cuplaj, această muchie poate fi doar între parent(u) şi un alt fiu al său w $\neq$ u.

Am arătat deci că u are un frate w a.î. w parent(u) $\in$ M.

***Caz 2:** u = rădăcină (adică este nodul din care s-a pornit parcurgerea DFS). Dacă nu s-a putut introduce în cuplaj nici una din muchiile dintre acest u şi un fiu v al său, atunci, în funcţia ConstruieşteCuplajPropriuDFS(T), codiţia de la prima instrucţiune **if** (cea din interiorul buclei **for**) este întotdeauna falsă, adică*

***not** saturat (fiu) **and not** saturat(rădăcină) = **false**.*

*Ştim deja că (**not** saturat(rădăcină)) = **true** $\Rightarrow$ **not** saturat(fiu) = **false***

*$\Rightarrow$ saturat(fiu) = **true** $\Rightarrow$ toţi fii rădăcinii sunt extremităţi ale unor muchii din cuplaj.*

Aceste muchii pot fi doar între respectivii fii şi nodurile de pe nivelul imediat inferior.

*Prin urmare, trecând un astfel de fiu **a** al rădăcinii pe primul nivel, şi considerând rădăcina ca fiu al acestuia, avem:*

- *saturat(a) = **true**, aşa cum am arătat mai sus $\Rightarrow$ $\exists$ b fiu al lui a a.î. ab $\in$ M $\Rightarrow$ în această configuraţie (acum parent(rădăcină) = a) vechea rădăcină, care este în continuare nod expus, are un frate, pe b, a.î. b parent(rădăcină) $\in$ M.*
- *nici un alt nod din arbore nu îşi schimbă starea (ceilalţi fii ai rădăcinii, dacă existau, erau deja noduri saturate); aşadar, oricare alt nod expus din arbore are în continuare proprietatea cerută, conform celor arătate la cazul 1.*

*Aşadar, cuplajul construit de algoritm este **cuplaj propriu** penrtu arborele T.*

Complexitatea algoritmului:

*Deoarece algoritmul se bazează pe o parcurgere DFS (celelalte operaţii efectuate la fiecare pas se execută în O(1)) a unui arbore dat prin liste de adiacenţă, complexitatea algoritmului este **O(m+n)**. În plus, pentru orice arbore, m = n - 1, deci m = O(n). În consecinţă, complexitatea timp a algoritmului prezentat mai sus este **O(n)**.*

GÎRDEA MARTA
URICARU RALUCA
A34
echipa 21

TEMA NR. 9
6 mai 2003

Problema 1:

Fie G = (S, T; E) un graf bipartit.

*Conform **teoremei lui Hall**, $\exists M \in \mathcal{M}_G$ a.î. $S \subseteq S(M) \Leftrightarrow \forall A \subseteq S, |N_G(A)| \geq |A|$.*

„⇒”

Ştim că, pentru orice întreg k, $0 \leq k \leq |S|$, G are un cuplaj M de cardinal cel puţin $|S|$ - k.

G este bipartit, deci orice muchie din E are o extremitate în S şi o etremitate în T. Această afirmaţie este evident valabilă şi pentru mulchiile din cuplajul M. Deoarece cardinalul lui M este cel puţin $|S|$ - k, se obţine că cel puţin $|S|$ - k noduri din S sunt saturate de M, prin urmare nu pot fi noduri izolate.

În consecinţă, mulţimea S poate conţine cel mult $|S| - (|S| - k) = k$ noduri izolate.

*Fie **X mulţimea nodurilor saturate din S** şi **Y = S – X,** $|Y| \leq k$ (nu este obligatoriu ca toate nodurile din Y să fie izolate).*

Construim graful G' = (X, T, E'). G' este subgraful indus în G de $X \cup T$.

Deoarece extremităţile tuturor muchiilor din M se găsesc în X şi T, $M \subseteq E'$.

Cuplajul M saturează toate nodurile din X în G, deci va satura toate nodurile din X în G', adică $X \subseteq S(M)$ în G'.

Atunci, conform teoremei lui Hall, $\forall A' \subseteq X, |N_G(A')| \geq |A'|$.

Fie o mulţime $A \subseteq S$ oarecare. Evident, $|A \cap Y| \leq k$.

Aşa cum am arătat mai sus, $|N_G(A \cap X)| \geq |A \cap X|$.

Dar $(A \cap Y) \cup (A \cap X) = A$ şi $X \cap|N_G(A')| \geq |A'|Y = \Phi \Rightarrow |A \cap X| = |A| - |A \cap X| \Rightarrow |A \cap X| \geq |A| - k$.

Se obţine deci că $|N_G(A)| \geq |A| - k$ pentru orice k submulţime a lui S, ceea ce trebuia demonstrat.

„⇐”

Arătăm că, în condiţiile date, există un cuplaj de cardinal $|S| – k$.

Construim următoarele şiruri de mulţimi:

$A_i \rightarrow$ o mulţime de k+i noduri din S.

$T_i \rightarrow$ o mulţime de i noduri din T pentru care $\forall A \subseteq T_i, |N_G(A)| \geq |A$

$S_i \rightarrow$ o mulţime de i noduri din S astfel încât $S_i \subseteq N_G(T_i)$ şi $S_i \subseteq A_i$.

Inducţie:

Pas de bază:

Arătăm că există un nod u_1 în T astfel încât $|N_G(u_1)| \geq 1$.

*Fie A_1 o submulţime oarecare de **k+1 noduri**. Din ipoteză ştim că $|N_G(A_1)| \geq |A_1| - k = 1 \Rightarrow \exists$ un nod u_1 în T, astfel încât $\exists v_1$ în A_1 cu $u_1v_1 \in E(G)$.*

Deci $T_1 = \{u_1\}$, $S_1 = \{v_1\}$.

Pas inductiv:

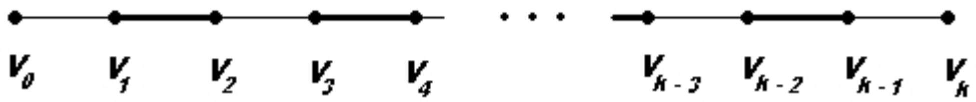
Presupunem că există T_i şi S_i pentru submulţimea A_i de k+i noduri.

Fie $A_{i+1} = A_i \cup \{x\}$, $x \in S - A_i$ oarecare.
Atunci $|N_G(A_{i+1})| \geq |A_{i+1}| - k = i+1 > |T_i|$.
Fie $T_{i+1}' = N_G(A_{i+1}) - T_i$. şi fie $A_{i+1}' = A_{i+1} - S_i$.
$|A_{i+1}| = k+i+1$ şi $|S_i| = i$, $S_i \subseteq A_{i+1} \Rightarrow |A_{i+1}'| = k+1 \Rightarrow$ (din ipoteză) $\exists\, u_{i+1} \in T_{i+1}'$ a.î. $\exists\, v_{i+1} \in A_{i+1}$ a.î. $u_{i+1}v_{i+1} \in E(G)$.
Deci $T_{i+1} = T_i \cup \{u_{i+1}\}$, $S_{i+1} = S_i \cup \{v_{i+1}\}$, iar muchiile u_jv_j, $1 \leq j \leq i+1$, formează chiar un cuplaj.

Formăm mulţimi atâta timp cât există noduri în $S - A_i$, deci putem forma $|S|$-k astfel de mulţimi, deoarece $|A_i| = i + k$, $|A_i| \leq |S|$, deci există în G un cuplaj de cardinal $|S| - k$, ceea ce trebuia demonstrat.

Problema 2:

*a) **Reducere la absurd:***
Presupunem că există un cuplaj M de grad maxim care nu este de cardinal maxim.
*Confirm **teoremei lui Berge**, un cuplaj M este de cardinal maxim în graful G $\Leftrightarrow$ G nu admite drumuri de creştere relativ la M.*
Vom presupune deci că există un cuplaj M de grad maxim care admite un drum de creştere P.



v_0 şi v_k sunt vârfuri expuse.
Construim cuplajul $M' = (M \setminus \{v_1v_2, \ldots, v_{k-2}v_{k-1}\}) \cup \{v_0v_1, \ldots, v_{k-1}v_k\}$ (cuplajul M' conţine aceleaşi muchii ca şi M, cu excepţia celor din $P \cap M$, care sunt înlocuite cu cele din $P \setminus M$).
Constatăm că $S(M') = S(M) \cup \{v_0, v_k\}$. (M' saturează toate vârfurile saturate de M; în plus M' saturează şi vârfurile v_0, v_k, varfuri expuse relativ la M).
În aceste condiţii avem:

$$\sum_{i \in S(M')} d(i) = \sum_{j \in S(M)} d(j) + d(v_0) + d(v_k).$$

$d(v_0) > 0$ şi $d(v_k) > 0$ (deoarece se găsesc pe un drum P)
deci

$$\sum_{i \in S(M')} d(i) > \sum_{j \in S(M)} d(j)$$

*adică **M nu este cuplaj de grad maxim**, ceea ce contrazice ipoteza.*
Presupunerea făcută este deci falsă.
*Am obţinut că **orice cuplaj de grad maxim este de cardinal maxim**.*

b)
***"$\Leftarrow$"**: Ştim că, în graful G, orice cuplaj de grad maxim saturează toate vâfurile de grad maxim. Deoarece în orice graf există întotdeauna cel puţin un cuplaj de grad maxim, se obţine, evident, că, în aceste condiţii, există în G un cuplaj care saturează toate vârfurile de grad maxim.*

***"$\Rightarrow$":** Ştim că există un cuplaj M_0 care saturează toate nodurile de grad maxim (în aceste condiţii, $\Delta(G) \neq 0$, deci G nu este graful nul).*
Reducere la absurd:
*Presupunem că există un cuplaj de grad maxim **M** care nu saturează toate nodurile de grad maxim.*

Fie $u \in V(G)$ a.î. $d(u) = \Delta(G)$ şi $u \in E(M)$ (u nod expus relativ la cuplajul M).

Caz 1: *Există $v \in N_G(u)$ a.î. $v \in E(M)$.*

*Construim cuplajul $M' = M \cup \{uv\}$. Atunci, $|M'| = |M| + 1$, adică $|M'| > |M|$, deci M nu este de cardinal maxim. Conform punctului **a)**, M nu este nici de grad maxim, ceea ce **contrazice ipoteza**.*

Caz 2: *Există $v \in N_G(u)$ a.î. $v \in S(M)$, $vw \in M$ ($w \in N_G(v)$) şi $d(w) < \Delta(G)$.*

Construim cuplajul $M' = (M \setminus \{vw\}) \cup \{uv\}$.

Atunci,

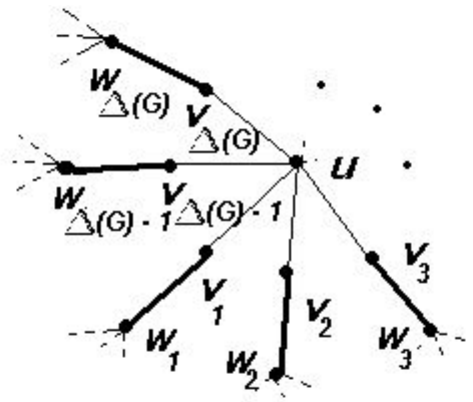
$$\sum_{i \in S(M')} d(i) = \sum_{j \in S(M)} d(j) - d(w) + d(u).$$

Întrucât $d(w) < \Delta(G) = d(u)$, se obţine

$$\sum_{i \in S(M')} d(i) > \sum_{j \in S(M)} d(j)$$

*adică se obţine faptul că M nu este cuplaj de grad maxim, ceea ce **contrazice ipoteza**.*

Caz 3: *Oricare $v \in N_G(u)$, $v \in S(M)$, $vw \in M$ ($w \in N_G(v)$) şi $d(w) = \Delta(G)$ (Toate nodurile vecine cu u sunt saturate, formând împreună cu alt nod $w \neq u$ de grad maxim câte o muchie din cuplaj).*



Ştim, din ipoteză, că există un cuplaj M_0 care saturează toate vârfurile de grad maxim. Există deci $e \in M_0$, $e \notin M$, a.î. u este extremitate a lui e.

Dacă $e = uv \in M_0$ şi $vw \in M$, evident $vw \notin M_0$. Dar M_0 saturează w (deoarece am presupus că w este nod de grad maxim), deci există $v' \in N_G(w)$ a.î. $v' \in S(M_0)$, adică $wv' \in M_0$.

Raţionând astfel în continuare pentru noul vârf de grad maxim w, se poate spune că există un drum P în G a.î.: $\boldsymbol{P \setminus M_0 \subseteq M}$.

Cu alte cuvinte, P este drum alternat în G relativ la cuplajul M, respectiv este drum alternat în G relativ la cuplajul M_0. În plus, $P \cap M \cap M_0 = \Phi$ (cele două cuplaje nu au muchii comune pe acest drum.

*În plus, P are număr **impar** de noduri, respectiv număr **par** de muchii (deoarece are o extremitate expusă relativ la M, şi anume pe u; de asemenea, am presupus că M este cuplaj de grad maxim, deci, conform **a)**, este cuplaj de cardinal maxim, deci nu admite drumuri de creştere).*

Cealaltă extremitate a drumului P, notată u', este deci nod saturat de M, deci este nod expus relativ la M_0. u' nu poate fi deci nod de grad maxim, deoarece, conform ipotezei, M_0 saturează toate nodurile de grad maxim.

Construim un nou cuplaj $\boldsymbol{M' = (M - P) \cup (P \cap M_0)}$, adică, înlocuind acele muchii comune lui P şi M cu acele muchii comune lui P şi M_0 în M'.

Conform celor precizate mai sus, M' saturează toate vârfurile saturate de M, cu excepţia lui u'. În plus, M' saturează u.

Aşadar avem:

$$\sum_{i \in S(M')} d(i) = \sum_{j \in S(M)} d(j) - d(u') + d(u).$$

Dar d(u') < Δ(G), deci d(u') < d(u). În consecinţă,

$$\sum_{i \in S(M')} d(i) > \sum_{j \in S(M)} d(j)$$

adică M nu este cuplaj de grad maxim, ceea ce contrazice ipoteza.

Presupunerea făcută este falsă şi în acest caz.

*Am obţinut deci că, **dacă există un cuplaj care saturează toate nodurile de grad maxim, atunci orice cuplaj de grad maxim saturează toate nodurile de grad maxim**.*

c) Fie H subgraful bipartit indus de mul'imea nodurilor de grad maxim.

O condiţie necesară pentru existenţa cuplajului este Δ(G) > 0 (graful diferit de graful nul). Vom arăta că există un cuplaj care saturează toate nodurile de grad maxim, arătând că orice cuplaj de grad maxim saturează toate nodurile de grad maxim în condiţiile ipotezei.

***Reducere la absurd**:*

Presupunem că există un cuplaj M de grad maxim, care nu saturează toate nodurile de grad maxim. Fie u ∈ V(G), cu d(u) = Δ(G) şi u ∈ E(M) (u este nod de grad maxim, expus relativ la cuplajul M). M este cuplaj de grad maxim ⇒ (conform punctului a) nu există drumuri de creştere relativ la M.

Deoarece u este nod expus şi u nu este izolat, atunci u este fie extremitatea unui drum alternat par, sau aparţine unui circuit alternat impar (nu poate fi extremitatea unui drum alternat impar, deoarece orice drum alternat impar cu o extremitate expusă este drum de creştere; nu poate fi într-un circuit alternat par, deoarece într-un astfel de circuit nu există noduri expuse).

***Caz I:** u ∈ unui circuit alternat impar. Deoarece H este bipartit, nu admite circuite impare. Aşadar, ∃ cel puţin un nod v în G, cu d(v) < Δ(G) care să se găsească pe acest circuit. Deoarece M este cuplaj de cardinal maxim, toate celelalte noduri din circuitul C cu excepţia lui u, sunt noduri saturate. Fie P unul din drumurile de la u la v. Construim cuplajul M' în care înlocuim acele muchii din P∩M cu muchiile din P – M. Atunci avem:*

$$\sum_{i \in S(M')} d(i) = \sum_{j \in S(M)} d(j) - d(v) + d(u).$$

Dar d(v) < d(u) = Δ(G) ⇒ se poate construi un cuplaj M' de grad mai mare decît al lui M, ceea ce contrazice ipoteza.

***Caz II:** u este extremitatea unui drum alternat par. Fie u'cealaltă extremitate a acestui drum. Arătăm că dacă toate extremităţile drumurilor alternate pare sunt noduri de grad maxim, atunci u face parte dintr-un circuit, caz ce se reduce la cazurile anterioare.*

Demonstraţie:

Presupunem că extremităţile tuturor drumurilor alternate sunt noduri de grad maxim.

Reducere la absurd:

Dacă nu există nici un circuit, atunci nu există nici un drum de la u la u ⇒ nu există drumuri de la nici un vecin de-ai lui u la nici un alt vecin dintre vecinii lui u.

Fie H' subgraful indus în H de mulţimea nodurilor accesibile dintr-un vecin v al lui u prin drumuri care nu trec prin u.

