

Cuprins

1	Secvențe de grade	3
1.1	Construcția unui multigraf neorientat cu secvența gradelor dată	3
1.2	Construcția unui graf neorientat cu secvența gradelor dată	4
2	Parcurgeri. Aplicații	7
2.1	Parcurgerea grafurilor	7
2.2	Parcurgea în lățime (BF - Breadth First)	7
2.2.1	Pseudocod	7
2.2.2	Corectitudinea calculului de distanțe folosind BFS	8
2.3	Parcurgea în adâncime (DF – Depth First)	10
2.3.1	Pseudocod	10
2.3.2	Proprietăți	12
2.3.3	Determinarea de cicluri și circuite folosind DFS	14
2.3.4	Componente tare conexe	14
3	Arbori	18
3.1	Construcția unui arbore cu secvența gradelor dată	19
4	Arbori parțiali de cost minim (apcm)	22
4.1	Algoritmul lui Kruskal	22
4.1.1	Clustering	22
4.2	Algoritmul lui Prim	24
5	Drumuri minime în grafuri orientate ponderate	27
5.1	Drumuri minime de sursă unică (de la un vârf dat la celelalte)	27
5.1.1	Algoritmul lui Dijkstra	27
5.1.2	Drumuri minime în grafuri fără circuite (DAGs= Directed Acyclic Graphs) . . .	29
5.2	Drumuri minime între oricare două vârfuri	31
6	Fluxuri în rețele de transport	32
6.1	Noțiuni introductive	32
6.2	Revizuirea fluxului	34
6.3	Algoritmul Ford-Fulkerson	35
6.4	Corectitudinea Algoritmului Ford-Fulkerson	36
6.5	Aplicații - probleme care se reduc la determinarea unui flux maxim	39
6.5.1	Cuplaje în grafuri bipartite	39
6.5.2	Construcția unui graf orientat cu secvențele de grade interne și externe date . . .	43

6.6	Conectivitate	44
-----	-------------------------	----

1 Secvențe de grade

Materialul din această secțiune urmează cartea [?].

Fie $s_0 = \{d_1, \dots, d_n\}$ o secvență de numere naturale.

Problemă. Să se construiască, dacă se poate, un (multi)graf neorientat G cu $s(G) = s_0$.

Observație 1.1. Deoarece suma gradelor vârfurilor într-un (multi)graf este egală cu dublul numărului de muchii, o condiție necesară pentru existența unui (multi)graf G cu $s(G) = s_0$ este ca suma

$$d_1 + \dots + d_n$$

să fie număr par.

❓ Este condiția din Observația 1.1 și suficientă?

1.1 Construcția unui multigraf neorientat cu secvența gradelor dată

Teorema 1.2. O secvență de $n \geq 2$ numere naturale $s_0 = \{d_1, \dots, d_n\}$ este secvența gradelor unui multigraf neorientat dacă și numai dacă suma $d_1 + \dots + d_n$ este număr par.

Proof. " \implies " Presupunem că există un multigraf neorientat G cu $s(G) = s_0$. Atunci

$$d_1 + \dots + d_n = 2|E(G)| \text{ este număr par.}$$

" \impliedby " Presupunem că $d_1 + \dots + d_n$ este număr par.

Rezultă că există un număr par de numere impare în secvența (multisetul) s_0

Construim un multigraf G cu $V(G) = \{x_1, \dots, x_n\}$ având $s(G) = s_0$ (mai exact cu $d_G(x_i) = d_i$) după următorul algoritm:

1. Adăugăm în fiecare vârf $x_i \in V(G)$ $\lfloor \frac{d_i}{2} \rfloor$ bucle.
2. Formăm perechi disjuncte cu vârfurile care trebuie să aibă gradul impar și unim vârfurile din aceste perechi cu câte o muchie.

Formalizând, dacă renotăm numerele din secvența s_0 astfel încât primele $2k$ numere din secvență: d_1, \dots, d_{2k} să fie impare și celelalte pare, definim

$$E(G) = \left\{ x_i x_i^{\lfloor \frac{d_i}{2} \rfloor} \mid i \in \{1, \dots, n\}, d_i > 0 \right\} \cup \{x_i x_{i+1} \mid i \in \{1, \dots, 2k-1\}\}.$$

Atunci, pentru i cu $1 \leq i \leq 2k$ avem d_i impar și

$$d_G(x_i) = 2 \left\lfloor \frac{d_i}{2} \right\rfloor + 1 = 2 \frac{d_i - 1}{2} + 1 = d_i,$$

iar pentru $2k+1 \leq i \leq n$ avem d_i par și

$$d_G(x_i) = 2 \left\lfloor \frac{d_i}{2} \right\rfloor = d_i,$$

deci $s(G) = s_0$. □

1.2 Construcția unui graf neorientat cu secvența gradelor dată

Dat un graf neorientat G , pentru a obține grafuri neorientate cu aceeași secvență de grade ca și G se poate folosi următoarea transformare t (pe care o vom numi de interschimbare pe pătrat). Fie x, y, u, v patru vârfuri distincte ale lui G astfel încât $xy, uv \in E(G)$, dar $xu, yv \notin E(G)$. Considerăm graful notat $t(G, xy, uv)$ definit astfel:

$$t(G, xy, uv) = G - \{xy, uv\} \cup \{xu, yv\}$$

Spunem că $t(G, xy, uv)$ este graful obținut din G prin aplicarea transformării t de interschimbare pe pătratul $xyvu$ - figura 1.

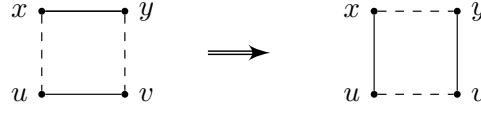


Figure 1: Transformarea t de interschimbare pe pătrat

Observație 1.3. Graful $t(G, xy, uv)$ are aceeași secvență de grade ca și G .

Teorema 1.4. (Havel-Hakimi) O secvență de $n \geq 2$ numere naturale $s_0 = \{d_1 \geq \dots \geq d_n\}$ cu $d_1 \leq n - 1$ este secvența gradelor unui graf neorientat (cu n vârfuri) dacă și numai dacă secvența $s'_0 = \{d_2 - 1, \dots, d_{d_1+1} - 1, d_{d_1+2}, \dots, d_n\}$ este secvența gradelor unui graf neorientat (cu $n - 1$ vârfuri).

Proof. " \Leftarrow " Presupunem că s'_0 este secvența gradelor unui graf neorientat. Fie $G' = (V', E')$ un graf neorientat cu $V' = \{x_2, \dots, x_n\}$ având secvența gradelor $s(G') = s'_0$, mai precis cu

$$d_{G'}(x_i) = \begin{cases} d_i - 1, & \text{dacă } i \in \{2, \dots, d_1 + 1\} \\ d_i, & \text{dacă } i \in \{d_1 + 2, \dots, n\}. \end{cases}$$

Construim pornind de la G' un nou graf $G = (V, E)$ adăugând un vârf nou x_1 pe care îl unim cu vârfurile x_2, \dots, x_{d_1+1} :

- $V = V' \cup \{x_1\}$
- $E = E' \cup \{x_1 x_i \mid i \in \{2, \dots, d_1 + 1\}\}$.

Pentru un $i \in \{1, \dots, n\}$ avem atunci

$$d_G(x_i) = \begin{cases} d_{G'}(x_i) + 1 = d_i - 1 + 1 = d_i, & \text{dacă } i \in \{2, \dots, d_1 + 1\} \\ d_{G'}(x_i) = d_i, & \text{dacă } i \in \{d_1 + 2, \dots, n\} \\ d_1, & \text{dacă } i = 1. \end{cases}$$

Rezultă că $s(G) = s_0$, deci s_0 este secvența gradelor unui graf neorientat.

" \Rightarrow " Presupunem că s_0 este secvența gradelor unui graf neorientat.

Fie $G = (V, E)$ un graf neorientat cu $V = \{x_1, \dots, x_n\}$ astfel încât $d_G(x_i) = d_i$ pentru orice $i \in \{1, \dots, n\}$. Vom construi un graf G' cu $s(G') = s'_0$ pornind de la G .

Pentru aceasta, construim întâi din G un graf G^* având secvența gradelor tot s_0 , dar în care vârful x_1 are mulțimea vecinilor $N_{G^*}(x_1) = \{x_2, \dots, x_{d_1+1}\}$.

Cazul 1. Dacă $N_G(x_1) = \{x_2, \dots, x_{d_1+1}\}$, atunci considerăm $G^* = G$.

Cazul 2. Există cel puțin un indice $i \in \{2, \dots, d_1 + 1\}$ cu $x_1 x_i \notin E$ (i.e. $x_i \notin N_G(x_1)$). Fie i minim cu această proprietate.

Deoarece $d_G(x_1) = d_1$, rezultă că există $j \in \{d_1 + 2, \dots, n\}$ cu $x_1 x_j \in E$. Mai mult, deoarece $j > d_1 + 1 \geq i$, avem $d_i = d_G(x_i) \geq d_G(x_j) = d_j$. În plus, x_1 este adiacent cu x_j , dar nu și cu x_i . Rezultă că există un alt vârf x_k cu $k \in \{2, \dots, n\} - \{i, j\}$ care este adiacent cu x_i ($x_i x_k \in E$), dar care nu este adiacent cu x_j ($x_j x_k \notin E$) - figura 2.

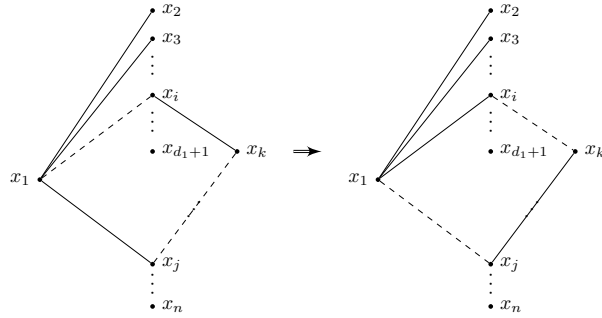


Figure 2: Transformare din demonstrația teoremei Havel-Hakimi

Considerăm graful G_i obținut din G prin aplicarea transformării de interschimbare t pentru pătratul $x_i x_k x_j x_1$:

$$G_i = t(G, x_i x_k, x_1, x_j) = G - \{x_i x_k, x_1 x_j\} \cup \{x_1 x_i, x_k x_j\}$$

Avem $N_{G_i}(x_1) \cap \{x_2, \dots, x_{d_1+1}\} = (N_G(x_1) \cap \{x_2, \dots, x_{d_1+1}\}) \cup \{x_i\}$ (x_1 are un vecin în plus în $\{x_2, \dots, x_{d_1+1}\}$) și, conform Observației 1.3, $s(G_i) = s(G) = s_0$.

Aplicând succesiv transformări de tip t pentru fiecare indice $i \in \{2, 3, \dots, d_1 + 1\}$ pentru care x_1 și x_i nu sunt adiacente obținem în final un graf G^* cu $s(G^*) = s_0$ și $N_{G^*}(x_1) = \{x_2, \dots, x_{d_1+1}\}$.

Fie $G' = G^* - x_1$. Atunci $V(G') = \{x_2, \dots, x_n\}$ și pentru orice $i \in \{2, \dots, n\}$:

$$d_{G'}(x_i) = \begin{cases} d_{G^*}(x_i) - 1 = d_i - 1, & \text{dacă } i \leq d_1 + 1 \\ d_{G^*}(x_i) = d_i, & \text{dacă } i \geq d_1 + 2, \end{cases}$$

deci $s(G') = s'_0$. Rezultă că $s'_0 = \{d_2 - 1, \dots, d_{d_1+1} - 1, d_{d_1+2}, \dots, d_n\}$ este secvența gradelor unui graf neorientat. \square

Din Teorema Havel-Hakimi se obține următorul algoritm de determinare a unui graf neorientat cu secvența gradelor dată.

Algoritmul Havel-Hakimi

Intrare: o secvență de n numere naturale d_1, \dots, d_n

Ieșire: un graf G cu $V(G) = \{x_1, \dots, x_n\}$ cu $s(G) = s_0$ dacă s_0 este secvența gradelor unui graf, sau mesajul NU altfel.

Idee: La un pas unim un vârf de grad maxim d din secvența s_0 cu vârfurile corespunzătoare următoarelor cele mai mari d elemente din s_0 diferite de d și actualizăm secvența s_0 ($s_0 = s'_0$). Se repetă pasul până când secvența conține numai 0 sau conține elemente negative.

Pseudocod:

Pasul 1. Dacă $d_1 + \dots + d_n$ este număr impar sau există în s_0 un număr $d_i > n - 1$, atunci scrie NU, STOP.

Pasul 2.

cât timp s_0 conține valori nenule execută

 alege d_k cel mai mare număr din secvența s_0

 elimină d_k din s_0

 fie $d_{i_1}, \dots, d_{i_{d_k}}$ cele mai mari d_k numere din s_0

 pentru $j \in \{i_1, \dots, i_{d_k}\}$:

 adaugă la G muchia $x_k x_j$

 înlocuiește d_j în secvența s_0 cu $d_j - 1$

 dacă $d_j - 1 < 0$, atunci scrie NU, STOP.

Observație. Pentru a determina ușor care este cel mai mare număr din secvență și care sunt cele mai mari valori care îi urmează, este util ca pe parcursul algoritmului secvența s_0 să fie ordonată descrescător.

Exemplu. - vezi curs + laborator

Teorema 1.5. (Extindere a teoremei Havel-Hakimi) Fie $s_0 = \{d_1 \geq \dots \geq d_n\}$, o secvență de $n \geq 2$ numere naturale cu $d_1 \leq n - 1$ și fie $i \in \{1, \dots, n\}$ fixat. Fie $s_0^{(i)}$ secvența obținută din s_0 prin următoarele operații:

- eliminăm elementul d_i

- scădem o unitate din primele d_i componente în ordine descrescătoare ale secvenței rămase.

Are loc echivalența:

s_0 este secvența gradelor unui graf neorientat \iff

$s_0^{(i)}$ este secvența gradelor unui graf neorientat

Proof. Demonstrația este similară cu cea a Teoremei Havel-Hakimi ([?] - exercițiul 2.11). □

Exercițiu ([?] - exercițiul 2.12) Fie G_1 și G_2 două grafuri neorientate cu mulțimea vârfurilor $V = \{1, \dots, n\}$. Atunci $s(G_1) = s(G_2)$ dacă și numai dacă există un șir de transformări t de interschimbare pe pătrat prin care se poate obține graful G_2 din G_1 .

