ELEMENTE DE TEORIA GRAFURILOR

1. Noțiuni generale

În general, pentru situațiile care necesită la rezolvare un oarecare efort mintal (şi un caz tipic este cel al celor din economie), se caută, în primul rând, o metodă de reprezentare a lor care să permită receptarea întregii probleme dintr-o privire (pe cât posibil) şi prin care să se evidențieze cât mai clar toate aspectele acesteia.

În acest scop se folosesc imagini grafice gen diagrame, schițe, grafice etc. O reprezentare dintre cele mai utilizate este cea prin **grafuri**. Acestea sunt utilizate în special pentru vizualizarea sistemelor şi situațiilor complexe. În general, vom reprezenta componentele acestora prin puncte în plan iar relațiile (legăturile, dependențele, influențele etc) dintre componente prin arce de curbă cu extremitățile în punctele corespunzătoare. Între două puncte pot exista unul sau mai multe segmente (în funcție de câte relații dintre acestea, care ne interesează, există) iar segmentelor li se pot asocia sau nu orientări (după cum se influențează cele două componente între ele), numere care să exprime intensitatea relațiilor dintre componente etc.

Este evident, totuşi, că această metodă are limite, atât din punct de vedere uman (prea multe puncte şi segmente vor face desenul atât de complicat încât se va pierde chiar scopul pentru care a fost creat – claritatea şi simplitatea reprezentării, aceasta devenind neinteligibilă) cât şi din punct de vedere al tehnicii de calcul (un calculator nu poate "privi" un desen ca un om).

Din acest motiv, alături de expunerea naiv-intuitivă a ceea ce este un graf, dată mai sus, se impune atât o definiție riguroasă cât şi alte modalități de reprezentare a acestora, adecvate în general rezolvărilor matematice.

***Definiția* 1** Se numeşte **multigraf** un triplet G = (X, A, *f*) în care X şi A sunt două mulțimi iar *f* este o funcție, definită pe produsul vectorial al lui X cu el însuşi ($X^2 = X \times X$), care ia valori în mulțimea părților mulțimii A (notată $P(A)$)

Mulțimea X se numeşte mulțimea nodurilor multigrafului şi elementele sale se numesc noduri (sau vârfuri) ale multigrafului, iar A reprezintă mulțimea relațiilor (legăturilor) posibile între două noduri ale multigrafului.

***Definiția* 2** Se numeşte **graf orientat** un multigraf în care mulțimea A are un singur element: A = {a}.

În acest caz mulțimea părților mulțimii A are doar două elemente: mulțimea vidă $\varnothing$ şi întreaga mulțime A. Dacă unei perechi orientate (x_i, x_j) din X^2 i se asociază prin funcția *f* mulțimea A atunci spunem ca există arc de la nodul x_i la nodul x_j iar perechea (x_i,x_j) se va numi **arcul** (x_i,x_j). Nodul x_i se numeşte **nod inițial** sau **extremitate inițială** a arcului (x_i,x_j) iar nodul x_j se numeşte **nod final** sau **extremitate finală** a arcului (x_i,x_j). Arcul (x_i,x_j) este **incident spre interior** vârfului x_j şi **incident spre exterior** vârfului x_i. Dacă pentru un arc nodul inițial coincide cu nodul final atunci acesta se numeşte **buclă**. Nodurile x_i şi x_j se vor numi **adiacente** dacă există cel puțin unul din arcele (x_i,x_j) şi (x_j,x_i).

Dacă unei perechi orientate (x_i, x_j) din X^2 i se asociază prin funcția *f* mulțimea vidă $\varnothing$ atunci spunem că nu există arc de la nodul x_i la nodul x_j.

Este evident că a cunoaşte un graf orientat este echivalent cu a cunoaşte vârfurile şi arcele sale. Din acest motiv putem defini un graf orientat prin perechea (X,U), unde X este mulțimea vârfurilor sale iar U mulțimea arcelor sale.

De asemenea, putem cunoaşte un graf orientat cunoscând mulțimea nodurilor şi, pentru fiecare nod, mulțimea arcelor incidente spre exterior. Din acest motiv putem defini un graf orientat ca o pereche (X,Γ) unde X este perechea nodurilor iar Γ este o funcție definită pe X cu valori în

mulțimea părților lui X, valoarea acesteia într-un nod x_i, $\Gamma(x_i) \subseteq X$ fiind mulțimea nodurilor adiacente nodului x_i, prin arce pentru care x_i este extremitatea inițială.

***Definiția* 3** Se numeşte **graf neorientat** un multigraf în care mulțimea A are un singur element iar funcția *f* are proprietatea:

$$f[(x_i,x_j)] = f[(x_j,x_i)] \text{ , oricare ar fi nodurile } x_i \text{ şi } x_j \text{ din X}$$

În aceste condiții, dacă $f[(x_i,x_j)] = f[(x_j,x_i)] = A$ atunci perechea neorientată $\{x_i,x_j\}$ este o **muchie** iar dacă $f[(x_i,x_j)] = f[(x_j,x_i)] = \varnothing$ spunem că nu există muchie între vârfurile x_i şi x_j.

Deoarece, în cele mai multe din cazurile practice care vor fi analizate în acest capitol, situația este modelată matematic printr-un graf orientat, vom folosi, pentru simplificarea expunerii, denumirea de graf în locul celei de graf orientat iar în cazul în care graful este neorientat vom specifica acest fapt la momentul respectiv.

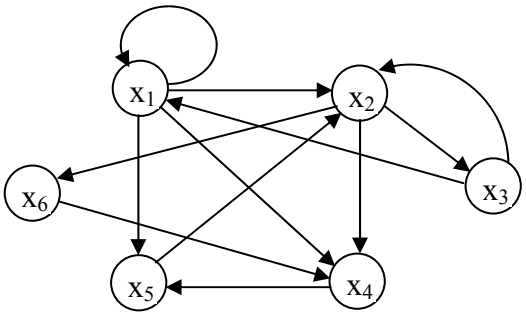
2. Moduri de reprezentare ale unui graf

A. O primă modalitate de reprezentare este listarea efectivă a tuturor nodurilor şi a arcelor sale.
B. Putem reprezenta graful dând pentru fiecare nod mulțimea nodurilor cu care formează arce în care el este pe prima poziție.
C. Putem reprezenta geometric graful, printr-un desen în plan, reprezentând fiecare nod printr-un punct(cerculeț) şi fiecare arc printr-un segment de curbă care are ca extremități nodurile arcului şi pe care este trecută o săgeată orientată de la nodul inițial spre cel final.
D. Putem folosi o reprezentare geometrică în care nodurile sunt reprezentate de două ori, în două şiruri paralele, de la fiecare nod din unul din şiruri plecând săgeți spre nodurile cu care formează arce în care el este pe prima poziție, de pe al doilea şir (reprezentarea prin corespondență).
E. Un graf poate fi reprezentat printr-o matrice pătratică booleană, de dimensiune egală cu numărul de noduri, în care o poziție a_{ij} va fi 1 dacă există arcul (x_i,x_j) şi 0 în caz contrar, numită matricea adiacențelor directe.
F. Un graf poate fi reprezentat printr-o matrice pătratică latină, de dimensiune egală cu numărul de noduri, în care pe o poziție a_{ij} va fi x_ix_j dacă există arcul (x_i,x_j) şi 0 în caz contrar.

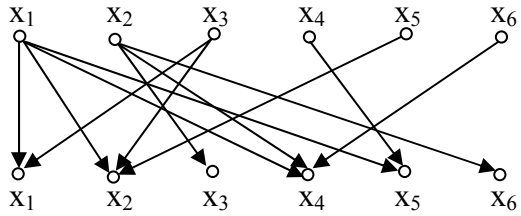
Exemplu: Dacă în reprezentarea A avem graful G = (X,U), unde $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$ şi $U = \{(x_1,x_1), (x_1,x_2), (x_1,x_4), (x_1,x_5), (x_2,x_3), (x_2,x_4), (x_2,x_6), (x_3,x_1), (x_3,x_2), (x_4,x_5), (x_5,x_2), (x_6,x_4)\}$, atunci în celelalte reprezentări vom avea:

B
$x_1 \rightarrow \{x_1, x_2, x_4, x_5\}$
$x_2 \rightarrow \{x_3, x_4, x_6\}$
$x_3 \rightarrow \{x_1, x_2\}$
$x_4 \rightarrow \{x_5\}$
$x_5 \rightarrow \{x_2\}$
$x_6 \rightarrow \{x_4\}$

C



D (reprezentarea prin corespondență)



E

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	1	1	0	1	1	0
x_2	0	0	1	1	0	1
x_3	1	1	0	0	0	0
x_4	0	0	0	0	1	0
x_5	0	1	0	0	0	0
x_6	0	0	0	1	0	0

F

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	x_1x_1	x_1x_2	0	x_1x_4	x_1x_5	0
x_2	0	0	x_2x_3	x_2x_4	0	x_2x_6
x_3	x_3x_1	x_3x_2	0	0	0	0
x_4	0	0	0	0	x_4x_5	0
x_5	0	x_5x_2	0	0	0	0
x_6	0	0	0	x_6x_4	0	0

3. Concepte de bază în teoria grafurilor

1. **semigraf interior** al unui nod x_k: este mulțimea arcelor $U^-_{x_k} = \{(x_j,x_k)/ (x_j,x_k) \in U\}$ care sunt incidente spre interior nodului x_k;
2. **semigraf exterior** al unui nod x_k: este mulțimea arcelor $U^+_{x_k} = \{(x_k,x_i)/ (x_k,x_i) \in U\}$ care sunt incidente spre exterior nodului x_k;
3. **semigradul interior** al unui nod x_k: este numărul arcelor care sunt incidente spre interior nodului x_k = cardinalul lui $U^-_{x_k}$ şi se notează cu $\delta^-_{x_k}$;
4. **semigradul exterior** al unui nod x_k: este numărul arcelor care sunt incidente spre exterior nodului x_k = cardinalul lui $U^+_{x_k}$ şi se notează cu $\delta^+_{x_k}$;
5. **gradul** unui nod x_k: este suma semigradelor nodului x_k: $\delta_{x_k} = \delta^+_{x_k} + \delta^-_{x_k}$;
6. **nod izolat**: este un nod cu gradul 0;
7. **nod suspendat**: este un nod cu gradul 1;
8. **arce adiacente**: arce care au o extremitate comună;
9. **drum** într-un graf: o mulțime ordonată de noduri ale grafului: $(x_1, x_2, ..., x_k)$, cu proprietatea că există în graf toate arcele de forma (x_i,x_{i+1}) i = 1,...,k-1;
10. **lungimea unui drum**: este numărul arcelor care îl formează;
11. **drum elementar**: un drum în care fiecare nod apare o singură dată;
12. **drum simplu**: un drum în care fiecare arc apare o singură dată;
13. **putere de atingere** a unui nod $x_i \in X$ în graful G: numărul de noduri la care se poate ajunge din x_i. Puterea de atingere se notează cu $p(x_i)$, $1 \le i \le n$ şi evident $p(x_i) \ge \delta^+_{x_i}$.
14. **drum hamiltonian**: un drum elementar care trece prin toate nodurile grafului;
15. **drum eulerian**: un drum simplu care conține toate arcele grafului;
16. **lanț**: un drum în care arcele nu au neapărat acelaşi sens de parcurgere;
17. **circuit**: un drum în care nodul inițial coincide cu cel final;
18. **circuit elementar**: un drum în care fiecare nod apare o singură dată, cu excepția celui final, care coincide cu cel inițial;
19. **circuit simplu**: un drum în care fiecare arc apare o singură dată;
20. **circuit hamiltonian**: un circuit care trece prin toate nodurile grafului;
21. **ciclu**: este un circuit în care arcele nu au neapărat acelaşi sens de parcurgere;
22. **ciclu elementar**: un ciclu în care fiecare nod apare o singură dată, cu excepția celui final, care coincide cu cel inițial;
23. **ciclu simplu**: un ciclu în care fiecare arc apare o singură dată;
    Observație: Într-un graf neorientat noțiunile de drum şi lanț sunt echivalente şi de asemenea cele de circuit şi ciclu.
24. **graf parțial** al unui graf G = (X,U): este un graf G'(X,U') cu $U' \subset U$;
25. **subgraf** al unui graf $G = (X,\Gamma)$: este un graf $G'(X',\Gamma')$ unde $X' \subset X$ şi $\Gamma'(x_i) = \Gamma(x_i) \cap X'$ pentru orice $x_i \in X'$;

26. **graf redus** al unui graf G = (X,U): este un graf $G^*(X^*,U^*)$ unde X^* este formată din mulțimile unei partiții de mulțimi nevide ale lui X, iar (X_i^*, X_j^*) există doar dacă $i \neq j$ şi există cel puțin un arc în U, de la un nod din X_i^* la un nod din X_j^*.
27. **graf tare conex**: este un graf în care între oricare două noduri există cel puțin un drum;
28. **graf simplu conex**: este un graf în care între oricare două noduri există cel puțin un lanț;

Observație: Pentru grafuri neorientat noțiunile de tare conex şi simplu conex sunt echivalente, graful numindu-se doar conex;

29. **componentă tare conexă** a unui graf G = (X,U): este un subgraf al lui G care este tare conex şi nu este subgraful nici unui alt subgraf tare conex al lui G (altfel spus, între oricare două noduri din componentă există cel puțin un drum şi nu mai există nici un nod în afara componentei legat printr-un drum de un nod al componentei).