Numărul de muchii din acest subgraf este (Δ(H') |H'| -1)/2. Pentru ca acesta să fie număr întreg, este necesar ca |H'| să fie număr impar. H' fiind bipartit (H' = (S', T'; Eh)), una din mulţimile S' şi T' este de cardinal par, iar cealaltă este de cardinal impar.*

Δ(G) >2 (este impar, aşa cum am arătat mai sus; este diferit de 1 deoarece există drumuri alternate pare, deci există noduri de grad cel puţin 2).

Aşadar, numărul de muchii care pleacă din S' este diferit de numărul de muchii care pleacă din T' $\Rightarrow$ ***contradicţie****.*
Deci H' nu poate fi subgraf indus în H, adică există un drum din u într-un nod din G – V(H).

Deci, dacă u nu face parte din nici un circuit, există un drum alternat par cu cealaltă extremitate u' nod ce nu are grad maxim.
În aceste condiţii, construim cuplajul M' în care înlocuim acele muchii din P $\cap$ *M cu muchiile din P – M. Atunci avem:*

$$\sum_{i \in S(M')} d(i) = \sum_{j \in S(M)} d(j) - d(u') + d(u).$$

Dar d(u') < d(u) = Δ(G) $\Rightarrow$ *se poate construi un cuplaj M' de grad mai mare decît al lui M, ceea ce contrazice ipoteza.*
În toate cazurile am obţinut contradicţii, deci presupunerea făcută este falsă, deci orice cuplaj de grad maxim saturează toate nodurile de grad maxim. Graful având un număr finit de noduri şi Δ(G) $\in$ *N, există un cuplaj de grad maxim oricare ar fi graful G.*

d) Vom demonstra că pentru un graf G bipartit E(G) poate fi partiţionată în Δ(G) cuplaje prin ***inducţie după Δ(G)****.*

Pasul de bază:
Δ(G) = 0. Atunci graful G este graful nul, aşadar există 0 cuplaje posibile pentru acest graf.
Δ(G) = 1 $\Rightarrow \forall i \in V(G)$*, d(i) = 0 sau d(i) = 1. Într-un astfel de graf fiecare nod are maxim un vecin, fiind incident cu maxim o muchie. Aşadar, E(G) este mulţime de muchii independente, adică E(G) este un* ***cuplaj****.*

Pasul inductiv:
Presupunem afirmaţia adevărată pentru toate grafurile G cu Δ(G) $\leq$ *k. Demonstrăm pentru grafurile cu Δ(G) = k+1:*
Caz 1: *există un singur nod u* $\in$ *V(G) cu d(u) = Δ(G).*
Atunci există un cuplaj oarecare M_0 *care satisface nodul u (de exemplu* $M_0 = \{uv\}$*, unde v* $\in N_G(u)$*; acest nod v există deoarece am presupus Δ(G) > 0).*
Prin eliminarea acestei muchii din graf se obţine un nou graf bipartit G' (deoarece orice graf parţial al unui graf bipartit este graf bipartit), pentru care Δ(G') = k. Din pasul inductiv rezultă că E(G') poate fi partiţionată în k cuplaje.
Aceste k cuplaje, împreună cu M_0 *menţionat mai sus, formează o mulţime de k+1 cuplaje ce partiţionează E(G).*
Caz 2: *există mai multe noduri* $u_i \in V(G)$ *cu* $d(u_i) = \Delta(G)$*.*
Aceste noduri induc un graf bipartit (deoarece orice subgraf cu cel puţin două noduri al unui graf bipartit este graf bipartit).
Atunci, aşa cum am arătat la ***c)****, există un cuplaj oarecare* M_1 *care satisface toate nodurile de grad maxim* u_i*.*
Prin eliminarea muchiilor din acest cuplaj din graf se obţine un nou graf bipartit G' (deoarece orice graf parţial al unui graf bipartit este graf bipartit), pentru care Δ(G') = k. Din pasul inductiv rezultă că E(G') poate fi partiţionată în k cuplaje.
Aceste k cuplaje, împreună cu M_1 *menţionat mai sus, formează o mulţime de k+1 cuplaje ce partiţionează E(G).*

Problema 3:

Vom arăta că există un algoritm care rezolvă această problemă de decizie în timp polinomial.

*Date **b** (numărul de muchii) şi **k** (numărul minim de noduri pe care trebuie să le aibă subgraful căutat) constatăm că:*

- *dacă, în graful G, |E(G)| < b sau nu există cel puţin k noduri care să nu fie izolate, atunci cu siguranţă nu există un subgraf cu proprietăţile cerute, deci răspunsul la problema de decizie este **negativ**; altfel se poate continua căutarea;*
- *dacă 2b < k, răspunsul este **negativ** (deoarece orice graf de ordin n fără noduri izolate are cel puţin [(n+1)/2] muchii); altfel se poate continua căutarea;*
- *dacă 2b = k, problema se reduce la a găsi un cuplaj de cardinal b în G;*
    - *dacă **ν(G) < b**, atunci răspunsul este **negativ**;*
    - *altfel, răspunsul este **pozitiv** (se poate construi un cuplaj de cardinal maxim din care se elimină, dacă este cazul, muchii, până când se obţine un cuplaj de cardinal b; subgraful astfel obţinut are b muchii şi 2b = k vârfuri, deci îndeplineşte condiţiile; deoarece este o problemă de decizie, nu este necesară construcţia grafului propriuzis);*
- *dacă 2b > k, atunci:*
    - *se construieşte un cuplaj de cardinal maxim M, obţinându-se astfel un subgraf cu **ν(G)** muchii şi **2ν(G)** noduri;*
    - *dacă **ν(G) ≥ b**, atunci: din cuplajul de cardinal maxim construit se elimină muchii (dacă este cazul) până la obţinerea unui nou cuplaj M' cu exact b muchii; un subgraf al lui G, notat H, pentru care E(H) = M şi V(H) = S(M) are exact b muchii şi 2b noduri; dar 2b > k, deci subgraful astfel obşinut are proprietăţile cerute; răspunsul dat în acest caz este **pozitiv**;*
    - *dacă **ν(G) < b,** trebuie adăugate muchii până se ajunge la cele b necesare; trebuie adăugate b - **ν(G)** muchii (şi cel puţin **k** - **2ν(G)** noduri, în cazul în care k > **2ν(G)**); întrucât M era cuplaj de cardinal maxim, orice altă muchie adăugată la mulţimea de muchii M va fi incidentă cu cel puţin o muchie din M; prin urmare, la adăugarea unei muchii se mai poate adăuga cel mult un vârf;*
        - *dacă k ≤ **2ν(G)**, atunci s-a obţinut deja numărul de noduri căutat; se vor adăuga la mulţimea M noi muchii (oricare din graf) până la b, obţimându-se o nouă mulţime de muchii M' cu |M'|=b; un subgraf H al lui G cu E(H) = M' şi V(H) = V(M') (V(M') are cardinalul cel puţin **2ν(G),** deci cel puţin **k**) îndeplineşte condiţiile cerute, deci răspunsul pentru acest caz este **pozitiv**;*
        - *dacă **k > 2ν(G)**, atunci există cel puţin **k** - **2ν(G)** noduri expuse relativ la M care nu sunt izolate (conform condiţiei iniţiale, există cel puţin **k** noduri neizolate în G); se adaugă la subgraf **k** - **2ν(G)** noduri expuse relativ la M, neizolate; la adăugarea fiecărui astfel de nod se **adaugă câte o muchie** (întrucât M este cuplaj de cardinal maxim, orice nod expus este fie izolat, fie vecinul unui nod saturat; aşadar, la adăugarea unui astfel de nod se adaugă şi muchia dintre el şi vecinul său saturat); se obţine astfel un subgraf cu k noduri neizolate şi **ν(G) + (k - 2ν(G)) = k – ν(G)** muchii. Dacă **k – ν(G) > b** (adică, adăugând numărul necesar de noduri, am depăşit numărul de muchii cerut), atunci răspunsul este **negativ**. Altfel, se mai pot adăuga muchii oarecare din E(G) până la b (ştim că există muchiile necesare, din condiţia iniţială), obţinându-se astfel un graf H cu b muchii şi cel puţin k noduri. Răspunsul în acest caz este deci **pozitiv**.*

*Un **algoritm** propriu-zis de rezolvare a problemei de decizie este:*

```
function DecizieP3(G, k, b)
begin
        m ←|E(G)|
        nNeiz ←NumărNoduriNeizolate
        /*Numărarea muchiilor şi a nodurilor neizolate din graful G se poate face printr-o
parcurgere BFS sau DFS a grafului G, deci se execută în timp polinomial (O(m+n) dacă G este
reprezentat prin liste de adiacenţă, O(n^2) în cazul cel mai nefavorabil)*/
        if (m < b or nNeiz < k)
        then
                return FALSE

        if (2b < k)
        then
                return FALSE

        if (2b = k)
        then
                M ←ConstruieşteCuplajDeCardinalMaxim(G)
                /*Construcţia unui cuplaj de cardinal maxim se poate efectua în timp polinomial, şi
anume în O(n^3)*/
                a ←Calculează ν(G)
                if (a < b)
                then
                        return FALSE;
                else
                        /*dacă se cere şi construcţia propriu-zisă a subgrafului :
                                construcţia grafului se poate face în O( ν(G) - b) = O(m) = O(n^2),
                                adică în timp polinomial*/
                        while (a > b) do
                                EliminăMuchieOarecareDinCuplaj(M)
                                a ←a – 1
                        E(H) ←M
                        V(H) ←S(M)
                        return TRUE

        /*dacă 2b > k, atunci:*/

        M ←ConstruieşteCuplajDeCardinalMaxim(G)
        /*Construcţia unui cuplaj de cardinal maxim se poate efectua în timp polinomial*/
        a ←Calculează ν(G)
        if (a >b)
        then /*răspuns pozitiv*/
                /*dacă se cere şi construcţia propriu-zisă a subgrafului :
                                construcţia grafului se poate face în O( ν(G) - b) = O(m) = O(n^2),
                                adică în timp polinomial*/
                while (a > b) do
```

```
            ElimimăMuchieOarecareDinCuplaj(M)
            a ← a – 1
        E(H) ←M
        V(H) ←S(M)
        return TRUE
    else
        /*suntem în cazul ν(G) < b*/
        if (k ≤ 2ν(G)) /*dacă avem deja nodurile necesare*/
        then /*răspuns pozitiv*/
            /*dacă se cere şi construcţia propriu-zisă a subgrafului :
                construcţia grafului se poate face în O( ν(G) - b) = O(m) = O(n²),
                adică în timp polinomial*/
            while (a < b) do
                AdaugăMuchieOarecareDinGînM(M)
                a ← a + 1
            E(H) ← V(M)
            V(H) ← M
            return TRUE
        else
            /*suntem în cazul k > 2ν(G )*/
                if (k  -  ν(G) > b)
                then return FALSE
                else    /*răspuns pozitiv*/
                        /*dacă se cere şi construcţia grafului:*/
                        E(H) ← S(M)
                        V(H) ← M
                        k' ← |V(H)|
                        /*adaugă vârfurile expuse necesare*/
                        /*operaţia se poate face în O(n+m), O(n²) în cazul cel mai
                        nefavorabil*/
                        for all i in V(G) do
                            if (i ∈ E(M))
                            then
                              V(H) ← V(H) ∪{i}
                              E(H) ← E(H) ∪{ ij} /*j este vecinul saturat al lui i*/
                              k ← k + 1
                              if (k = k') then break
                        /*adaugă alte muchii necesare din graf, până când |E(H)|
                        devine b; dacă este necesar, se vor aăuga şi noi vârfuri, fapt
                        ce nu afectează proprietăţile lui H*/
                        /*operaţia se poate executa în O(m), O(n²) în cazul cel mai
                        nefavorabil*/
                        b' ←  |E(H)|
                        while (b' < b) do
                                AdaugăMuchieOarecareDinG (E(H))
                                b' ← b' + 1
                        return TRUE
end
```

***function** ConstruieşteCuplajDeCardinalMaxim (G)*
begin
M ← găseşte_cuplaj_oarecare()
while** (∃ P drum de creştere relativ la M) **do
M←M Δ P
***return** M*
end

Complexitatea** algoritmului de mai sus este $O(n^2) + O(n^3) =$ **$O(n^3)$**. Am arătat deci că **această problemă de decizie se poate rezolva în timp polinomial.

Problema 4:

Fie G = (V, E) un graf 2-muchie conex 3-regulat.

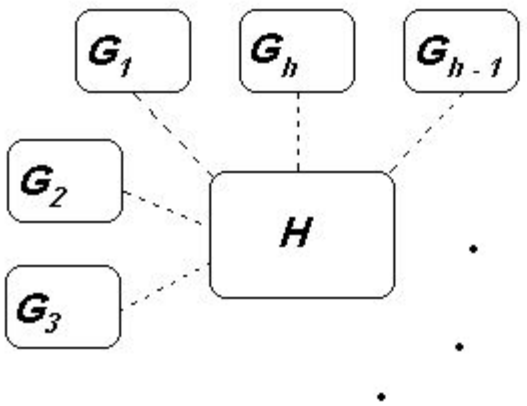
*Conform **teoremei lui Tutte**, un graf G are un cuplaj perfect* $\Leftrightarrow \forall S \subseteq V, S \neq V, q(G - S) \leq |S|$, *unde q(H) este numărul de componente conexe de ordin impar ale grafului H.*

Aşadar, pentru a arăta că graful G are un cuplaj perfect, este suficient să demonstrăm că, $\forall S \subseteq V, S \neq V, q(G - S) \leq |S|$.

Fie S o mulţime oarecare de noduri din V, $S \neq V$, $k = |S|$ *şi* $H = \langle S \rangle_G$.

*Prin eliminarea tuturor nodurilor ce compun mulţimea S din graful G se obţine un număr **h de componente conexe** G_i, $1 \leq i \leq h$, din care **x sunt de ordin impar**.*

Întrucât graful G este 2-muchie conex (deci implicit conex), există muchii între noduri din G_i şi noduri din H, $\forall i, 1 \leq i \leq h$.



1. *Arătăm că există cel puţin 2 muchii între H şi G_i:*

   *Subgraful indus de mulţimea de vârfuri V – S are h componente conexe, deci **oricare două noduri u şi v,** $u \in V(G_i)$, $v \in V(G_j)$, $i \neq j$, $1 \leq i, j \leq h$, $uv \notin E(G)$.*

   *Dacă nu ar exista nici o muchie **în G** între subgraful H şi subgraful G_i, atunci G nu ar fi conex → contradicţie.*

   *Dacă ar exista o singură muchie **în G** între subgraful H şi subgraful G_i, prin eliminarea acesteia graful G ar deveni neconex → contradicţie cu faptul că G este 2-muchie conex.*

2. *Fie G_j o componentă conexă impară a subgrafului indus* $\langle V - S \rangle_G$, *şi* $y = |G_j|$, *y evident impar. Suma gradelor nodurilor din $V(G_j)$ **în G** este **3y**, deoarece G este 3-regulat. Fie z numărul de muchii dintre G_j şi H.*

*Numărul de muchii din G_j **în graful <V – S>$_G$** este $|E(G_j)| = \dfrac{3y - z}{2}$.*

*Deoarece $|E(G_j)| \in \mathbf{N}$, $3y - z$ este număr **par**.*
*Dar $y = |G_j|$ este impar, deci $3y$ este impar. În consecinţă, **z trebuie să fie număr impar**.*

*Aşa cum am arătat mai sus, z trebuie să fie cel puţin 2 şi z impar $\Rightarrow$ **z ≥ 3**. (Adică, **între nodurile din H şi nodurile dintr-o componentă conexă impară G_j a grafului <V – S>$_G$, trebuie să existe în graful G cel puţin 3 muchii**).*

*În consecinţă, cel puţim **3x** muchii din graful G se utilizează pentru legarea nodurilor din S de noduri din componentele impare G_j.*

*3. Suma gradelor vârfurilor din S **în G** este **3|S| = 3k**. Fiecare nod din s poate fi extremitatea unor muchii (cel mult 3) de legătură dintre subgraful H şi subgraful G_j de ordin impar). Aşadar, $3k \geq 3x$, adică **x ≤ k (numărul de componente conexe impare din G – S q(G – S) ≤ k = |S|**).*

*Am obţinut deci că, $\forall S \subseteq V$, $S \neq V$, $q(G - S) \leq |S|$, deci, conform **teoremei lui Tutte**, graful G **are un cuplaj perfect**.*

GÎRDEA MARTA
URICARU RALUCA
A34
echipa 21

TEMA NR. 10
13 mai 2003

1. *Fie G un graf conex cu n vârfuri şi T_G familia arborilor săi parţiali. Se consideră graful $H = (T_G, E(H))$ 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.

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.