2 Parcurgeri. Aplicații

2.1 Parcurgerea grafurilor

Scop: Dat un graf și un vârf de start s , să se determine toate vârfurile accesibile din s (printr-un lanț/drum, după cum graful este neorientat/orientat).

Ideea: pornim din vârful de start s ; dacă știm că un vârf i este accesibil din s și există muchie (arc în caz orientat) de la i la j , atunci și j este accesibil din s .

2.2 Parcurgea în lățime (BF - Breadth First)

Parcurgerea pe lățime BF urmărește vizitarea vârfurilor în ordinea crescătoare a distanțelor lor față de s (distanța este dată de numărul de muchii).

Se pornește din vârful de start s , se vizitează vecinii acestuia (vârfurile adiacente cu el), apoi vecinii nevizitați anterior ai acestora, procesul repetându-se până nu mai există vârf cu vecini nevizitați.

2.2.1 Pseudocod

Pentru a reține ce vârfuri au fost vizitate (descoperite) deja vom folosi un vector viz.

Pentru gestionarea vârfurilor parcurse care mai pot avea vecini nevizitați se va folosi o structură de tip coadă.

Muchiile folosite de algoritm pentru a descoperi noi vârfuri accesibile din s formează un arbore de rădăcină s . Pentru a reține acest arbore putem folosi legături de tip tată (predecesor). Atunci când vârful j este descoperit ca vecin nevizitat al lui i și introdus în coadă vom atribui lui $tata[j]$ valoarea i (j este fiu al lui i în arbore).

Vom demonstra că pentru orice vârf i accesibil din s lanțul/drumul determinat prin parcurgerea BF de la s la i (din arbore) este minim (ca număr de muchii). Putem reconstitui un astfel de drum folosind legătura $tata$, mergând înapoi din i spre s . Pentru a memora doar lungimea acestui drum, adică distanța de la s la i putem memora suplimentar un vector d cu semnificația

$d[i]$ = lungimea drumului determinat de algoritm de la s la i = nivelul lui i în arborele asociat parcurgerii

Avem $d[j] = d[tata[j]] + 1$ (nivelul lui j este cu 1 mai mare decât al vârfului din care a fost descoperit)

Algoritmul BFS

Inițializări

pentru fiecare $x \in V$ executa

$viz[x] = 0$

$tata[x] = 0$

$d[x] = \text{inf}$

BFS(s)

```

coada  $C = \emptyset$ 
adauga( $s, C$ )
 $viz[s] = 1; d[s] = 0$  //s devine gri, este in explorare
cat timp  $C \neq \emptyset$  executa
     $i = \text{extrage}(C)$ 
    afiseaza( $i$ )
    pentru  $j$  vecin al lui  $i$  ( $ij \in E$ )
        daca  $viz[j] == 0$  atunci
            adauga( $j, C$ )
             $viz[j] = 1$ 
             $tata[j] = i$ 
             $d[j] = d[i] + 1$ 
//s devine negru, explorarea sa s-a incheiat

```

Apel

Pentru un vârf s procedura de parcurgere se apelează $\text{BFS}(s)$.

Pentru a parcurge toate vârfurile grafului se reia apelul subprogramului BFS pentru vârfurile rămase nevizitate:

```

pentru fiecare  $x \in V$  executa
    daca  $viz[x] == 0$  atunci
         $\text{BFS}(x)$ 

```

Complexitatea algoritmului depinde de modul în care este memorat graful și se obține însumând timpul necesar inițializărilor $O(n)$ cu timpul necesar parcurgerii vecinilor pentru fiecare vârf. Dacă graful este memorat cu matrice de adiacență, vecinii unui vârf se determină în $O(n)$, deci ai tuturor vârfurilor în $O(n^2)$. Dacă graful este memorat cu liste de adiacență, vecinii unui vârf se determină în $O(\text{gradul lui } i)$, deci ai tuturor vârfurilor în $O(\text{suma gradelor}) = O(m)$; astfel, parcurgerea va avea complexitatea $O(n + m)$ pentru reprezentarea grafului cu liste de adiacență.

2.2.2 Corectitudinea calculului de distante folosind BFS

Următoarele observații de pot demonstra ușor prin reducere la absurd.

Observație 2.1. *Fie G un graf orientat/neorientat*

Dacă P este un drum/lanț minim de la s la u , atunci P este drum/lanț elementar (altfel, dacă s-ar repeta un vârf u , am putea elimina porțiunea de drum dintre două apariții ale lui u și obținem un drum cu mai puține vârfuri).

Dacă P este un drum/lanț minim de la s la u și z este un vârf al lui P , atunci subdrumul lui P de la s la z este drum/lanț minim de la s la z .

Lema 2.2. Dacă la un moment al execuției algoritmului BFS, în coada C avem vârfurile : v_1, v_2, \dots, v_r , atunci

$$d[v_1] \leq d[v_2] \leq \dots \leq d[v_r] \leq d[v_1] + 1$$

Proof. Vom demonstra că după fiecare operație asupra cozii C relația rămâne adevărată.

Prima operație este inserarea vârfului de start s în C ; după această operație în coadă avem doar un vârf și deci relația este adevărată.

Presupunem afirmația adevărată la pasul curent. Notăm v_1, v_2, \dots, v_r vârfurile din coadă la acest pas.

La pasul curent se extrage din coadă vârful v_1 și se adaugă în coadă vecinii lui nevizitați w_1, \dots, w_k . După extragerea vârfului v_1 , în coadă rămân vârfurile v_2, \dots, v_r care verifică relația:

$$d[v_2] \leq \dots \leq d[v_r] \leq d[v_1] + 1 \leq d[v_2] + 1$$

Un vecin $w_i, 1 \leq i \leq k$ a lui v_1 va avea eticheta $d[w_i] = d[v_1] + 1 \leq d[v_2] + 1$, deci după inserarea acestuia în coadă, coada va avea elementele $v_2, \dots, v_r, w_1, \dots, w_k$ care verifică relația:

$$d[v_2] \leq \dots \leq d[v_r] \leq d[v_1] + 1 = d[w_1] = \dots = d[w_k] = d[v_1] + 1 \leq d[v_2] + 1$$

□

Lema 2.3. 1. Dacă $d[v] = k < \infty$, atunci există în G un drum de la s la v de lungime k și acesta se poate determina din vectorul $tata$, mai exact $tata[v] = \text{predecesorul lui } v \text{ pe un drum de la } s \text{ la } v \text{ de lungime } k$.

2. $d[v] \geq \delta(s, v)$

3. Dacă $d[v] = \infty$, atunci nu există în G drum de la s la v .

Proof. Putem demonstra prin inducție că atunci când pentru toate nodurile descoperite până la un pas afirmația este adevărată.

Inițial este vizitat vârful s și se setează $d[s] = 0$, corespunzător drumului $[s]$. Presupunem că la pasul curent afirmația este valabilă pentru toate vârfurile deja descoperite.

Fie u vârf curent (extras din coadă la pasul curent). Atunci pentru orice vecin v al său nevizitat până acum avem $d[v] = d[u] + 1$ și $tata[v] = u$ și există un drum de la s la v de lungime $d[u] + 1$ obținut din drumul de la s la u de lungime $d[u]$ (care există conform ipotezei de inducție) la care se adaugă vârful v (în care u este predecesor al lui v)

A doua afirmație este o consecință a primeia, deoarece $d[u]$ este lungimea unui drum de la s la u , iar $\delta(s, u)$ este lungimea unui drum minim. □

Teorema 2.4. Fie $G = (V, E)$ un graf neorientat și $s \in V$ fixat. La finalul algoritmului de parcurgere BF din s avem

$$d[u] = \delta(s, u) \text{ pentru orice } u \in V$$

și tata memorează un arbore al distanțelor față de s .

Proof. Presupunem prin absurd că afirmația nu este adevărată.

Fie y vârful cel mai apropiat de s cu $d[y]$ calculat incorect. Atunci, din Lema 2.3, rezultă că

$$d[y] > \delta(s, y).$$

Fie P un drum minim de la s la y și x predecesorul lui y pe acest drum. Atunci, conform Observației ??, subdrumul lui P de la s la x este drum minim, deci are lungimea egală cu $\delta(s, x)$ și avem:

$$\delta(s, x) < l(P) = \delta(s, x) + 1 = \delta(s, y).$$

Atunci x este mai apropiat de s decât y , deci eticheta lui este calculată corect: $\delta(s, x) = d[x]$. Avem atunci

$$l(P) = d[x] + 1 = \delta(s, y) < d[y].$$

Din modul de funcționare al BFS, când x este scos din coadă, y este în una din situațiile:

- deja vizitat și extras din coadă (negru): atunci $d[y] \leq d[x]$ (conform relației din Lema 2.3, deoarece y a fost extras și deci și inserat în coadă înaintea lui x)
- deja vizitat și încă în coadă (gri): atunci $d[y] \leq d[x] + 1$ (Lema 2.3)
- este nevizitat încă (alb): atunci y va fi vizitat din x (x fiind nodul curent explorat), deci $d[y] = d[x] + 1$.

Rezultă $d[y] \leq d[x] + 1 = l(P) = \delta(s, y) < d[y]$, contradicție.

□

2.3 Parcurgea în adâncime (DF – Depth First)

Pentru parcurgea în adâncime a componentei conexe a unui vârf de start s se pornește din acest vârf și se trece mereu la primul dintre vecinii vârfului curent nevizitați anterior, dacă un astfel de vecin există. Dacă toți vecinii vârfului curent au fost deja vizitați se merge înapoi pe drumul de la s la vârful curent până se ajunge la un vârf care mai are vecini nevizitați; se trece la primul dintre aceștia și se reia procedeul.

2.3.1 Pseudocod

Algoritmul DFS Inițializări - ca și la BFS

DFS(x)

```
//incepe explorarea varfului x, devine gri  
viz[x] = 1  
pentru fiecare  $xy \in E$  //y vecin al lui x  
daca viz[y] == 0 atunci  
    tata[y] = x
```

$d[y] = d[x] + 1$ //nivel, nu distanta
 $DFS(y)$
 //s-a finalizat explorarea varfului x, devine negru

Apel - ca și la BFS

Complexitatea - ca și la BFS

Pseudocod complet

Un vârf are una dintre următoarele stări în timpul parcurgerii DFS:

1. **nevizitat** – culoarea **albă**
2. **în explorare** – culoare **gri**
3. **finalizat** – culoare neagră

Dacă menținem pe parcursul algoritmului o variabilă de timp care se modifică atunci când se schimbă starea unui vârf (îl descoperim sau îl finalizăm), putem asocia fiecărui vârf două valori:

1. $desc[v]$ – momentul la care vârfurile a fost descoperit (a devenit gri)
2. $fin[v]$ - momentul la care vârfurile a fost finalizat (a devenit negru)

Dacă dorim determinarea unor elemente de structură în graf precum circuite, componente tare conexe etc poate fi util să memorăm în parcurgerea DFS culoarea fiecărui vârf și/sau momentul în care a fost descoperit sau finalizat.

Dacă memorăm culoarea fiecărui vârf, nu mai este necesar și vectorul vizitat, deoarece

$$viz[x] = 0 \iff culoare[x] = alb$$

Un pseudocod complet este următorul:

$DFS(x)$

$culoare[x] = gri$
 $timp = timp + 1$
 $desc[x] = timp$ //incepe explorarea varfului x
 pentru fiecare $xy \in E$ //y vecin al lui x
 daca $culoare[y] == 0$ atunci
 $tata[y] = x$
 $d[y] = d[x] + 1$ //nivel, nu distanta
 $DFS(y)$
 $culoare[x] = negru$
 $timp = timp + 1$
 $fin[x] = timp$ //s-a finalizat explorarea varfului x

Apel:

pentru fiecare $x \in V$ executa

$culoare[x] = alb$

```

    timp = 0
    pentru fiecare  $x \in V$  executa
        daca  $culoare[x] = alb$  atunci
            DFS( $x$ )

```

2.3.2 Proprietăți

Din modul de funcționare a parcurgerii DF rezultă următoarea proprietate.

Propoziția 2.5. *Vârful v este un descendent al lui u în pădurea de adâncime DF pentru un graf G (orientat sau neorientat) dacă și numai dacă vârful v este descoperit în timp ce u este gri (este încă în explorare)*

Au loc rezultate mai generale:

Teorema 2.6. (Teorema parantezelor) *În orice căutare în adâncime a unui graf (orientat sau neorientat) $G = (V, E)$, pentru orice două vârfuri u și v cu*

$$desc[u] < desc[v]$$

exact una din următoarele condiții este adevărată:

1. *intervalele $[desc[u], fin[u]]$ și $[desc[v], fin[v]]$ sunt disjuncte și u și v nu sunt unul descendentul celuilalt în arborele/pădurea DF*
2. *$[desc[v], fin[v]] \subset [desc[u], fin[u]]$, caz în care v este un descendent al lui u în arborele/pădurea DF*

Proof. Din ipoteză avem $desc[u] < desc[v]$. Distingem două cazuri:

1. $desc[v] > fin[u]$: intervalele sunt disjuncte; atunci u este deja negru când v a fost descoperit, deci v nu este descendent al lui u și nici invers
2. $desc[v] < fin[u]$: v este descoperit în timp ce u este gri, deci este descendent al lui u ; atunci v va fi finalizat înaintea lui u , deci $[desc[v], fin[v]] \subset [desc[u], fin[u]]$

□

Propoziția 2.7. (Incluziunea intervalelor descendenților) *Vârful v este un descendent al lui u în pădurea de adâncime DF pentru un graf G (orientat sau neorientat) dacă și numai dacă vârful v este descoperit în timp ce u este gri (este încă în explorare) dacă și numai dacă $[desc[v], fin[v]] \subset [desc[u], fin[u]]$*

În raport cu pădurea DF muchiile/arcele se împart în 4 categorii (v.fig. 3):

1. **de arbore DF (de avansare directă/ de arbore):** (u, v) cu v fiu al lui u (v a fost descoperit din u)
2. **de întoarcere (înapoi):** (u, v) cu v strămoș al lui u

3. **de avansare (de avansare înainte):** (u, v) cu v descendent al lui u care nu este însă fiu al lui
4. **de traversare (transversale):** toate celelalte muchii; pot fi între vârfuri din același arbore DF cu condiția ca unul să nu fie strămoșul celuilalt sau pot uni vârfuri din arbori DF diferiți

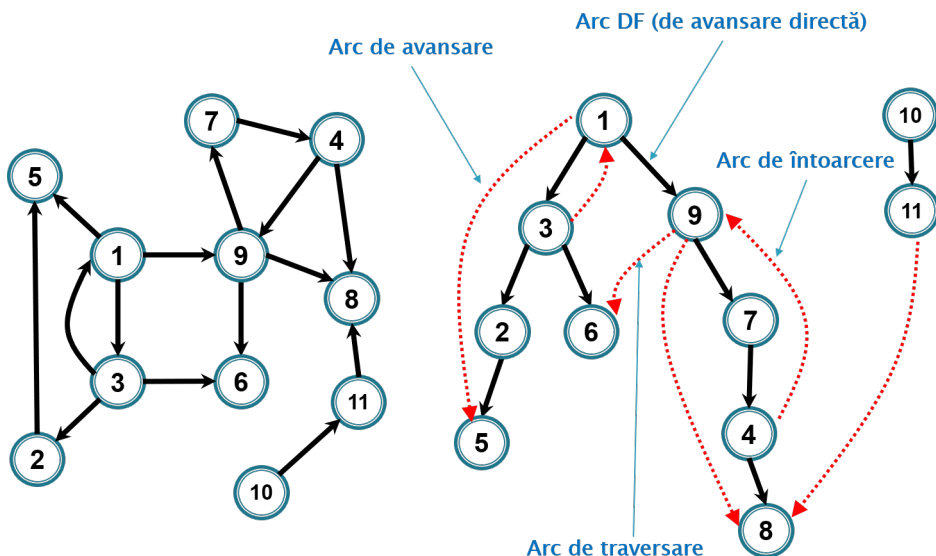


Figure 3

În cazul unui graf neorientat are loc următoarea proprietate, care rezultă din modul de funcționare al parcurgerii DF: dacă un vârf x nu este descendent al unui vârf y în pădurea DF, atunci nu există muchie de la x la y , altfel dacă x a fost descoperit primul, y ar fi fost descoperit în timpul explorării lui x și ar fi devenit descendent al acestuia (similar, dacă y ar fi fost descoperit primul, x ar fi devenit descendent al acestuia).