4. Găsirea drumurilor într-un graf orientat

Dacă privim graful ca imagine a unui sistem, nodurile reprezentând componentele sistemului, atunci o interpretare imediată a unui arc (x_i,x_j) este că, componenta x_i influențează direct componenta x_j. Dacă nodurile au semnificația de stări posibile ale unui sistem atunci un arc (x_i,x_j) semnifică faptul că sistemul poate trece direct din starea x_i în starea x_j. În ambele cazuri se vede că avem de-a face doar cu informații despre legături directe; totuşi, chiar dacă o componentă x_i nu influențează direct componenta x_j ea o poate influența prin intermediul altor componente, existând un şir de componente intermediare: x_1 x_2 ,..., x_k, fiecare influențând-o direct pe următoarea şi x_i direct pe x_1 iar x_k direct pe x_j. Astfel, dacă dintr-o stare x_i nu se poate trece direct într-o stare x_j s-ar putea totuşi în mai multe etape, prin alte stări intermediare. Deoarece găsirea acestor influențe sau treceri posibile este de obicei foarte importantă iar pentru un sistem cu mii sau zeci de mii de componente acest lucru nu mai poate fi făcut "din ochi", este necesară formalizarea noțiunii de "influențe" şi "treceri" posibile, nu neapărat directe. Acest lucru a şi fost făcut mai sus, deoarece este evident că "x_i influențează x_j" sau "din starea x_i se poate trece în starea x_j" este echivalent cu existența în graf a unui drum de la nodul x_i la nodul x_j.

În continuare vom da un algoritm prin care putem găsi toate drumurile dintr-un graf orientat cu un număr finit de noduri.

Pasul 1. Se construieşte matricea booleană a adiacențelor directe corespunzătoare grafului, notată cu A. În aceasta se află, evident, toate drumurile de lungime 1.

Este interesant de văzut ce legătură există între această matrice şi drumurile de lungime 2. Fie două noduri x_i şi x_j oarecare din graf. Existența unui drum de lungime 2 între ele presupune existența unui nod x_k, din graf, cu proprietatea că există atât arcul (x_i,x_k) cât şi arcul (x_i,x_k). Pentru a vedea dacă acesta există, luăm pe rând fiecare nod al grafului şi verificăm dacă există sau nu ambele arce ((x_i,x_k) şi (x_i,x_k)). Aceasta este echivalent cu a verifica dacă, în matricea booleană a adiacențelor directe, există vreun indice k astfel încât elementul k al liniei i şi elementul k al coloanei j să fie ambele egale cu 1. Dacă folosim operațiile algebrei booleene de adunare şi înmulțire:

$\dot{+}$	0	1
0	0	1
1	1	1

$\dot{\times}$	0	1
0	0	0
1	0	1

atunci verificările de mai sus sunt echivalente cu a verifica dacă elementul de pe poziția (i,j) din A^2 este egal cu 1. Valoarea 1 spune doar că există cel puțin un drum de lungime 2 de la x_i la x_j. Dacă dorim să vedem şi câte sunt, vom folosi regulile de înmulțire şi adunare obişnuită.

De asemenea, se poate observa că existența unui drum de lungime 3 de la x_i la x_j presupune existența unui nod x_k astfel încât să existe un drum de lungime 2 de la x_i la x_k şi un arc de la x_k la x_j, care este echivalent cu a verifica dacă există vreun indice k astfel încât elementul k al liniei i din

matricea A^2 şi elementul k al coloanei j din A sunt ambele egale cu 1 sau, mai simplu, dacă elementul (i,j) din A^3 este 1.

Din cele de mai sus se observă că existența drumurilor de lungime k este dată de valorile matricei A^k, dacă s-au folosit regulile algebrei booleene şi numărul lor este dat de A^k, dacă s-au folosit regulile obişnuite.

Pasul 2. Vom calcula succesiv puterile lui A până la puterea A^{n-1}

Dacă între nodurile x_i şi x_j există un drum de lungime ≥ n atunci el va conține un număr de noduri mai mare sau egal nu n+1 şi, cum în graf sunt doar n vârfuri, este clar că cel puțin unul, să zicem x_k, va apărea de două ori. Vom avea în acest caz un drum de la x_i până la prima apariție a lui x_k, şi un drum de la ultima apariție a lui x_k la x_j. Eliminând toate nodurile dintre prima apariție a lui x_k şi ultima apariție a sa vom obține un drum de la x_i la x_j, în care x_k apare o singură dată. Aplicând acest procedeu pentru toate nodurile care apar de mai multe ori pe drum, vom obține un drum de la x_i la x_j, în care fiecare nod apare o singură dată (deci un drum elementar), care are evident cel mult n-1 arce. În concluzie, dacă există vreun drum de la x_i la x_j atunci există şi un drum elementar şi, deci, va exista o putere a lui A, între A^1 şi A^{n-1}, în care poziția (i,j) este diferită de 0. Pentru deciderea existenței unui drum între oricare două noduri este suficientă, deci, calcularea doar a primelor n-1 puteri ale lui A.

Pasul 3. Se calculează matricea $D = A + A^2 + ... + A^{n-1}$

Dacă ne interesează doar existența drumurilor dintre noduri, nu şi numărul lor, vom folosi înmulțirea şi adunarea booleană şi conform observației de mai sus:

$$d_{ij} = \begin{cases} 1 & \text{dacă există cel putin un drum de } x_i \text{ la } x_j \\ 0 & \text{dacă nu există nici un drum de } x_i \text{ la } x_j \end{cases}$$

În acest caz, observând că:

$$A\cdot(A + I)^{n-2} = C^0_{n-2}\cdot A + C^1_{n-2}\cdot A^2 + C^2_{n-2}\cdot A^3 + ... + C^{n-2}_{n-2}\cdot A^{n-1} = A + A^2 + A^3 + ... + A^{n-1} = D$$

rezultă că e suficient să calculăm doar puterea n-2 a matricei A + I şi apoi s-o înmulțim cu A. Avantajul acestei metode, în ceea ce privește economia de timp, este susținut şi de următoarea observație: dacă D conține toate perechile de arce între care există drum atunci:

$$D = (A + A^2 + ... + A^{n-1}) + A^n + A^{n+1} + ... + A^{n+k} = D \text{ oricare ar fi } k \geq 0 \Rightarrow$$
$$\Rightarrow A\cdot(A + I)^{n-2+k} = (A + A^2 + ... + A^{n-1}) + A^n + A^{n+1} + ... + A^{n+k-1} = D = A\cdot(A + I)^{n-2} \Leftrightarrow$$
$$\Leftrightarrow A\cdot(A + I)^{n-2+k} = A\cdot(A + I)^{n-2} \text{ oricare ar fi } k \geq 0$$

deci de la puterea k = n-2 toate matricile A^k sunt egale. Putem, deci, calcula direct orice putere a lui A+I mai mare sau egală cu n-1 (de exemplu calculând $(A+I)^2$, $(A+I)^4$, $(A+I)^8$, ..., $(A+I)^{2^r}$, r fiind prima putere a lui 2 pentru care $2^r \geq n-2$).

Procedeul de mai sus nu asigură decât aflarea faptului dacă există sau nu drum între două noduri, eventual ce lungime are şi câte sunt de această lungime. Totuşi, în problemele practice cel mai important este să ştim care sunt efectiv aceste drumuri. Deoarece toate drumurile pot fi descompuse în drumuri elementare şi în problemele practice în general acestea sunt cele care interesează, paşii următori ai algoritmului vor fi dedicați găsirii lor. Pentru găsirea acestora se foloseşte reprezentarea grafului prin matricea latină de la cazul F.

Pasul 4. Construim matricea latină L asociată grafului, unde:

$$l_{ij} = \begin{cases} x_i x_j & \text{dacă există arcul } (x_i, x_j) \\ 0 & \text{dacă nu există arcul } (x_i, x_j) \end{cases}$$

şi matricea $\widetilde{L}$, definită prin:

$$\widetilde{l}_{ij} = \begin{cases} x_j & \text{dacă există arcul } (x_i, x_j) \\ 0 & \text{dacă nu există arcul } (x_i, x_j) \end{cases}$$

numită **matricea latină redusă**.

Găsirea unui drum de lungime 2 de la x_i la x_j presupune găsirea unui nod cu proprietatea că există arcele (x_i,x_k) şi (x_k,x_j) şi memorarea vectorului (x_i, x_k, x_j). Aceasta este echivalent cu a găsi un indice k astfel încât elementul de pe poziția k a liniei i, din matricea L, să fie x_i,x_k şi elementul de pe poziția k al coloanei j, din matricea $\widetilde{L}$, să fie x_j. Vom înmulți deci matricea L cu matricea $\widetilde{L}$, folosind însă nişte reguli de calcul speciale, numite înmulțire şi adunare latină.

Definiția 1: Se numeşte **alfabet** o mulțime de semne numite **simboluri** sau **litere** $\{s_i/i \in I\}$ unde I este o mulțime oarecare de indici, finită sau nu.

Definiția 2: Se numeşte **cuvânt** un şir finit de simboluri notat $s_{i_1} s_{i_2} ... s_{i_n}$.

Definiția 3: Se numeşte **înmulțire latină** o operație definită pe mulțimea cuvintelor unui alfabet, notată "$\times_L$", astfel:

$$s_{i_1} s_{i_2} ... s_{i_n} \times_L s_{j_1} s_{j_2} ... s_{j_m} = s_{i_1} s_{i_2} ... s_{i_n} s_{j_1} s_{j_2} ... s_{j_m}$$

(produsul a două cuvinte se obține prin concatenarea lor)

Înmulțirea latină este asociativă, are ca element neutru cuvântul vid, nu e comutativă şi un element este inversabil doar dacă este cuvântul vid.

Definiția 3: Se numeşte **adunare latină** o funcție definită pe mulțimea cuvintelor unui alfabet cu valori în mulțimea părților mulțimi cuvintelor, notată "$+_L$" astfel:

$$s_{i_1} s_{i_2} ... s_{i_n} +_L s_{j_1} s_{j_2} ... s_{j_m} = \left\{ s_{i_1} s_{i_2} ... s_{i_n} \right\} \left\{ s_{j_1} s_{j_2} ... s_{j_m} \right\}$$

(suma a două cuvinte este mulțimea formată din cele două cuvinte)

Pasul 5. Se calculează succesiv matricile:

$$L^2 = L \times_L \widetilde{L} \quad , \quad L^3 = L^2 \times_L \widetilde{L} \quad , \quad ... \quad , L^{k+1} = L^k \times_L \widetilde{L}$$

folosind operațiile de înmulțire şi adunare latină, alfabetul fiind mulțimea nodurilor grafului, unde operația de înmulțire este uşor modificată, produsul dintre două elemente ale matricilor fiind 0, dacă unul dintre ele este 0 sau au un nod comun şi este produsul latin al lor, în caz contrar.

Din felul cum a fost construită, matricea L^k va conține toate drumurile elementare de lungime k. Cum un drum elementar poate avea cel mult n noduri (câte are graful cu totul) rezultă că:

- primele n-1 puteri ale lui L conțin toate drumurile elementare din graf;
- puterile lui L mai mari sau egale cu n au toate elementele egale cu 0;
- matricea L^{n-1} conține toate drumurile hamiltoniene din graf (dacă există).

Observație: Deoarece obținerea matricii D prin metoda de mai sus presupune un volum foarte mare de calcule (de exemplu, dacă graful are 100 de noduri, ridicarea unei matrici de 100×100 la puterea 100) pentru obținerea acesteia se poate aplica şi următorul algoritm:

Pas 1. Se construieşte matricea de adiacență A;

Pas 2. Pentru fiecare linie i se adună boolean la aceasta toate liniile j pentru care $a_{ij} = 1$.

Pas 3. Se reia pasul 2 până când, după o aplicare a acestuia, matricea rămâne aceeaşi (nu mai apare nici un 1)

Ultima matrice obținută este matricea drumurilor D numită şi matricea conexiunilor totale.

Această metodă, deşi mai simplă nu spune însă şi care sunt aceste drumuri, pentru găsirea lor aplicându-se, de exemplu, înmulțirea latină

5. ARBORI. Problema arborelui de valoare optimă

În acest subcapitol grafurile vor fi considerate neorientate.