Soluţie:

*a) Iniţial vom demonstra că graful H obţinut este **conex**, şi anume că există drum între oricare 2 noduri ale sale.*

*Se observă că în cazul mulţimilor de muchii ale arborilor parţiali, **cardinalul diferenţei lor simetrice este număr par**. Aceasta are loc întrucât mulţimile de muchii ale celor 2 arbori pentru care calculăm diferenţa simetrică au acelaşi cardinal (arborii parţiali ai unui graf au exact n-1 muchii). Deci dacă un arbore diferă de un al doilea printr-o muchie este evident că şi cel de-al doilea are o muchie diferită de cele ale primului.*

*Acest lucru se poate demonstra prin **reducere la absurd**.*

Presupunem prin reducere la absurd că există T_1 şi T_2, 2 arbori parţiali pentru un graf G, astfel încât $|E(T_1) \Delta E(T_2)| = 2k + 1$, $k \geq 0$. Aceasta înseamnă că un arbore diferă de celălalt printr-un număr par de muchii (să zicem prin 2s muchii), iar celălalt arbore diferă de primul printr-un număr impar de muchii (2t+1 muchii), astfel încât 2s+2t+1 = 2k+1.

Putem face presupunerea că T_1 diferă de T_2 prin 2s muchii. Deci restul muchiilor din T_1, şi anume n-1-2s, apar şi în T_2. Dar mai sus am obţinut că T_2 are 2t+1= 2k+1-2s muchii diferite de cele din T_1. Deci T_2 are 2k+1-2s+n-1-2s = n+2k-4s muchii. Ştiam că T_2 are, asemenea tuturor arborilor parţiali, n-1 muchii. Obţinem că 4s-2k = 1, ceea ce este absurd, întrucât 4s-2k este număr par.

Aşadar presupunerea iniţială este falsă şi deci putem spune că $|E(T_1) \Delta E(T_2)| = 2k$, $k>0$ pentru oricare 2 arbori parţiali T_1 şi T_2.

*Vom demonstra că graful H este conex şi are diametrul cel mult n – 1 prin **inducţie după cardinalul K = 2k al diferenţei simetrice a mulţimilor de muchii**.*

Pasul de bază:

La acest pas vom verifica dacă pentru oricare 2 arbori, pentru care diferenţa simetrică dintre mulţimile lor de muchii are cardinalul 2, există drum între ei.

Din ipoteză ştim că între doi arbori parţiali care diferă printr-o singură muchie există muchie în graful H. Aşadar există drum între oricare doi arbori parţiali cu această proprietate.

Pasul inductiv:

Presupunem că există drum între oricare doi arbori care diferă prin k-1 muchii *(adică au cardinalul diferenţei simetrice egal cu 2(k-1)).*

Vom demonstra că şi între oricare doi arbori care diferă prin k muchii *(adică au cardinalul diferenţei simetrice egal cu 2k)* ***există drum în graful H****.*

Pentru aceasta vom găsi un arbore parţial T_3 care diferă prin k-1 muchii de T_1 şi printr-o muchie de T_2. Dacă reuşim acest lucru, obţinem un drum de la T_1 la T_2 care trece prin T_3 (ştim, conform ipotezei inductive, că există drum de la T_1 la T_3, întrucât aceştia diferă prin k-1 muchii; între T_3 şi T_2 avem muchie în H, întrucât cei 2 arbori diferă printr-o singură muchie).

Alegem o muchie dintre muchiile care aparţin diferenţei simetrice, şi anume una dintre muchiile lui T_1 care nu se găsesc şi în T_2*, pe care o notăm cu* ***e_1****. Adăugând această muchie la arborele T_2 obţinem exact un circuit (după cum am explicat şi la* ***problema 2*** *de la* ***tema 8****).* ***Alegem o muchie e_2, dintre muchiile circuitului obţinut, care nu aparţine şi lui T_1****. Va exista întotdeauna o astfel de muchie întrucât, altfel, ar însemna că toate muchiile circuitului aparţin lui T_1 şi deci arborele parţial T_1 admite un circuit, ceea ce este absurd.*

Considerăm arborele T_3 determinat de muchiile lui T_2, mai puţin e_2, la care adăugăm muchia e_1 din T_1. Acesta este tot un arbore parţial întrucât are tot n-1 muchii şi este conex (am eliminat o muchie din circuit dar toate celelalte noduri care formează circuitul rămân legate). Arborele T_3 diferă de T_2 prin muchia e_1 şi de arborele T_1 prin k-1 muchii (cele k muchii prin care diferă T_2 de T_1, mai puţin muchia e_2).

Deci între T_2 şi T_3 avem muchie în graful H, întrucât diferă printr-o muchie, iar de la T_1 la T_3 avem drum conform ipotezei inductive, întrucât diferă prin k-1 muchii. Aşadar avem drum şi de la T_1 la T_2, prin T_3.

Deci între oricare 2 arbori parţiali care diferă prin k muchii putem construi un drum în graful H şi ***conform metodei inducţiei putem spune că între oricare 2 arbori care diferă printr-un număr de k muchii (cu k între 1 şi n-1) există drum în graful H.***

Aşadar graful H este conex.

Ştiind că numărul de muchii dintr-un arbore parţial este n-1, diferenţa simetrică dintre doi arbori parţiali are cel mult 2n-2 muchii. Putem ajunge dintr-un arbore parţial în altul reducând la fiecare pas cu 2 numărul de muchii din diferenţa simetrică dintre cei doi arbori (fiecare pas reprezintă o muchie în H din drumul dintre cei doi arbori).

Demonstraţie:

Pas de bază:

Dacă doi arbori parţiali diferă printr-o muchie ($|T_1 \Delta T_2| = 2$), atunci în H există o muchie între cei doi arbori parţiali, deci se poate ajunge de la T_1 la T_2 printr-o muchie, deci există un drum de lungime 1 între cei doi arbori.

Pas inductiv:

Fie T_1 şi T_2 doi arbori parţiali. După cum am arătat mai sus, putem alege orice muchie din $T_1 – T_2$ şi o muchie (aleasă din ciclul format prin adăugarea muchiei alese din

T_1 la T_2) din $T_2 – T_1$, astfel încât să obţinem un nou arbore parţial T_0, astfel încât diferenţa simetrică dintre T_0 şi T_2 are cardinalul mai mic cu 2.

Procedând inductiv, în $|T_1 \Delta T_2|/2$ paşi (muchii din drumul de la T_1 la T_2) putem ajunge de la T_1 la T_2, şi cum $max(|T_1 \Delta T_2|) = 2(n-1) \Rightarrow$ între orice doi arbori există un drum de lungime mai mică sau egală cu n-1 $\Rightarrow$ ***diametrul(H)$\leq$ n-1***

b) Vom demonstra că subgraful C determinat de arborii parţiali de cost minim este conex, indiferent de funcţia de cost c, prin ***inducţie după cardinalul diferenţei simetrice a mulţimilor de muchii ca la punctul a)****.*

Alegem o funcţie de cost c.

Pasul de bază:

La acest pas vom verifica dacă între oricare 2 arbori parţiali de cost minim, pentru care diferenţa simetrică dintre mulţimile lor de muchii are cardinalul 2, există drum.

Din ipoteză ştim că între doi arbori parţiali care diferă printr-o singură muchie există muchie în graful H. Aşadar există drum şi în C între oricare doi arbori parţiali cu această proprietate.

Pasul inductiv:

Presupunem că există drum, în C, între oricare doi arbori parţiali de cost minim care diferă prin k-1 muchii *(adică au cardinalul diferenţei simetrice egal cu 2(k-1)).*

Vom demonstra că şi între oricare doi arbori care diferă prin k muchii *(adică au cardinalul diferenţei simetrice egal cu 2k)* ***există drum în subgraful C****.*

Pentru aceasta vom găsi un arbore parţial de cost minim (în cazul funcţiei c alese) T_3 care diferă prin k-1 muchii de T_1 şi printr-o muchie de T_2. Dacă reuşim acest lucru, obţinem un drum de la T_1 la T_2 care trece prin T_3 (ştim, conform ipotezei inductive, că există drum de la T_1 la T_3, întrucât aceştia diferă prin k-1 muchii; între T_3 şi T_2 avem muchie în H, întrucât cei 2 arbori diferă printr-o singură muchie).

Alegem muchia de cost minim dintre muchiile care aparţin diferenţei simetrice *(pe care o notăm cu e_1)* ***şi presupunem că e din T_1****. Dacă ar fi fost din T_2 ar fi fost o situaţie analogă doar că am fi creat un arbore parţial de cost minim care ar fi diferit printr-o muchie de T_1 şi prin k-1 de T_2.*

Adăugând această muchie la arborele T_2 obţinem exact un circuit (analog punctului a)). ***Alegem o muchie e_2, dintre muchiile circuitului obţinut, care nu aparţine şi lui T_1****. Va exista întotdeauna o astfel de muchie întrucât, altfel, ar însemna că toate muchiile circuitului aparţin lui T_1 şi deci arborele parţial T_1 admite un circuit, ceea ce este absurd.*

Considerăm arborele T_3 determinat de muchiile lui T_2, mai puţin e_2, la care adăugăm muchia e_1 din T_1. Acesta este tot un arbore parţial întrucât are tot n-1 muchii şi este conex (am eliminat o muchie din circuit dar toate celelalte noduri care formează circuitul rămân legate). Costul lui T_3 este $c(T_2)-c(e_2)+c(e_1)$, care evident este cel mult $c(T_2)$ întrucât alesesem e_1 ca fiind muchia de cost minim din diferenţa simetrică. Aşadar *şi arborele T_3 este de cost minim deoarece $c(T_2)$ este minimă şi deci $c(T_3)$ este minimă. Arborele T_3 diferă de T_2 prin muchia e_1 şi de arborele T_1 prin k-1 muchii (cele k muchii prin care diferă T_2 de T_1, mai puţin muchia e_2).*

***Deci între T_2 şi T_3 avem muchie în graful H şi deci şi în subgraful C, întrucât diferă printr-o muchie, iar de la T_1 la T_3 avem drum în C conform ipotezei inductive, întrucât diferă prin k-1 muchii. Aşadar avem drum şi de la T_1 la T_2, prin T_3, drum care se păstrează în C**.*

*Deci între oricare 2 arbori parţiali de cost minim care diferă prin k muchii putem construi un drum în subgraful C şi **conform metodei inducţiei putem spune că între oricare 2 arbori parţiali de cost minim, pentru o funcţie c, care diferă printr-un număr de k muchii (cu k între 1 şi n-1) există drum în subgraful C.***

Aşadar subgraful C este conex, indiferent de funcţia c.

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 ca 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.

Soluţie:

Iniţial, vom explica intuitiv de ce nu pot avea loc simultan cele două situaţii.

Să presupunem că am avea îndeplinită prima situaţie, şi anume, că am avea un circuit cu respectivele proprietăţi. Dacă am încerca să construim o partiţie de forma exprimată la punctul ii) ar trebui ca nodurile din circuit între care sunt arce verzi să se afle în aceeaşi mulţime (în S sau în T) iar cele între care se află arce galbene să aibă originea în T şi destinaţia în S sau să se afle în aceeaşi mulţime. Dar oricum am aranja vârfurile în cele 2 mulţimi am obţine măcar două vârfuri în mulţimi diferite şi între care este fie arc verde, fie arc galben dar nu în direcţia dorită (de la T la S).

*Presupunem prin **reducere la absurd** că este posibilă o asemenea situaţie, şi deci că există o asemenea partiţie. Aşadar toate nodurile între care există arce verzi, se află două câte două în aceeaşi mulţime iar arcele galbene din circuit sunt fie cu sursa în T şi destinaţia în S (dacă au orientarea corectă), fie au ambele extremităţi într-o mulţime.*

Dacă avem măcar un arc galben uv din circuit în cadrul partiţiei şi anume cu $u \in T$ şi $v \in S$ am avea toate arcele verzi care intră sau ies din u în T, iar cele care intră sau ies din v în S. Însă aceste noduri fac parte dintr-un circuit şi deci mai există un arc care ar trebui să facă legătura între nodurile (circuitului) din cele două mulţimi, altfel ar rămâne un nod cu gradul 1; obţinem un arc verde sau un arc galben de la S la T, ceea ce contrazice presupunerea făcută că există o astfel de partiţie (S,T) a lui V.

Dacă nu avem nici un arc galben din circuit în cadrul partiţiei, înseamnă că toate nodurile din circuit între care există arc sunt puse două câte două în aceeaşi mulţime. Dar atunci ar însemna că unul dintre noduri nu are decât un vecin, pentru că altfel ar trebui ca toate nodurile să fie în aceeaşi mulţime pentru că altfel ar trebui ca toate nodurile să fie în aceeaşi mulţime (dar s şi t sunt cu certitudine în muţimi diferite). Dar acest lucru este imposibil întrucât într-un circuit toate nodurile au gradul 2. Şi din nou se contrazice presupunerea făcută şi deci nu există o astfel de partiţie.

Dacă am avea îndeplinită a doua situaţie, adică am avea o asemenea partiţie, am obţine aceeaşi contradicţie după cum am explicat şi anterior.

Vom încerca să construim o astfel de partiţie şi vom arăta că în cazul în care nu putem construi o asemenea partiţie, obţinem un circuit cu proprietăţile de la punctul i). Iniţial, introducem în mulţimea T nodul t iar în mulţimea S toate celelalte noduri, inclusiv nodul s. Algoritmul gestionează o coadă în care reţine nodurile din T şi pe măsură ce termină de lucrat cu un nod îl extrage din coadă. Se preia fiecare nod v din coadă (primul introdus dintre nodurile rămase în coadă) şi se verifică dacă mai are vreun nod adiacent cu el în mulţimea S. Când se găseşte un nod w în S astfel încât $vw \in E(G(H))$*, unde vw are culoarea verde sau wv, arc de culoare galbenă, se introduce nodul w în mulţimea T şi în coadă. Conţinuăm până când se goleşte coada sau până când introducem nodul s în T. Dacă suntem în situaţia de a introduce s în T ne putem opri şi spune că nu se poate construi o astfel de partiţie dar există un circuit ca la punctul i).*

procedure *Construieşte_i_sau_ii(H)*
begin

   /* iniţializăm mulţimile S şi T, precum şi coada C*/

   $T \leftarrow \{t\}$

   $S \leftarrow V-\{t\}$

   $C \leftarrow \{t\}$

   do

      $v \leftarrow C.first$

   /* luăm primul nod din coadă şi verificăm vecinii adiacenţi cu el din S, care respectă condiţiile pe care le-am explicat mai sus*/

      for *every w in S* ***do***

         if *((vw* $\in$ *E(H) and vw-verde) or (wv* $\in$ *E(H) and (wv-verde or wv-galben)))* ***then***

            if $(w \neq s)$ ***then***

               $T \leftarrow T \cup \{w\}$

               S ← S ¥ {w}

               C.adaugă(w)

      C.extrage(v)

/* ne oprim când coada este vidă şi deci am terminat de examinat toate nodurile din T şi nu am mai găsit nepotriviri sau când am ajuns să-l punem pe s*/

***while** ((C nu e vidă) and (w $\neq$ s))*

***if** (w $\neq$ s) **then** „am obţinut o partiţie (S,T)"*

***else** „nu putem obţine o astfel de partiţie, însă există un circuit care corespunde punctului i)"*

end;

Algoritmul se termină întotdeauna, deoarece se face cel mult o verificare a adiacenţei a oricare două noduri, iar numărul de noduri este finit.

*În continuare vom explica de ce în cazul în care nu reuşim să construim o astfel de partiţie şi se iese din bucla **do while** prin nesatisfacerea condiţiei w $\neq$ s, există un circuit cu proprietăţile de la i).*

*Nodul **s** a fost descoperit de un nod cu care forma un arc care respecta cerinţele circuitului. Dacă se încearcă aducerea acestui nod în T, atunci există un arc verde de la un nod v din T la s sau un arc verde sau galben se la s la un nod v din T. Pornindu-se din t, s-a ajuns la s prin astfel de arce. Dar există arc galben de la t la s, acest arc închizând un circuit care respectă proprietatea i).*

*Dacă s-a ieşit din bucla **do while** prin neatisfacerea condiţiei „C nu este vidă", atunci s-a reuşit construirea unei partiţii (S, T) ce respectă condiţiile din enunţ, adică între S şi T nu există decât arce roşii în orice sens, şi arce galbene orientate de la T la S, iar nodul s nu aparţine lui T, deci s $\in$ S. Toate arcele galbene fiind orientate de la T la S, nu există un arc care să închidă un eventual circuit cu proprietatea i).*

Evident, doar una dintre cele două condiţii poate fi falsă la un moment dat, deci nu se pot construi atât un circuit cât şi o partiţie.

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 m – independente şi |A| < |B|, atunci există b $\in$ B – A a.î. A $\cup$ {b} este m – independentă.*

Soluţie:

Fie A şi B două mulţimi m – independente cu |A| < |B|. În aceste condiţii, B – A $\neq$ Φ, adică ∃ b $\in$ B – A.

În plus, se ştie că toate mulţimile m – independente maximale au acelaşi cardinal. Întrucât există cel puţin o mulţime m – independentă (de exemplu B) de cardinal mai mare decât A, A nu este maximală, deci ∃ v $\in$ V – A a.î. A $\cup$ {v} este m – independentă.