Propoziția 2.8. (Tipuri de muchii în cazul neorientat) În cazul unui graf neorientat există doar două tipuri de muchii în raport cu pădurea DF:

- muchii de arbore (de avansare)
- muchii de întoarcere (către un ascendent)

Algoritmic, o muchie este detectată în timpul parcurgerii DF ca fiind de întoarcere dacă unește vârful curent cu un vârf gri (ascendent al său) care nu este tatăl lui.

În cazul unui graf neorientat pot exista toate cele 4 tipuri de arce.

Propoziția 2.9. (Caracterizarea celor 4 tipuri de arce în raport cu pădurea DF) Fie G un graf orientat. Un arc (u, v) al său este detectat prin parcurgere DF ca fiind:

- de avansare (de arbore): dacă v este nevizitat (alb) atunci când este parcurs arcul (u, v) ; în acest caz

$$desc[u] < desc[v] < fin[v] < fin[u]$$

- *de avansare înainte: dacă v este negru (finalizat) atunci când este parcurs arcul (u, v) și este descendent al lui u , adică $desc[u] < desc[v]$; în acest caz*

$$desc[u] < desc[v] < fin[v] < fin[u]$$

- *de întoarcere: dacă v este încă în explorare (gri, nefinalizat) atunci când este parcurs arcul (u, v) ; în acest caz*

$$desc[v] < desc[u] < fin[u] < fin[v]$$

- *de traversare: dacă v este negru (finalizat) atunci când este parcurs arcul (u, v) și nu este descendent al lui u , adică $desc[u] > desc[v]$; în acest caz*

$$desc[v] < fin[v] < desc[u] < fin[u]$$

2.3.3 Determinarea de cicluri și circuite folosind DFS

Determinarea unui ciclu/circuit într-un graf neorientat/orientat folosind DFS se bazează pe următoarea observație:

Observație 2.10. *Un graf neorientat/orientat G are ciclu/circuit dacă și numai dacă există muchie/arc de întoarcere în pădurea DFS (care unește un vârf de un strămoș al lui)*

2.3.4 Componente tare conexe

Fie G un graf orientat.

În determinarea componentelor tare conexe ale lui G , deoarece este necesar ca între două vârfuri u și v să existe și drum de la u la v , și drum de la v la u , este util să considerăm graful transpus G^T obținut din G prin inversarea orientării arcelor:

$$G^T = (V, E^T), E^T = \{yx | xy \in E\}$$

Atunci componenta tare conexă a lui x se obține intersectând mulțimea vârfurilor accesibile din x în G cu mulțimea vârfurilor accesibile din x în G^T .

Dacă facem însă parcurgeri pentru fiecare vârf în G și în G^T , complexitatea algoritmului va fi $O(n^2 + nm)$.

Există însă algoritmi prin care se pot determina componentele tare conexe ale unui graf folosind doar o parcurgere (Algoritmul lui Tarjan) sau două (Algoritmul lui Kosaraju): una în G și cealaltă în G^T , dar parcurgerea lui G^T se face într-o ordine particulară a vârfurilor (în funcție de ordinea în care au fost finalizate în parcurgerea DF a lui G).

Vom prezenta în continuare Algoritmul lui Kosaraju, fiind un algoritm simplu. Ideea Algoritmului lui Tarjan este similară celei de la algoritmul de determinare a componentelor biconexe ale unui graf neorientat (v.seminar).

Algoritmul lui Kosaraju- Pseudocod:

Pasul 1. Parcurgem în adâncime graful G și adăugăm într-o stivă S fiecare vârf la momentul la care este finalizat (pentru a obține o ordonare descrescătoare a vârfurilor după timpul de finalizare)

Pasul 2. Construim graful G^T

Pasul 3. Parcurgem în adâncime graful G^T considerând vârfurile în ordinea în care sunt extrase din S (descrescătoare după timpul de finalizare de la Pasul 1):

cat timp S este nevida

$x = pop(S)$

dacă x este nevizitat atunci

$DFS(G^T, x)$

afiseaza componenta tare conexă formată cu vârfurile vizitate în $DFS(G^T, x)$

Corectitudine

Notăm cu $fin[v]$ momentul la care este finalizat un nod în parcurgerea DFS a grafului G de la Pasul 1.

Un exemplu este dat în figura 4 unde sunt indicați timpii de finalizare și componentele tare conex care vor fi obținute la Pasul 3 al algoritmului, pentru a evidenția relațiile între timpii de finalizare pentru vârfurile din componente diferite, relații cu ajutorul cărora vom demonstra corectitudinea algoritmului. Principala relație pe care o vom demonstra este că dacă C_1 și C_2 sunt două componente tare conex ale lui G astfel încât există un arc/drum (u, v) de la C_1 la C_2 ($u \in C_1, v \in C_2$), atunci

$$\max\{fin[x] | x \in C_1\} > \max\{fin[x] | x \in C_2\}$$

Este evident, dar important, că dacă există un arc de la C_1 la C_2 , nu există arc și de la C_2 la C_1 (alfel vârfurile din C_1 și C_2 ar fi în aceeași componentă tare conexă).

Lema 2.11. Fie C o componentă tare conexă a lui G și v un vârf din G care nu aparține lui C , dar există un drum de la C la v (adică de la un vârf din C la v). Atunci

$$\max\{fin[u] | u \in C\} > fin[v]$$

(există în C un vârf cu timp de finalizare mai mare decât v)

Proof. Fie x primul vârf din $C \cup \{v\}$ vizitat la pasul 1.

Cazul 1: $x = v$. Deoarece $v \notin C$, nu există arc(nici drum) de la v la un vârf din C . Rezultă că v va fi finalizat înainte de a fi descoperit vreun vârf din C , deci $fin[v] < \max\{fin[u] | u \in C\}$ (de fapt în acest caz $fin[v] < fin[u]$ pentru orice $u \in C$).

Cazul 2: $x \in C$. Deoarece v este accesibil dintr-un vârf din C , v este accesibil și din x și este nevizitat la momentul apelului $DFS(x, G)$. Rezultă că v va fi descoperit și finalizat în cadrul acestui apel (ca

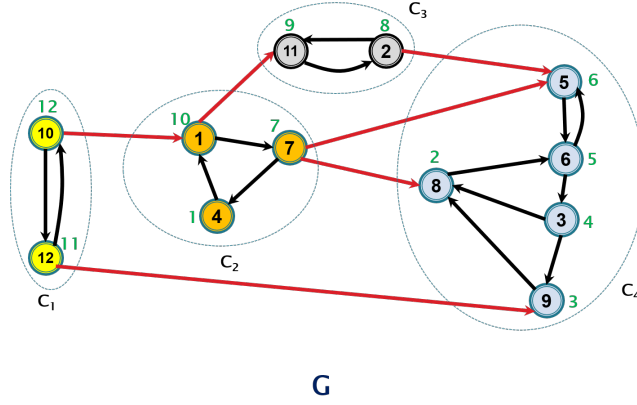


Figure 4

descendent al lui x), deci $fin[x] > fin[v]$ (toate vârfurile din $C \cup \{v\}$ or fi descoperite și finalizate în acest apel, x fiind ultimul finalizat).

□

Corolar 2.12. Fie C_1 și C_2 două componente tare conexe ale lui G . Dacă există un arc/drum (u, v) de la C_1 la C_2 ($u \in C_1, v \in C_2$), atunci

$$\max\{fin[x] | x \in C_1\} > \max\{fin[x] | x \in C_2\}$$

Proof. Deoarece există un drum de $u \in C_1$ la un vârf $v \in C_2$, atunci, din definiția componentei tare conexe, există un drum de la u la fiecare vârf $x \in C_2$. Folosind Lema 2.11 rezultă că

$$\max\{fin[u] | u \in C_1\} > fin[x], \forall x \in C_2,$$

de unde rezultă relația din enunț.

□

Corolar 2.13. Fie C_1 și C_2 două componente tare conexe ale lui G . Dacă

$$\max\{fin[u] | u \in C_1\} > \max\{fin[v] | u \in C_2\}$$

atunci

- nu există drum de la C_2 la C_1 în G și
- nu există drum de la C_1 la C_2 în G^T

Proof. Din Corolarul anterior, dacă am avea drum de la C_2 la C_1 , ar avea loc inegalitatea inversă:

$$\max\{fin[u] | u \in C_1\} < \max\{fin[v] | u \in C_2\}.$$

În plus nu există drum de la C_2 la C_1 în G dacă și numai dacă nu există drum de la C_1 la C_2 în G^T . □

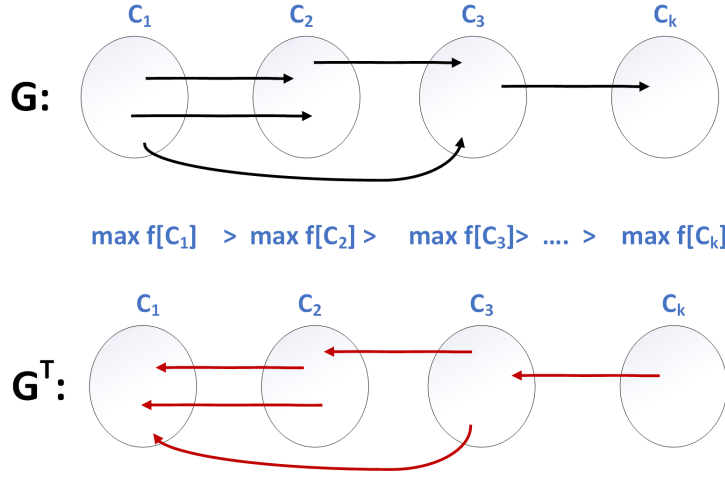


Figure 5

Teorema 2.14. Fie C_1, C_2, \dots, C_k componentele tare conexe ale lui G , astfel încât

$$\max\{fin[u] | u \in C_1\} > \max\{fin[u] | u \in C_2\} > \dots > \max\{fin[u] | u \in C_k\}$$

Fie x cel de al i -lea vârf pentru care se apelează $DFS(x, G^T)$ la Pasul 3. Atunci, la momentul acestui apel toate vârfurile din componentele C_1, \dots, C_{i-1} au fost vizitate, iar cele din C_i, \dots, C_k nu au fost vizitate. În plus, în apelul $DFS(x, G^T)$ se vor vizita toate vârfurile componentei C_i și doar acestea.

Proof. Demonstrăm prin inducție după i .

Pentru $i = 1$: Primul vârf pentru care se apelează $DFS(x, G^T)$ este cel cu timpul de finalizare $fin[x]$ maxim. LA acest moment niciun vârf nu este vizitat. Prin apelul $DFS(x, G^T)$ se vor vizita toate vârfurile accesibile din x în G^T , deci și toate vârfurile din C_1 . În plus, deoarece

$$\max\{fin[u] | u \in C_1\} > \max\{fin[u] | u \in C_j\} \text{ pentru orice } j \neq 1.$$

conform Corolarului anterior nu există drum în G^T de la x la un vârf care nu este în C_1 , deci vârf vizitate în apelul $DFS(x, G^T)$ sunt chiar vârfurile din C_1 .

Presupunem afirmațiile adevărate până la pasul i , $1 \leq i < k$ și demonstrăm că rămân adevărate și la pasul $i + 1$. Din ipoteza de inducție, până la acest pas au fost vizitate doar vârfurile din componentele C_1, \dots, C_i (și doar acelea), iar aceste componente au fost corect afișate. Fie x vârf pentru care se apelează $DFS(x, G^T)$ la pasul $i + 1$. Vârful x are cel mai mare timp de finalizare dintre vârfurile rămase, deci $x \in C_{i+1}$. Ca și la pasul de verificare, din Corolarul anterior avem, nu există drum în G^T de la un vârf din C_{i+1} la unul din $C_{i+2} \cup \dots \cup C_k$, deci în apelul $DFS(x, G^T)$ vor fi vizitate doar vârfurile din C_{i+1} (toate) și afișate ca formând a $i + 1$ -a componentă tare conexă. \square

3 Arbori

Definiție 3.1. *Un arbore este un graf neorientat conex fără cicluri.*

Lema 3.2. *Orice arbore cu $n \geq 2$ vârfuri are cel puțin două vârfuri terminale (de grad 1).*

Proof. Fie T un arbore cu $n \geq 2$ vârfuri. Fie P cel mai lung lanț elementar în T . Extremitățile lui P sunt vârfuri terminale, altfel am putea extinde lanțul P cu o muchie sau se formează un ciclu elementar în T . Într-adevăr, fie x, y extremitățile lui P . Presupunem prin absurd că $d_T(x) \geq 2$. Atunci există un vârf v adiacent cu x astfel încât $xv \notin E(P)$.