5.1. Noțiunea de arbore

Un **arbore** este un graf neorientat, finit, conex şi fără cicluri. Grafurile din fig. 4.1. sunt arbori.

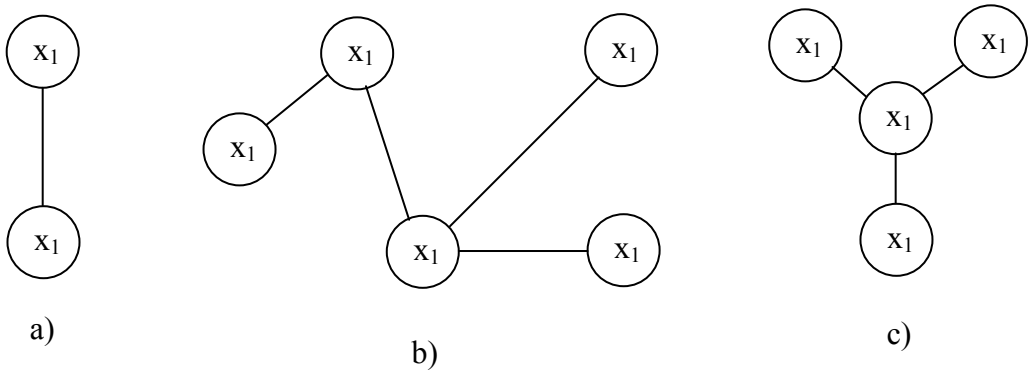


Figura 4.1

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

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

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

Deoarece definiția arborelui este dificil de aplicat pentru deciderea faptului că un graf este arbore sau nu (şi în special sunt greu de verificat conexitatea şi mai ales existența ciclurilor) există mai multe caracterizări posibile ale unui arbore, acestea fiind date de teorema de mai jos:

Teoremă. Dacă H este un graf neorientat finit, atunci următoarele afirmații sunt echivalente:

1) H este arbore;
2) H nu conține cicluri şi, dacă se unesc printr-o muchie două noduri neadiacente, se formează un ciclu (şi numai unul). Arborele este, deci, pentru o mulțime de noduri dată, graful cu numărul maxim de arce astfel încât să se păstreze proprietatea că nu are cicluri);
3) H este conex şi dacă i se suprimă o muchie se creează două componente conexe (arborele este graful conex cu numărul minim de arce);

4) H este conex şi are n-1 muchii;
5) H este fără cicluri şi are n-1 muchii;
6) Orice pereche de noduri este legată printr-un lanț şi numai unul.

Demonstrație :

1)⇒ 2). între cele două noduri adiacente noii muchii introduse exista deja un drum în fostul graf. Acest drum, împreună cu noul arc va forma evident un ciclu şi afirmația 2) a fost demonstrată.

2)⇒3). Pentru oricare două vârfuri neunite printr-o muchie, adăugând muchia dintre cele două vârfuri s-ar crea, conform ipotezei, un ciclu care conține această muchie, deci două drumuri între cele două noduri, din care unul nu conține noua muchie, adică în graful inițial exista un drum între cele două noduri. Dacă nu există cicluri înseamnă că între oricare două noduri există un singur drum. Pentru două noduri unite printr-o muchie, aceasta este chiar drumul corespunzător celor două noduri. Dacă suprimăm această muchie între cele două noduri nu va mai exista nici un drum, formându-se două componente conexe.

3)⇒4). Demonstrația se face prin inducție după n = numărul de noduri ale grafului. Pentru n=2 este evident. Presupunem afirmația adevărată pentru toate grafurile cu cel mult n noduri. Dacă graful are n+1 noduri, prin suprimarea unei muchii se formează două componente conexe fiecare având cel mult n noduri ($n_1 \le n$, $n_2 \le n$ şi $n_1 + n_2 = n+1$) şi deci au $n_1 - 1$ respectiv $n_2 - 1$ muchii. În concluzie graful inițial a avut $(n_1 - 1) + (n_2 - 1) + 1 = n_1 + n_2 - 1 = (n+1)-1$ muchii, ceea ce era de demonstrat.

4)⇒5). Dacă ar avea un ciclu atunci prin suprimarea unui arc al acestuia ar rămâne de asemenea conex. Eliminăm acest arc apoi repetăm procedeul pentru graful parțial rămas şi tot aşa până când nu mai rămâne nici un ciclu. În acest moment graful rămas este conex şi nu are cicluri deci este arbore şi deci are n-1 arce, în contradicție cu faptul că el avea n-1 arce înainte de a începe suprimarea arcelor;

5)⇒6). Dacă între două noduri ar exista două drumuri atunci acestea ar forma la un loc un ciclu. Deci între 2 noduri este cel mult un drum. Dacă între două noduri nu ar exista nici un drum ar fi cel puțin două componente conexe în graf, fiecare fiind arbore (pentru că nu există cicluri) şi deci fiecare ar avea un număr de arce cu 1 mai mic decât numărul de noduri. Făcând adunarea, ar rezulta că în graf sunt strict mai puțin de n-1 arce.

6)⇒1). Dacă H ar avea un ciclu, între două noduri ale acestuia ar exista două lanțuri, în contradicție cu ipoteza.

Presupunem că avem un graf pentru care am verificat deja dacă este conex. Dacă nu este atunci acesta, evident, nu are nici un graf parțial care să fie arbore.

Presupunem de asemenea că fiecărei muchii îi este asociată o valoare reală.

5.2. Algoritmi pentru găsirea arborelui de valoare optimă

Vom da mai jos trei algoritmi pentru determinarea unui graf parțial al grafului, care să fie arbore şi pentru care suma valorilor arcelor sale să fie minimă (sau maximă).

Toți algoritmii descrişi în continuare extrag arborele prin colectarea una câte una a muchiilor acestuia.

A. Algoritmul lui Kruskal

Pasul 1. Dintre toate muchiile grafului se alege muchia de valoare minimă (maximă). Dacă minimul este multiplu se alege la întâmplare una din muchiile respective. Deoarece acest "la întâmplare" trebuie cumva tradus în limbajul calculatorului, în cazul implementării unui

program bazat pe acest algoritm, vom perturba din start valorile muchiilor, la k muchii cu aceiaşi valoare V adunând respectiv valorile ε, 2ε, ... , kε, unde ε este foarte mic (în orice caz, kε mai mic decât diferența dintre valoarea acestor arce si valoarea imediat superioară a unui arc), pozitiv.

Pasul 2. Dintre toate muchiile rămase, se alege cea de valoare minimă (maximă);

Pasul 3. Dintre toate muchiile rămase, se alege cea de valoare minimă (maximă), astfel încât să nu se formeze cicluri cu cele deja alese;

Pasul 4. Se reia algoritmul de la pasul 3 până se colectează n-1 muchii.

Deşi s-a demonstrat că algoritmul găseşte întotdeauna arborele optim, el are dezavantajul că este foarte laborios (de fiecare dată trebuie calculat minimul unei mulțimi mari sau foarte mari – există situații în practică în care graful are sute de mii de arce) şi, în plus, trebuie aplicat un algoritm special ca să respectăm condiția de a nu se forma cicluri, la alegerea unui nou arc.

O metodă posibilă este ca, după adăugarea fiecărui arc, să se împartă graful în componente conexe şi să alegem apoi un arc care nu are ambele extremitățile în aceeaşi componentă conexă.

De asemenea este clar că, în cazul existenței arcelor de valori egale, deoarece se alege la întâmplare, există mai multe variante de evoluție a alegerii arcelor. Totuşi, cu toate că pot fi mai multe grafuri la care se poate ajunge prin acest algoritm, ele vor avea toate aceeaşi valoare (minima (sau maxima) posibilă).

B. Algoritmul lui Sollin

Pasul 1. Pentru fiecare nod se alege muchia adiacentă de valoare minimă (maximă).

Pasul 2. Se evidențiază componentele conexe, existente în graful parțial format din arcele alese până în acest moment.

Pasul 3. Pentru fiecare componentă conexă se alege muchia adiacentă de valoare minimă (maximă). Prin muchie adiacentă unei componente conexe înțelegem o muchie care are o singură extremitate printre nodurile componentei respective.

Pasul 4. Se reia algoritmul de la pasul 2 până rămâne o singură componentă conexă. Aceasta este arborele optim căutat.

Acest algoritm asigură de asemenea găsirea arborelui optim, necesită mult mai puține calcule (la fiecare alegere se calculează minimul doar pentru muchiile adiacente unui singur nod), evită automat formarea ciclurilor, dar, pentru grafuri foarte mari, la un moment dat pot exista atât de multe componente conexe care trebuie memorate succesiv, încât calculul devine greoi sau, pe calculator, depăşeşte posibilitățile de memorare ale calculatorului.

C. O variantă a algoritmului lui Kruskal

Pasul 1. Dintre toate muchiile grafului se alege cea de valoare minimă (maximă);

Pasul 2. Dintre toate muchiile adiacente componentei conexe formată din arcele alese până în acest moment, se alege cea de valoare minimă (maximă);

Pasul 3. Se reia pasul 2 până se colecționează n-1 muchii.

Algoritmul are toate avantajele algoritmului lui Sollin şi, în plus, lucrează cu o singură componentă conexă, fiind mult mai uşor de implementat pe calculator şi mult mai rapid în execuție.

***Exemplu*:** Administrația unei localități montane a hotărât construirea unor linii de teleferic care să lege oraşul de cele 8 puncte turistice importante din jurul acestuia. În urma unui studiu au

fost puse în evidența toate posibilitățile şi costurile de conectare a obiectivele turistice între ele şi cu oraşul, acestea fiind prezentate în figura 4.2.

Se cere găsirea variantei de construcție de cost minim, care să asigure accesul din oraş la oricare din obiectivele turistice.

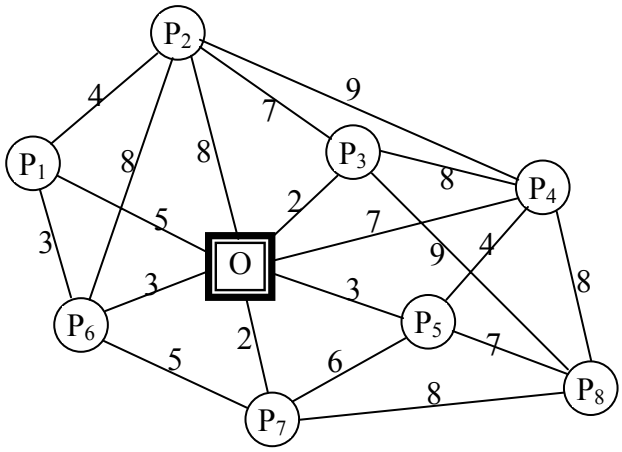


Figura 4.2

Rezolvare

Condiția de cost minim implică două obiective:

1. Să se construiască minimul de arce necesare;
2. Să se construiască cele mai ieftine legături.

Referitor la numărul de arce necesar, facem observația că, dacă din oraş se va putea ajunge la orice obiectiv turistic, atunci se va putea ajunge şi de la orice stațiune la oricare alta (trecând prin oraş), deci trebuie ca arcele alese pentru construcție să formeze la un loc un graf conex.

În concluzie, căutăm un graf parțial conex cu un număr minim de arce, adică un arbore. În plus, suma costurilor arcelor sale trebuie să fie minimă. Vom aplica pe rând cei trei algoritmi pentru găsirea acestuia:

A. Kruskal

La primul pas poate fi ales unul din arcele OP_3 sau OP_7, ele având valoarea minimă 2. Putem alege oricum primul arc dintre cele două pentru că la al doilea pas va fi ales celălalt.

La pasul trei poate fi ales unul din arcele OP_5, OP_6 sau P_1P_6 care au valoarea minimă 3. Nici în acest caz nu are vre-o importanță ordinea alegerii, deoarece pot fi alese succesiv toate trei fără a se forma nici un ciclu.

Al şaselea arc poate fi ales dintre arcele P_4P_5 şi P_1P_2, care au valoarea minimă 4. Nici în acest caz nu are vre-o importanță ordinea alegerii, deoarece pot fi alese succesiv ambele, fără a se forma nici un ciclu.

Următoarea valoare disponibilă a unui arc este 5, dar arcul opt nu poate fi ales dintre arcele OP_1, P_6P_7, deşi au valoarea minimă 5. Arcul OP_1 nu poate fi ales deoarece s-ar forma ciclul OP_1P_6, iar P_6P_7 ar duce la ciclul OP_6P_7. Următoarea valoare minimă este 6, pentru arcul P_5P_7 dar nu poate fi ales deoarece se formează ciclul OP_5P_7.