În aceste condiţii, căutarea unui b în B – A a.î. A $\cup$ {b} să fie m – independentă are sens.

***Observaţia 1:** O mulţime X $\subseteq$ V este m – independentă dacă există un cuplaj M_0 al lui G astfel încât X $\subseteq$ S(M_0). Dar, dacă un cuplaj M_0 saturează toate nodurile din* *mulţimea X, atunci M_0 saturează toate nodurile din orice submulţime a lui X. În consecinţă, orice submulţime a unei mulţimi m – independente este m – independentă.*

Aşadar, dacă A şi B sunt două mulţimi m – independente cu proprietăţile de mai sus, şi, în plus, $A \subseteq B$, atunci, $\forall b \in B - A$, $A \cup \{b\} \subseteq B$, deci $A \cup \{b\}$ este m – independentă.

Notaţii:

- $\mathcal{M}_A = \{M \mid A \subseteq S(M)\}$;
- $\mathcal{M}_B = \{M \mid B \subseteq S(M)\}$.

Fie $M_A \in \mathcal{M}_A$ un cuplaj care saturează A, cu proprietatea că

$$|M_A| = max(|M|, M \in \mathcal{M}_A\}$$

respectiv $M_B \in \mathcal{M}_B$ un cuplaj care saturează B, cu proprietatea că

$$|M_B| = max(|M|, M \in \mathcal{M}_B\}.$$

*Vom arăta că **M_A şi M_B sunt cuplaje de cardinal maxim în G**.*

Reducere la absurd:

Presupunem că M_A nu este cuplaj de cardinal maxim. Conform teoremei lui Berge, M_A admite un drum de creştere P. Fie x şi y cele două noduri expuse relativ la M_A (extremităţile acestui drum de creştere). Construim $M' = (M_A - (P \cap M_A)) \cup (P - M_A)$.

$S(M_A) \subseteq S(M')$, deci $M' \in \mathcal{M}_A$. În plus, $|M'| = |M_A| + 1$, ceea ce contrazice faptul că M_A este cuplaj cu cardinalul maxim între cuplajele care saturează A.

Analog se demonstrează că M_B este cuplaj de cardinal maxim.

Se construieşte subgraful $H = (S(M_A) \cup S(M_B), M_A \cup M_B)$. Evident,

$$A \subseteq S(M_A) \cup S(M_B) \text{ şi } B \subseteq S(M_A) \cup S(M_B).$$

Deoarece H este obţinut din reuniunea a două cuplaje,

$$\forall u \in S(M_A) \cup S(M_B)\ \mathbf{d_H(u) \leq 2}.$$

(deoarece un nod v poate fi extremitatea a cel mult 2 muchii în H).

În aceste condiţii, orice componentă conexă C din H este fie drum, fie circuit (orice altă structură este imposibilă din cauza faptului că gradele tuturor nodurilor sunt cel puţin 1, pt că am considerat doar nodurile saturate de cuplaje, şi cel mult 2).

Fie C o componentă conexă oarecare din G.

1. *Fie C **circuit**. Întrucât muchiile acestui circuit sunt muchiile a două cuplaje de cardinal maxim din G, oricare două muchii incidente sunt din cuplaje diferite.*

   *Acest circuit este in G (şi în H) un **circuit alternat** relativ la oricare din cele două cuplaje. Aşadar C nu poate fi circuit impar (dacă ar fi circuit alternat impar relativ la M_A, atunci ar exista în circuit un nod u expus relativ la M_A. Dar, u făcând parte din acest circuit, $d_H(u) = 2$. Atunci u este incident cu două muchii* *din E(H) – M_A, adică din M_B – M_A. De aici reiese că există două muchii incidente în M_B $\rightarrow$ contradicţie cu faptul că M_B este cuplaj. Analog se arată că C nu poate fi circuit alternat impar relativ la M_B).*

În consecinţă, orice circuit alternat din H este circuit alternat par, şi toate nodurile acestui circuit sunt saturate de ambele cuplaje.

Aşadar, dacă $\boldsymbol{\exists\, b \in (B - A) \cap V(C)}$**,** *atunci cuplajul M_A saturează b, deci saturează $A \cup \{b\}$, ceea ce trebuia demonstrat. Altfel, se obţine că*

$$(B - A) \cap V(C) = \Phi$$

adică $\forall b$, dacă $b \in B \cap V(C) \Rightarrow b \in A \cap V(C)$, adică $\boldsymbol{|A \cap V(C)| \geq |B \cap V(C)|}$.

2. *Fie C* ***drum.*** *(Deoarece c este componentă conexă a grafului H, evident C este drum maximal în raport cu incluziunea).*

Observaţie*: Dacă este drum de lungime 1, atunci:*

$\boldsymbol{C = v, e, u}$ *cu $e \in M_A$ sau $e \in M_B$. Vom arăta că $e \in M_A \cap M_B$.*

Reducere la absurd: *Presupunem $e \in M_A$ şi $e \notin M_B$ (celălalt caz se demonstrează analog). $d_H(u) = d_H(v) = 1$ (adică, în G, muchia e nu este incidentă cu nici o altă muchie din M_B.*

În aceste condiţii putem construi $M' = M_B \cup \{e\}$. M' este cuplaj, după cum am arătat mai sus. În plus, $S(M_B) \subseteq S(M')$, deci $B \subseteq S(M')$, şi $|M'| = |M_B| + 1$, ceea ce contrazice faptul că M_B este cuplaj de cardinal maxim.

Aşadar, orice drum de lungime 1 este reprezentat de o muchie din intersecţia celor 2 cuplaje şi extremităţile acesteia, aceste extremităţi fiind deci saturate de ambele cuplaje.

Prin urmare, dacă $\boldsymbol{\exists\, b \in (B - A) \cap V(C)}$, *atunci cuplajul M_A saturează b, deci saturează $A \cup \{b\}$, ceea ce trebuia demonstrat. Altfel, se obţine că*

$$(B - A) \cap V(C) = \Phi$$

adică $\forall b$, dacă $b \in B \cap V(C) \Rightarrow b \in A \cap V(C)$, adică $\boldsymbol{|A \cap V(C)| \geq |B \cap V(C)|}$.

Fie C un drum de lungime ***strict mai mare decât 1****. Întrucât muchiile acestui drum sunt muchiile a două cuplaje de cardinal maxim din G, oricare două muchii incidente sunt din cuplaje diferite.*

Acest drum este in G (şi în H) un ***drum alternat*** *relativ la oricare din cele două cuplaje.*

Presupunem că C este ***drum alternat impar****. Fie u şi v extremităţile acestui drum alternat. Din modul de construcţie a grafului H $\Rightarrow$ C este drum alternat impar relativ la ambele cuplaje în G.*

Presupunem u şi $v \in S(M_A)$. Evident, u şi $v \notin S(M_B)$. Atunci, în graful G, c este drum de creştere relativ la M_B, deci M_B admite drumuri de creştere, ceea ce contrazice faptul că M_B este cuplaj de cardinal maxim. Aşadar presupunerea făcută este ***falsă*** $\Rightarrow$ ***C nu poate fi drum alternat impar****.*

*Atunci C este **drum alternat par**. Fie u şi v extremităţile acestui drum. Presupunem că $u \in S(M_A)$, celălalt caz tratându-se similar. Evident, deoarece componenta conexă C a grafului H este **drum alternat par**, $u \in S(M_A) - S(M_B)$ şi $v \in S(M_B) - S(M_A)$.*

Întrucât $B \subseteq S(M_B)$ şi $A \subseteq S(M_A)$, este evident că $u \notin B$ şi $v \notin A$.

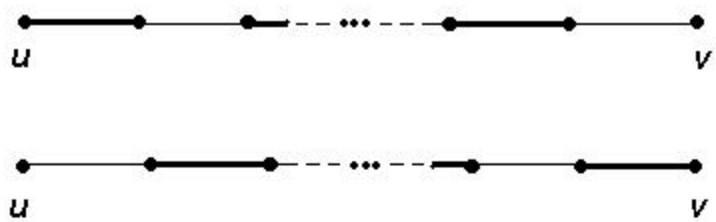
Presupunem că, studiind celelalte tipuri de componente conexe, nu am descoperit nici un nod b din mulţimea $B – A$.

Atunci, toate nodurile din această mulţime se găsesc în aceste componente conexe care sunt drumuri alternate pare.

Dacă există un nod b din $B – A$ care nu este extremitate a acestui drum, atunci, găsindu-se în interiorul drumului, este extremitate a unei muchii din M_A, deci este saturat de M_A. Prin urmare, în acest caz, $A \cup \{v\}$ este m – independentă.

Vom studia în continuare cazul când toate nodurile din $B – A$ sunt extremităţi ale drumurilor alternate pare C.

Dacă există o componentă conexă de acest tip (drum alternat par cu extremitatea v din mulţimea $B - A$) în care extremitatea u este din $S(M_A) – A$, atunci:



Putem construi un nou cuplaj M' astfel:

$M' = (M_A – (M_A \cap E(C))) \cup (E(C) - M_A)$.

Avem: $S(M') = (S(M_A) – \{u\}) \cup \{v\}$. Dar $A \subseteq S(M_A) – \{u\}$ şi $v \in B – A$, deci M' saturează $A \cup \{v\}$, adică $A \cup \{v\}$ este m – independentă, ceea ce trebuia demonstrat.

Vom arăta acum că, dacă toate componentele conexe de celelalte tipuri nu conţin noduri din $B – A$, atunci între drumurile alternate pare cu extremitatea v din $B - A$ există cel puţin unul pentru care u este din $S(M_A) – A$.

Reducere la absurd:

Presupunem toate componentele conexe care sunt drumuri alternate pare cu o extremitate v în $B – A$ au cealaltă extremitate u în A.

Aşa cum am arătat mai sus, $u \notin B$, deci $u \in A – B$.

Atunci, oricare ar fi o astfel de componentă conexă C, vom avea

$|(A – B) \cap V(C)| \geq |(B – A) \cap V(C)|$.

Întrucât am presupus că nici o altă componentă conexă a lui H nu mai conţine noduri din $B – A$, se obţine că

$$|(A – B) \cap V(H)| \geq |(B – A) \cap V(H)|.$$

Dar A şi B sunt incluse în V(H) (reiese din construcţia lui H), deci

*$|(A-B))| \geq |(B-A)|$, adică $|$**A** $|\geq|$**B** $|$**, ceea ce contrazice ipoteza.***
Prin urmare, presupunerea făcută este falsă.

*În toate cazurile am obţinut deci că **există b $\in$ B – A a.î. A $\cup$ {b} este m – independentă.***

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: $b_if_j \in M \Leftrightarrow f_j = M(b_i) \Leftrightarrow b_i = M(f_j)$. Vom presupune că:*

$\forall\, b \in B$ are o ordonare a preferinţelor sale pe F: $f_{i1} <_b f_{i2} <_b ... <_b f_{in}$ şi

$\forall\, f \in F$ are o ordonare a preferinţelor sale pe B: $b_{i1} <_f b_{i2} <_f ... <_f b_{in}$.

*Un cuplaj perfect M al lui $K_{n,n}$ se numeşte **stabil** dacă:*

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

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

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

Soluţie:

Un cuplaj perfect M al lui $K_{n,n}$ se numeşte stabil dacă nu există o pereche (f, b) astfel încât fiecare să se prefere unul pe celălalt în locul partenerilor cu care au fost cuplaţi de algoritm. Astfel de perechi determină un blocaj şi dacă există un astfel de blocaj putem spune că M, cuplajul perfect obţinut, nu este stabil.

Există mai multe posibilităţi de a alege tipurile de date cu care lucrăm. Am avea nevoie de următoarele structuri:

1. *o structură în care să reţinem preferinţele fiecărui băiat şi ale fiecărei fete şi care poate fi de două tipuri:*

- ***vector sau listă de preferinţe** în care, se reţin, pentru fiecare băiat respectiv fată, toate fetele, respectiv băieţii în ordinea descrescătoare a preferinţelor (adică persoana aflată pe prima poziţie este şi cea mai preferată de posesorul listei). Cel mai bine s-ar lucra cu vectori, întrucât s-ar realiza mai uşor accesul la elementele sale. Se observă că, folosind acest tip de structură, avem **timpul de acces la "cea mai preferată" persoană în O(1)**, însă dacă dorim să **comparăm dacă o persoană este preferată altei persoane accesul se face în O(n)**.*
- ***vector sau listă de grade de preferinţă**, în care reţinem pentru fiecare băiat sau fată, toate fetele, respectiv băieţii, în ordine, şi pentru fiecare este specificat un grad de preferinţă. **Dacă o persoană este preferată altei persoane, atunci va avea gradul de preferinţă mai mare decât al respectivei persoane dar nu va fi neapărat înaintea sa în cadrul structurii**. Şi aceasta poate fi implementată cu vector sau listă însă de această dată vectorul este optim în cazul în care **dorim să comparăm gradele de preferinţa ale două persoane; accesul se va face în O(1)** în cazul **vectorului** şi în **O(n)** în cazul **listei** (în comparaţie cu primul tip de structură, unde aveam O(n) în ambele cazuri).*

Pentru a găsi persoana "cea mai preferată" pentru un băiat sau o fată vom avea nevoie de O(n) *în cazul cel mai nefavorabil (în timp ce în prima situaţie accesul se făcea în O(1)).*

*Vom alege varianta cu vectorul de preferinţe, pentru fiecare băiat şi fată. Se reţin de fapt două matrice n*n, una pentru fete şi una pentru băieţi : prF, prB în care avem prF[i] – vectorul de preferinţe pentru fata i şi analog pentru băieţi.*

2. ***o structură în care să reţinem cuplurile formate la fiecare pas*** *şi în care să obţinem în final perechile stabile. De preferat este ca această structură să fie implementată prin 2 vectori, unul pentru fete şi unul pentru băieţi,* ***cF, cB****. În cF(i) avem perechea fetei i (-1 în cazul în care aceasta este singură) şi analog şi pentru băieţi. Accesul la informaţiile despre o anumită persoană se face în O(1).*
3. ***un vector primaProp*** *cu intrări pentru fiecare băiat din listă, şi* ***în care se reţine prima fată din lista de preferinţe a fiecărui băiat (poziţia ei în vectorul de preferinţe a băiatului respectiv)****. Accesul la elemente se face tot în O(1). Iniţial în acest vector se reţin numai valori de 0.*
4. ***o structură în care să reţinem băieţii liberi****. Cel mai bine ar fi ca multimea de băieţi liberi să fie reţinută într-o* ***stivă BăieţiLiberi*** *astfel încât atunci când se adaugă un băiat liber, el să fie împins în stivă iar când dorim să alegem un băiat liber care să facă o propunere, acesta să fie preluat din vârful stivei. Despre o fată putem afla dacă e liberă sau nu din vectorul în care se reţin cuplurile formate la fiecare pas: fata i este liberă dacă cF(i) este –1.*

Algoritmul se bazează pe următoarea idee:

Iniţial toate persoanele sunt libere
cât timp există un băiat liber
    considerăm un băiat liber
    fie f prima fată de pe lista de propuneri a lui b, căreia nu i-a propus deja
    dacă f este liberă
        cuplează f şi b
    altfel
    dacă f îl preferă pe b actualei ei perechi b'
        eliberează-l pe b'
        cuplează pe f cu b
    altfel
    f îl refuză pe b.