- Dacă $v \in V(P)$, atunci există în T ciclul $[x \overset{P}{-} v, x]$, contradicție (T este aciclic).
- Dacă $v \notin V(P)$, atunci lanțul $P' = [v, x \overset{P}{-} y]$ este elementar și $l(P') = l(P) + 1$ contradicție (P este lanț elementar maxim în T). \square

Lema 3.3. *Fie T un arbore cu $n \geq 2$ vârfuri și v un vârf terminal în T . Atunci $T - v$ este arbore.*

Proof. Afirmția rezultă din faptul că un vârf terminal nu poate fi vârf intern al unui lanț elementar, deci pentru orice $x, y \neq v$ un $x - y$ lanț elementar din T este lanț și în $T - v$. \square

Propoziția 3.4. *Un arbore cu n vârfuri are $n - 1$ muchii.*

Proof. Demonstrăm afirmația prin inducție.

Pentru $n = 1$, un arbore cu n vârfuri are un vârf și zero muchii. Pentru $n = 2$, un arbore cu 2 vârfuri are o singură muchie (este izomorf cu P_2).

" $n - 1 \implies n$ " Presupunem afirmația adevărată pentru un arbore cu $n - 1$ vârfuri.

Fie T un arbore cu n vârfuri. Conform Lemei 3.2, există un vârf terminal v în T . Atunci $T' = T - v$ este arbore cu $n - 1$ vârfuri (Lema 3.3) și $|E(T')| = |E(T)| - 1$. Din ipoteza de inducție, T' are $|E(T')| = |V(T')| - 1 = n - 2$ muchii. Rezultă $|E(T)| = |E(T')| + 1 = n - 1$. \square

Lema 3.5. *Fie G un graf neorientat conex și C un ciclu în G . Fie $e \in E(C)$ o muchie din ciclul C . Atunci $G - e$ este tot un graf conex.*

Proof. Afirmția rezultă din definiția unui graf conex și din următoarea observație: dintr-un $x - y$ lanț care conține muchia e în G se poate obține un $x - y$ lanț în $G - e$ înlocuind muchia e cu $C - e$. \square

Propoziția 3.6. *Fie T un graf neorientat cu $n \geq 1$ vârfuri. Următoarele afirmații sunt echivalente.*

1. T este arbore (conex și aciclic)
2. T este conex muchie-minimal (prin eliminarea unei muchii din T se obține un graf care nu mai este conex)
3. T este aciclic muchie-maximal
4. T este conex și are $n - 1$ muchii
5. T este aciclic și are $n - 1$ muchii

6. Între oricare două vârfuri din T există un unic lanț elementar.

Proof. " $1 \iff 2$ ": Pentru $n = 1$ este evident. Presupunem $n \geq 2$.

$1 \implies 2$: Presupunem că T este arbore (conex și aciclic).

Fie $e = xy \in E(T)$. Arătăm că $T - e$ nu este conex (deci T este conex muchie-minimal). Presupunem prin absurd că $T - e$ este conex. Atunci există un lanț elementar P în $T - e$ de la x la y . Atunci $P + xy = [x \overset{P}{-} y, x]$ este ciclu în T , contradicție.

$2 \iff 1$: Presupunem că T este conex muchie-minimal. Demonstrăm că T este aciclic.

Presupunem prin absurd că T conține un ciclu C . Fie $e \in E(C)$. Din Lema 3.5 rezultă că $T - e$ este conex, contradicție (T este conex muchie-minimal).

" $1 \iff i$ " pentru $i \in \{3, 4, 5, 6\}$ - Exerciții ([?] - v. Secțiunea 2.1, [?] - v. Secțiunea 4.1) □

Corolar 3.7. Orice graf conex conține un arbore parțial (un graf parțial care este arbore).

3.1 Construcția unui arbore cu secvența gradelor dată

Teorema 3.8. O secvență de $n \geq 2$ numere naturale **strict pozitive** $s_0 = \{d_1, \dots, d_n\}$ este secvența gradelor unui arbore dacă și numai dacă $d_1 + \dots + d_n = 2(n - 1)$.

Proof. " \implies " Presupunem că există un arbore T cu $s(T) = s_0$. Atunci

$$d_1 + \dots + d_n = 2|E(T)| = 2(n - 1) \text{ (conform Propoziției 3.6)}$$

" \impliedby " **Varianta 1.** Demonstrăm prin inducție după n că o secvență de n numere naturale (strict) pozitive $s_0 = \{d_1, \dots, d_n\}$ cu proprietatea că $d_1 + \dots + d_n = 2(n - 1)$ este secvența gradelor unui arbore.

Pentru $n = 2$ avem $d_1, d_2 > 0$ și $d_1 + d_2 = 2(2 - 1) = 2$, deci $d_1 = d_2 = 1$. Există un arbore cu secvența gradelor $s_0 = \{1, 1\}$, izomorf cu graful P_2 (lanț cu două vârfuri).

Presupunem afirmația adevărată pentru $n - 1$. Fie $s_0 = \{d_1, \dots, d_n\}$ o secvență de $n \geq 3$ numere naturale (strict) pozitive cu proprietatea că $d_1 + \dots + d_n = 2(n - 1)$. Arătăm că există un arbore cu secvența gradelor s_0 .

Presupunem (fără a restrânge generalitatea) că $d_1 \geq \dots \geq d_n$. Demonstrăm că $d_1 > 1$ și $d_n = 1$.

- Presupunem prin absurd că $d_1 = 1$. Atunci $d_1 = \dots = d_n = 1$. Rezultă $d_1 + \dots + d_n = n$, de unde $2(n - 1) = n$, deci $n = 2$, contradicție ($n \geq 3$).
- Presupunem prin absurd că $d_n \geq 2$. Atunci $d_1 \geq \dots \geq d_n \geq 2$. Rezultă $d_1 + \dots + d_n \geq 2n$, de unde $2(n - 1) \geq 2n$, contradicție.

Considerăm secvența $s'_0 = \{d_1 - 1, d_2, \dots, d_{n-1}\}$. Numerele din secvență sunt pozitive și avem

$$d_1 - 1 + d_2 + \dots + d_{n-1} = d_1 + \dots + d_n - (1 + d_n) = 2(n - 1) - (1 + 1) = 2((n - 1) - 1).$$

Atunci putem aplica ipoteza de inducție pentru secvența s'_0 . Rezultă că există un arbore T' cu mulțimea vârfurilor $V' = \{x_1, \dots, x_{n-1}\}$ având secvența gradelor $s(T') = s'_0$, mai exact cu:

$$d_{T'}(x_i) = \begin{cases} d_i - 1, & \text{dacă } i = 1 \\ d_i, & \text{dacă } i \in \{2, \dots, n-1\}. \end{cases}$$

Construim pornind de la T' un nou arbore $T = (V, E)$ adăugând un vârf nou x_n pe care îl unim cu vârful x_1 : $V = V' \cup \{x_n\}$, $E = E' \cup \{x_1x_n\}$. Pentru un $i \in \{1, \dots, n\}$ avem atunci

$$d_T(x_i) = \begin{cases} d_{T'}(x_i) + 1 = d_i - 1 + 1 = d_i, & \text{dacă } i = 1 \\ d_{T'}(x_i) = d_i, & \text{dacă } i \in \{2, \dots, n-1\} \\ 1 = d_n, & \text{dacă } i = n. \end{cases}$$

Rezultă că $s(T) = s_0$, deci s_0 este secvența gradelor unui arbore.

Varianta 2. Presupunem (fără a restrânge generalitatea) că $d_1 \geq \dots \geq d_n$. Avem $d_1 \geq 1$ și $d_n = 1$ (ca în varianta 1). Fie k astfel încât $d_k > 1$ și $d_{k+1} = 1$.

Construim un arbore T de tip omidă cu mulțimea vârfurilor $\{x_1, \dots, x_n\}$ având secvența gradelor s_0 astfel.

1. Unim printr-un lanț vârfurile x_1, \dots, x_k (care trebuie să fie neterminale - formează corpul omizii), în această ordine
2. Considerăm mulțimea de vârfuri $\{x_{k+1}, \dots, x_n\}$ (care trebuie să fie terminale) și unim
 - x_1 cu primele $d_1 - 1$ vârfuri din această mulțime
 - x_2 cu următoarele $d_2 - 2$ vârfuri
 - ...
 - x_{k-1} cu următoarele $d_{k-1} - 2$ vârfuri
 - x_k cu ultimele $d_k - 1$ vârfuri din această mulțime.

$$\text{Deoarece } d_1 - 1 + \sum_{i=2}^{k-1} (d_i - 2) + d_k - 1 = \sum_{i=1}^n d_i - 2k + 2 - (n - k) = n - k = |\{x_{k+1}, \dots, x_n\}|,$$

construcția este corectă.

Avem $s(T) = s_0$. □

Din demonstrația Teoremei 3.8 se desprind următorii algoritmi de determinare a unui arbore cu secvența gradelor dată.

Algoritm de construcție a unui arbore cu secvența de grade dată

Intrare: o secvență de n numere naturale pozitive d_1, \dots, d_n

Ieșire: un arbore T cu $V(T) = \{x_1, \dots, x_n\}$ cu $s(T) = s_0$ dacă s_0 este secvența gradelor unui arbore, sau mesajul NU altfel.

Idee: La un pas unim un vârf de grad 1 cu un vârf de grad mai mare decât 1 și actualizăm secvența s_0 . Se repetă de $n - 2$ ori, în final rămânând în secvență două vârfuri de grad 1, care se unesc printr-o muchie.

Pseudocod:

Varianta 1.

Pasul 1. Dacă $d_1 + \dots + d_n \neq 2(n - 1)$, atunci scrie NU, STOP.

Pasul 2.

Cât timp s_0 conține valori mai mari decât 1 execută //sau pentru $i = 1, n - 2$

alege un număr $d_k > 1$ și un număr $d_t = 1$ din secvență s_0 și adaugă la T muchia $x_k x_t$.

elimină d_t din s_0

înlocuiește d_k în secvența s_0 cu $d_k - 1$

Pasul 3.

fie d_k, d_t unicele elemente nenule (egale cu 1) din s_0 ; adaugă la T muchia $x_k x_t$.

Varianta 2. Corespunde variantei a doua de demonstrare a teoremei anterioare - construim un arbore de tip omidă.

4 Arbori parțiali de cost minim (apcm)

Pentru această secțiune se poate consulta cartea [?].

4.1 Algoritmul lui Kruskal

Intrare: $G = (V, E, w)$ - graf conex ponderat

Ieșire: un apcm T al lui G

Idee: La un pas este selectată o muchie de cost minim din G care nu formează cicluri cu muchiile deja selectate în T (care unește două componente conexe din graful deja construit)

Pseudocod:

$T = (V, \emptyset)$: inițial fiecare vârf formează o componentă conexă în T
pentru $i = 1, n - 1$ execută
 alege o muchie $e_i = uv$ de cost minim din G astfel încât u și v sunt în componente conexe
 diferite în T
 $E(T) = E(T) \cup \{uv\}$: reunește componenta lui u și componenta lui v

Exemplu. Complexitate. Detalii implementare: v slide-uri+laborator.

4.1.1 Clustering

Algoritm de determinare a unui k -clustering cu grad de separare minim [?]

Intrare:

- o mulțime de n obiecte $S = \{o_1, \dots, o_n\}$
- o funcție de distanță $d : S \times S \rightarrow \mathbb{R}_+$
- k un număr natural (numărul de clase)

Ieșire: un k -clustering (o partiționare \mathcal{C} a lui S în k clase) cu grad de separare maxim
($\text{sep}(\mathcal{C}) = \min\{d(o, o') \mid o, o' \in S, o \text{ și } o' \text{ sunt în clase diferite ale lui } \mathcal{C}\}$)

V. slide-uri pentru detalii si exemplu

Pseudocod:

Inițial fiecare obiect formează o clasă
pentru $i = 1, n - k$ execută
 alege două obiecte o_r, o_t din clase diferite cu $d(o_r, o_t)$ minimă
 reunește clasa lui o_r și clasa lui o_t
returnează cele k clase obținute

Asociem problemei un graf complet $G \sim K_n$, în care vârfurile corespund obiectelor, iar o muchie între două vârfuri o_i, o_j are ponderea $w(o_i o_j) = d(o_i, o_j)$.

Atunci algoritmul este echivalent cu **algoritmul lui Kruskal**, oprit când numărul de muchii selectate este $n - k$, mai exact are următorul pseudocod:

Pseudocod:

Inițial fiecare vârf formează o componentă conexă (clasă): $T' = (V, \emptyset)$

pentru $i = 1, n - k$ execută

alege o muchie $e_i = uv$ de cost minim din G astfel încât u și v sunt în componente conexe diferite în T'

reunește componenta lui u și componenta lui v : $E(T') = E(T') \cup \{uv\}$

returnează cele k mulțimi formate cu vârfurile celor k componente conexe ale lui T'

Observație 4.1. Algoritmul este echivalent cu următorul, mai general:

Pseudocod - variantă:

- determină un arbore parțial de cost minim T

- consideră mulțimea $\{e_{n-k+1}, \dots, e_{n-1}\}$ formată cu $k - 1$ muchii cu cele mai mari ponderi în T

- fie pădurea $T' = T - \{e_{n-k+1}, \dots, e_{n-1}\}$

- definește clasele k -clustering-ului ca fiind mulțimile vârfurilor celor k componente conexe ale pădurii astfel obținute

Propoziția 4.2. Corectitudine Algoritm Clustering

Algoritmul descris anterior determină un k -clustering cu grad de separare maxim.

Proof. La finalul algoritmului $E(T') = \{e_1, \dots, e_{n-k}\}$, cu $w(e_1) \leq \dots \leq w(e_{n-k})$, și T' este o pădure cu k componente conexe, vârfurile componentelor determinând clasele lui \mathcal{C} .

Avem

$$\text{sep}(\mathcal{C}) = \min\{w(e) \mid e = uv \in E(G) \text{ muchie din } G \text{ ce unește două componente conexe din } T'\}$$

Fie e_{n-k+1} următoarea muchie care ar fi fost selectată de algoritm dacă ar fi continuat cu $i = n - k + 1$, adică muchia de cost minim care unește două componente conexe din T' . Atunci $\text{sep}(\mathcal{C}) = w(e_{n-k+1})$.

Presupunem că există un alt k -clustering \mathcal{C}' cu $\text{sep}(\mathcal{C}') > \text{sep}(\mathcal{C})$. Atunci există două obiecte p, q care sunt în aceeași clasă în \mathcal{C} , dar sunt în două clase diferite C'_i și C'_j în \mathcal{C}' .