Valoarea următoare, 7, o au arcele OP_4, P_2P_3 şi P_5P_8. OP_4 nu poate fi ales deoarece s-ar forma ciclul OP_5P_4. Arcul P_2P_3 nu poate fi ales deoarece s-ar forma ciclul $OP_6P_1P_2P_3$. Arcul P_5P_8 nu formează nici un ciclu şi el va fi al optulea arc ales. În acest caz, deoarece s-au adunat 8 arce într-un graf cu 9 noduri, am obținut graful căutat.

Acest arbore este reprezentat în figura 4.3.

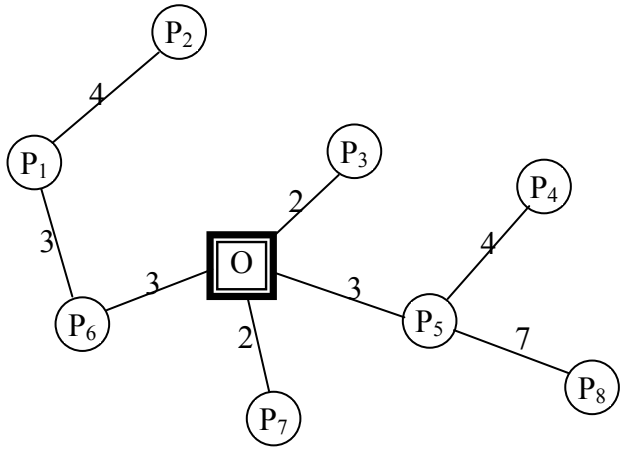


Figura 4.3

B. Sollin

Vom alege:	pentru nodul O	$\rightarrow$	arcul OP_3
	pentru nodul P_1	$\rightarrow$	arcul P_1P_6
	pentru nodul P_2	$\rightarrow$	arcul P_1P_2
	pentru nodul P_3	$\rightarrow$	arcul OP_3
	pentru nodul P_4	$\rightarrow$	arcul P_4P_5
	pentru nodul P_5	$\rightarrow$	arcul OP_5
	pentru nodul P_6	$\rightarrow$	arcul P_1P_6
	pentru nodul P_7	$\rightarrow$	arcul OP_7
	pentru nodul P_8	$\rightarrow$	arcul P_5P_8

Rezultă graful parțial:

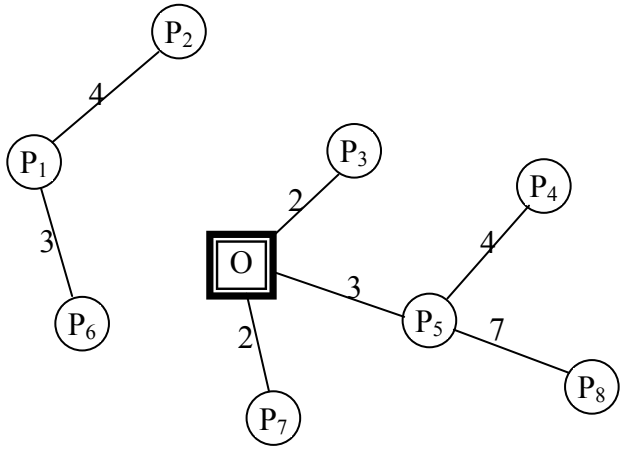


Figura 4.4

După cum se vede, s-au format două componente conexe: $C_1 = \{P_1,P_2,P_6\}$
$C_2 = \{O,P_3,P_4,P_5,P_7,P_8\}$.

Vom alege: pentru $C_1 \rightarrow$ arcul OP_6
pentru $C_2 \rightarrow$ arcul OP_6

şi obținem o singură componentă conexă, care este arborele căutat.

C. Varianta algoritmului lui Kruskal

Succesiunea alegerii arcelor va fi:

1 $\rightarrow$ OP_3
2 $\rightarrow$ OP_7
3 $\rightarrow$ OP_6
4 $\rightarrow$ OP_5
5 $\rightarrow$ P_1P_6
6 $\rightarrow$ P_1P_2
7 $\rightarrow$ P_4P_5
8 $\rightarrow$ P_5P_8

6. Cuplajul a două mulțimi disjuncte. Probleme de afectare (de repartiție)

În practica economică sunt foarte des întâlnite probleme în care se doreşte asocierea optimă a elementelor unei mulțimi $X = \{x_1, x_2, ... , x_n\}$ cu elementele unei alte mulțimi $Y = \{y_1, y_2, ... , y_m\}$ în condițiile unor limitări existente (şi cunoscute) ale posibilităților de asociere.

În general, fiecare asociere posibilă $x_i \leftrightarrow y_j$ aduce un anumit efect a_{ij} (profit, cost etc) care poate fi calculat şi vom presupune că este cunoscut.

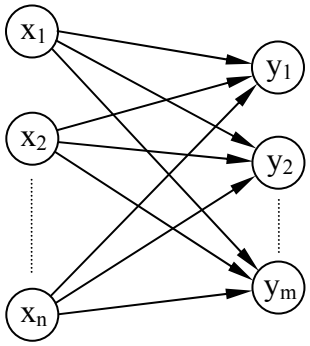
Limitările asupra asocierilor se traduc de obicei prin faptul că:

1. Un element x_i poate fi asociat doar cu anumite elemente din Y şi reciproc;
2. La sfârşit, fiecărui element din X i s-a asociat cel mult un element din Y şi reciproc.

Asocierea optimă presupune, de obicei, două obiective:

1. Să se facă maximul de asocieri;
2. Suma efectelor asocierilor să fie maximă (sau minimă, în funcție de semnificația acestora).

Reprezentarea geometrică a situației de mai sus este un graf de forma:



numit **graf bipartit**.

Definiția 1: Se numeşte **graf bipartit** un graf $G = (X, U)$ în care mulțimea nodurilor poate fi împărțită în două mulțimi disjuncte A şi B astfel încât orice arc are extremitatea inițială în A şi cea finală în B.

Definiția 2: Se numeşte **cuplaj** al unui graf bipartit o submulțime de arce $W \subseteq U$ cu proprietatea că nu există două arce adiacente (sau altfel spus, pentru orice nod există **cel mult** un arc incident acestuia).

Definiția 3: Se numeşte **cuplaj maxim** un cuplaj cu proprietatea că orice arc care nu face parte din cuplaj este adiacent cu un arc din cuplaj ($\Leftrightarrow$ orice arc am adăuga, nu mai rămâne cuplaj $\Leftrightarrow$ nu există nici un cuplaj în care să se includă strict $\Leftrightarrow$ conține numărul maxim de arce neadiacente)

Este evident că numărul de arce ale unui cuplaj este mai mic sau egal cu numărul de elemente din fiecare din mulțimile A şi B ($\leq \min(|A|, |B|)$. Este interesant de văzut însă cât de mare este el efectiv şi în ce condiții este egal chiar cu $\min(|A|, |B|)$.

Referitor la prima întrebare, în 1931 König a demonstrat o teoremă care permite stabilirea numărului de arce ale unui cuplaj maxim:

Teoremă: Numărul maxim de arce ale unui cuplaj într-un graf bipartit G = (A∪B, Γ) este egal cu $\min_{C \subset A}\left(|A - C| + |\Gamma(C)|\right)$

În ceea ce priveşte a doua problemă, observăm mai întâi că putem presupune că întotdeauna $|A| \le |B|$, în caz contrar inversând sensul tuturor arcelor grafului, problema rămânând aceeaşi.

În acest caz:

$$\min_{C \subset A}\left(|A - C| + |\Gamma(C)|\right) = |A| \Leftrightarrow \min_{C \subset A}\left(|A - C| + |\Gamma(C)|\right) - |A| = 0 \Leftrightarrow \min_{C \subset A}\left(-|C| + |\Gamma(C)|\right) = 0 \Leftrightarrow$$

$$\Leftrightarrow \max_{C \subset A}\left(|C| - |\Gamma(C)|\right) = 0 \Leftrightarrow \Gamma(C) \ge |C| \text{ oricare ar fi } C \subseteq A$$

sau altfel spus, pentru orice submulțime C a lui A, mulțimea nodurilor atinse de arce care pleacă din nodurile sale, adică Γ(C), are cel puțin atâtea elemente cât C.

De exemplu, la repartizarea angajaților pe posturi, fiecare angajat poate obține un post dorit dacă şi numai dacă oricare ar fi mulțimea de r angajați există cel puțin r posturi diferite din care pot alege.

Presupunem, în continuare, că s-a asociat fiecărui arc (x_i,x_j) o valoare v_{ij}.

Definiția 4: Se numeşte valoare a unui cuplaj suma valorilor arcelor care îl formează.

În acest moment putem spune că determinarea unei asocieri optime a mulțimilor X şi Y de la început este echivalentă matematic cu determinarea unui cuplaj maxim de valoare optimă (minimă sau maximă) în graful bipartit asociat.

Dintre problemele întâlnite în practica economică, ce se reduc matematic la găsirea unui cuplaj maxim de valoare optimă, amintim:

1. Problema repartizării muncitorilor unei secții la utilajele acesteia în funcție de pregătirea şi preferințele muncitorilor, complexitatea maşinilor etc;
2. transferarea unor informații într-un grup;
3. Repartizarea angajaților pe posturi;
4. Formarea grupelor de lucru după afinitățile dintre membrii colectivului.

În 1955, bazându-se pe teorema lui König, H.W. Kuhn a elaborat un algoritm, cunoscut în literatura de specialitate sub denumirea de **algoritmul ungar**, cu ajutorul căruia se poate determina un cuplaj maxim de valoare minimă într-un graf bipartit pentru care $|A| = |B| = n$.

El se bazează pe observația că, dacă se adună (sau scade) aceeaşi număr la toate valorile arcelor, nu se modifică ierarhia cuplajelor maxime, în ceea ce priveşte valoarea lor.

Vom prezenta algoritmul concomitent cu rezolvarea unui caz particular, pentru o mai bună receptare a acestuia:

"Într-o secție produsele finite se obțin în urma efectuării succesive a 6 operații pe 6 maşini. În această secție sunt angajați 6 muncitori, fiecare fiind calificat pentru efectuarea oricărei din cele 6 operații. Pentru a optimiza activitatea în secție cei 6 muncitori au fost supuşi la un test în care fiecare a prelucrat un număr de piese, pe toate cele şase maşini. În final, calculându-se timpul mediu în care muncitorul M_i efectuează operația O_j s-au obținut valorile (în ore) date în tabelul de mai jos:

	M_1	M_2	M_3	M_4	M_5	M_6
O_1	4	3	6	2	6	8
O_2	5	4	8	3	8	9
O_3	5	6	8	2	8	7
O_4	4	5	7	2	7	8
O_5	4	6	6	3	6	7
O_6	6	6	8	3	8	9

Să se găsească acea repartiție a muncitorilor la maşini astfel încât timpul în care o piesă se prelucrează succesiv pe cele 6 maşini să fie minim."

Pasul 1. Se construieşte matricea pătratică M care are elementele:

$$m_{ij} = \begin{cases} \text{valoarea arcului } (x_i, x_j) & \text{daca exista arcul } (x_i, x_j) \\ \infty & \text{daca nu exista arcul } (x_i, x_j) \end{cases}$$

Pentru exemplul ales vom avea: $M = \begin{pmatrix} 4 & 3 & 6 & 2 & 6 & 8 \\ 5 & 4 & 8 & 3 & 8 & 9 \\ 5 & 6 & 8 & 2 & 8 & 7 \\ 4 & 5 & 7 & 2 & 7 & 8 \\ 4 & 6 & 6 & 3 & 6 & 7 \\ 6 & 6 & 8 & 3 & 8 & 9 \end{pmatrix}$

Pasul 2. Se scade din fiecare linie minimul acesteia apoi, în matricea obținută, din fiecare coloană minimul acesteia (se poate face şi invers, rezultatul final va fi acelaşi). Pentru exemplul dat vom obține succesiv matricile:

$$M_1 = \begin{pmatrix} 2 & 1 & 4 & 0 & 4 & 6 \\ 2 & 1 & 5 & 0 & 5 & 6 \\ 3 & 4 & 6 & 0 & 6 & 5 \\ 2 & 3 & 5 & 0 & 5 & 6 \\ 1 & 3 & 3 & 0 & 3 & 4 \\ 3 & 3 & 5 & 0 & 5 & 6 \end{pmatrix} \text{ şi apoi } M_2 = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 2 \\ 1 & 0 & 2 & 0 & 2 & 2 \\ 2 & 3 & 3 & 0 & 3 & 1 \\ 1 & 2 & 2 & 0 & 2 & 2 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 0 & 2 & 2 \end{pmatrix}$$

Ultima matrice este cea asupra căreia se aplică următoarele calcule. În acest moment pe fiecare linie şi pe fiecare coloană se află cel puțin un 0, care corespunde celui mai mic timp. Se încearcă în continuare folosirea doar a acestor repartizări:

Pasul 3. În ordinea crescătoare a numărului de zerouri şi de sus în jos (în cazul existenței mai multor linii cu acelaşi număr de zerouri) se încadrează pentru fiecare linie zeroul a cărui coloană conține cele mai puține zerouri (primul de la stânga dintre acestea, în caz de egaliate) şi se barează celelalte zerouri de pe linia şi coloana acestuia. Pe parcursul algoritmului sunt luate în considerare la numărare doar zerourile neîncadrate şi nebarate încă. În final, pe fiecare linie şi pe fiecare coloană va fi cel mult un zero încadrat. Dacă în final sunt n (= dimensiunea matricei) zerouri, atunci arcele corespunzătoare formează cuplajul căutat. Dacă sunt mai puține se trece la pasul 4.