În continuare, vom prezenta algoritmul într-o formă mai concretă şi mai detaliată.

function *Prob_Căsătoriilor(n)*
begin
    // iniţializarea structurilor
    for *i ← 0 to n* ***do***
    //Iniţial nici un băiat şi nici o fată nu sunt cuplaţi

//Băieţii nu fac încă nici o propunere
//Iniţializările se fac în O(n)
cB[i] ← -1
cF[i] ← -1
primaProp[i] ← 0
BăieţiLiberi.push(i)
//cât mai avem încă băieţi liberi
while** (nu_e_vidă(BăieţiLiberi)) **do

/*numărul de iteraţii executate de această buclă nu poate depăşi n^2 deoarece:
Atunci când un băiat liber este cuplat cu o fată, cel mult un băiat este "eliberat", deci reintrodus în BăieţiLiberi. Prin urmare, la fiecare iteraţie, dimensiunea stivei BăieţiLiberi nu poate depăşi dimensiunea de la iteraţia anterioară.
Fiecare băiat poate propune o singură dată unei fete, şi poate fi refuzat de fiecare fată cel mult o dată.*/

/*celelalte operaţii din cadrul buclei (mai puţin funcţia de preferinţă se execută în O(1) întrucât sunt simple accesări*/
//se alege primul băiat din stivă dar nu se extrage încă din stivă
b ← BăieţiLiberi.top()
/*se alege prima propunere a lui b dintre fetele cărora nu le-a făcut deja propuneri*/
f ← prB[b][primaProp[b]]
(primaProp [b])++
//se trece la următoarea propunere din lista preferinţelor lui b
//verificăm dacă fata căreia îi face b propunerea este liberă
if** (cF[f] ≠ -1) **then
/*dacă fata este liberă se formează cuplul (f, b) şi se extrage b din lista băieţilor liberi*/
cF[f] ← b
cB[b] ← f
BăieţiLiberi.pop()
else
/* Verificăm dacă f preferă mai mult băiatul b decât îl preferă pe actualul ei logodnic*/
//Verificarea preferinţei se face în O(n)
if** (preferă(f, b, cF(f)) **then
/* funcţia returnează true dacă într-adevăr f îl preferă mai mult pe b decât pe cF(f) */
//îl scoatem pe b din rândul băieţilor liberi şi îl introducem pe cF(f)
BăieţiLiberi.pop()
BăieţiLiberi.push(cF(f));

cB[cF[f]] ← -1
//îi cuplăm pe f şi pe b
cF[f] = b
cB[b] = f
end;

//funcţia care determină dacă o fată preferă mai mult un băiat decât pe altul şi care are are evident complexitatea O(n) întrucât în cel mai rău caz se parcurge întreaga listă de preferinţe a fetei f */

function preferă (f, b1, b2)
begin
i ←0
//înaintăm în vectorul de preferinţe până când găsim unul dintre cei doi băieţi
//primul băiat întâlnit este preferat de fata f în locul celuilalt
while** (prF[f][i] ≠ b1 || prF[f][i] ≠ b2) **do
i++
if** (prF[f][i] = b1) **then
***return** true*
***else return** false*
end;

*Conform celor explicate, **complexitatea** algoritmului este*
$$O(n) + O(n^2)(O(n) + O(1)) = \mathbf{O(n^3)}.$$

Algoritmul alege primul băiat liber din stivă (şi anume acel băiat aflat în vârful stivei). Acesta rămâne în stivă până când este cuplat cu o fată. În momentul în care un cuplu se rupe un băiat rămâne liber şi unul liber este cuplat; se va scoate din stivă băiatul care se cuplează la acel pas şi va fi introdus cel care este despărţit de pereche. Evident un băiat va rămâne în stivă până când i se găseşte o pereche stabilă, deci până când nu mai poate fi scos de nici un băiat dintre cei rămaşi.

Fiecare băiat este acceptat de o fată aşadar algoritmul se termină cu un cuplaj. Acest cuplaj este întotdeauna stabil şi este favorabil din punctul de vedere al băieţilor. Presupunem că acest cuplaj obţinut este M. Trebuie să arătăm că într-adevăr nu se formează nici un blocaj.

Să presupunem că există o fată f pe care un băiat b o preferă mai mult decât pe perechea cu care a fost grupat în urma execuţiei algoritmului. Atunci, după modul în care lucrează algoritmul, putem trage concluzia că această fată f a refuzat în momentul în care băiatul b i-a făcut propunerea, sau l-a acceptat la un moent dat, dar a fost înlocuit ulterior cu un altul în condiţiile precizate în algoritm, deci putem spune că f îl preferă pe partenerul ei curent. Aşadar f şi b nu formează un blocaj şi cum am ales băiatul b arbitrar, putem spune că nu există blocaje şi deci că M este un cuplaj stabil.

Observaţie:

În cazul în care numărul de fete nu este egal cu numărul de băieţi, putem adăuga nume fictive. Dacă avem mai puţine fete se adaugă un număr de fete fictive iar altfel, se adaugă un număr fictiv de băieţi. Pentru fetele sau băieţii adăugaţi vom alcătui listele de preferinţe aleator numai că:

- *dacă adăugăm fete, atunci fetele adăugate trebuie puse ultimele în listele de preferinţe ale băieţilor;*
- *dacă adăugăm băieţi, atunci fetele trebuie să îi introducă ultimii în listele lor de preferinţe.*

Facem această alegere întrucât avem nevoie ca cei adăugaţi să nu fie luaţi în consideraţie decât de persoanele cele mai puţin dorite din cealaltă tabără. În final se vor exclude persoanele introduse în plus, urmând în tabăra cealaltă să rămână şi băieţi sau fete fără pereche.

GÎRDEA MARTA
URICARU RALUCA
A34
echipa 21

TEMA NR. 11
20 mai 2003

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 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.*

Soluţie:

Fie G = (V, E) un graf oarecare.

Se ştie că M este cuplaj perfect pentru un graf G dacă S(M) = V (un cuplaj este perfect dacă saturează toate nodurile).

De asemenea, orice muchie din cuplaj saturează exact două noduri şi orice nod poate fi saturat de cel mult o muchie.

Aşadar, dacă M este cuplaj perfect, atunci $|M| = \frac{|G|}{2}$.

O condiţie necesară pentru ca G să aibă un cuplaj perfect este ca acesta să aibă ordin par. O altă condiţie necesară este ca G să nu aibă noduri izolate. Evident, aceste condiţii nu sunt condiţii suficiente.

Vom căuta deci un cuplaj perfect într-un graf convenabil ales astfel încât să putem obţine cuplajul maxim în G.

Varianta I:

Ţinând cont de condiţiile precizate mai sus, se va încerca construirea cuplajului de cardinal maxim pentru graful G astfel:

***Pas 1**: Construim un nou graf G' = (V', E'). Iniţial, G' = G.*

***Pas 2**: Se va asocia o funcţie de pondere* $\boldsymbol{p: E' \to R_+}$ *muchiilor grafului G':*

$$\forall e \in E(G'),\ p(e) = 1.$$

***Pas 2**: Dacă |G| este număr impar, nu are sens aplicarea algoritmului de căutare a cuplajului perfect de pondere minimă (conform celor precizate mai sus). Se adaugă un nou nod w la V(G'), diferit de toate nodurile din V(G'), astfel fiind satisfăcută condiţia ca G' să aibă ordin impar.*

Noul nod w va fi legat prin muchii de toate nodurile din V(G'). Se extinde funcţia de pondere la noul graf astfel:

$$p(wi) = \boldsymbol{N},\ \forall e \in V(G') - \{w\},$$

(unde N este un număr mare, de exemplu n(n-1)/2).

***Justificarea** alegerii acestei funcţii de pondere:*

Intrucât se doreşte obţinerea unui cuplaj de cardinal maxim pentru G prin calculul unui cuplaj perfect de pondere minimă în G', vom încerca să includem în acest cuplaj perfect cât mai multe muchii din G. Aşadar, muchiile din G vor avea o pondere *mică, pentru a se facilita introducerea lor în cuplajul de pondere minimă, iar muchiile adăugate ulterior vor avea pondere mult mai mare, astfel încât să se evite pe cât posibil introducerea lor în cuplajul construit. Este suficient dacă ponderea unei astfel de muchii depăşeşte suma ponderilor tuturor muchiilor din G).*

Dacă |G| este număr par, nu se trece la pasul următor.

***Pas 3:** Se aplică algoritmul de căutare a unui cuplaj perfect de pondere minimă în G'. Dacă un astfel de cuplaj Mp este găsit, atunci se aleg din Mp acele muchii care apar în G, obţimându-se asfel un cuplaj de cardinal maxim pentru G, după cum vom demonstra mai jos.*

În caz de eşec, se procedează astfel:

- *se adaugă două noi noduri u şi v la V(G') (vom adăuga două noduri pentru ca ordinul grafului să rămână par; evident, u şi v $\notin$ V(G') şi sunt distincte la fiecare pas);*
- *aceste noduri vor fi legate de toate celelalte noduri din G' (şi între ele) prin muchii cărora li se asociază ponderea N, ca mai sus.*
- *se repetă pasul 3 până când în G' se obţine un cuplaj perfect de pondere minimă Mp; din acesta se extrage cuplajul M de cardinal maxim pentru G, aşa cum am arătat mai sus.*

***Algoritmul** propriu-zis de construcţie a cuplajului M este:*

```
function ConstruieşteCuplajDeCardinalMaxim(G)
begin
    /*construcţia lui G'*/
    G' ← G
    n ← |G|
    N ← n(n-1)/2
    if (|V(G)| este impar)
    then
        V(G') ← V(G') ∪ {w}
        E(G') ← E(G') ∪ {wi| i ∈ V(G'), i ≠ w}
    /*asocierea funcţiei de pondere*/
    for all e in E(G') do
        if (e ∈ E(G)) then p(e) ← 1
            else p(e) ← N
    /*aplicarea algoritmului de calcul al cuplajului perfect de pondere minima*/
    while (not ExistăCuplajPerfect (G', p)) do
        V(G') ← V(G') ∪ {u, v}
        E(G') ← E(G') ∪ {ui, vi| i ∈ V(G'), i ≠ u, v}
    /*la ieşirea din while se obţine cuplajul perfect de pondere minimă pentru G'*/
    M' ← CuplajPerfectDePondereMinima(G', p)ş
    /*alegerea din M' a muchiilor care apar în G*/
```

$M \leftarrow M' \cap E(G)$
***return** M*

end

Corectitudinea algoritmului:

***1.** Vom arăta că **algoritmul se opreşte**, adică, modificând astfel graful G', se obţine după un număr de paşi un graf care are un cuplaj perfect.*

Conform Teoremei lui Tutte, un graf $G = (V, E)$ are un cuplaj perfect dacă şi numai dacă, $\forall S \subseteq V, S \neq V, q(G - S) \leq |S|$.

Fie B_k mulţimea tuturor bipartiţiilor posibile $(S, V' - S)$ ale mulţimii V'_k a grafului G'_k construit şi verificat de algoritm la pasul k din bucla while, pentru care $S \neq V'_k$.

$B_k = \{(S_0, V'_k - S_0), \ldots, (S_{nk}, V'_k - S_{nk})\}$.

Pp că G_k nu are cuplaj perfect. Atunci există cel puţin o mulţime $S_j \subseteq V'_k$, $S_j \neq V'_k$, pentru care $q(G'_k - S_j) > |S_j|$.

Fie $a = q(G'_k - S_j)$ şi $b = |S_j|$.

Urmând paşii algoritmului de construcţie a cuplajului de cardinal maxim în G, se vor adăuga în G'_k două noduri care vor fi legate prin muchii atât între ele, cât şi cu toate celelalte noduri din V'_k, obţinându-se astfel un nou graf G'_{k+1}.

La introducerea a două noi noduri în graful G_k, acestea vor trebui împărţite între cele două mulţimi ale bipartiţiilor. Aşadar, B_{k+1} va avea forma:

$B_{k+1} = \{(S_i, (V_k - S_i) \cup \{u, v\}), (S_i \cup \{u\}), (V_k - S_i) \cup \{v\}), (S_i \cup \{v\}), (V_k - S_i) \cup \{u\}), (S_i \cup \{u, v\}), V_k - S_i) | \ (S_i, V_k - S_i) \in B_k\}$.

Vom studia fiecare variantă de astfel de cupluri din B_{k+1}:

- *<u>dacă bipartiţia este de forma: $(S_i, (V_k - S_i) \cup \{u, v\})$:</u>*

Deoarece nodurile u şi v sunt adiacente cu toate nodurile din G_k, subgraful indus de $(V_k - S_i) \cup \{u, v\}$ este conex.

Dacă $|S_i|$ este par, atunci $|(V_k - S_i) \cup \{u, v\}|$ este par, deci nu există nici o componentă conexă impară în $G_{k+1} - S_i$, adică $q(G_{k+1} - S_i) = 0 \leq |S_i|$.

Dacă $|S_i|$ este impar, atunci $|(V_k - S_i) \cup \{u, v\}|$ este impar, deci există exact o componentă conexă impară în $G_{k+1} - S_i$, adică $q(G_{k+1} - S_i) = 1 \leq |S_i|$.

Prin urmare, la fiecare pas, acest tip de mulţimi satisfac condiţiile teoremei lui Tutte.

- *<u>dacă bipartiţia este de forma: $(S_i \cup \{u\}), (V_k - S_i) \cup \{v\})$ sau $(S_i \cup \{v\}), (V_k - S_i) \cup \{u\})$:</u>*

Deoarece nodurile u şi v sunt adiacente cu toate nodurile din G_k, subgraful indus de $(V_k - S_i) \cup \{u\}$ sau de $(V_k - S_i) \cup \{v\}$ este conex.

Dacă | $S_i \cup \{u\}$| este par, atunci $|(V_k - S_i) \cup \{v\}|$ este par, deci nu există nici o componentă conexă impară în $G_{k+1} - (S_i \cup \{u\})$, adică $q(G_{k+1} - S_i \cup \{u\}) = 0 \leq | S_i \cup \{u\}|$.

Dacă | $S_i \cup \{u\}$| este impar, atunci $|(V_k - S_i) \cup \{v\}|$ este impar, deci există exact o componentă conexă impară în $G_{k+1} - (S_i \cup \{u\})$, adică $q(G_{k+1} - (S_i \cup \{u\})) = 1 \leq | S_i \cup \{u\}|$.

Analog pentru $S_i \cup \{v\}$.

Prin urmare, la fiecare pas, acest tip de mulţimi satisfac condiţiile teoremei lui Tutte.

- dacă bipartiţia este de forma: $(S_i \cup \{u, v\}), V_k - S_i)$:

Atunci, $q(V_k - S)$ este acelaşi ca şi la pasul anterior, (deoarece nu se adaugă şi nu se elimină noduri sau muchii în $V - S$). În schimb cardinalul celeilalte mulţimi a bipartiţiei creşte cu 2.

Aşadar, pentru o astfel de mulţime de noduri S_l, dacă vom adăuga succesiv la graful G' câte două noduri, după un anumit număr de paşi se va mai elimina un caz de mulţime S care nu satisface condiţiile teoremei lui Tutte.

De precizat că trebuie adăugate atâtea noduri cât este diferenţa iniţială dintre $q(V' - S)$ şi $|S|$.

După cum am arătat, indiferent câte noduri se adaugă în S, nu se modifică numărul de componente conexe impare din $V - S$ (deoarece nu se adaugă şi nu se elimină noduri sau muchii în $V - S$). Aşadar, valoarea lui $q(V_k - S)$, pentru orice G_k construit de algoritm şi orice S submulţime proprie a lui V_k, este mărginită de maximul dintre valorile $q(V' - S)$, unde V' este mulţimea vârfurilor grafului G' de la primul pas. În cazul cel mai nefavorabil, fiecare nod din V' este o componentă conexă impară, deci $q(V_k - S)$ nu poate depăşi $|V'|$ (notat n).

Aşadar, după adăugrarea a cel mult n noi noduri la graful G' se obţine un graf care are un cuplaj perfect.

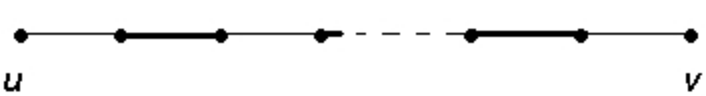
Întrucât, în bucla **while**, la fiecare iteraţie se adaugă două noduri, condiţia din **while** devine falsă după cel mult n/2 iteraţii.

Evident, dacă există cuplaj perfect, atunci există cuplaj perfect de pondere minimă, deci s-a găsit cuplajul căutat şi se poate trece la construcţia cuplajului de cardinal maxim pentru G.

2. Vom arăta că M obţinut din din intersecţia cuplajului perfect de pondere minimă al lui G' cu mulţimea muchiilor lui G **este cuplaj de cardinal maxim** pentru G.

Reducere la absurd:

Presupunem că algoritmul de mai sus nu construieşte un cuplaj de cardinalmaxim pentru G. Conform teoremei lui Berge, aceasta înseamnă că G admite un drum de creştere P relativ la M.



Nodurile u şi v sunt noduri expuse relativ la cuplajul M.

Dar G este subgraf al lui G', iar M este submulţime a cuplajului M', care este cuplaj perfect în G'. Aşadar, muchiile din P care apar în m sunt şi în M'.

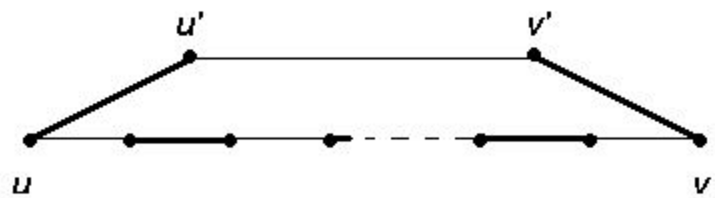
În plus, deoarece M' este cuplaj perfect în G', nu există în G' noduri expuse relativ la M'. Aşadar, există u', v' în V' a.î.:

$u' \in N_{G'}(u)$, $v' \in N_{G'}(v)$ *şi* $uu' \in M'$, *respetiv* $vv' \in M'$.

Dacă măcar unul din nodurile u' şi v' (de exemplu u') ar fi din V (din G), atunci, din modul de construcţie a cuplajului M, ar rezulta că uu' este muchie în cuplajul M (deoarece, la construcţia lui G' nu s-au adăugat muchii între nodurile care erau iniţial în G, deci dacă există muchia uu', atunci aceasta face parte din mulţimea E a grafului G; aşadar, $uu' \in E$ *şi* $uu' \in M' \Rightarrow uu' \in E \cap M'$, *adică* $uu' \in M$*). Dar acest lucru contrazice presupunerea făcută, şi anume că u este nod expus relativ la M.*

În consecinţă, ***u' şi v' ∈ V' – V*** *(sunt noduri adăugate ulterior, la construcţia lui G').*

Dar, din modul de construcţie al lui G' (la fiecare pas există muchie între orice nod din V' – V şi toate celelalte noduri din V') reiese că există în G' muchia ***u'v'****.*



Evident, această muchie nu este din cuplajul M', deoarece extremităţile sale sunt saturate de alte muchii din M'.