Deoarece p, q sunt în aceeași clasă a lui \mathcal{C} , atunci vârfurile p și q sunt în aceeași componentă conexă a lui T' , deci există P un lanț de la p la q în T' . Mai mult, deoarece p și q sunt în clase diferite ale lui \mathcal{C}' , există o muchie e a lui P cu o extremitate în C'_i și cealaltă în C'_j . Rezultă că $\text{sep}(\mathcal{C}') \leq w(e)$. Dar $e \in E(P) \subseteq E(T')$, deci $w(e) \leq w(e_{n-k}) \leq w(e_{n-k+1}) = \text{sep}(\mathcal{C})$, de unde obținem $\text{sep}(\mathcal{C}') \leq \text{sep}(\mathcal{C})$, contradicție.

□

4.2 Algoritmul lui Prim

Intrare: $G = (V, E, w)$ - graf conex ponderat

Ieșire: un apcm T al lui G

Idee: Se pornește de la un vârf s (care formează arborele inițial). La un pas este selectată o muchie de cost minim de la un vârf deja adăugat la arbore la unul neadăugat; se adaugă acest vârf la arbore împreună cu muchia selectată.

Pseudocod:

```
Fie  $s$ - vârful de start;  $T = (\{s\}, \emptyset)$ 
pentru  $i = 1, n - 1$  execută
    alege o muchie  $uv$  cu cost minim astfel încât  $u \in V(T)$  și  $v \notin V(T)$ 
     $E(T) = E(T) \cup \{uv\}$ 
     $V(T) = V(T) \cup \{v\}$ : adaugă  $v$  la arbore
```

Exemplu. Complexitate. Detalii implementare: v slide-uri+laborator.

Corectitudinea algoritmilor lui Kruskal și Prim

Notăție. Pentru o mulțime de muchii $A \subseteq E(G)$, scriem $A \subseteq \text{apcm}$ dacă există un apcm T al lui G astfel încât $A \subseteq E(T)$ (A poate fi extinsă până devine mulțimea muchiilor unui apcm).

Atât algoritmul lui Kruskal, cât și cel al lui Prim funcționează după următoarea schemă:

Schemă:

```
 $A = \emptyset$  (mulțimea muchiilor selectate în arborele construit)
pentru  $i = 1, n - 1$  execută
    alege o muchie  $e$  astfel încât  $A \cup \{e\} \subseteq \text{apcm}$ 
     $A = A \cup \{e\}$ 
returnează  $T = (V, A)$ 
```

Vom demonstra un criteriu de alegere a muchiei e la un pas astfel încât dacă $A \subseteq \text{apcm}$, atunci și $A \cup \{e\} \subseteq \text{apcm}$ și vom demonstra că algoritmiile lui Kruskal și Prim aplică acest criteriu.

Propoziția 4.3. Fie $G = (V, E, w)$ un graf conex ponderat.

Fie $A \subseteq E$ astfel încât $A \subseteq \text{apcm}$.

Fie $S \subset V$ astfel încât orice muchie din A are ambele extremități în S sau ambele extremități în $V - S$.

Fie e o muchie de cost minim cu o extremitate în S și cealaltă în $V - S$ (v.fig. 6).

Atunci $A \cup \{e\} \subseteq \text{apcm}$.

Proof. Fie T_{\min} un apcm astfel încât $A \subseteq E(T_{\min})$.

Dacă $e \in E(T_{\min})$, atunci $A \cup \{e\} \subseteq E(T_{\min})$.

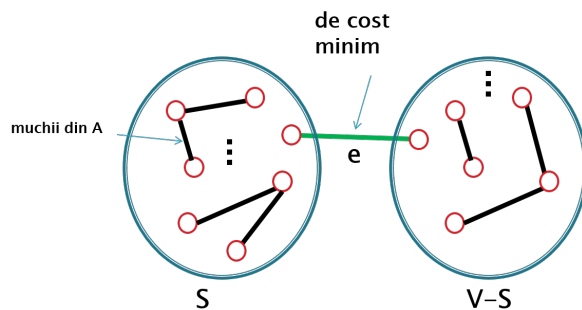


Figure 6

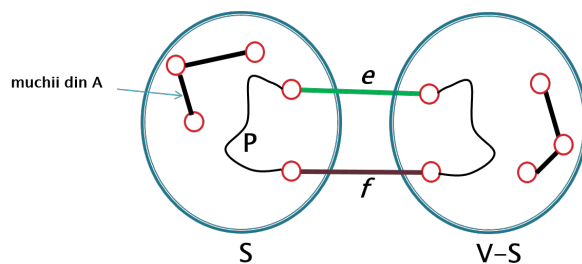


Figure 7

Dacă $e \notin E(T_{min})$, construim pornind de la T_{min} un alt apcm care să conțină și muchia e (și pe cele din A).

Notăm cu u și v extremitățile muchiei e , astfel încât $u \in S$ și $v \in V - S$.

Deoarece T_{min} este arbore, există un lanț elementar P de la u la v în T_{min} . Lanțul P are o extremitate în mulțimea S (extremitatea u) și cealaltă extremitate în mulțimea $V - S$ (extremitatea v). Rezultă că P conține o muchie f cu o extremitate în S și una în $V - S$ (v. fig. 7). Deoarece, conform ipotezei, A nu conține nicio muchie de la S la $V - S$, avem $f \notin A$.

Considerăm graful $T' = T_{min} + e - f$. Deoarece $T_{min} + e$ conține un unic ciclu elementar, format din lanțul P și muchia e , iar f aparține acestui ciclu, graful $T' = T_{min} + e - f$ este un graf conex și aciclic, deci un arbore.

Avem:

- $A \subseteq E(T')$, deoarece $f \notin A$
- $w(T') = w(T_{min}) + w(e) - w(f) \leq w(T_{min})$, deoarece, conform ipotezei muchia e are cel mai mic cost dintre muchiile de la S la $V - S$, deci $w(e) \leq w(f)$. Rezultă $w(T') = w(T_{min})$, deci T' este tot apcm.

Am obținut astfel un apcm T' cu $A \cup \{e\} \subseteq E(T')$, deci $A \cup \{e\} \subseteq \text{apcm}$.

□

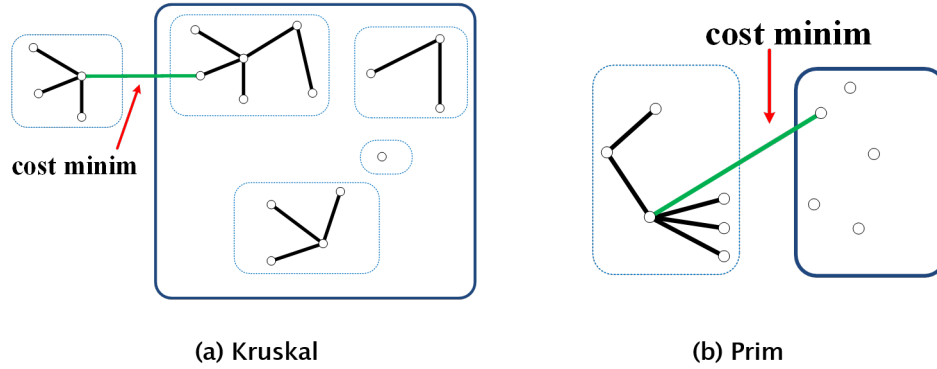


Figure 8

Teorema 4.4. - Corectitudinea algoritmului lui Kruskal. Fie $G = (V, E, w)$ un graf conex ponderat. Algoritmul lui Kruskal determină un arbore parțial de cost minim al lui G .

Proof. Fie $n = |V(G)|$. Algoritmul construiește un graf T conex cu $n - 1$ muchii (și aciclic), deci un arbore (la fiecare dintre cele $n - 1$ etape există o muchie în G care unește două componente din T , deoarece G este conex și T este graf parțial al lui G).

Folosim Propoziția 4.3 pentru a demonstra că după fiecare pas $i = \{1, \dots, n - 1\}$, avem $E(T) \subseteq \text{apcm}$. În final avem chiar egalitate, deoarece $|E(T)| = n - 1$, deci T este apcm.

Inițial $E(T) = \emptyset \subseteq \text{apcm}$.

Presupunem că înainte de etapa i avem $E(T) \subseteq \text{apcm}$.

Fie $e = uv$ muchia aleasă de algoritm la pasul i și S = componenta conexă a lui u în pădurea curentă T - v. fig. 8 (a). Atunci $E(T)$ și e verifică proprietățile din Propoziția 4.3 (toate muchiile din $E(T)$ sunt în interiorul componentelor conexe ale lui T , deci nu există muchie în $E(T)$ de la S la $V - S$).

Rezultă $E(T) \cup \{e\} \subseteq \text{apcm}$. □

Teorema 4.5. - Corectitudinea algoritmului lui Prim. Fie $G = (V, E, w)$ un graf conex ponderat. Algoritmul lui Prim determină un arbore parțial de cost minim al lui G .

Proof. Algoritmul construiește un graf T conex și aciclic, deci un arbore.

Ca și în demonstrația Teoremei 4.4, faptul că T este un apcm rezultă aplicând la fiecare pas Propoziția 4.3 pentru mulțimea $S = V(T)$ = mulțimea vârfurilor deja adăugate la T până la pasul curent - v. fig. 8 (b). □

5 Drumuri minime în grafuri orientate ponderate

5.1 Drumuri minime de sursă unică (de la un vârf dat la celelalte)

5.1.1 Algoritmul lui Dijkstra

Intrare: $G = (V, E, w)$ - graf orientat ponderat cu ponderi **pozitive** ($w : E \rightarrow \mathbb{R}_+^*$), s - vârf de start

Ieșire:

- vector d de distanțe, cu semnificația $d[x] =$ distanța de la s la vârful x
- vectorul $tata$, reprezentând un arbore al distanțelor față de s (din care se poate determina câte un drum minim de la s la fiecare vârf x)

Idee: Fiecare vârf u are asociată o etichetă de distanță $d[u] =$ costul minim al unui drum de la s la u descoperit până la acel pas = estimare superioară pentru distanța de la s la u .

La un pas este ales ca vârf curent vârful u care estimat a fi cel mai apropiat de s și se descoperă noi drumuri către vecinii lui u , actualizându-se etichetele de distanță ale acestora.

Pseudocod:

```
inițializează mulțimea vârfurilor nevizitate  $Q$  cu  $V$ 
pentru fiecare  $u \in V$  executa
     $d[u] = \infty$ ;  $tata[u] = 0$ 
 $d[s] = 0$ 
cat timp  $Q \neq \emptyset$  executa
     $u =$  extrage un vârf cu eticheta  $d$  minimă din  $Q$ 
    pentru fiecare  $uv \in E$  executa //relaxarea arcului  $uv$ 
        dacă  $d[u] + w(u, v) < d[v]$  atunci
             $d[v] = d[u] + w(u, v)$ 
             $tata[v] = u$ 
scrie  $d, tata$ 
//Variantă: scrie drum minim de la  $s$  la  $t$  un vârf  $t$  dat folosind  $tata$ 
```

Exemplu. Complexitate. Detalii implementare: v slide-uri+laborator.

Corectitudinea algoritmului lui Dijkstra

Următoarele observații generale privind un drum minim între două vârfuri sunt evidente, dar importante în demonstrarea corectitudinii algoritmilor de drumuri minime.

- Observație 5.1.**
1. Dacă P este un drum minim de la s la u , atunci P este drum elementar.
 2. Dacă P este un drum minim de la s la u și z este un vârf al lui P , atunci subdrumul lui P de la s la z este drum minim de la s la z .
Mai general, dacă x și y sunt două vârfuri din P , atunci subdrumul lui P de la x la y este drum minim de la x la y .

Lema 5.2. Pentru orice $u \in V$, la orice pas al algoritmului lui Dijkstra avem:

- (a) dacă $d[u] < \infty$, atunci există un drum de la s la u în G de cost $d[u]$ și acesta se poate determina din vectorul $tata$, mai exact $tata[u] = \text{predecesorul lui } u \text{ pe un drum de la } s \text{ la } u \text{ de cost } d[u]$.
- (b) $d[u] \geq \delta(s, u)$

Proof. a) Demonstrăm afirmația prin inducție după numărul de iterații (execuții ale ciclului "cat timp").

Inițial singura etichetă finită este $d[s] = 0$, corespunzătoare drumului $[s]$. La prima iterație este extras din Q vârful s și pentru el afirmația se verifică.

Presupunem afirmația adevărată până la un pas.

Fie $u \in Q$ vârful extras la iterația curentă. Fie v cu $uv \in E$ un vecin al lui u căruia i se modifică eticheta la acest pas prin relaxarea arcului uv . Avem:

- după relaxarea arcului uv : $d[v] = d[u] + w(uv)$; $tata[v] = u$
- din ipoteza de inducție există P un drum de la s la u de cost $d[u]$.

Atunci drumul $R = [s \xrightarrow{P} u, v]$ este un drum de la s la v de cost

$$w(R) = w(P) + w(uv) = d[u] + w(uv) = d[v]$$

și $u = tata[v]$ este predecesorul lui v pe acest drum.

□

Corolar 5.3. Dacă la un pas al algoritmului pentru un vârf u avem relația $d[u] = \delta(s, u)$, atunci $d[u]$ nu se mai modifică până la final.

Proof. Afirmația rezultă din Lema anterioară (nu există drum de la s la u cu cost mai mic decât cel minim, adică decât $\delta(s, u)$). □

Teorema 5.4. Fie $G = (V, E, w)$ un graf orientat ponderat cu $w : E \rightarrow \mathbb{R}_+^*$ și $s \in V$ fixat.

La finalul algoritmului lui Dijkstra avem:

$$d[u] = \delta(s, u) \text{ pentru orice } u \in V$$

și $tata$ memorează un arbore al distanțelor față de s .

Proof. Vom demonstra prin inducție după numărul de iterații (execuții ale ciclului "cat timp") că după fiecare iterație avem $d[x] = \delta(s, x)$ pentru orice $x \in V - Q$ (eticheta vârfurilor deja selectate este corect calculată).

Inițial avem $d[s] = \delta(s, s)$ și primul vârf extras din Q este s , deci afirmația se verifică după prima iterație.

Presupunem că afirmația este adevărată până la iterația curentă: $d[x] = \delta(s, x)$ pentru orice $x \in V - Q$.

Fie $u \in Q$ vârful extras la această iterație. Demonstrăm că $d[u] = \delta(s, u)$ (adică eticheta lui u este corect calculată, fiind egală cu distanța de la s la u).

Dacă u nu este accesibil din s atunci, din Lema 5.2, $d[u] = \infty = \delta(s, u)$.