În exemplul nostru avem trei linii cu câte un zero (a 3-a, a 4-a şi a 5-a) .Îl încadrăm pe cel de pe linia 3 (prima dintre ele) şi barăm restul zerourilor de pe linia 3 şi coloana 3, obținând:

$$\begin{pmatrix} 1 & 0 & 1 & \cancel{0} & 1 & 2 \\ 1 & 0 & 2 & \cancel{0} & 2 & 2 \\ 2 & 3 & 3 & \boxed{0} & 3 & 1 \\ 1 & 2 & 2 & \cancel{0} & 2 & 2 \\ 0 & 2 & 0 & \cancel{0} & 0 & 0 \\ 2 & 2 & 2 & \cancel{0} & 2 & 2 \end{pmatrix}$$

În acest moment pe liniile 1 şi 2 se află un zero. Se încadrează cel de pe linia 1 şi se barează celelalte de pe linia 1 şi coloana 2, obținând:

$$\begin{pmatrix} 1 & \boxed{0} & 1 & \not{0} & 1 & 2 \\ 1 & \not{0} & 2 & \not{0} & 2 & 2 \\ 2 & 3 & 3 & \boxed{0} & 3 & 1 \\ 1 & 2 & 2 & \not{0} & 2 & 2 \\ 0 & 2 & 0 & \not{0} & 0 & 0 \\ 2 & 2 & 2 & \not{0} & 2 & 2 \end{pmatrix}$$

Ultima linie cu zerouri este linia 5 din care îl încadrăm pe primul şi le barăm pe celelalte:

$$\begin{pmatrix} 1 & \boxed{0} & 1 & \not{0} & 1 & 2 \\ 1 & \not{0} & 2 & \not{0} & 2 & 2 \\ 2 & 3 & 3 & \boxed{0} & 3 & 1 \\ 1 & 2 & 2 & \not{0} & 2 & 2 \\ \boxed{0} & 2 & \not{0} & \not{0} & \not{0} & \not{0} \\ 2 & 2 & 2 & \not{0} & 2 & 2 \end{pmatrix}$$

În total nu sunt 6 zerouri încadrate (sunt doar trei) şi deci trecem la pasul 4.

Pasul 4. La acest pas se va stabili numărul minim posibil de linii şi coloane care să conțină toate zerourile matricii. În acest sens vom proceda astfel:

a) se marchează liniile care nu au nici un zero încadrat;
b) se marchează coloanele care au un zero barat pe o linie marcată;
c) se marchează liniile care au un zero încadrat pe o linie marcată (dacă există);
Se repetă operațiile b) şi c) până nu mai poate fi marcată nici o linie şi nici o coloană.

În cazul nostru vom avea: a) → se marchează liniile 2, 4 şi 6;
b) → se marchează coloanele 2 şi 4;
c) → se marchează liniile 1 şi 3;
b) → nu mai marcăm nici o coloană deoarece nu mai există nici un zero barat pe liniile 1 şi 3, care să corespundă unei coloane nemarcate;
c) → nu mai marcăm nici o linie, deoarece nu a mai apărut nici o coloană marcată.

Rezultă:

$$\begin{array}{c} \quad\ \ \text{☆} \quad\ \ \text{☆} \\ \begin{pmatrix} 1 & \boxed{0} & 1 & \not{0} & 1 & 2 \\ 1 & \not{0} & 2 & \not{0} & 2 & 2 \\ 2 & 3 & 3 & \boxed{0} & 3 & 1 \\ 1 & 2 & 2 & \not{0} & 2 & 2 \\ \boxed{0} & 2 & \not{0} & \not{0} & \not{0} & \not{0} \\ 2 & 2 & 2 & \not{0} & 2 & 2 \end{pmatrix} \begin{matrix} \text{☆} \\ \text{☆} \\ \text{☆} \\ \text{☆} \\ \\ \text{☆} \end{matrix} \end{array}$$

Pasul 5. Se taie liniile nemarcate şi coloanele marcate:

$$\begin{array}{c} \quad\ \ \text{☆} \quad\ \ \text{☆} \\ \begin{pmatrix} 1 & \boxed{0} & 1 & \not{0} & 1 & 2 \\ 1 & \not{0} & 2 & \not{0} & 2 & 2 \\ 2 & 3 & 3 & \boxed{0} & 3 & 1 \\ 1 & 2 & 2 & \not{0} & 2 & 2 \\ \boxed{0} & 2 & \not{0} & \not{0} & \not{0} & \not{0} \\ 2 & 2 & 2 & \not{0} & 2 & 2 \end{pmatrix} \begin{matrix} \text{☆} \\ \text{☆} \\ \text{☆} \\ \text{☆} \\ \\ \text{☆} \end{matrix} \end{array}$$

Pasul 6. Se împart elementele matricei în trei grupe:

G_1 = elemente aflate la intersecții de linii netăiate cu coloane netăiate;

G_2 = elemente situate la intersecții de linii tăiate cu coloane netăiate sau de linii netăiate cu coloane tăiate;
G_3 = elemente situate la intersecții de coloane tăiate cu linii tăiate

Pasul 7. Se găseşte minimul grupei G_1, care se scade din fiecare element al lui G_1 şi se adună la fiecare element al grupei G_3. Elementele grupei G_2 rămân neschimbate.
Pentru exemplul dat, minimul lui G_1 este 1 şi obținem noua matrice:

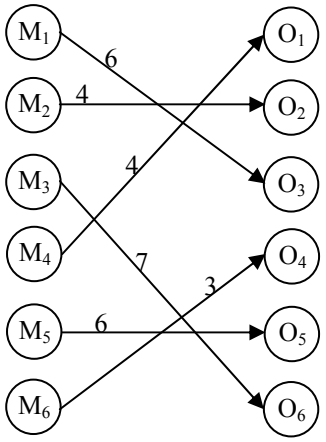
$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 3 & 2 & 0 & 2 & 0 \\ 0 & 2 & 1 & 0 & 1 & 1 \\ 0 & 3 & 0 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Pasul 8. Se reia algoritmul de la pasul 3.

Vom avea după marcare:

$$\begin{pmatrix} \not{0} & \not{0} & \boxed{0} & \not{0} & \not{0} & 1 \\ \not{0} & \boxed{0} & 1 & \not{0} & 1 & 1 \\ 1 & 3 & 2 & \not{0} & 2 & \boxed{0} \\ \boxed{0} & 2 & 1 & \not{0} & 1 & 1 \\ \not{0} & 3 & \not{0} & 1 & \boxed{0} & \not{0} \\ 1 & 2 & 1 & \boxed{0} & 1 & 1 \end{pmatrix}$$

Deoarece avem 6 zerouri încadrate, am obținut cuplajul maxim de valoare minimă căutat, căruia îi va corespunde repartizarea muncitorilor pe operații de mai jos:

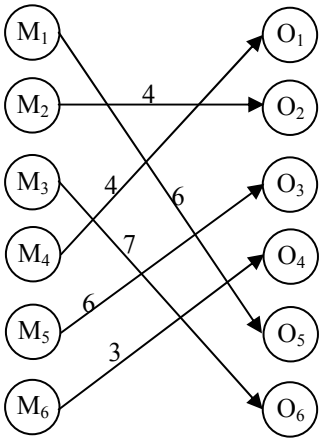


care duce la o durată totală a prelucrării unei piese de 6 + 4 + 7 + 6 + 3 = 26 ore

Observație: Deoarece regula de a alege de sus în jos la linii cu acelaşi număr de zerouri este arbitrară şi de asemenea alegerea primului zero de la stânga, putem ajunge şi la alte cuplaje maxime, dar toate vor avea aceeaşi valoare, cea minimă. De exemplu, un alt cuplaj optim este:

$$\begin{pmatrix} \not{0} & \not{0} & \not{0} & \not{0} & \boxed{0} & 1 \\ \not{0} & \boxed{0} & 1 & \not{0} & 1 & 1 \\ 1 & 3 & 2 & \not{0} & 2 & \boxed{0} \\ \boxed{0} & 2 & 1 & \not{0} & 1 & 1 \\ \not{0} & 3 & \boxed{0} & 1 & \not{0} & \not{0} \\ 1 & 2 & 1 & \boxed{0} & 1 & 1 \end{pmatrix}$$

adică



care are de asemenea valoarea 26.

Observația 1. Dacă dorim un cuplaj de valoare maximă atunci vom calcula la pasul 1 matricea M astfel:

1. Construind matricea A de elemente:

$$a_{ij} = \begin{cases} \text{valoarea arcului } (x_i, x_j) & \text{daca exista arcul } (x_i, x_j) \\ -\infty & \text{daca nu exista arcul } (x_i, x_j) \end{cases}$$

2. Matricea M va avea componentele: $m_{ij} = \max_{1 \le i, j \le n} (a_{ij}) - a_{ij}$

apoi aplicăm în continuare algoritmul.

Observația 2. Dacă $|A| \neq |B|$ atunci aplicăm același algoritm cu singura diferență că ne vom opri când vom obține un număr de zerouri egal cu min ($|A|, |B|$).

7. Drumuri şi circuite hamiltoniene

Una dintre cele mai cunoscute probleme economice este problema comis voiajorului. Comis voiajorul este un individ care trebuie să prezinte s-au să distribuie marfa comandată la o serie de centre distribuite în general neliniar pe o anumită zonă teritorială (localităţile dintr-un judeţ, magazinele dintr-un cartier, persoanele dintr-un sat etc). Dacă numărul de obiective care trebuie vizitate este mare sau foarte mare iar timpul disponibil foarte limitat atunci devine vitală o asemenea organizare a trecerii pe la fiecare obiectiv încât să se efectueze în timpul minim posibil. Acest timp minim se traduce prin drumul cel mai scurt, iar cel mai scurt drum este evident cel în care se trece pe la fiecare obiectiv o singură dată. În plus, la sfârşit trebuie să se afle în punctul iniţial, adică sediul firmei la care lucrează.

O reprezentare a regiunii aprovizionate, în care centrele pe la care se trece sunt vizualizate prin puncte iar căile de acces la acestea prin segmente de curbe, va fi evident un graf, problema reducându-se la a găsi circuitul hamiltonian de lungime minimă.

În timp, s-au evidenţiat o multitudine de probleme reductibile la găsirea unui drum (sau circuit) hamiltonian într-un graf, cum ar fi:

1. Problema poştaşului (găsirea traseului cel mai scurt care trece pe la toate locuinţele ce aparţin de oficiul poştal la care lucrează acesta);
2. Problema adunării deşeurilor (cel mai scurt drum care trece pe la toate punctele de depozitate a deşeurilor);
3. Problema succesiunii operaţiilor (executarea mai multor operaţii pe o maşină în acea ordine în care suma timpilor consumaţi cu pregătirea maşinii pentru trecerea de la o operaţie la următoarea să fie minim)
4. Ordinea lipirii unor componente electronice pe o placă, etc;

Determinarea drumurilor hamiltoniene

Problema determinării drumului (circuitului) hamiltonian de valoare optimă s-a dovedit deosebit de dificilă, neexistând nici acum un algoritm care să rezolve problema în timp polinomial şi nici măcar o metodă simplă prin care să se decidă dacă într-un graf dat există sau nu drumuri hamiltoniene.

Există însă mai mulţi algoritmi, unii exacţi alţii heuristici, care reuşesc, într-un caz sau altul, să rezolve problema satisfăcător şi în timp util.

A. Algoritmul lui Foulkes

Pasul 1. Se scrie matricea booleană A asociată grafului G.

Pasul 2. Se determină matricea D a drumurilor grafului G prin procedeul expus la începutul capitolului şi apoi matricea $M = I + D$.

Pasul 3. Se împarte mulţimea nodurilor grafului în submulţimi disjuncte astfel:

1. Se consideră în matricea M liniile pline (cu toate elementele 1). Nodurile ce corespund liniilor pline cu 1 formează submulţimea C_1.
2. Se elimină liniile şi coloanele care corespund nodurilor din submulţimea stabilită.
3. Se reia raţionamentul de la punctul 1 pe matricea redusă obţinută la punctul 2 obţinându-se următoarea submulţime şi în continuare toate celelalte până se epuizează toate liniile matricei.