Am obţinut deci un circuit alternat par C în G' relativ la M'.

Fie 2k + 1 numărul muchiilor de pe drumul P în G. Evident, dintre acestea, k se găsesc în cuplajul M (şi în M').

Ponderea muchiilor din M' de pe acest circuit C este:

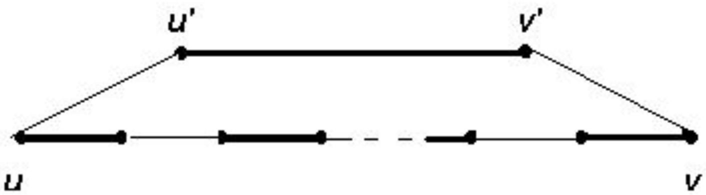
$$\boldsymbol{p(C \cap M') = k * 1 + 2 * N.}$$

(deoarece ponderile muchiilor de pe P sunt 1, acestea făcând parte din G, iar ponderile muchiilor adăugate ulterior, adică uu' şi vv', sunt N).

Construim un nou cuplaj M'' în G', astfel:

$$M'' = (M' - (C \cap M')) \cup (C - M').$$

(adică eliminăm muchiile de pe C, înlocuindu-le cu acele muchii de pe C care nu erau iniţial în M').



Evident, M'' este cuplaj perfect, întrucât S(M'') = S(M'), iar S(M') = V'.

Ponderea cuplajului M'' este:

$$p(M'') = p(M') - p(M' \cap C) + p(C - M') \Rightarrow$$

$$p(M'') = p(M') - (k*1 + 2*N) + ((k + 1)*1 + N) \text{ şi } N > 1 \Rightarrow$$

$\Rightarrow$ p(M'') < p(M'), ceea ce contrazice faptul că M' era cuplaj de ponedere minimă.

*În consecinţă, presupunerea făcută este falsă. G nu admite drumuri de creştere relativ la M, deci **M este cuplaj de cardinal maxim în G**.*

Varianta a II-a:

Construim G' = (V', E') astfel:

- *dacă |V| este par, atunci V' = V; altfel V' = V $\cup$ {w} (se adaugă un nou nod pentru îndeplinirea condiţiei necesare de existenţă a cuplajului perfect, referitoare la paritatea ordinului grafului);*
- *E' este mulţimea tuturor muchiilor posibile între nodurile lui G' (G' este graf complet).*

Evident, un graf complet de ordin par are un cuplaj perfect.

*Vom asocia o funcţie de pondere **p: E' $\rightarrow$ R$_+$** astfel:*

- *p(e) = 1 dacă e $\in$ E(G);*
- *p(e) = |E'| dacă e $\in$ E(G') – E(G).*

***Justificarea** alegerii acestei funcţii de pondere:*

Intrucât se doreşte obţinerea unui cuplaj de cardinal maxim pentru G prin calculul unui cuplaj perfect de pondere minimă în G', vom încerca să includem în acest cuplaj perfect cât mai multe muchii din G. Aşadar, muchiile din G vor avea o pondere mică, pentru a se facilita introducerea lor în cuplajul de pondere minimă, iar muchiile adăugate ulterior vor avea pondere mult mai mare (este suficient dacă ponderea unei astfel de muchii depăşeşte suma ponderilor tuturor muchiilor din G).

După construcţia cuplajului perfect de pondere minimă M' în G', se aleg din acest cuplaj acele muchii care sunt în G. Mulţimea acestor muchii va fi un cuplaj de cardinal maxim în G.

***Algoritmul** de construcţie a unui cuplaj de cardinal maxim, utilizând un algoritm de calcul al cuplajului perfect este:*

***function** ConstruieşteCuplajDeCardinalMaxim(G)*

begin

*/*construcţia G'*/*

$V(G') \leftarrow V(G)$

if *(|V(G)| este impar)*

then $V(G') \leftarrow V(G') \cup \{w\}$

$E(G') \leftarrow \{ij \mid i, j \in V(G'), i \neq j\}$

*/*asocierea funcţiei de pondere*/*

for all *e* **in** *E(G')* **do**

if $(e \in E(G))$ **then** $p(e) \leftarrow 1$

else $p(e) \leftarrow |E(G')|$

*/*aplicarea algoritmului de calcul al cuplajului perfect de pondere minima*/*

$M' \leftarrow$ *CuplajPerfectDePondereMinima(G', p)*

*/*alegerea din M' a muchiilor care apar în G*/*

$M \leftarrow M' \cap E(G)$

return *M*

end

Vom demonstra că mulţimea M construită de algoritm este chiar cuplaj de cardinal maxim în G.

Reducere la absurd:

Presupunem că M nu este cuplaj de cardinal maxim.

Fie $m = |M|$.

Ştim că $M \subseteq M'$ *(din construcţia cuplajului M).*

M' este cuplaj perfect în G' $\Rightarrow |M'| = \dfrac{|G'|}{2}$.

Ponderea acestui cuplaj este:

$$\sum_{e \in E(G)} p(e) + \sum_{e \in E(G') - E(G)} p(e) = m * 1 + (|M'| - m) * |E'|.$$

Deoarece am presupus că M nu este cuplaj de cardinal maxim, $\exists M_1 \in \mathcal{M}_G$ *a.î.* $|M_1| > |M|$. *Fie* $|M_1| = |M| + a$, $\boldsymbol{a > 0}$.

Atunci, întrucât G' este graf complet, $\exists M_1'$ *cuplaj perfect în G' a.î.* $M_1 \subseteq M_1'$.

(Demonstraţie:

M_1 *este cuplaj în graful G şi G este subgraf al lui G'* $\Rightarrow M_1$ *este cuplaj în G'.*

$|S(M_1)|$ *şi* $|G'|$ *sunt numere pare, deci* $|G' - S(M_1)|$ *este număr par (adică mulţimea nodurilor expuse în G' relativ la* M_1, $E(M_1)$ *are cardinal par). În plus,* $E(M_1)$ *induce un subgraf complet în G'. Acesta, având ordin par, are un cuplaj perfect* M_2.

Dar:

$$S(M_1) \cup S(M_2) = V(G') \text{ şi } S(M_1) \cap S(M_2) = \Phi$$

$\Rightarrow M_1' = M_1 \cup M_2$ *este* ***cuplaj pefect*** *în G'.*

Am arătat deci că, pornind de la M_1, *se poate construi un cuplaj perfect în G', ceea ce trebuia demonstrat).*

Ponderea cuplajului M_1' este:

$$(m + a) * 1 + (|M'| - m - a) * |E'|.$$

adică $p(M_1') = p(M') - a(|E'| - 1)$.

Dar $a > 0$ şi $|E'| > 1$, deci

$$\mathbf{p(M_1') < p(M')}$$

ceea ce contrazice alegerea cuplajului M' ca fiind cuplaj perfect de pondere minimă în G'.

Presupunerea făcută este deci falsă.

Prin urmare, algoritmul de mai sus construieşte un cuplaj de cardinal maxim într-un graf G oarecare.

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).*

Soluţie:

Fie M o matrice oarecare de 0 şi 1, $M \in \{0, 1\}^{m \times n}$. Această matrice are deci m linii, numerotate de la 0 la m – 1, şi n coloane, numerotate de la 0 la n – 1.

Vom asocia aceste matrice un graf bipartit G = (X, Y; E) astfel:

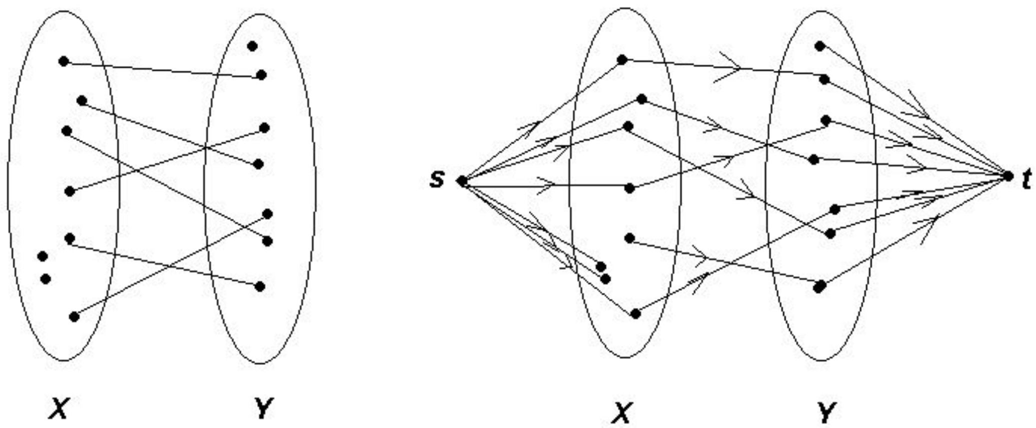
- *$X = \{0, 1, \ldots, m - 1\}$, adică fiecărei linii din M îi corespunde un nod în X;*
- *$Y = \{m+ 0, m+ 1, \ldots, m+ n - 1\}$, adică fiecărei coloane din M îi corespunde un nod în Y;*
- *dacă elementul de la intersecţia liniei i cu coloana j este 0, atunci se introduce în E muchia $(i, j + m)$.*

Graful G este evident bipartit, putând exista intersecţii doar între o linie şi o coloană, deci nu există muchii între două noduri din aceeaşi mulţime a bipartiţiei.

Observăm deci că fiecărui element 0 de la intersecţia liniei i cu coloana j în matricea M îi corespundeo muchie e în G cu extremităţile i şi j+m.

În aceste condiţii, problema găsirii unei mulţimi de cardinal maxim de elemente egale cu 0 şi care să nu se găsească pe aceeaşi linie sau coloană în M se reduce la găsirea unei mulţimi de muchii de cardinal maxim în G în care oricare două muchii au extremităţile distincte, adică la ***găsirea unui cuplaj de cardinal maxim*** *în graful bipartit G.*

Problema găsirii unui cuplaj de cardinal maxim într-un graf bipartit se poate rezolva cu ajutorul unui algoritm de flux maxim.



Se asociază problemei o reţea de transport $R_G = (G', c, s, t)$ construită astfel:

- *$G' = (V', E')$ digraf unde :*
  - *$V' = X \cup Y \cup \{s, t\}$,*
  - *$E' = \{ij \mid i \in X, j \in Y, ij \in E\} \cup \{si \mid i \in X\} \cup \{jt \mid t \in X\}$;*
- *funcţia de cost $c:E' \rightarrow R_+$ definită prin $c(a) = 1$, $\forall a \in E'$.*

Aplicând un algoritmul Ford Fulkerson pentru determinarea unui flux maxim pe reţeaua astfel definită, se poate determina întotdeauna un flux de valoare maximă cu componente întregi, deoarece capacităţile arcelor din reţea sunt întregi.

Aşadar, fluxul x_{ij}, cu i din X şi j din Y poate avea valoarea 0 sau 1.

Evident, dintre toate muchiile cu o extremitate i comună, i din X, doar una va avea fluxul 1, iar celelalte 0. Analog, dintre toate muchiile cu o extremitate j comună, j din Y, doar una va avea fluxul 1, iar celelalte 0.

În consecinţă, mulţimea arcelor dintre X şi Y pe care fluxul este 1 este mulţime stabilă de muchii, deci este cuplaj pentru G. Fluxul descoperit fiind maxim, acest cuplaj este de cardinal maxim.

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.*

Soluţie:

Vom porni de la reţeaua de procesoare şi vom construi o reţea căreia să-i putem asocia un flux care să rezolve problema.

Se consideră un digraf G ca fiind graful ce reprezintă conexiunile dintre procesoare în reţea.

Se calculează media încărcărilor procesoarelor adunând încărcările tuturor procesoarelor şi împărţind suma la numărul acestora.

Vom adăuga două noduri fictive, o sursă s şi o destinaţie t. De sursa s vom lega toate procesoarele supraîncărcate, iar de destinaţia t vom lega toate procesoarele subîncărcate. Notăm cu S_i procesoarele supraîncărcate, cu E_i cele cu încărcarea potrivită şi cu U_i cele aflate sub medie.

Putem vedea cele două noduri fictive adăugate ca două procesoare, procesorul s fiind procesorul din care să plece surplusul din procesoarele S_i iar procesorul t ar fi procesorul în care ajunge toată această încărcătură. În procesoarele supraîncărcate păastrăm numai încărcătura pe care trebuie să o aibă în final (media).

Muchiilor din digraful asociat acestei reţele de calculatoare le asociem capacităţi pentru a obţine o reţea căreia să-i calculăm fluxul maxim.

Reţeaua va avea următoarele proprietăţi:

-Arcele care pleacă din s şi ajung în S_i vor avea capacităţile egale cu load(S_i)-medie (în reţeaua iniţială de calculatoare această încărcătură nu pleacă din s, ci din procesoarele S_i, însă a fost necesară această convenţie pentru a realiza reţeaua căreia să-i ataşăm fluxul de care avem nevoie pentru a echilibra încărcătura procesoarelor);

-Pentru arcele care pleacă din S_i sau ajung în S_i dintr-o sursă diferită de s, cât şi pentru arcele care intră în U_i sau ies din U_i şi se duc într-o destinaţie diferită de t, precum şi pentru toate celelalte arce între celelalte procesoare vom asocia capacităţile egale cu ∞, pentru a putea permite procesoarelor, în momentul în care se construieşte fluxul maxim, să trimită şi să primească cât este nevoie pentru a putea obţine echilibrarea în final;

-Un flux asociat acestei reţele repreznta de fapt modul în care circulă încărcătura de la procesoarele supraîncărcate la cele subîncărcate.

-Arcele care pleacă din U_i şi ajung în t vor avea asociate capacităţile medie-load(U_i), adică exact de cât are nevoie fiecare procesor pentru a atinge media. (noi ştim că dacă toată încărcătura din s a ajuns în t înseamnă că procesoarelor subîncărcate li s-a transmis exact cantitatea de înărcare de care aveau nevoie)

Pentru această reţea se va utiliza un algoritm de căutare a fluxului maxim, de exemplu algoritmul lui Edmonds şi Karp. Notăm fluxul maxim găsit cu x*.

Fie x fluxul asociat acestei reţele pentru care v(x) este:

$\sum_{i\in S} load(i) - medie = \sum_{i\in U} medie - load(i)$ *(cantitatea care pleacă din s trebuie să ajungă în întregime în t, adică cantitatea de încărcare care prisoseşte procesoarelor supraîncărcate trebuie să ajungă în cele subîncărcate)*

Vom arăta că acest flux este cel de care avem nevoie pentru a obţine o echilibrare a încărcărilor procesoarelor şi că dacă fluxul maxim x obţinut pentru reţeaua aleasă este chiar acest flux x atunci am obţinut o echilibrare a încărcăturilor procesoarelor.*

Să presupunem că în urma aplicării algoritmului obţinem chiar acest flux x.

Procesoarele care erau supraîncărcate primesc numai încărcătura care le prisoseşte. Evident, nodurile care reprezintă procesoarele supraîncărcate vor avea în final încărcătura medie, întrucât, după cum am ales reţeaua, încărcătura în exces intră în S_i şi iese din S_i după regula conservării fluxului. La fel şi încărcătura care intră în *aceste noduri, venită din alte noduri decât s. Nodurile din E_i, care erau deja echilibrate vor rămâne astfel întrucât tot ceea ce a intrat în ele prin acest flux a şi ieşit. Nodurile din U_i care au primit cantitatea de încărcare necesară pentru a atinge media au transmis-o lui t, în întregime. Ceea ce au primit în plus au transmis prin alte arce pentru a respecta regula conservării (nu puteau transmite mai mult lui t întrucât erau limitate de capacităţile arcelor către t). Aşadar fiecare procesor subîncărcat îi va trimite lui t exact cât mai are nevoie până va fi echilibrat. Deci tot ceea ce a fost surplus în celelalte procesoare a ajuns în final în t şi deci putem spune că a ajuns în procesoarele care merau iniţial subîncărcate. Aşadar putem scoate muchii fictive şi obţinem că urmînd drumurile determinate de acest flux obţinem o modalitate de echilibrare. Deci fluxul maxim x* ne dă chiar modalitatea de transmitere a cantităţii de încărcare între procesoare.*

Formăm o partiţie (S, T) a lui $V \cup \{s, t\}$, $s \in S$ şi $t \in T$. De exemplu putem alege partiţia ({s}, V-{s}). Aceasta determină o secţiune care este de altfel şi secţiune minimă în reţeaua aleasă intrucât dacă am mai dăuga un nod v în S am putea pierde eventual un arc de la s la v, însă am mai obţine cel puţin un arc deoarece nodul v trebuia să transmită unui alt nod ce a primit de la s, conform legii de conservare a fluxului. Din teorema secţiunii minime şi fluxului maxim ştim că c(S,T)=v(x), unde (S, T) este secţiune de capacitate minimă iar x* este fluxul maxim.*

Vom presupune acum că fluxul maxim x este strict mai mare decât fluxul x. Dar valoarea lui x este exact capacitatea secţiunii (S,T).*

c(S, T)= $\sum_{i \in S_i} load(i) - medie$ *(capacitatea muchiilor între S şi T, adică între s şi nodurile din S_i) = v(x). Conform teoremei amintite mai sus obţinem că x este flux maxim, ceea ce contrazice presupunerea făcută.*

Dacă presupunem că x este mai mare decât x, atunci cel puţin pe una din muchiile care pleacă din s fluxul este subcapacitar. Întrucât*

$$\sum_{j \in N_g^+(i)} x_{ij} = \sum_{j \in N_g-(i)} xji$$

vom obţine în acest caz un procesor i supraîncărcat care nu va transmite tot surplusul de încărcătură, deci nu va ajunge la medie. Fluxul necesar pentru echilibrare este deci mai mare decât fluxul maxim posibil în această reţea. În consecinţă, o astfel de reţea de procesoare nu va putea fi echilibrată.