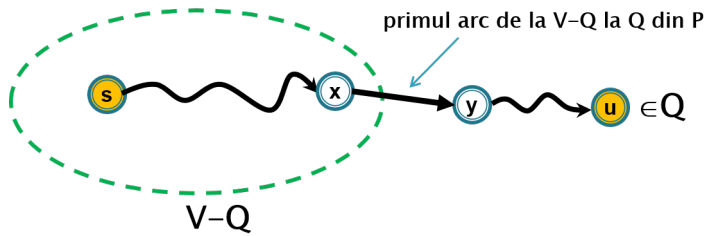


Figure 9

Altfel, fie P un drum minim de la s la u (avem $w(P) = \delta(s, u)$). Deoarece P are o extremitate $s \in V - Q$ și cealaltă extremitate $u \in Q$, rezultă că P conține un arc cu o extremitate în $V - Q$ și cealaltă în Q . Fie xy primul astfel de arc din P (cu $x \in V - Q$ și $y \in Q$) - v. fig. 9.

Din ipoteza de inducție și Observația 5.1 avem

$$d[x] = \delta(s, x) = w([s \overset{P}{-} x]).$$

La iterația la care a fost extras din Q vârful x , după relaxarea arcului xy avem:

$$d[y] \leq d[x] + w(xy) = w([s \overset{P}{-} x]) + w(xy) = w([s \overset{P}{-} y]).$$

Deoarece arcele au capacități pozitive, avem $w([s \overset{P}{-} y]) \leq w(P)$ și deci

$$d[y] \leq w(P) = \delta(s, u) \leq d[u].$$

Dar, deoarece la pasul curent și u și y sunt în Q , din modul în care algoritmul alege vârful u avem

$$d[u] \leq d[y].$$

Rezultă că $d[u] = d[y] = w(P) = \delta(s, u)$ (și P are vârfuri interne doar din mulțimea vârfurilor deja selectate).

Din Lema 5.2, pentru fiecare u , $d[u]$ este ponderea drumului de la s la u memorat în vectorul $tata$, deci acesta corespunde unui arbore al distanțelor față de s .

□

5.1.2 Drumuri minime în grafuri fără circuite (DAGs= Directed Acyclic Graphs)

Intrare: $G = (V, E, w)$ - graf orientat ponderat (ponderi reale) **fără circuite**, s - vârf de start

Ieșire:

- vector d de distanțe, cu semnificația $d[x] = \text{distanța de la } s \text{ la vârful } x$
- vectorul $tata$, reprezentând un arbore al distanțelor față de s (din care se poate reconstrui câte un drum minim de la s la fiecare vârf x)

Idee: Fiecare vârf u are asociată o etichetă de distanță $d[u]$ = costul minim al unui drum de la s la u descoperit până la acel pas (ca și la Dijkstra).

Etichetele vârfurilor sunt calculate în ordinea dată de sortarea topologică. La un pas sunt relaxate toate arcele care ies din vârful curent.

Pseudocod:

```

pentru fiecare  $u \in V$  executa
     $d[u] = \infty$ ;  $tata[u] = 0$ 
 $d[s] = 0$ 
 $SortTop = \text{sortare-topologica}(G)$ 
pentru fiecare  $u \in SortTop$ 
    pentru fiecare  $uv \in E$  executa
        daca  $d[u] + w(u, v) < d[v]$  atunci //relaxam  $uv$ 
             $d[v] = d[u] + w(u, v)$ 
             $tata[v] = u$ 
scrie  $d$ ,  $tata$ 

```

Exemplu. Complexitate. Detalii implementare: v slide-uri+laborator.

Corectitudine.

Are loc un rezultat similar celui pentru algoritmul lui Dijkstra

Lema 5.5. Pentru orice $u \in V$, la orice pas al algoritmului de drumuri minime în grafuri fără circuite are loc proprietatea:

- (a) dacă $d[u] < \infty$, atunci există un drum de la s la u în G de cost $d[u]$ și acesta se poate determina din vectorul $tata$, mai exact $tata[u] = \text{predecesorul lui } u \text{ pe un drum de la } s \text{ la } u \text{ de cost } d[u]$.
- (b) $d[u] \geq \delta(s, u)$

Teorema 5.6. Fie $G = (V, E, w)$ un graf orientat ponderat fără circuite (DAG) și $s \in V$ fixat.

La finalul algoritmului de drumuri minime în grafuri fără circuite avem:

$$d[u] = \delta(s, u) \text{ pentru orice } u \in V$$

și $tata$ memorează un arbore al distanțelor față de s .

Proof. Presupunem că toate vârfurile sunt accesibile din s . Altfel, eticheta lor va rămâne ∞ pe tot cursul algoritmului, deci este corect calculată.

Vom demonstra prin inducție după numărul de iterații (execuții ale ciclului "pentru fiecare $u \in SortTop$ ") că la fiecare iterație avem $d[x] = \delta(s, x)$ pentru orice x deja considerat (aflat în ordonare topologică înaintea lui u).

Inițial avem $d[s] = 0 = \delta(s, s)$, deci afirmația se verifică pentru prima iterație.

Presupunem că afirmația este adevărată până la iterația curentă. Fie $u \in SortTop$ vârful considerat la această iterație. Demonstrăm că $d[u] = \delta(s, u)$ (adică eticheta lui u este corect calculată, fiind egală cu distanța de la s la u).

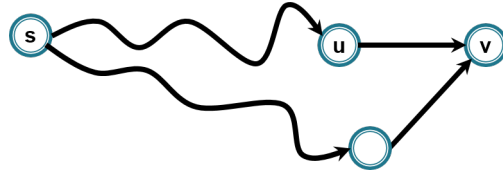


Figure 10

Variantă 1. Are loc relația (v. fig. 10) :

$$\delta(s, u) = \min\{\delta(s, x) + w(xu) | xu \in E\}, \forall u \neq s.$$

La pasul curent eticheta $d[u]$ calculată de algoritmul prin relaxări de arce este

$$d[u] = \min\{d[x] + w(xu) | x \text{ înaintea lui } u \text{ în ordonarea topologică}\}.$$

Deoarece dacă $xu \in E$, x este înaintea lui u în ordonarea topologică, atunci rezultă:

$$d[u] = \min\{d[x] + w(xu) | xu \in E\}$$

și, din ipoteza de inducție, $d[x] = \delta(s, x)$. Obținem

$$d[u] = \min\{d[x] + w(xu) | xu \in E\} = \min\{\delta(s, x) + w(xu) | xu \in E\} = \delta(s, u).$$

Varianta 2 (similar Dijkstra).

Fie P un drum minim de la s la u (avem $w(P) = \delta(s, u)$) și x predecesorul lui u pe acest drum. Atunci x este înaintea lui u în ordonarea topologică deci, din ipoteza de inducție și Observația 5.1 avem

$$d[x] = \delta(s, x) = w([s \overset{P}{-} x]).$$

La iterația la care a fost considerat x , după relaxarea arcului xu avem:

$$d[u] \leq d[x] + w(xu) = w([s \overset{P}{-} x]) + w(xu) = w([s \overset{P}{-} u]) = \delta(s, u).$$

Din Lema 5.5, pentru fiecare u $d[u]$ este ponderea drumului de la s la u memorat în vectorul $tata$, deci acesta corespunde unui arbore al distanțelor față de s . □

5.2 Drumuri minime între oricare două vârfuri

V. slide

6 Fluxuri în rețele de transport

6.1 Noțiuni introductive

Definiție 6.1. O rețea de transport este un cvintuplu $N = (G, S, T, I, c)$ unde

- $G = (V, E)$ este un graf orientat cu $V = S \cup I \cup T$, unde S, I, T sunt mulțimi disjuncte, nevide, cu semnificația:
 - S - mulțimea surselor (intrărilor)
 - T - mulțimea destinațiilor (ieșirilor)
 - I - mulțimea vârfurilor intermediare
- $c : E \longrightarrow \mathbb{N}$ este funcția capacitate, cu semnificația: $c(e) =$ cantitatea maximă care poate fi transportată prin arcul e .

În cele ce urmează vom considera doar rețele cu următoarele proprietăți

- $S = \{s\}$ - sursă unică (notată s)
- $T = \{t\}$ - destinație unică (notată t)
- $d^-(s) = 0$ - în sursă nu intră arce
- $d^+(t) = 0$ - din destinație nu ies arce

(oricărei rețele de transport i se poate asocia o rețea de transport care verifică aceste ipoteze, echivalentă din punct de vedere al valorii fluxului, prin adăugarea unei surse noi și a unei destinații noi - v. fig 11).

În plus, vom presupune că orice vârf al lui G este accesibil din s .

În continuare vom nota o rețea de transport $N = (G, s, t, I, c)$.

Notății. Fie rețeaua $N = (G, s, t, I, c)$, X și Y două mulțimi de vârfuri și o funcție $f : E \longrightarrow \mathbb{N}$. Notăm:

- $f(X, Y) = \sum_{xy \in E, x \in X, y \in Y} f(xy)$
- $f^+(X) = f(X, V - X)$
- $f^-(X) = f(V - X, X)$.
- Pentru un vârf v notăm $f^+(v) = f^+(\{v\})$ și $f^-(v) = f^-(\{v\})$.

Definiție 6.2. Fie o rețea de transport $N = (G, s, t, I, c)$ (cu proprietățile amintite: $d^-(s) = d^+(t) = 0$).

Un $s - t$ flux în N (numit în continuare și doar **flux**) este o funcție $f : E \longrightarrow \mathbb{N}$ cu proprietățile:

1. $0 \leq f(e) \leq c(e), \forall e \in E(G)$ - **condiția de mărginire**
2. pentru orice vârf intermediar $v \in I$, fluxul (total) care intră în vârful v este egal cu fluxul (total) care iese din vârful v - **condiția de conservare a fluxului**:

$$\sum_{uv \in E} f(uv) = \sum_{vu \in E} f(vu),$$

sau, cu notațiile anterioare,

$$f^-(v) = f^+(v), \forall v \in I.$$

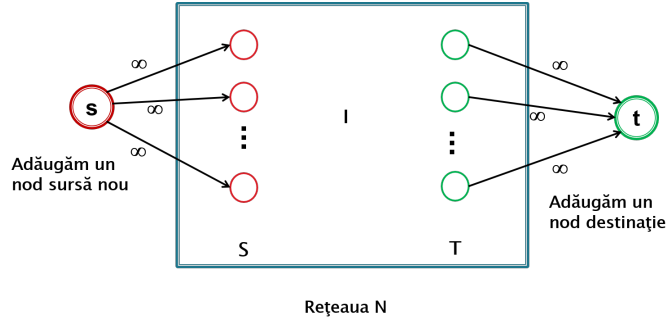


Figure 11

Valoarea fluxului f , notată $val(f)$, se definește ca fiind suma fluxului pe arcele care ies din sursa s :

$$val(f) = \sum_{sv \in E} f(sv) = f^+(s).$$

Vom demonstra ulterior că are loc relația

$$val(f) = f^+(s) = f^-(t).$$

Observație 6.3. În orice rețea există cel puțin un flux, și anume fluxul vid $f = 0$.

Definiție 6.4. Un flux f^* în rețeaua N se numește **flux maxim** sau **flux de valoare maximă**, dacă:

$$val(f^*) = \max\{val(f) | f \text{ flux în } N\}.$$

Problema fluxului maxim MAX-FLOW. Dată o rețea de transport $N = (G, s, t, I, c)$, să se determine un flux maxim (de valoare maximă) în N .

O problemă legată de problema fluxului maxim este problema determinării unei tăieturi minime.

Definiție 6.5. O $s - t$ **tăietură** K în rețeaua N (numită în cele ce urmează doar **tăietură**) este o partiționare a mulțimii vârfurilor V în două submulțimi X și $Y = V - X$ (disjuncte) astfel încât $s \in X$ și $t \in Y$ și se notează $K = (X, Y)$. Un arc cu o extremitate în X și cealaltă în Y (de la X la Y) se numește **arc** (sau **arc direct**) al tăieturii $K = (X, Y)$, iar un arc de la Y la X se numește **arc invers** pentru tăietura $K = (X, Y)$.

Capacitatea unei tăieturi $K = (X, Y)$, notată cu $c(K)$, se definește ca fiind suma capacităților arcelor directe ale tăieturii (=suma capacităților arcelor care ies din X):

$$c(K) = c^+(X) = \sum_{xy \in E, x \in X, y \in Y} c(xy).$$

O tăietură \tilde{K} se numește **tăietură minimă (de capacitate minimă)** în N dacă

$$c(\tilde{K}) = \min\{c(K) | K \text{ este } s - t \text{ tăietură în } N\}$$

Problema tăieturii minime MIN-CUT. Dată o rețea de transport $N = (G, s, t, I, c)$, să se determine o tăietură minimă (de capacitate minimă) în N .

Vom demonstra că pentru orice flux f și orice tăietură K în N are loc relația:

$$val(f) \leq c(K).$$

Va rezulta că, dacă are loc egalitate în această relație, f este flux maxim și K este tăietură minimă. Astfel, algoritmul Ford-Fulkerson pentru problema fluxului maxim va găsi și o soluție pentru problema tăieturii minime.

Pentru a prezenta algoritmul Ford-Fulkerson de determinare a unui flux maxim, vom defini noțiunile necesare.

Fie N o rețea de transport și f un flux în N .

- Un lanț într-un graf orientat G corespunde unui lanț în graful neorientat asociat acestuia (ignorând orientarea arcelor), mai exact este o succesiune de vârfuri și arce

$$P = [v_0, e_1, v_1, \dots, v_{k-1}, e_k, v_k]$$

în care $e_i \in E(G)$ și fie $e_i = v_{i-1}v_i$, fie $e_i = v_i v_{i-1}$ pentru fiecare $i \in \{1, \dots, k\}$. Dacă $e_i = v_{i-1}v_i$, arcul e_i se numește **arc direct** sau **arc înainte** în P , altfel se numește **arc invers** sau **înapoi**.

- Fie P un $s - v$ lanț în N (în G), cu $v \in V$:

$$P = [s = v_0, e_1, v_1, \dots, v_{k-1}, e_k, v_k = v]$$

Asociem fiecărui arc e din P o capacitate $i_P(e)$, numită **capacitate reziduală în raport cu P** , astfel:

$$i_P(e) = \begin{cases} c(e) - f(e), & \text{dacă } e \text{ este arc direct în } P \\ f(e), & \text{dacă } e \text{ este arc invers în } P. \end{cases}$$

($i_P(e)$) care reprezintă cantitatea maximă cu care se poate modifica fluxul f pe arcul e dacă folosim pentru revizuirea fluxului lanțul P).