Pasul 4. Se construieşte graful G' în care:

1. Nodurile care formează o submulțime sunt reprezentate prin puncte în interiorul unui dreptunghi şi între acestea se trasează arcele existente în graful inițial G.
2. Se trasează legăturile dintre submulțimi. Ele sunt reprezentate prin arcele existente în graful inițial G între nodurile submulțimii C_1 şi cele ale submulțimii C_2, între nodurile submulțimii C_2 şi cele ale submulțimii C_3 etc.

Pasul 5. Se găsesc drumurile hamiltoniene

Un drum hamiltonian se găseşte plecând de la un vârf din submulțimea C_1, trecând prin toate vârfurile acesteia cu un drum hamiltonian, din ultimul vârf la care se ajunge în C_1 trecând la un vârf din C_2, parcurgând în continuare un drum hamiltonian în a doua submulțime şi tot aşa, trecând prin toate submulțimile şi parcurgând, deci, toate nodurile grafului inițial, o singură dată. Aplicând acest procedeu în toate modurile posibile se obțin toate drumurile hamiltoniene din graful inițial G. (*Observație*: poate să nu existe nici un drum hamiltonian în graful G, caz în care algoritmul se opreşte deoarece la un anumit pas nu mai exista nici o linie plina cu 1).

Observație. Algoritmul lui Foulkes reduce găsirea drumurilor hamiltoniene în graful inițial G (care în problemele practice este foarte mare) la găsirea mai multor drumuri hamiltoniene mai mici în componente tare conexe ale grafului. Dacă un graf are o singură componentă tare conexă, algoritmul lui Foulkes nu este eficient, în acest caz trebuind aplicați alți algoritmi cum ar fi cel bazat pe înmulțirea latină.

B. Algoritmul lui Chen pentru determinarea drumurilor hamiltoniene în grafuri fără circuite

Fie G = (X,U) un graf orientat fără circuite, cu n noduri: $X = \{x_1, x_2, \dots, x_n\}$. Vom considera că am calculat matricea drumurilor D şi puterile de atingere ale tuturor nodurilor.

Dacă în graful G există un drum de la nodul x_i la nodul x_j atunci evident $p(x_i) > p(x_j)$, deoarece în orice vârf în care se poate ajunge din x_j se poate ajunge şi din x_i dar din x_j nu se poate ajunge în x_j pentru că nu există circuite.

Teorema 2.3 (Chen) *Un graf cu n noduri, fără circuite conține un drum hamiltonian dacă şi numai dacă există relația:*

$$\sum_{i=1}^{n} p(x_i) = \frac{n(n-1)}{2}$$

Demonstrație

"$\Rightarrow$" Fie *H* un drum hamiltonian şi presupunem că nodurile grafului au fost notate în ordinea în care apar în acest drum. Atunci din orice nod x_i se poate ajunge în toate nodurile cu indice mai mare şi numai în acestea (altfel ar exista circuite) şi deci puterea unui nod x_i este n – i, de unde:

$$\sum_{i=1}^{n} p(x_i) = (n-1) + (n-2) + \dots + 1 + 0 = \frac{n(n-1)}{2}$$

"$\Leftarrow$" Ordonând vârfurile în ordinea descrescătoare a puterii lor de atingere ($i > j \Leftrightarrow p(x_i) < p(x_j)$) şi cum graful nu are circuite, vom obține o matrice D cu toate zerourile deasupra diagonalei (evident pe o poziție (i,i) nu se află nici un 1 iar dacă ar fi un 1 pe poziția (i,j) cu i > j ar însemna că

din x_i se poate ajunge în x_j, deci în toate nodurile în care se poate ajunge din x_j, iar din x_j nu se poate ajunge în x_i, deci $p(x_i) > p(x_j)$ în contradicție cu ipoteza de ordonare a nodurilor). Cum deasupra diagonalei sunt $\frac{n(n-1)}{n}$ poziții iar suma puterilor vârfurilor este chiar $\frac{n(n-1)}{n}$ rezultă că toate pozițiile de deasupra diagonalei sunt 1. Aceasta înseamnă că există toate arcele de forma (x_i,x_{i+1}) (altfel n-ar exista drum de la x_i la x_{i+1}, deoarece toate drumurile au indicii nodurilor în ordine descrescătoare) şi deci drumul hamiltonian $(x_1, x_2, \dots, x_n)$ q.e.d.

Teorema 2.4 Dacă într-un graf orientat fără circuite există un drum hamiltonian atunci acesta este unic.

Demonstrație Deoarece un drum hamiltonian se identifică cu o permutare a nodurilor grafului, existența a două drumuri hamiltoniene implică existența a două permutări distincte a nodurilor grafului şi cum două permutări distincte diferă prin cel puțin o inversiune vor exista două noduri x_i şi x_j în ordinea $x_i \to x_j$ pe un drum şi invers pe celălalt, existând deci un drum atât de la x_i la x_j cât şi de la x_j la x_i, cele două formând împreună un circuit, în contradicție cu ipoteza.

Pe aceste teoreme se bazează **algoritmul lui Chen** de determinare a drumului hamiltonian într-un graf orientat fără circuite:

Pasul1. Se scrie matricea de adiacență A
Pasul2. Se calculează matricea drumurilor D
Pasul3. Dacă există un indice i cu $d_{ii} = 1$ atunci graful are circuite, nu se poate aplica algoritmul lui Chen şi algoritmul se opreşte. Dacă nu, se trece la pasul 4.
Pasul4. Se calculează puterile de atingere pentru fiecare nod.
Pasul5. Dacă nu se verifică relația $\sum_{i=1}^{n} p(x_i) = \frac{n(n-1)}{2}$ atunci graful nu are drumuri hamiltoniene şi algoritmul se opreşte, altfel se trece la pasul 6.
Pasul6. Se ordonează nodurile în ordinea descrescătoare a puterilor lor de atingere şi obținem drumul hamiltonian căutat.

C. Algoritmul lui Kaufmann

Pasul 1. Construim matricea latină L asociată grafului, unde:

$$l_{ij} = \begin{cases} x_i x_j & \text{dacă există arcul } (x_i, x_j) \\ 0 & \text{dacă nu există arcul } (x_i, x_j) \end{cases}$$

Pasul 2. Construim matricea $\tilde{L}$, definită prin:

$$\tilde{l}_{ij} = \begin{cases} x_j & \text{dacă există arcul } (x_i, x_j) \\ 0 & \text{dacă nu există arcul } (x_i, x_j) \end{cases}$$

numită **matricea latină redusă**.

Pasul 3. Se calculează succesiv matricile:

$$L^2 = L \times_L \tilde{L},\ L^3 = L^2 \times_L \tilde{L},\ \dots,\ L^{k+1} = L^k \times_L \tilde{L},\ \dots$$

folosind operațiile de înmulțire şi adunare latină, alfabetul fiind mulțimea nodurilor grafului, unde operația de înmulțire este uşor modificată, produsul dintre două elemente ale matricilor fiind 0, dacă unul dintre ele este 0 sau au un nod comun, şi este produsul latin al lor, în caz contrar.

Din felul cum a fost construită, matricea L^k va conține toate drumurile elementare de lungime k. Cum un drum elementar poate avea cel mult n noduri (câte are graful cu totul) rezultă că:

- primele n-1 puteri ale L conțin toate drumurile elementare din graf;
- puterile lui L mai mari sau egale cu n au toate elementele egale cu 0;
- matricea L^{n-1} conține toate drumurile hamiltoniene din graf.

Pasul 4. Dacă se doresc şi circuitele atunci se verifică pentru fiecare drum hamiltonian dacă poate fi completat până la un circuit (adică dacă există în graf arcul care uneşte nodul final cu cel inițial);

Pasul 5. Dacă se doreşte şi drumul (sau circuitul) de valoare optimă (maximă sau minimă) se calculează suma valorilor pentru fiecare drum şi/sau circuit şi se alege cel cu valoarea optimă.

În concluzie, metoda înmulțirii latine (A. Kaufmann – J. Melgrange) determină toate drumurile elementare din graf, prin calcularea matricelor $M^{(1)}$, $M^{(2)}$, $M^{(3)}$, …, $M^{(n-1)}$.

În matricea $M^{(n-1)}$ se citesc drumurile hamiltoniene.

Această metodă a înmulțirii latine (algoritmul lui Kaufmann) este utilă, mai ales, în cazul grafurilor tare conexe, unde algoritmul lui Foulkes nu este eficient. Totuşi, metoda este greu de aplicat în grafuri cu un număr mare de noduri. În acest caz este preferabil să se construiască graful condensat, să se determine drumurile hamiltoniene în fiecare în parte cu algoritmul lui Kaufmann şi apoi, ca la algoritmul lui Foulkes, să se caute drumurile hamiltoniene în graful inițial.

D. Un algoritm bazat pe algoritmul ungar

Fie G = (X,U) un graf orientat cu n noduri $X = \{x_1, x_2, \dots, x_n\}$.

Pasul 1. Se construieşte graful bipartit $H = (A \cup B, V)$ în care A = B = X şi V = U (adică am folosit pentru G reprezentarea prin corespondență).

Pasul 2. Se găseşte pentru graful H cuplajul maxim de valoare minimă.

Pasul 3. Se construieşte graful parțial al lui G format doar cu arcele cuplajului găsit. Este uşor de demonstrat că, componentele tare conexe ale acestuia sunt toate nişte circuite. Dacă s-a format un singur circuit acesta este circuitul hamiltonian de valoare minimă. Dacă s-au format mai multe se trece la pasul 4.

Pasul 4. Pentru fiecare arc aflat pe circuitul de lungime minimă (dacă sunt mai multe se iau în considerare arcele tuturor) se reia algoritmul de la pasul 1 pentru graful parțial rezultat din G prin eliminarea acestui arc.

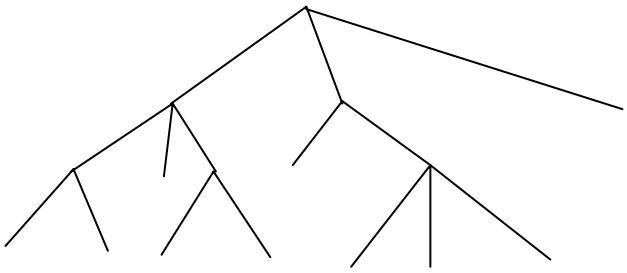
Pasul 5. Pentru fiecare graf parțial se continuă procedeul până se ajunge la unul din cazurile:

Cazul1. Cuplajului maxim găsit îi corespunde un singur circuit. Dacă acest circuit este primul obținut atunci valoarea sa i se atribuie unei variabile Z şi circuitul este păstrat. Dacă nu este primul atunci valoarea sa se compară cu Z şi, dacă este mai mică, ea devine noua valoare a lui Z şi circuitul se păstrează, eliminându-l pe cel corespunzător fostei valori a lui Z. În caz contrar se trece la alt graf parțial neanalizat încă.

Cazul2. Cuplajul maxim are o valoare mai mare decât Z. Pentru acest graf parțial se abandonează ramificarea.

Pasul 6. Se continuă analiza grafurilor parțiale până sunt analizate toate ramificațiile. Valoarea Z finală este valoarea circuitului de valoare minimă iar circuitul corespunzător este cel optim.

Analiza de mai sus poate fi schematizată printr-un arbore de tipul:



în care fiecare nod este un graf parțial de analizat, iar pentru fiecare arc, nodul inferior este un graf parțial care provine din graful corespunzător nodului superior, prin suprimarea unui arc de pe circuitele de lungime minimă corespunzătoare cuplajului maxim de valoare minimă al acestuia.

Observație: Algoritmul asigură găsirea circuitului de valoare minimă iar în cazul în care algoritmul lui Foulkes nu funcționează este o alternativă mai bună decât algoritmul lui Kaufmann. Totuşi el nu lucrează în timp polinomial şi în unele cazuri (de exemplu cazuri în care se formează foarte multe cicluri cu lungime minimă) necesită un număr imens de calcule.

În aceste cazuri se pot folosi metode euristice prin care se elimină din start o serie de arce, considerate a avea valori prea mari pentru a se putea afla pe circuitul hamiltonian de valoare minimă, apoi se aplică în graful parțial rămas unul din algoritmii de mai sus.

8. Drumuri optime într-un graf

În marea majoritate a problemelor care pot fi modelate prin grafuri nu ne interesează numai dacă există sau nu legături între componentele reprezentate prin nodurile grafului ci şi intensitatea acestora. Această intensitate are semnificația unei valori numerice (pozitive sau negative) asociate arcului corespunzător legăturii a cărei intensitate o măsoară.

În aplicațiile economice această valoare poate fi:

- lungimea drumului dintre două localități;
- costul parcurgerii rutei reprezentate prin arcul corespunzător;
- durata parcurgerii rutei respective;
- cantitatea transportată pe ruta respectivă;
- capacitatea maximă a rutei respective;
- câştigul realizat prin trecerea de la o stare la alta a sistemului;
- consum de energie pentru efectuarea trecerii respective;
- punctaj realizat etc.