Deci putem realiya echilibrarea procesoarelor numai în cazul în care obţinem fluxul maxim egal cu fluxul căutat de noi, x.

4. *Să se determine fluxul de valoare maximă în reţeaua din figură (explicând funcţionarea algoritmului lui Edmonds-Karp).*

Soluţie:

Algoritmul lui Edmonds-Karp are la bază algoritmul lui Ford & Fulkerson, căruia i s-au adus îmbunătăţiri. Se porneşte de la un flux iniţial şi se efectuează creşteri succesive până când nu mai obţinem drumuri de creştere. În acest caz putem spune că am ajuns la un flux maxim. Pentru depistarea drumurilor de creştere utilizăm etichete:

nodul i este etichetat dacă există un drum de creştere de la s la acel nod.

În cazul algoritmului lui Edmonds Karp se gestionează o coadă, pe care o folosim pentru parcurgerea BFS a vârfurilor etichetate. Acest algoritm se bazează pe utilizarea drumurilor minime (cu număr minim de arce) de creştere a fluxului curent.

La fiecare pas se selectează primul vârf din coadă (care este primul vârf etichetat şi necercetat) şi se încearcă etichetarea vecinilor săi. În momentul în care ajungem în t înseamnă că am obţinut un drum de creştere de la s la t. Se determină capacitatea reziduală a acestui drum obţinut la pasul curent şi se adaugă la vechiul flux, obţinându-se un nou flux, pentru care repetăm algoritmul.

Vom aplica algoritmul în cazul grafului nostru.

Pornim prin a alege un flux iniţial x, şi anume fluxul cu $x_{ij}=0$, $\forall$ ij arc din digraful nostru. Evident acesta este un flux întrucât pentru fiecare nod valoarea fluxului de intrare este egală cu valoarea fluxului de ieşire, iar în cazul nostru aceste valori sunt 0.

Etichetarea nodurilor se face astfel:

- pe prima poziţie vom pune nodul din care a fost obţinut nodul pentru care facem etichetarea;

- pe a doua poziţie vom scrie tipul arcului prin care am descoperit nodul (direct sau invers);

- a treia poziţie va conţine minimul dintre valoarea de pe poziţia a treia a etichetei nodului prin care am descoperit nodul pe care îl etichetăm la pasul curent şi capacitatea reziduală a respectivului arc.

*Vom eticheta sursa **s** cu **(NULL, NULL, ∞)**, întrucât considerăm sursa s ca nod iniţial şi deci ea nu este obţinută din nici un alt nod pentru a obţine în final capacitatea reziduală minimă.*

Introducem sursa s în coadă şi pornim etichetarea nodurilor adiacente cu s.

Prin parcurgere bfs găsim nodul a şi apoi nodul b.

*Pentru **a** avem etichetarea **(s, direct, 4)** iar pentru **b** obţinem **(s, direct, 3)**. Capacităţile reziduale sunt egale în acest caz cu capacităţile muchiilor, întrucât avem x_{ij}= 0 pentru fiecare arc din digraf.*

*Vom scoate din coadă nodul s şi vom cerceta următorul nod etichetat, în cazul nostru a. Din a putem ajunge în s, în b şi în c, însă b şi s sunt deja etichetate aşa că îl vom introduce în coadă şi eticheta numai pe c. Eticheta pentru **c** este **(a, direct, 4)**.*

*Îl scoatem pe a din coadă şi b devine primul nod din coadă, etichetat şi necercetat. Din b găsim neetichetat nodul **d**, pe care îl etichetăm prin **(b, direct, 3)**. Îl scoatem pe b şi îl cercetăm pe c, din care îl obţinem pe **t**, pe care îl etichetăm cu **(c, direct, 3)**.*

Am ajuns în t, aşadar am obţinut un ***drum de creştere de la s la t*** *care este:*

P=s→a→c→t.

Acest drum are capacitatea reziduală egală cu $\mathbf{min(r_{ij})_{ij\text{-arc de pe drumul }P}=3}$.

Această capacitate reziduală o adunăm la fiecare x_{ij} *de pe drumul de creştere pentru fluxul x şi obţinem un nou flux* x^1*, pentru care repetăm algoritmul.*

$\mathbf{x^1_{sa}= 3, x^1_{ac}= 3, x^1_{ct}= 3}$***, restul arcelor având valorile anterioare*** *(de la fluxul x).*

Se etichetează din nou sursa ***s cu (NULL, NULL, ∞).***

Din s descoperim nodurile a şi b. Nodul ***a*** *va fi etichetat cu* ***(s, direct, 1)*** *iar b cu* ***(s, direct, 3)****. Din a îl descoperim pe* ***c*** *pe care îl etichetăm cu* ***(a, direct, 1)*** *şi pe b care este etichetat cu* ***(a, direct, 1)****. Din c se descoperă b, d şi t. însă b era deja etichetat şi t nu poate fi etichetat întrucât* $c_{ct}=3=x_{ct}$*. Aşadar îl etichetăm numai pe* ***d*** *cu* ***(c, direct, 1)*** *şi îl introducem în coadă. Deci b devine următorul nod etichetat şi necercetat, însă din b nu mai putem eticheta nici un nod. Din d îl descoperim pe* ***t*** *care poate fi etichetat cu* ***(d, direct, 1)****.*

Întrucât l-am descoperit pe t, putem spune că am găsit un nou ***drum de creştere de la s la t,***

s→a→c→d→t,

care are capacitatea reziduală ***1****.*

Trebuie să aplicăm din nou algoritmul, pentru fluxul x^2 *obţinut prin adăugarea capacităţii reziduale la fluxul* x^1*, pentru arcele din drumul de creştere obţinut anterior. De data aceasta se obţine un flux cu valorile:*

$\mathbf{x^2_{sa}= 4, x^2_{ac}= 4, x^2_{cd}= 1, x^2_{dt}= 1, x^2_{ct}= 3}$ ***iar restul rămân 0****.*

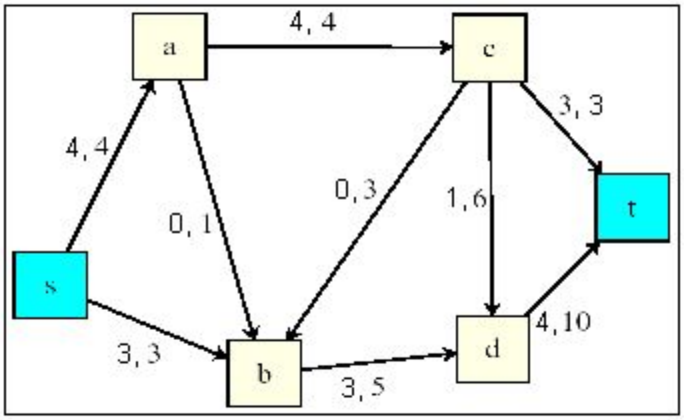
Etichetăm din nou ***s*** *cu* ***(NULL, NULL, ∞).*** *Din s îi găsim pe a şi pe b. Nodul* ***a*** *nu poate fi etichetat întrucât* $\mathbf{c(sa) = x^2_{sa} = 4}$ *iar pe* ***b*** *îl etichetăm cu* ***(s, direct, 3)****. Din b îl obţinem pe a, c şi d (s fiind deja etichetat). Pe a nu îl putem eticheta deoarece de la b la a avem arc invers cu valoarea asociată 0* ***,*** *pe* ***c*** *îl etichetăm cu* ***(b, invers, 3)****, şi pe* ***d*** *cu* $\mathbf{(b, direct, min(c(bd)-x^2_{bd} = 5, e_3(b)=3)) = (b, direct, 3)}$*. Din c îi obţinem pe a şi t. Pe* ***a*** *îl etichetăm cu* ***(c, invers, 3)****, pe iar pe t nu îl putem eticheta deoarece avem capacitatea muchiei egală cu* $x^2_{ct}=3$*. Nodul d devine primul nod din coadă şi îl descoperă pe* ***t*** *care are eticheta* $\mathbf{(d, direct, min(c(dt)-x^2_{dt} = 6, e_3(d)=3))= (d, direct, 3).}$ *Am găsit un nou drum de creştere de la s la t:*

s→b→ d→t

cu capacitatea reziduală 3. Se obţine un nou flux x^3 *cu valorile* $\mathbf{x^3_{sa}= 4, x^3_{sb}= 3, x^3_{ac}= 4, x^3_{bd}= 3, x^3_{cd}= 1, x^3_{dt}= 4, x^3_{ct}= 3}$ ***iar restul sunt 0****.*

Etichetăm ***s*** *cu* ***(NULL, NULL, ∞).*** *Din s îi găsim pe a şi pe b. Nodul* ***a*** *nu poate fi etichetat întrucât* $\mathbf{c(sa) = x^3_{sa} = 4}$ *iar pe* ***b*** *de asemenea nu poate fi etichetat întrucât* $\mathbf{c(sb)=x^3_{sb}=3}$***. Aşadar nu mai putem obţine un drum de creştere de la s la t şi deci fluxul obţinut anterior este maxim.***

$\mathbf{x^3_{ij}}$ ***sunt reprezentate în figura de mai jos:***


a
c
s
t
b
d
4, 4
4, 4
0, 1
0,3
1,6
3, 3
3, 3
3, 5
4,10

GÎRDEA MARTA
URICARU RALUCA
A34
echipa 21

ALGORITMICA GRAFURILOR

TEMA NR. 12
27 mai 2003

1. *Fie v valoarea fluxului maxim în reţeaua R = (G, c, s , t). Demonstraţi că există k-st drumuri în G,* $P_1, P_2, \ldots, P_k$ *(* $0 \le k \le |E(G)|$ *), şi numerele reale nenegative* $v_1, v_2, \ldots, v_k$ *astfel încât* $x:E(G) \to R$, *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.*

Soluţie:

Vom demonstra această proprietate prin inducţie după numărul de arce pentru care fluxul are valoare pozitivă, pentru un flux f de valoare maximă.

Pas de bază

În cazul în care fluxul nostru maxim are un singur arc, de valoare v_1*, pozitivă, este evident că putem alege* P_1 *ca fiind chiar acest arc iar valoarea* v_1*, valoarea asociată arcului* x_{st}*.*

Pas inductiv

*Presupunem că la **pasul curent i** este adevărată proprietatea: pentru un flux* f_i *obţinut la pasul i, dacă adăugăm toate valorile* $v_1, \ldots, v_i$*, asociate şirului* $P_1, \ldots P_i$ *obţinute până la acest pas obţinem un flux de valoare maximă.*

*Vom demonstra că şi pentru **pasul i+1** dacă alegem un st-drum direct* P_{i+1} *şi o valoare convenabilă obţinem un nou flux* f_{i+1} *care are aceeaşi proprietate cu fluxul* f_i*.*

*Alegem arcul **uv** cu valoarea pozitivă cea mai mică pentru fluxul nostru* f_i *şi construim un drum direct* P_{i+1} *de la s la t care să conţină şi arcul **uv** astfel încât toate arcele de pe acest drum să aibă valori pozitive.*

Considerăm un flux f_{i+1} *care să conţină arcele fluxului* f_i *cu proprietatea că arcele care nu apar în drumul* P_{i+1} *vor avea valoarea pe care o au în fluxul* f_i *iar arcele care apar în drumul* P_{i+1} *vor primi valoarea egală cu diferenţa dintre valoarea arcului respectiv pentru fluxul f şi valoarea lui **uv** în fluxul f. Adăugăm acest drum* P_{i+1} *la şirul de drumuri determinate până la pasul curent iar valorea asociată lui va fi* $v_{i+1} = f_{uv}$*. Noul flux obţinut are valoarea egală cu valoarea lui* f_i*, din care scădem v, de câte ori apare un arc din P în fluxul f.*

Când nu mai avem nici un arc cu valaore pozitivă în fluxul nostru ne putem opri şi putem spune că valoarea fluxului maxim este chiar suma din cerinţa problemei.

Am putea încerca şi o altă abordare a acestei probleme, şi anume:

- *putem găsi toate drumurile de creştere directe şi, pornind de la un flux a cărui valaoare o iniţializăm cu 0, putem construi fluxul conform algoritmului lui Edmonds . Vom obţine un flux cu valoarea mai mică sau egală cu fluxul de valoare maximă;*

- *dacă nu mai avem drumuri de creştere, înseamnă că fluxul obţinut este chiar cel maxim, drumurile P_i sunt drumurile directe descoperite iar valorile asociate sunt valorile care se adaugă la fiecare pas în algoritmul lui Edmonds.*
- *dacă mai există drumuri de creştere, care au arce inverse, transformăm fiecare drum de creştere obţinut, în două drumuri de creştere directe. Vom descoperi arcele inverse dintr-un drum de creştere de la dreapta la stânga , şi pentru fiecare arc invers obţinut împărţim drumul nostru de creştere în 2 drumuri: unul direct (din care am scos deja toate arcele inverse, aflat la dreapta arcului invers obţinut) şi un alt drum din care eliminăm arcul invers de la pasul curent, ocolindu-l. Noul drum de creştere este de fapt o concatenare a celor 2 drumuri: unul direct şi unul care mai poate avea încă arce inverse. De aceste arce inverse scăpăm la un pas ulterior.*
- *când nu mai avem drumuri de creştere putem spune că am găsit fluxul maxim. In final vom obţine numai drumuri de creştere directe în şirul P_i iar valorile lor asociate sunt valorile cu care au contribuit aceste drumuri la construcţia fluxului maxim.*

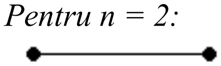
2. *Numim GP – descompunere a grafului complet K_n orice mulţime $A = \{B_1, \ldots, B_{k(A)}\}$, unde: fiecare B_i este subgraf bipartit complet al lui K_n, orice două grafuri B_i şi B_j au mulţimile de muchii disjuncte şi $\cup_{i=1,\, k(A)} E(B_i) = E(K_n)$. Arătaţi că orice GP - descompunere a lui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

Soluţie:

*1. Vom arăta prin **inducţie** că pentru $\forall\, n \geq 2$, există o GP - descompunere A_n a lui K_n a.î. $k(A_n) = n - 1$.*

Pasul de bază:

Pentru n = 2:



Graful K_2 este graf bipartit complet ($K_{1,1}$), deci putem considera $B = K_{1,1}$ şi $A = \{B\}$. Evident, $k(A_2) = 1 = 2 - 1$.

Pasul inductiv:

Presupunem afirmaţia adevărată pentru $k \leq n$. Demonstrăm pentru $k = n + 1$.

Fie K_{n+1}.

Vom construi mulţimea A astfel:

- *se construieşte un graf biartit $B_1 = (S, T; E_1)$ unde: $S = \{u\}$, u un nod oarecare din $V(K_n)$, $T = V(K_n) - \{u\}$. B_1 trebuie să fie graf bipartit complet, deci $B_1 = K_{1,n}$.*
- *se construieşte un nou graf G' din care se elimină toate muchiile ce apar în B_1. Deoarece $B_1 = K_{1,n}$, nodul u rămâne nod izolat în G', deci nu va putea face parte dintr-un subgraf bipartit complet. Aşadar vom elimina şi nodul u din G'. Se constată că G' devine subgraf indus al lui K_{n+1} şi $|G'| = n$, deci **G' = K_n**. Dar, **din ipoteza inductivă, pentru K_n există o GP - descompunere***

A_n astfel încât $k(A_n) = n - 1$. *De asemenea, din construcţie reiese că $E(B_1) \cap E(G') = \Phi$, deci, $\forall B \in A_n$, $E(B_1) \cap E(B) = \Phi$. În plus, tot din construcţie, avem $E(B_1) \cup E(G') = E(K_{n+1})$. Aşadar, mulţimea*

$$A_{n+1} = \{B_1\} \cup A_n$$

este GP – descompunere pentru K_{n+1}, iar

$$k(A_{n+1}) = 1 + k(A_n) = 1 + n - 1 = (n+1) - 1.$$

Aşadar, pentru orice $n \geq 2$, se poate construi o GP - descompunere A_n a lui K_n a.î. $k(A_n) = n - 1$.