- **Capacitatea reziduală $i(P)$** a unui $s - v$ lanț P se definește ca fiind

$$i(P) = \min\{i_P(e) | e \in E(P)\}$$

($i(P)$) reprezintă cantitatea maximă cu care se poate revizui fluxul f de-a lungul lanțului P).

Prin convenție, dacă P are un singur vârf, $i(P) = \min(\emptyset) = \infty$.

6.2 Revizuirea fluxului

Fie f un flux în rețeaua N .

Un $s - t$ lanț cu $i(P) > 0$ se numește lanț f -nesaturat sau **f -drum de creștere** (*augmenting path*).

Fie P un $s - t$ lanț f -nesaturat. **Fluxul revizuit de-a lungul lanțului P** se definește ca fiind funcția $f_P : E \rightarrow \mathbb{N}$,

$$f_P(e) = \begin{cases} f(e) + i(P), & \text{dacă } e \text{ este arc direct al lui } P \\ f(e) - i(P), & \text{dacă } e \text{ este arc invers al lui } P \\ f(e), & \text{altfel} \end{cases}$$

Propoziția 6.6. *Fluxul revizuit f_P este flux în N și*

$$\text{val}(f_P) = \text{val}(f) + i(P).$$

În plus, lanțul P este f_P -saturat.

Pentru $e = uv$ notăm cu $e^{-1} = vu$ inversul lui e (obținut din e inversând orientarea).

În loc de a considera lanțuri cu arce inverse în rețeaua N (în graful G), putem gestiona capacitățile reziduale ale arcelor asociind rețelei și fluxului f un graf numit graf rezidual, care conține pe lângă arce din G și arce obținute din cele din G inversând orientarea (inverse ale arcelor din G).

Mai exact, **(multi)graful rezidual G_f** asociat fluxului f din N este un graf orientat ponderat, cu funcția de pondere (de capacitate) c_f , construit astfel:

- $V(G_f) = V(G)$
- Pentru fiecare arc $e = uv$ cu $c(e) - f(e) > 0$, adăugăm arcul e la G_f cu ponderea asociată $c_f(e) = c(e) - f(e)$
- Pentru fiecare arc $e = uv$ cu $f(e) > 0$, adăugăm la G_f arcul $e^{-1} = vu$ (inversul arcului e) cu ponderea asociată $c_f(e^{-1}) = f(e)$.

Capacitatea unui drum P în G_f se definește ca fiind

$$c_f(P) = \min\{c_f(e) | e \in E(P)\}.$$

Notăm $E^{-1}(G_f) = \{e \in E(G) | e^{-1} \in E(G_f)\}$.

Observație 6.7. *O succesiune de vârfuri $P = [v_1, \dots, v_k]$ este un $s - t$ lanț f -nesaturat în G dacă și numai dacă este $s - t$ drum în G_f ; în plus, capacitatea reziduală $i(P)$ a lanțului P în G este egală cu capacitatea $c_f(P)$ a drumului P în G_f .*

Un arc e este arc invers în lanțul P în $G \iff e^{-1}$ este arc în drumul P din G_f .

Vom demonstra că un flux este maxim dacă și numai dacă nu există un $s - t$ lanț f -nesaturat în G (\iff nu există $s - t$ drum în G_f), de aici rezultând și corectitudinea algoritmului Ford-Fulkerson de determinare a unui flux maxim într-o rețea de transport.

6.3 Algoritmul Ford-Fulkerson

Intrare: $N = (G, s, t, I, c)$ - rețea de transport cu capacități numere naturale

Ieșire: un flux maxim f în G (+ o $s - t$ tăietură minimă în G)

Idee: La un pas este revizuit fluxul curent pe un $s-t$ lanț nesaturat (drum de creștere). Se repetă revizuirea până când nu mai există un astfel de lanț.

Pseudocod:

1. Fie f un flux în N (spre exemplu $f = 0$ fluxul vid: $f(e) = 0, \forall e \in E$)
2. Cât timp există un $s-t$ lanț f -nesaturat P în G (\iff există un $s-t$ drum în G_f)
determină un astfel de lanț P în G (\iff un $s-t$ drum în G_f)
revizuiește fluxul f de-a lungul lanțului P
3. Fie $X =$ mulțimea vârfurilor accesibile din s prin lanțuri f -nesaturate
returnează f , $K = (X, V - X)$

Complexitate Determinarea unui lanț - complexitate $O(m)$ (parcursare)

La fiecare pas $val(f)$ crește cu $i(P) \geq 1$.

Deoarece $val(f)$ este mai mică sau egală cu capacitatea oricărei tăieturi, putem determina ordinul de complexitate în funcție de capacitatea tăieturii minime, notată $L \implies O(mL)$.

Putem determina limite superioare pentru L : $L \leq c^+(s)$ sau $L \leq nU$ unde C este capacitatea maximă a unui arc ($O(nmC)$)

6.4 Corectitudinea Algoritmului Ford-Fulkerson

Lema 6.8. Fie $N = (G, s, t, I, c)$ o rețea de transport, f flux în N și $K = (X, Y = V - X)$ o tăietură în N . Are loc relația

$$val(f) = f^+(X) - f^-(X) = f^-(Y) - f^+(Y).$$

Proof. Pentru $x \in X - \{s\}$, din condiția de conservare a fluxului avem

$$f^+(x) - f^-(x) = 0.$$

Din definiția valorii fluxului și deoarece $f^-(s) = 0$ (deoarece în s nu intră arce), avem

$$f^+(s) - f^-(s) = val(f).$$

Adunând relațiile obținem

$$val(f) = \sum_{x \in X} (f^+(x) - f^-(x)).$$

Dar

$$\sum_{x \in X} f^+(x) = f(X, V) = f(X, V - X) + f(X, X) = f^+(X) + f(X, X).$$

Analog,

$$\sum_{x \in X} f^-(x) = f(V, X) = f^-(X) + f(X, X).$$

Rezultă

$$val(f) = f^+(X) + f(X, X) - (f^-(X) + f(X, X)) = f^+(X) - f^-(X).$$

În plus, deoarece $Y = V - X$, avem

$$f^+(X) = f(X, V - X) = f(X, Y) = f(V - Y, Y) = f^-(Y)$$

și, analog,

$$f^-(X) = f^+(Y)$$

Rezultă astfel că

$$val(f) = f^+(X) - f^-(X) = f^-(Y) - f^+(Y)$$

□

Corolar 6.9. Fie $N = (G, s, t, I, c)$ o rețea de transport și f flux în N . Are loc relația

$$val(f) = f^+(s) = f^-(t).$$

Proof. Considerăm în Lema 6.8 tăietura $K = (X, Y)$ cu $X = V - \{t\}$, $Y = \{t\}$. Rezultă $val(f) = f^-(t) - f^+(t) = f^-(t)$ (din t nu ies arce). □

Propoziția 6.10. Fie $N = (G, s, t, I, c)$ o rețea de transport, f flux în N și $K = (X, Y = V - X)$ o tăietură în N .

a) Are loc relația

$$val(f) \leq c(K)$$

cu egalitate dacă și numai dacă $f^+(X) = c^+(X)$ (toate arcele directe din K au fluxul egal cu capacitatea) și $f^-(X) = 0$ (toate arcele inverse pentru K au fluxul 0).

b) Dacă are loc egalitatea $val(f) = c(K)$, atunci f este flux maxim și K este tăietură minimă.

Proof. a) Din Lema 6.8 și condiția de mărginire pentru f avem:

$$val(f) = f^+(X) - f^-(X).$$

Dar $f^+(X) \leq c^+(X) = c(K)$ și $f^-(X) \geq 0$. Rezultă

$$val(f) \leq c(K) - 0 = c(K)$$

cu egalitate doar dacă $f^+(X) = c^+(X)$ și $f^-(X) = 0$.

b) Presupunem $val(f) = c(K)$.

Fie f^* flux maxim și \tilde{K} tăietură minimă. Din definițiile fluxului maxim/ tăieturii minime rezultă relațiile

$$val(f) \leq val(f^*) \leq c(\tilde{K}) \leq c(K) = val(f),$$

deci toate relațiile sunt de egalitate: $val(f) = val(f^*)$ și $c(\tilde{K}) = c(K)$. Rezultă că f are valoare maximă și K are capacitate minimă. □

Propoziția 6.11. (Tăietura minimă asociată unui flux maxim) Fie $N = (G, s, t, I, c)$ o rețea de transport, f flux în N .

Dacă nu există un $s - t$ lanț f -nesaturat în N , atunci există în N o tăietură K_f cu $\text{val}(f) = c(K_f)$ și, mai mult, f este flux maxim și K_f este tăietură minimă.

Proof. Fie $X = \{x \in V \mid \exists s - x \text{ lanț } f\text{-nesaturat în } G\}$
(sau, echivalent, $X = \{x \in V \mid \exists s - x \text{ drum în } G_f\}$). Fie $K = (X, V - X)$.

Avem $s \in X$ și $t \notin X$ (deoarece, conform ipotezei, nu există $s - t$ lanț f -nesaturat în G), deci K_f este o $s - t$ tăietură.

Din Lema 6.8 avem

$$\text{val}(f) = f^+(X) - f^-(X).$$

Arătăm că avem $f^+(X) = c^+(X)$ (adică $f(e) = c(e)$ pentru orice arc e de la X la $Y = V - X$) și $f^-(X) = 0$ (adică $f(e) = 0$ pentru orice arc e de la $Y = V - X$ la X). Rezultă atunci că

$$\text{val}(f) = c^+(X) = c(K).$$

Fie $e = xy$ cu $x \in X$ și $y \in V - X$ un arc direct în K . Din definiția lui X , rezultă că există Q un $s - x$ lanț f -nesaturat în G : $i(Q) > 0$. Considerăm $s - y$ lanțul $P = [s \overset{Q}{-} x, e, y]$. Deoarece $y \notin X$, avem $i(P) = 0 = \min\{i(Q), i_P(e)\}$. Dar $i(Q) > 0$ și e este arc direct în P , deci $i_P(e) = c(e) - f(e) = 0$. Rezultă $f(e) = c(e)$.

Fie $e = yx$ cu $x \in X$ și $y \in V - X$ un arc invers în K . Raționând analog, avem lanțul $P = [s \overset{Q}{-} x, e, y]$ în care e este acum arc invers. Obținem $i_P(e) = f(e) = 0$

□

Din demonstrația Propoziției anterioare rezultă următorul algoritm de determinare a unei tăieturi minime dat un flux maxim.

Algoritm de determinare a unei tăieturi minime dat un flux maxim

Intrare: $N = (G, s, t, I, c)$, f - flux maxim

Ieșire: o tăietură minimă $K = (X, Y)$ în N

Idee: X = mulțimea vârfurilor x accesibile din s prin $s - x$ lanțuri f -nesaturate. Aceste vârfuri se determină prin parcurgerea grafului G pornind din vârful s și considerând doar arce cu capacitatea reziduală pozitivă (în raport cu lanțurile construite prin parcurgere, memorate cu vectorul $tata$) - v. alg Ford-Fulkerson.

Observație 6.12. În finalul algoritmului Ford-Fulkerson mulțimea X care definește tăietura minimă $K = (X, Y)$ este chiar mulțimea vârfurilor vizitate la ultima parcurgere a grafului (în care nu s-a mai gasit un $s - t$ lanț f -nesaturat).

Teorema 6.13. (Caracterizarea unui flux maxim) Fie $N = (G, s, t, I, c)$ o rețea de transport, f flux în N . Are loc echivalența

f este flux maxim \iff nu există $s - t$ lanț f -nesaturat în G .

Proof. " \implies " Presupunem că f este flux maxim.

Presupunem prin absurd că există P un $s-t$ lanț f -nesaturat în G . Fie f_P fluxul obținut prin revizuirea lui f de-a lungul lui P . Conform Propoziției 6.6 avem:

$$val(f_P) = val(f) + i(P) > val(f),$$

ceea ce contrazice faptul că f este flux maxim.

" \Leftarrow " Presupunem că nu există $s-t$ lanț f -nesaturat în G . Atunci, conform Propoziției 6.11, există o tăietură K cu $val(f) = c(K)$ și f este flux maxim. \square

Teorema 6.14. (Corectitudinea Algoritmului Ford-Fulkerson) Algoritmului Ford-Fulkerson se termină într-un număr finit de pași și fluxul f determinat de algoritmul este flux maxim. În plus, mulțimea X a vârfurilor accesibile din s prin lanțuri f -nesaturate determină o tăietură $K = (X, V - X)$ de capacitate minimă.

Proof. Fie L capacitatea minimă a unei $s-t$ tăieturi în N . Fie f un flux în N . Avem

$$val(f) \leq L.$$

Conform Propoziției 6.6, dacă P este un $s-t$ lanț f -nesaturat, avem $val(f_P) = val(f) + i(P)$. Dar, deoarece $i(P) > 0$ și $i(P) \in \mathbb{N}$, avem $i(P) \geq 1$ și deci

$$val(f_P) \geq val(f) + 1.$$

Rezultă că la fiecare iterație a algoritmului Ford-Fulkerson valoarea fluxului crește cu cel puțin 1. Deoarece această valoare este mai mică sau egală cu L , rezultă că algoritmul are cel mult L etape.

În final fluxul f obținut verifică proprietatea: nu există $s-t$ lanț f -nesaturat, deci, conform Propoziției 6.13, este flux maxim. \square

Teorema 6.15. (Teorema Ford-Fulkerson / MAX-FLOW, MIN-CUT Theorem) Într-o rețea de transport $N = (G, s, t, I, c)$ valoarea unui $s-t$ flux maxim este egală cu capacitatea unei $s-t$ tăieturi minime.

Proof. Fie f^* flux maxim. Din Propoziția 6.13 rezultă că nu există $s-t$ lanț f^* -nesaturat. Atunci, conform Propoziției 6.11, există o tăietură minimă \tilde{K} cu $val(f^*) = c(\tilde{K})$. \square

6.5 Aplicații - probleme care se reduc la determinarea unui flux maxim

6.5.1 Cuplaje în grafuri bipartite

Fie $G = (V, E)$ un graf neorientat. Un **cuplaj** în G este o mulțime de muchii $M \subseteq E$ neadiacente două câte două.

Un graf neorientat $G = (V, E)$ se numește **bipartit** dacă există o partiție a lui V în două submulțimi V_1, V_2 (bipartiție):

$$\begin{aligned} V &= V_1 \cup V_2 \\ V_1 \cap V_2 &= \emptyset \end{aligned}$$

astfel încât orice muchie $e \in E$ are o extremitate în V_1 și cealaltă în V_2 .