Una din problemele care poate apărea în aceste situații este găsirea, pentru o anumită pereche de noduri (sau mai multe perechi), a drumului optim între acestea.

Pentru formalizarea problemei vom introduce noțiunea de **valoare a unui drum**, care este egală cu suma valorilor arcelor care îl compun. Vom nota în continuare valoarea unui arc (x_i,x_j) cu $v(x_i,x_j)$ sau cu v_{ij}. În aceste condiții putem enunța problema drumului optim astfel:

"***Dat un graf G = (X,U) şi o funcție care asociază fiecărui arc o valoare reală, să se găsească, pentru o pereche dată de noduri, drumul (drumurile) de valoare optimă (minimă sau/şi maximă) între cele două noduri şi valoarea acestuia (acestora)***"

Deoarece este vorba de găsirea minimului unei mulțimi de numere reale, prima întrebare care se pune este dacă aceasta admite minim. Dacă mulțimea nodurilor grafului este infinită atunci pot exista o infinitate de drumuri elementare distincte între cele două noduri şi mulțimea valorilor acestora poate avea orice formă (închisă sau nu, mărginită sau nu) devenind foarte greu de caracterizat cazurile când minimul dorit există. Deoarece totuşi majoritatea covârşitoare a problemelor economice se modelează prin grafuri cu număr finit de noduri, ne vom limita în continuare doar la acestea.

Un număr finit de noduri *n* atrage după sine existența unui număr finit de arce (cel mult n^2) şi a unui număr finit de drumuri elementare (cel mult $n \cdot n! \cdot \sum_{k=1}^{n-1} \frac{1}{k!}$). Deoarece oricărui drum *d* îi corespunde un drum elementar d_e (obținut prin eliminarea tuturor subcircuitelor lui *d*) putem calcula valoarea oricărui drum ca sumă între valoarea drumului elementar corespunzător şi valorile unor subcircuite ale sale, fiecare înmulțită cu numărul de parcurgeri ale circuitului respectiv.

În concluzie, dacă există un circuit de valoare negativă înseamnă că există drumuri de valoare oricât de mică (cele care conțin acest circuit), obținută prin parcurgerea acestuia de oricâte ori dorim) şi, deci, mulțimea valorilor drumurilor este nemărginită inferior, neexistând drum de valoare minimă. Dacă există un circuit de valoare pozitivă atunci există drumuri de valoare oricât de mare şi mulțimea valorilor drumurilor este nemărginită superior, neexistând drum de valoare maximă.

Dacă nu există circuite de valoare negativă atunci valoarea oricărui drum este mai mare sau egală cu a drumului elementar corespunzător, deci drumul de valoare minimă (dacă există) va fi un drum elementar. Cum mulțimea drumurilor elementare este finită (şi deci şi mulțimea valorilor lor) va avea minorant şi am lămurit problema compatibilității problemei. Analog, dacă nu există circuite de valoare pozitivă atunci valoarea oricărui drum este mai mică sau egală cu a drumului elementar

corespunzător, deci drumul de valoare maximă (dacă există) va fi un drum elementar. Cum mulțimea drumurilor elementare este finită (şi deci şi mulțimea valorilor lor), va avea majorant.

Obs. 1. Dacă în graf nu există decât arce de valoare pozitivă atunci există drum de valoare minimă.

Obs. 1. Dacă în graf nu există decât arce de valoare negativă atunci există drum de valoare maximă.

Obs. 1. Dacă în graf nu există circuite atunci există şi drum de valoare minimă şi drum de valoare maximă.

Deoarece din cele de mai sus se sesizează importanța existenței circuitelor într-un graf vom da în continuare un **algoritm de depistare a existenței circuitelor într-un graf**:

Pasul 1. Se construieşte mulțimea A formată din nodurile pentru care toate arcele incidente sunt incidente spre interior (noduri în care toate arcele "intră" sau, altfel spus, noduri din care nu "pleacă" nici un arc).

Pasul 2. Se găsesc toate nodurile care nu sunt din A pentru care toate arcele incidente au cealaltă extremitate în A (noduri din care se poate "ajunge" doar in A). Dacă nu există nici un astfel de arc se trece la pasul 4.

Pasul 3. Se adaugă arcele găsite la pasul 2 la mulțimea A apoi se reia algoritmul de la pasul 2, pentru noua mulțime A.

Pasul 4. Dacă A conține mulțimea tuturor nodurilor atunci graful nu conține circuite. Dacă au rămas noduri în afara lui A atunci graful conține circuite.

Algoritmi de găsire a drumului optim

Din cauza varietății nelimitate a grafurilor posibile, nu există un algoritm care să rezolve orice problemă în timp util, dar s-au elaborat o mulțime de algoritmi, fiecare fiind cel mai eficace în anumite cazuri. Aceşti algoritmi pot fi grupați în cinci categorii:

1. Algoritmi prin calcul matricial (Bellman-Kalaba, I. Tomescu, Bellman-Schimbell);
2. Algoritmi prin ajustări succesive: (Ford);
3. Algoritmi prin inducție (Dantzig);
4. Algoritmi prin ordonare prealabilă a vârfurilor grafului;
5. Algoritmi prin extindere selectivă (Dijkstra).

În continuare vom prezenta trei dintre aceşti algoritmi.

A. Algoritmul lui Bellman - Kalaba

Algoritmul se aplică în grafuri finite care nu au circuite de valoare negativă (pentru o problemă de minim) sau care nu au circuite de valoare pozitivă (într-o problemă de maxim) şi găseşte drumurile de valoare minimă (maximă) de la toate nodurile grafului la un nod oarecare, fixat. Dacă dorim să cunoaştem drumurile de valoare minimă (maximă) între oricare două noduri vom aplica algoritmul, pe rând, pentru fiecare nod al grafului.

Fie $G = \{x_1, x_2, ... ,x_n\}$ un graf orientat finit. Presupunem (fără a restrânge generalitatea, că am numerotat nodurile astfel încât nodul spre care căutăm drumurile de valoare minimă (maximă) de la celelalte noduri să fie x_n.

Pasul 1. Se construieşte matricea pătratică M cu dimensiunea egală cu numărul de noduri ale grafului ale cărei elemente sunt:

$$m_{ij} = \begin{cases} \text{valoarea arcului } (x_i, x_j) & \text{daca exista arcul } (x_i, x_j) \text{ si } i \neq j \\ 0 & \text{daca } i = j \\ +\infty \text{ (într - o problema de minim)} \\ -\infty \text{ (într - o problema de maxim)} & \text{daca nu exista arcul } (x_i, x_j) \end{cases}$$

Pasul 2. Se adaugă succesiv liniile L_i la matricea M, elementele acestora calculându-se prin relațiile de recurență:

1. $L_{1j} = m_{jn}$ $j = 1,...,n$ (prima linie este ultima coloană, transpusă, a matricii M)
2. $L_{ij} = \min (L_{i-1,j}, \min\limits_{k=1,n} (m_{jk} + L_{i-1,k}))$ într-o problemă de minim

sau $L_{ij} = \max (L_{i-1,j}, \max\limits_{k=1,n} (m_{jk} + L_{i-1,k}))$ într-o problemă de maxim

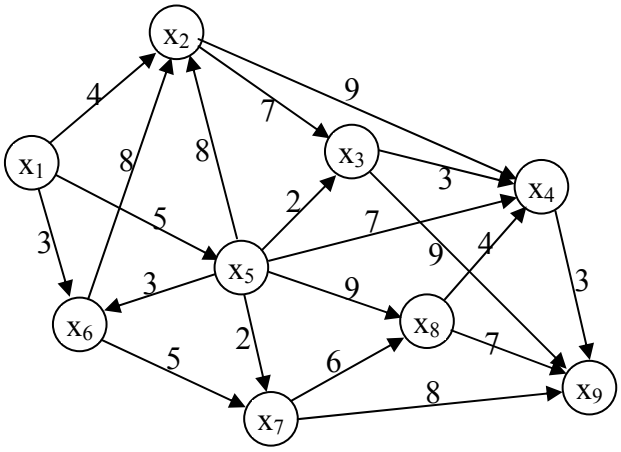
Pasul 3. După calcularea fiecărei linii noi se compară elementele ei cu cele ale precedentei:

- Dacă $L_{ij} = L_{i-1,j}$ pentru orice $j = 1,...,n$ atunci se oprește recurența și ultima linie calculată conține valorile minime ale drumurilor de la celelalte noduri la nodul x_n.
- Dacă există cel puțin un indice j cu $L_{ij} \neq L_{i-1,j}$ se trece la calcularea noii linii L_{i+1}

Pasul 4. Pentru găsirea drumului care dă valoarea minimă de la un nod x_j la nodul x_n se găsesc, începând înapoi de la ultima linie, pe care s-au obținut valorile finale, notată L_f, nodurile $x_{k_1}, x_{k_2}, ..., x_{k_r}$ care formează drumul căutat, unde $x_{k_1} = x_j$, $x_{k_r} = x_n$ și fiecare alt indice k_{i+1} este cel pentru care s-a obținut minimul(maximul) de pe poziția k_i al liniei L_i.

Observație: Pentru grafuri foarte mari, algoritmul necesită un volum mare de memorie, prin necesitatea memorării matricei M, care este greu de manipulat. Chiar dacă din cele n^2 arce posibile graful ar avea doar un procent foarte mic matricea grafului va avea tot n^2 poziții de memorat și analizat.

Exemplu: Presupunem dat graful orientat de mai jos, în care se dorește găsirea drumului de valoare minimă de la nodul x_1 la nodul x_9.



Matricea M va fi

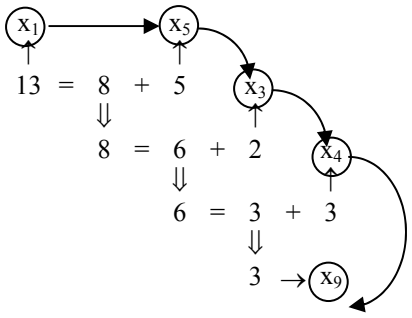
$$\begin{pmatrix} 0 & 4 & \infty & \infty & 5 & \infty & \infty & \infty & \infty \\ \infty & 0 & 7 & 9 & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & 3 & \infty & \infty & \infty & \infty & 9 \\ \infty & \infty & \infty & 0 & \infty & \infty & \infty & \infty & 3 \\ \infty & 8 & 2 & 7 & 0 & 3 & 2 & 9 & \infty \\ \infty & 8 & \infty & \infty & \infty & 0 & 5 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & 6 & 8 \\ \infty & \infty & \infty & 4 & \infty & \infty & \infty & 0 & 7 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$

iar după calcularea liniilor L_i obținem:

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
x_1	0	4	∞	∞	5	∞	∞	∞	∞
x_2	∞	0	7	9	∞	∞	∞	∞	∞
x_3	∞	∞	0	3	∞	∞	∞	∞	9
x_4	∞	∞	∞	0	∞	∞	∞	∞	3
x_5	∞	8	2	7	0	3	2	9	∞
x_6	∞	8	∞	∞	∞	0	5	∞	∞
x_7	∞	∞	∞	∞	∞	∞	0	6	8
x_8	∞	∞	∞	4	∞	∞	∞	0	7
x_9	∞	∞	∞	∞	∞	∞	∞	∞	0
L_1	∞	∞	9	3	∞	∞	8	7	0
L_2	∞	12	6	3	10	13	8	7	0
L_3	15	12	6	3	8	13	8	7	0
L_4	13	12	6	3	8	13	8	7	0
L_5	13	12	6	3	8	13	8	7	0

Deoarece $L_4 = L_5$ oprim calcularea liniilor după calcularea liniei 5. În această linie se află valorile celor mai scurte de la toate nodurile la nodul x_9. Drumul dorit de noi ($x_1 \rightarrow x_9$) are valoarea dată de prima poziție a liniei 5, fiind egal cu 13.

Pentru a găsi acest drum, plecăm înapoi de la linia 4 şi avem:



B. Algoritmul lui Ford simplificat

Algoritmul lui Ford simplificat ***se aplică doar în grafuri care nu admit circuite***. Cu ajutorul lui se găseşte drumul de valoare optimă între două noduri fixate x_i şi x_j. Printr-o eventuală renumerotare a nodurilor putem presupune că nodul de la care porneşte drumul este x_1, care va fi numit nod inițial, iar nodul la care se termină este x_n, numit nod final.

Algoritmul este:

Pasul 1. I se dă vârfului inițial valoarea 0 (zero): $w(x_0) = 0$

Pasul 2. Se construieşte mulțimea A formată din nodul inițial: $A = \{x_1\}$

Pasul 3. Se analizează nodurile din afara mulțimii A.