2. Vom arăta că nu se poate construi pentru K_n o GP – descompunere A_n pentru care $k(A_n) < n - 1$.

Observaţie: *Orice GP – descompunere A_n a grafului K_n conţine cel puţin un graf bipartit $B = (S, T; E)$ pentru care $|S| = 1$ sau $|T| = 1$.*

Demonstraţie:

Reducere la absurd:

Presupunem că există o GP-descompunere a grafului K_n care nu are proprietatea de mai sus.

Observăm că, orice nod poate apărea în cel mult $(n - 1) / 2$ grafuri bipartite (deoarece cel puţin două muchii incidente cu el apar într-un astfel de graf, iar mulţimile de muchii ale grafurilor sunt disjuncte).

Considerând un graf bipartit B, dacă vom încerca să găsim grafuri bipartite în care se află muchiile ce nu apar în B şi care au proprietatea $|S| > 1$ şi $|T| > 1$, vom constata că fie muchiile acestor grafuri se suprapun, fieunele dintre aceste grafuri nu sunt complete.

Aşadar nu este posibilă o astfel de descompunere.

Luând în considerare observaţia de mai sus, vom demonstra proprietarea cerută prin ***inducţie:***

Pasul de bază:

Pentru $n = 2$ propoziţia de mai sus este evidentă (este necesar cel puţin un subgraf bipartit pentru acoperirea mulţimii de muchii, al cărei cardinal este nenul).

Pasul inductiv:

Presupunem afirmaţia adevărată pentru $k \leq n$. Demonstrăm pentru $k = n + 1$.

Fie K_{n+1}.

Reducere la absurd:

Presupunem că există o GP – descompunere A_{n+1} a acestuia pentru care $k(A_{n+1}) < (n + 1) - 1$.

Întrucât am arătat mai sus că orice GP – descompunere A_{n+1} a acestuia conţine cel puţin un graf bipartit $B = (S, T; E)$ pentru care $|S| = 1$ sau $|T| = 1$, există un nod u în $V(K_n)$ astfel încât există B în A_{n+1} cu $S = \{u\}$ sau $v = \{u\}$.

Vom elimina din K_n acest nod u şi toate muchiile care îl leagă de celelalte noduri din graf. Se va obţine astfel graful K_n. Putem obţine o GP – descompunere pentru K_n din GP descompunerea grafului de mai sus, considerând intersecţia dintre $E(K_n)$ şi fiecare *graf bipartit din A_{n+1}. Evident, intersecţia dintre B şi $E(K_n)$ este vidă, deci nici un subgraf al acestuia nu va apărea în acest A_n. Aşadar, $k(A_n) \leq k(A_{n+1}) - 1$, adică $k(A_n) \leq n - 2$, ceea ce contrazice ipoteza inductivă.*

Aşadar presupunerea făcută este falsă. Pentru orice n, orice GP – descompunere A a lui K_n are $k(A) \geq n - 1$.

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

Soluţie:

1. Demonstrăm că $\omega(G) \leq |f(V)|$.

Reducere la absurd:

Presupunem că, pentru graful $G = (V, E)$, există o funcţie f a.î. $\omega(G) > |f(V)|$.

Fie $A \subseteq V$ clică cu $|A| = \omega(G)$.

$\forall i, j \in A$, $ij \in E \Rightarrow f(i)f(j) \in E$.

$f(A) \subseteq f(V)$ şi $\omega(G) > |f(V)| \Rightarrow \omega(G) > |f(A)| \Rightarrow |A| > |f(A)| \Rightarrow$

$\exists u, v \in A$ a.î. $f(u) = f(v) = w \in V$.

Dar $u, v \in A$ şi A clică $\Rightarrow$

$uv \in E \Rightarrow f(u)f(v) \in E$ (din definitia funcţiei f) $\Rightarrow ww \in E \rightarrow$ ***contradicţie.***

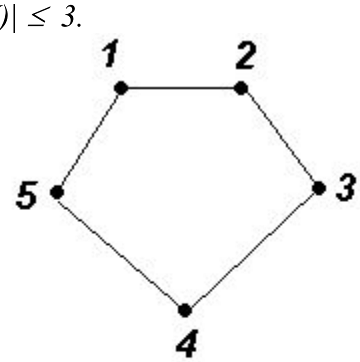
Presupunerea făcută este falsă $\Rightarrow$ ***$\omega(G) \leq |f(V)|$.***

2. Arătăm că există grafuri pentru care orice funcţie f cu proprietatea de mai sus are $|f(V)| > \Delta(G) + 1$.

Vom considera graful C_5.

$\Delta(C_5) = 2$.

Vom arăta că este imposibilă construcţia unei funcţii f cu proprietatea de mai sus şi cu $|f(V)| \leq 3$.



Construim funcţia f:

Fie $f(1) = v1$.

Fie $f(2) = v2$. Evident, $v2 \neq v1$, deoarece 12 este muchie, deci şi v1v2 trebuie să fie muchie.

Fie f(3) = v3.

v3 ≠ v2, deoarece 23 este muchie, deci şi v2v3 trebuie să fie muchie.

Presupunem că putem alege v3 = v1.

Fie f(4) = v4.

v4 ≠ v1, deoarece 34 este muchie, deci şi v3v4 trebuie să fie muchie, adică v1v4 trebuie să fie muchie (am ales v3 = v1).

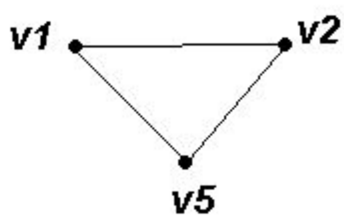
Presupunem că putem alede v4 = v2.

Fie f(5) = v5.

v5 ≠ v1, deoarece 51 este muchie, deci şi v5v1 trebuie să fie muchie.

v5 ≠ v2, deoarece 45 este muchie, deci şi v4v5 trebuie să fie muchie, adică v2v5 trebuie să fie muchie (am ales v4 = v2).

Întrucât v1v5, v2v5, v1v2 sunt muchii în C_5*, am obţine că graful de mai jos este subgraf al lui* C_5*, ceea ce este imposibil.*



Vom face un pas înapoi, alegând altă valoare pentru v4.

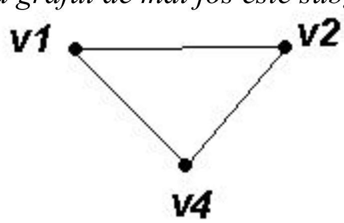
Vom alege v4 ≠ v1 şi v4 ≠ v2.

Fie f(5) = v5.

v5 ≠ v1, deoarece 51 este muchie, deci şi v5v1 trebuie să fie muchie.

v5 ≠ v4, deoarece 45 este muchie, deci şi v4v5 trebuie să fie muchie.

Dacă vom alege v5 = v2, atunci, întrucât v1v2, v2v4, v1v4 sunt muchii în C_5*, am obţine că graful de mai jos este subgraf al lui* C_5*, ceea ce este imposibil.*



Mai facem un pas înapoi şi alegem pentru v3 o valoare diferită de v1.

Fie f(4) = v4.

v4 ≠ v3, deoarece 34 este muchie, deci şi v3v4 trebuie să fie muchie.

Presupunem că putem alege v4 = v2.

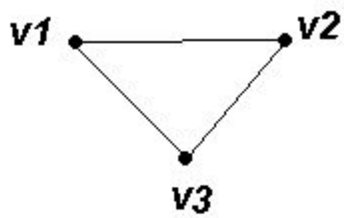
Fie f(5) = v5.

v5 ≠ v1, deoarece 51 este muchie, deci şi v5v1 trebuie să fie muchie.

v5 ≠ v2, deoarece 45 este muchie, deci şi v4v5 trebuie să fie muchie, adică v2v5 trebuie să fie muchie (am ales v4 = v2).

Presupunem că putem alege v5 = v3.

Întrucât v1v3, v2v3, v1v2 sunt muchii în C_5, am obţine că graful de mai jos este subgraf al lui C_5, ceea ce este imposibil.



Alegem altă valoare pentru v4. Deoarece acesta nu poate fi v3 sau v2, după cum am arătat, vom alege v1 (pentru a ne încadra în numărul de valori cerut, adică 3).

Deci v4 = v1.

Deoarece, în aceste condiţii, v1v2, v2v3, v3v4 (adică v3v1) sunt muchii în C_5, conform definiţiei lui f, obţinem că graful de mai sus este subgraf în C_5, ceea ce este evident imposibil.

Am arătat deci că, indiferent de modul de alegere a valorilor pentru f, ***nu se poate găsi pentru C_5 o astfel de funcţie pentru care $|f(V)| \leq 3$.***

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. Arătaţi că graful circuit C_n (n>2) nu are partiţii speciale. Descrieţi un algoritm polinomial care să testeze dacă un graf dat are partiţii speciale.*

Soluţie:

Considerăm T o mulţime de noduri ale circuitului, astfel încât subgraful indus de T în G să fie neconex. Oricum am alege această mulţime putem face următoarea ***observaţie,*** *care se bazează pe neconexitatea subgrafului:*

există cel puţin o pereche de noduri din T : (v_1, v_2) astfel încât, între v_1 şi v_2, să nu existe drum in subgraful indus de mulţimea T.

Fie u şi v o pereche de noduri din T cu această proprietate, alese astfel încât pe unul din cele 2 drumuri de la u la v în circuitul C_n să nu existe alte noduri din T. Este evident că putem găsi 2 noduri cu această proprietate întrucât dacă alegem 2 noduri oarecare x şi y între care nu există drum în $<T>_G$ înseamnă că între acestea, în circuitul C_n, există cel puţin un nod din S, pe fiecare din cele 2 drumuri de la x la y (noduri care întrerup cele 2 drumuri). Alegem unul din drumurile de la x la y în C_n. Dacă pe acest drum mai există cel puţin un nod z din T, atunci z devine noul x şi verificăm pe drumul de la noul x la y dacă mai avem noduri din T. Aplicăm această metodă până când nu mai avem nici un nod din T între x şi y. Dar valorile iniţiale ale lui x şi y au fost alese în aşa fel încât să nu existe drum de la x la y în $<T>_G$ şi deci noii x şi y obţinuţi nu sunt adiacenţi (între ei trebuie să existe cel puţin un nod din S în circuitul C_n)

Nodurile x şi y obţinute sunt 2 noduri cu proprietatea enunţată mai sus, şi deci le putem considera u, respectiv v. Fie w_1,..., w_k nodurile din S aflate între u şi v în C_n (pe drumul de la u la v pe care nu mai există noduri din T). Evident şi celălalt drum de la u la v din C_n conţine noduri din S care să-l întrerupă. Fie t_1, ..., t_r aceste noduri din S. In graful iniţial C_n, nici unul dintre nodurile dintre w_1,..., w_k nu este adiacent cu nici unul dintre nodurile t_1, ..., t_r, deoarece sunt de o parte şi de alta a lui u şi v. Deci în graful complementar al lui C_n avem muchie între fiecare nod t_i şi fiecare nod w_j. Aceste noduri acoperă toată mulţimea S şi avem următoarea proprietate: de la fiecare nod w_i se poate ajunge la fiecare nod t_j, de la fiecare nod t_i se ajunge la un alt nod t_s printr-un nod w_l (care evident este legat de amândouă) şi analog pentru oricare două noduri w_j, w_s, care sunt legate tot printr-un oricare nod t_i. Aşadar între oricare 2 noduri din S există drum în subgraful indus de S în graful complementar lui C_n. Deci acest subgraf este conex.

Oricum am alege o mulţime T care să genereze un subgraf neconex în C_n, mulţimea S obţinută generează un subgraf conex în graful complemetar. Deci pentru orice graf circuit C_n cu n>2 nu avem partiţii speciale. Pentru n egal cu 2 nu putem alege o mulţime T care să genereze un subgraf neconex (întrucât mulţimea T nu poate avea decât cardinalul 1 şi deci am obţine o singură componentă conexă).

Întrucât nu am reuşit să obţinem un algoritm care să determine în timp polinomial o partiţie specială pentru un graf dat vom utiliza un algoritm de tip backtracking , deci un algoritm care să rezolve în timp exponenţial.

Reţinem într-un vector caracteristic v, de dimensiune egală cu numărul de noduri ale grafului nostru G, valori de 0 şi 1 care simbolizează dacă un nod aparţine sau nu mulţimii T. Spunem că un nod i aparţine lui T dacă v[i] are valoarea 0. în cazul în care nodul j nu aparţine lui T atunci pe poziţia j în v vom avea 1 şi deci acest nod va aparţine lui S. Prin algoritmul backtracking pe care îl vom prezenta mai jos vom încerca toate posibilităţile în care putem forma mulţimea T. Vom verifica după ce construim vectorul v făcută dacă se păstrează proprietatea de neconexitate în subgraful determinat de mulţimea T în G şi în subgraful determinat de mulţimea S în complementarul lui G. Se vor încerca toate posibilităţile de creare a vectorului cu valori de 0 şi 1. Condiţia este să avem cel puţin 2 valori distincte . Pornim cu vectorul umplut cu –1. Dacă cele 2 mulţimi de la pasul curent respectă proprietatea algoritmul se opreşte şi se anunţă că graful G admite partiţii speciale.

```
procedure verificare_partitii_speciale ()
begin
        /*iniţializăm vectorul v cu valori de -1
        if (a[0][1] = 1 ) then
                if ( verifica_neconexitate(1) = true ) //verificăm neconexitatea pentru T
                  then
                        afisează „ am găsit o partiţie specială”
                        return
```

```
i ← 0
/* cât timp nu am epuizat toate posibilităţile  intrăm în buclă*/
while ( i>-1 ) do
        ok ← false
        while ( v[i] < 1 and ok = false ) do
                v[i] ← v[i]+1
                if ( i > n-3) then
                                /* verificăm dacă nu avem toate elementele din
urmă cu aceeaşi valoare şi dacă suntem pe n-2 şi valoarea celorlalte elemente
este 0 atunci v[n-2] va fi setat pe 1 iar dacă celelalte elemente au valoarea 1
atunci acesta va fi obligatoriu 0 (dacă s-a verificat deja această variantă, atunci
algorimtul se opreşte întrucât nu mai avem nici o variantă de verificat); dacă
suntem pe poziţia n-1 şi suntem în situaţia că în spate avem n-2 elemente cu
aceeaşi valoare şi unul de o valoare diferită, atunci şi v[n-1] trebuie setat pe
acea valoare*/
                                if (i=n-2) then
                                        if (toate elementele din urmă au aceeaşi
                                valoare val ) then
                                                if ( val = 0  and v[i] = 1) then
                                                        v[i] ← 2
                                                else
                                                        v[i] ← (val+1) mod 2
                                                        ok ← true
                                        else ok ← true
                                else if (i=n-1) then
                                        if (toate n-3 elemente din urmă au aceeaşi
                                valoare val şi unul are valoarea val1 ) then
                                                if ( val1 = 0  and v[i] = 1) then
                                                        v[i] ← 2
                                                else
                                                        v[i] ← (val+1) mod 2
                                                        ok ← true
                                        else ok ← true
                else ok ← true
        if ( ok = true ) then
        /* dacă am găsit o valoare corespunzătoare pentru poziţia respectivă*/
                if ( i=n-1 ) then
        /* dacă am completat vectorul  până la ultima poziţie*/
                        if (verifica_neconexitate(1)=true and
                verifica_neconexitate(0) = true) then
        /* dacă am găsit două mulţimi care corespund*/
                        afişează „graful admite partiţii speciale”
```

return
else
i ← i – 1
/* încercăm să completăm din nou poziţiile din vector cu alte valori*/
else
/* trecem la poziţia următoare în vector şi încercăm să o completăm cu o nouă valoare*/
i ← i + 1
v[i] ← -1
/* dacă nu mai putem continua ne întoarcem */
else *i ← i - 1*

/* dacă am ajuns până aici, înseamnă ca nu am găsit nici o partiţie specială*/
afişează „ graful nu admite partiţii speciale”
end;

Funcţia de verificare a neconexităţii *(se face în timp liniar n+m, unde n este numărul de vârfuri pentru care se verifică neconexitatea iar m este numărul de muchii între aceste vârfuri) este funcţia standard de determinare a componentelor conexe, cu precizarea că se verifică neconexitatea numai pentru subgraful determinat de acele noduri care au valoarea în vectorul v egală cu valoarea transmisă drept parametru funcţiei. Pentru valoarea 0 (elementele din T) verificăm neconexitatea în graful G iar pentru valoarea 1 (elementele din S) verificăm neconexitatea în graful* $\overline{G}$ *(de fapt lucrăm cu negaţiile valorilor din matricea de adiacenţă). Se găsesc de fapt componentele conexe pentru subgraful dorit şi în momentul în care se descoperă 2 componente conexe, funcţia returnează* ***true****. Altfel returnează* ***false****.*