Notăm atunci $G = (V_1 \dot{\cup} V_2, E)$.

Propoziția 6.16. Definiție echivalență a unui graf bipartit *Un graf neorientat $G = (V, E)$ este bipartit dacă și numai dacă există o 2-colorare proprie a vârfurilor lui G (bicolorare):*

$$c : V \longrightarrow \{1, 2\}$$

(adică astfel încât pentru orice muchie $e = xy \in E$ avem $c(x) \neq c(y)$)

Proof. " \implies " Dacă $G = (V_1 \dot{\cup} V_2, E)$, colorăm cu 1 vârfurile din V_1 și cu 2 vârfurile din V_2 . " \impliedby " Considerăm $V_1 = c^{-1}(\{1\})$ mulțimea vârfurilor de culoare 1 și $V_2 = c^{-1}(\{2\})$. Atunci (V_1, V_2) este o bipartiție pentru G . \square

Teorema 6.17. Teorema lui König de caracterizare a grafurilor bipartite *Fie $G = (V, E)$ un graf cu $n \geq 2$ vârfuri. Avem G este bipartit \Leftrightarrow toate ciclurile elementare din G sunt pare (nu are cicluri elementare impare)*

Proof. " \implies " Presupunem că G este bipartit. Atunci există $c : V \longrightarrow \{1, 2\}$ o bicolorare a lui G . Presupunem prin absurd că G conține un ciclu impar $C = [v_1, \dots, v_{2k+1}, v_1]$. Atunci c este bicolorare pentru C și avem $c(v_i) \neq c(v_{i+1})$. Rezultă

$$c(v_1) = c(v_3) = \dots = c(v_{2k+1})$$

și

$$c(v_2) = c(v_4) = \dots = c(v_{2k})$$

Dar atunci $c(v_{2k+1}) = c(v_1)$ și $v_{2k+1}v_1 \in E(C) \subseteq E(G)$, contradicție.

" \impliedby " Presupunem că toate ciclurile elementare din G sunt pare.

Presupunem fără a restrânge generalitatea că G este conex. Fie T un arbore parțial al lui G . Arborele T este un graf bipartit, o colorare proprie c a lui T cu 2 culori putând fi obținută astfel: fixăm o rădăcină în T , colorăm vârfurile de pe niveluri pare cu 1 și cele de pe niveluri impare cu 2.

Arătăm că 2-colorarea proprie c a lui T este colorare proprie și pentru G .

Fie $e = xy \in E(G) - E(T)$. Fie P unicul $x - y$ lanț elementar de la x la y în T . Atunci $P + e$ este ciclu elementar în G , deci este par. Rezultă că P are lungime impară ($P = [x = v_1, \dots, v_{2k} = y]$), și deci, deoarece c este bicolorare pentru T (deci și pentru P), $c(x) \neq c(y)$. \square

Observație 6.18. Algoritm de verificare dacă un graf este bipartit. *Demonstrația implicației " \Leftarrow " furnizează un algoritm pentru a testa dacă un graf este bipartit și, în caz afirmativ, pentru a determina o bicolorare a sa.*

Astfel, se colorează alternativ vârfurile printr-o parcurgere a grafului și se testează dacă se obține o colorare proprie (muchii care nu sunt în arborele parțial asociat parcurgerii au și ele extremitățile colorate diferit).

Problema cuplajului maxim (Maximum-Matching). Dat un graf neorientat G , să se determine un cuplaj de cardinal maxim în G .

Vom descrie un algoritm care determină un cuplaj maxim într-un graf **bipartit** reducând această problemă la problema determinării unui flux maxim într-o rețea asociată lui N .

Algoritm de determinare a unui cuplaj maxim într-un graf bipartit

Intrare: $G = (V = X \cup Y, E)$ un graf bipartit

Ieșire: un cuplaj de cardinal maxim în G

Pseudocod:

1. Asociem grafului G o rețea $N_G = (G', s, t, X \cup Y, c)$ astfel:
 - $V(G') = V(G) \cup \{s, t\}$
 - adăugăm la $E(G')$ arce xy cu $x \in X, y \in Y$ (orientate de la x la y) corespunzătoare muchiilor xy din G
 - adăugăm la $E(G')$ arcele $sx, x \in X$
 - adăugăm la $E(G')$ arcele $yt, y \in Y$
 - asociem arcelor capacitatea 1 ($c = 1$)
2. Determinăm f un flux maxim în N .
3. Fie $M = \{xy \in E(G) \mid f(xy) = 1, x, y \in X \cup Y\}$.
4. returnează M

Complexitate - În N_G capacitatea arcelor este 1. Algoritmul Ford-Fulkerson pentru o astfel de rețea are complexitatea $O(nmC) = O(nm)$.

Corectitudine

Propoziția 6.19. Fie $G = (V = X \cup Y, E)$ un graf bipartit și N_G rețeaua de transport asociată în algoritm.

a) Fie M un cuplaj în G . Atunci există în N_G un flux f cu valoarea $val(f) = |M|$ definit astfel:

- pentru fiecare $xy \in M, x \in X, y \in Y$: $f(xy) = 1$
- pentru fiecare $x \in X \cap V(M)$: $f(sx) = 1$
- pentru fiecare $y \in Y \cap V(M)$: $f(yt) = 1$
- pentru toate celelalte arce $f(e) = 0$

b) Fie f un flux în N_G . Atunci există în G un cuplaj de cardinal $val(f)$, și anume

$$M = \{xy \in E(G) \mid x \in X, y \in Y, f(xy) = 1\}$$

Proof. Observație: Deoarece $c = 1$, fluxul pe un arc în N_G poate fi doar 0 sau 1.

a) Arătăm că f definit ca în enunț este flux. Din definiția lui f avem $f(e) \leq 1 \leq c(e)$ pentru orice arc e , deci este satisfăcută condiția de mărginire.

Fie $x \in X$.

- Dacă $x \notin V(M)$ atunci $f^+(x) = f^-(x) = 0$.
- Dacă $x \in V(M)$, deoarece M este cuplaj, există un unic y cu $xy \in M$, deci un unic arc $e = xy$ care iese din x cu flux pozitiv. Avem atunci $f^+(x) = f(xy) = 1$ și $f^-(x) = f(sx) = 1$ și deci $f^+(x) = f^-(x) = 1$.

Analog, pentru $y \in Y$, avem

$$f^+(y) = f^-(y) = \begin{cases} 1, & \text{dacă } y \in V(M) \\ 0, & \text{dacă } y \notin V(M) \end{cases}$$

În plus

$$val(f) = \sum_{sx \in N_G} f(sx) = \sum_{x \in X \cap V(M)} 1 = |X \cap V(M)| = |M|.$$

b) Arătăm că M este cuplaj.

- Fie $x \in V(M) \cap X$. Atunci există cel puțin o muchie $xy \in M$, deci un arc cu $f(xy) = 1$. Avem

$$\begin{aligned} f^+(x) &= f^-(x) \\ f^-(x) &= f(sx) \leq 1 \\ f^+(x) &= \sum_{xz \in N_G} f(xz) \geq f(xy) = 1 \end{aligned}$$

Rezultă $f^+(x) = f^-(x) = 1 = f(sx) = f(xy)$ și xy este unicul arc care iese din x cu flux pozitiv.

Rezultă că xy este unica muchie din M incidentă în x .

Analog, pentru $y \in V(M) \cap Y$ avem $f(yt) = 1$ și xy este unica muchie din M incidentă în y .

Rezultă că M este cuplaj.

- În plus, dacă $x \in X - V(M)$, avem, din definiția lui M , $f^+(x) = 0$, deci și $f^-(x) = 0$, adică $f(sx) = 0$.

(f este chiar fluxul definit la a))

Ca și la a) avem atunci $val(f) = |V(M) \cap X| = |M|$.

(sau, folosind tăieturi, avem:

$$val(f) = f^+(X \cup \{s\}, Y \cup \{t\}) - f^-(X \cup \{s\}, Y \cup \{t\}) = |M| - 0 = |M|.)$$

□

Corolar 6.20. Fie f^* un flux maxim în N_G . Atunci cuplajul M^* corespunzător, definit prin

$$M^* = \{xy \in E(G) | x \in X, y \in Y, f^*(xy) = 1\}$$

este cuplaj maxim în G .

Proof. Din Propoziția 6.19 avem $|M^*| = \text{val}(f^*)$.

Dacă prin absurd ar exista un cuplaj M' cu $|M'| > |M^*|$, atunci fluxul f' corespunzător lui M' (definit ca în Propoziția 6.19) are valoarea

$$\text{val}(f') = |M'| > |M^*| = \text{val}(f^*),$$

ceea ce contrazice faptul că f^* este maxim. □

6.5.2 Construcția unui graf orientat cu secvențele de grade interne și externe date

Algoritm de determinare a unui graf orientat cu secvențele de grade interne și externe date

Intrare: Două secvențe de n numere naturale: $s_0^+ = \{d_1^+, \dots, d_n^+\}$ și $s_0^- = \{d_1^-, \dots, d_n^-\}$ cu

$$d_1^+ + \dots + d_n^+ = d_1^- + \dots + d_n^-$$

Ieșire: Un graf orientat G cu $s^+(G) = s_0^+$ și $s^-(G) = s_0^-$ dacă există, mesajul NU altfel.

Pseudocod:

1. Construim o rețea $N = (G', s, t, X \cup Y, c)$ astfel:
 - $G' = (X \cup Y \cup \{s, t\}, E')$ unde
 - $X = \{x_1, \dots, x_n\}$
 - $Y = \{y_1, \dots, y_n\}$
 - $E' = \{x_i y_j | i, j \in \{1, \dots, n\}, i \neq j\} \cup \{s x_i | x_i \in X\} \cup \{y_j t | y_j \in Y\}$
 - Definim capacitățile arcelor astfel:
 - $c(s x_i) = d_i^+$, pentru $s x_i \in E$
 - $c(y_j t) = d_j^-$, pentru $y_j t \in E$
 - $c(x_i y_j) = 1$, pentru $x_i y_j \in E$
2. Determinăm f un flux maxim în N .
3. Dacă $\text{val}(f) \neq d_1^+ + \dots + d_n^+$, atunci scrie "NU", STOP.
4. Definim graful orientat $G = (V, E)$ cu $V = \{1, \dots, n\}$ și $E = \{ij | f(x_i y_j) = 1\}$
5. returnează G

Complexitate - Capacitatea minimă a unei tăieturi $L \leq c^+(s) = d_1^+ + \dots + d_n^+ = |E(G)| \implies O(m^2)$.

Corectitudine

Propoziția 6.21. Fie $s_0^+ = \{d_1^+, \dots, d_n^+\}$ și $s_0^- = \{d_1^-, \dots, d_n^-\}$ două secvențe de n numere naturale cu $d_1^+ + \dots + d_n^+ = d_1^- + \dots + d_n^-$.

Fie N rețeaua construită în algoritm.

Atunci există un graf orientat G cu $s^+(G) = s_0^+$ și $s^-(G) = s_0^-$ dacă și numai dacă valoarea fluxului maxim în N este $d_1^+ + \dots + d_n^+$.

Proof. Observație: Un flux f în N cu $\text{val}(f) = d_1^+ + \dots + d_n^+$ este flux maxim, deoarece în acest caz $\text{val}(f) = c^+(s)$.

" \implies " Presupunem că există un graf orientat G cu $V(G) = \{1, \dots, n\}$ având $s^+(G) = s_0^+$ și $s^-(G) = s_0^-$. Definim un flux f în N astfel:

- $f(x_i y_j) = 1$ pentru $ij \in E(G)$
- $f(sx_i) = d_i^+$, pentru $i \in \{1, \dots, n\}$
- $f(y_j t) = d_j^-$, pentru $j \in \{1, \dots, n\}$

Arătăm că f este corect definit. Evident f satisface condiția de mărginire.

Fie $i \in \{1, \dots, n\}$ fixat. Avem

$$f^+(x_i) = \sum_{ij \in E(G)} f(x_i y_j) = |\{ij | j \in V, ij \in E(G)\}| = d_G^+(i) = d_i^+.$$

Dar și

$$f^-(x_i) = f(sx_i) = d_i^+,$$

deci f satisface condiția de conservare în x_i .

Analog, pentru $j \in \{1, \dots, n\}$ avem $f^-(y_j) = d_j^- = f(y_j t) = f^+(y_j)$.

Rezultă că f este flux cu $val(f) = f^+(s) = f(sx_1) + \dots + f(sx_n) = d_1^+ + \dots + d_n^+$.

" \impliedby " Presupunem că există un flux f în N cu $val(f) = d_1^+ + \dots + d_n^+$. Avem atunci

$$\begin{aligned} d_1^+ + \dots + d_n^+ &= val(f) = f^+(s) = \\ &= f(sx_1) + \dots + f(sx_n) \leq c(sx_1) + \dots + c(sx_n) = \\ &= d_1^+ + \dots + d_n^+, \end{aligned}$$

deci au loc egalitățile $f(sx_i) = d_i^+$ pentru orice $i \in \{1, \dots, n\}$. Analog,

$$\begin{aligned} d_1^- + \dots + d_n^- &= d_1^+ + \dots + d_n^+ = val(f) = f^-(t) = \\ &= f(y_1 t) + \dots + f(y_n t) \leq c(y_1 t) + \dots + c(y_n t) = \\ &= d_1^- + \dots + d_n^-, \end{aligned}$$

deci $f(y_j t) = d_j^-$ pentru orice $j \in \{1, \dots, n\}$.

Definim graful orientat $G = (V, E)$ astfel: $V = \{1, \dots, n\}$ și

$$E = \{ij | f(x_i y_j) > 0\} = \{ij | f(x_i y_j) = 1\}.$$

Pentru $i \in \{1, \dots, n\}$ fixat avem

$$\begin{aligned} f^+(x_i) &= f^-(x_i) \\ f^-(x_i) &= f(sx_i) = d_i^+ \\ f^+(x_i) &= |\{x_i y_j | j \in \{1, \dots, n\}, f(x_i y_j) = 1\}| = |\{ij | j \in \{1, \dots, n\}, ij \in E\}| = d_G^+(i) \end{aligned}$$

deci $d_i^+ = d_G^+(i)$. Rezultă $s^+(G) = s_0^+$.

Analog $d_j^- = f^-(y_j) = f^+(y_j) = f(y_j t) = d_j^-$, deci $s^-(G) = s_0^-$. □

6.6 Conectivitate

- Suplimentar, v. slide.