– Dacă există noduri în care se poate ajunge prin arce directe doar de la nodurile mulțimii A, acestea se adaugă la mulțimea A, cu valoarea:

$$w(x_i) = \min_{\substack{x_j \\ \exists(x_j, x_i)}} \left(w(x_j) + v(x_j, x_i)\right), \text{ în problemele de minim}$$

sau $w(x_i) = \max_{\substack{x_j \\ \exists(x_j,x_i)}} \left(w(x_j) + v(x_j, x_i)\right)$, în problemele de maxim

apoi se trece la pasul 4

- Dacă nu există nici un nod de acest tip atunci nu există nici un drum de la x_1 la x_n. STOP

Pasul 4. Se analizează mulțimea A:

- Dacă $x_n \in A$ atunci valoarea sa reprezintă valoarea drumului de valoare optimă de la x_1 la x_n. Pentru găsirea acestui drum se pornește înapoi de la nodul final x_n și se găsesc nodurile $x_{k_1}, x_{k_2}, ..., x_{k_r}$ care formează drumul căutat, unde $x_{k_1} = x_n$, $x_{k_r} = x_1$ și fiecare alt indice k_{i+1} este cel pentru care:

$$w(x_{k_{i+1}}) + v(x_{k_{i+1}}, x_{k_i}) = w(x_{k_i}) \quad \text{STOP}$$

- Dacă $x_n \notin A$ se reia algoritmul de la pasul 3.

Exemplu: Pentru același graf și aceeași pereche de noduri din exemplul rezolvat cu algoritmul lui Bellman-Kalaba vom avea succesiv:

pas1: $w(x_1) = 0$
pas2: $A = \{x_1\}$
pas3: Nodurile în care se poate ajunge doar din x_1: $\{x_5\} \neq \emptyset$
$w\{x_5) = \min(w(x_1) + v(x_1,x_5)) = 0 + 5 = 5$
pas4: $x_9 \notin A$
pas3: $A = \{x_1,x_5\}$ și nodurile în care se poate ajunge prin arce directe doar din x_1 și x_5 sunt: $\{x_6\} \neq \emptyset$
$w\{x_6) = \min(w(x_1) + v(x_1,x_6), w(x_5) + v(x_5,x_6)) = \min(0 + 3 , 5 + 3) = 3$
pas4: $x_9 \notin A$
pas3: $A = \{x_1,x_5,x_6\}$ și nodurile în care se poate ajunge prin arce directe doar din x_1, x_5 și x_6 sunt: $\{x_2,x_7\} \neq \emptyset$
$w\{x_2) = \min(w(x_1) + v(x_1,x_2), w(x_5) + v(x_5,x_2), w(x_6) + v(x_6,x_2)) = \min(0 + 4{,}5 + 8{,}3 + 8) = 4$
$w\{x_7) = \min(w(x_5) + v(x_5,x_7), w(x_6) + v(x_6,x_7)) = \min(5 + 2{,}3 + 5) = 7$
pas4: $x_9 \notin A$
pas3: $A = \{x_1,x_2,x_5,x_6,x_7\}$ și nodurile în care se poate ajunge prin arce directe doar din x_1, x_2, x_5, x_6 și x_7 sunt: $\{x_3,x_8\} \neq \emptyset$
$w\{x_3) = \min(w(x_2) + v(x_2,x_3), w(x_5) + v(x_5,x_3)) = \min(4 + 7{,}5 + 2) = 7$
$w\{x_8) = \min(w(x_5) + v(x_5,x_8), w(x_7) + v(x_7,x_8)) = \min(5 + 9{,}7 + 6) = 13$
pas4: $x_9 \notin A$
pas3: $A = \{x_1,x_2,x_3,x_5,x_6,x_7,x_8\}$ și nodurile în care se poate ajunge prin arce directe doar din x_1, x_2,x_3,x_5, x_6, x_7 și x_8 sunt: $\{x_4\} \neq \emptyset$
$w\{x_4) = \min(w(x_2) + v(x_2,x_4), w(x_3) + v(x_3,x_4), w(x_5) + v(x_5,x_4), w(x_8) + v(x_8,x_4)) = \min(4 + 9{,}7 + 3{,}5 + 7{,}13 + 4) = 10$
pas4: $x_9 \notin A$
pas3: $A = \{x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8\}$ și nodurile în care se poate ajunge prin arce directe doar din x_1, x_2, x_3, x_4, x_5, x_6, x_7 și x_8 sunt: $\{x_9\} \neq \emptyset$
$w\{x_9) = \min(w(x_3) + v(x_3,x_9), w(x_4) + v(x_4,x_9), w(x_7) + v(x_7,x_9), w(x_8) + v(x_8,x_9)) = \min(7 + 9, 10 + 3, 7 + 8, 13 + 7) = 13$
pas4: $x_9 \in A$ și urmează să găsim drumul care are lungimea 13.
Avem succesiv:

$$w(x_9) = w(x_4) + v(x_4,x_9)$$
$$w(x_4) = w(x_3) + v(x_3,x_4)$$
$$w(x_3) = w(x_5) + v(x_5,x_3)$$

$$w(x_5) = w(x_1) + v(x_1,x_5)$$

deci drumul căutat este: $x_1 \rightarrow x_5 \rightarrow x_3 \rightarrow x_4 \rightarrow x_9$

Observaţia 1. Dacă graful are un circuit atunci se poate demonstra uşor că nu vom putea da valoare nici unui nod al acestuia şi dacă există vreun drum de la x_1 la x_n care trece prin unul din nodurile circuitului nu vom putea da valoare nici lui x_n, cu toate că există drum de la x_1 la x_n.

Observaţia 2: Algoritmul necesită pentru memorare şi manipulare doar cunoaşterea, pentru fiecare nod, a nodurilor spre care "pleacă" arce din acesta şi valorile acestor arce, fiind mult mai uşor de aplicat sau implementat pe calculator. El are însă dezavantajul că se poate aplica doar în grafuri fără circuite.

C. Algoritmul Ford generalizat

Algoritmul lui Ford generalizat a fost creat cu scopul de a putea găsi drumul optim şi în grafurile care au circuite. Cu ajutorul lui se găseşte drumul de valoare optimă între două noduri fixate x_i şi x_j. Printr-o eventuală renumerotare a nodurilor putem presupune că nodul de la care porneşte drumul este x_1, care va fi numit nod inițial, iar nodul la care se termină este x_n, numit nod final.

Algoritmul este:

Pasul 1. I se dă vârfului inițial valoarea 0 (zero): $w(x_0) = 0$ şi tuturor celelalte valoarea $+\infty$ (într-o problemă de minim) sau $-\infty$ (într-o problemă de maxim).

Pasul 2. În ordinea crescătoare a indicilor nodurilor se calculează pentru fiecare nod, pe bază fostelor valori, noile valori cu formula:

$$w^*(x_i) = \min\left(w(x_i),\ \min_{\exists(x_j,x_i)} \left(w(x_j) + v(x_j,x_i) \right) \right) \text{ în problemele de minim}$$

sau $$w^*(x_i) = \max\left(w(x_i),\ \max_{\exists(x_j,x_i)} \left(w(x_j) + v(x_j,x_i) \right) \right) \text{ în problemele de maxim}$$

Pasul 3. Se compară noile valori $w^*(x_i)$ cu fostele valori $w(x_i)$:

- Dacă $w^*(x_i) = w(x_i)$ pentru orice nod x_i atunci:
  - dacă $w(x_n) < \infty$ (la problema de minim) sau $w(x_n) > -\infty$ (la problema de maxim), valoarea nodului x_n reprezintă valoarea drumului de valoare minimă(maximă) de la x_1 la x_n. Pentru găsirea acestui drum se porneşte înapoi de la nodul final x_n şi se găsesc nodurile $x_{k_1}, x_{k_2}, ..., x_{k_r}$ care formează drumul căutat, unde $x_{k_1} = x_n$, $x_{k_r} = x_1$ şi fiecare alt indice k_{i+1} este cel pentru care:
  $$w(x_{k_{i+1}}) + v(x_{k_{i+1}}, x_{k_i}) = w(x_{k_i}) \quad \text{STOP}$$
  - dacă $w(x_n) = +\infty$ ($-\infty$) atunci nu există nici un drum de la x_1 la x_n. STOP
- Dacă există cel puțin un nod pentru care $w^*(x_i) < w(x_i)$ se reia algoritmul de la pasul 2 pentru noile valori ale vârfurilor.

Observaţie: Algoritmul poate găsi drumul şi în grafuri cu circuite dar este evident mult mai lent decât cel simplificat. Pentru scurtarea duratei de execuţie se poate modifica algoritmul în sensul că o valoare nou calculată a unui vârf va fi folosită imediat ca atare la calculul noilor valori ale celorlalte, nu doar după ce se calculează noile valori ale tuturor vârfurilor.

D. Algoritmul lui Dijkstra

În algoritmul Ford simplificat, pentru a găsi valoarea nodului final, deci a drumului minim, plecăm de la nodul inițial în toate direcțiile posibile, păstrând de fiecare dată toate nodurile analizate. Acest fapt duce la un consum inutil de timp, deoarece foarte multe din aceste noduri nu vor face parte din drumul optim. Pentru a elimina acest neajuns, algoritmul lui Dijkstra încearcă să păstreze, la fiecare iterație, mulțimea minimă de noduri care să le conțină pe toate cele care vor forma efectiv drumul optim. În plus, algoritmul se poate aplica şi în drumuri cu circuite. Ca un minus este faptul că se aplică doar la probleme de minim. Algoritmul are următorii paşi:

Pasul 1. I se dă vârfului inițial valoarea 0 (zero): $w(x_0) = 0$

Pasul 2. Se construieşte mulțimea A formată din nodul inițial: $A = \{x_1\}$

Pasul 3. Se analizează nodurile din afara mulțimii A.

- Dacă există noduri ***în care se poate ajunge prin arce directe de la noduri din A*** (nu ***doar*** de la nodurile mulțimii A, ca la algoritmul lui Ford simplificat) se calculează pentru toate acestea:

$$w(x_i) = \min_{\substack{x_j \in A \\ \exists (x_j, x_i)}} \left(w(x_j) + v(x_j, x_i) \right) \text{ în problemele de minim}$$

  dar, spre deosebire de algoritmul lui Ford simplificat, *se adaugă la mulțimea A* ***doar*** *cel pentru care se obține valoarea minimă*, apoi se trece la pasul 4.
- Dacă nu există nici un nod de acest tip atunci nu există nici un drum de la x_1 la x_n. STOP

Pasul 4. Se analizează mulțimea A:

- Dacă $x_n \in A$ atunci valoarea sa reprezintă valoarea drumului de valoare optimă de la x_1 la x_n. Pentru găsirea acestui drum se porneşte înapoi de la nodul final x_n şi se găsesc nodurile $x_{k_1}, x_{k_2}, ..., x_{k_r}$ care formează drumul căutat, unde $x_{k_1} = x_n$, $x_{k_r} = x_1$ şi fiecare alt indice k_{i+1} este cel pentru care:

$$w(x_{k_{i+1}}) + v(x_{k_{i+1}}, x_{k_i}) = w(x_{k_i}) \quad \text{STOP}$$

- Dacă $x_n \notin A$ se reia algoritmul de la pasul 3.

Exemplu Vom aplica algoritmul la acelaşi graf folosit la ceilalți algoritmi, pentru a putea face comparații:

pas1: $w(x_1) = 0$
pas2: $A = \{x_1\}$
pas3: Nodurile în care se poate ajunge şi din x_1: $\{x_2, x_5, x_6\} \neq \varnothing$
$w\{x_2) = \min(w(x_1) + v(x_1,x_2)) = 0 + 4 = 4$
$w\{x_5) = \min(w(x_1) + v(x_1,x_5)) = 0 + 5 = 5$
$w\{x_6) = \min(w(x_1) + v(x_1,x_6)) = 0 + 3 = 3$
$\min(w\{x_2),w\{x_5),w\{x_6)) = w\{x_6) = 3$
pas4: $x_9 \notin A$
pas3: $A = \{x_1,x_6\}$ şi nodurile în care se poate ajunge prin arce directe din x_1 sau x_6 sunt: $\{x_2,x_5,x_7\} \neq \varnothing$
$w\{x_2) = \min(w(x_1) + v(x_1,x_2), w(x_6) + v(x_6,x_2)) = \min(0 + 4 , 3 + 8) = 4$
$w\{x_5) = \min(w(x_1) + v(x_1,x_5)) = \min(0 + 5) = 5$
$w\{x_7) = \min(w(x_6) + v(x_6,x_7)) = \min(3 + 5) = 8$
$\min(w\{x_2),w\{x_5),w\{x_7)) = w\{x_2) = 4$
pas4: $x_9 \notin A$

pas3: A = $\{x_1,x_2,x_6\}$ şi nodurile în care se poate ajunge prin arce directe din x_1, x_2 sau x_6 sunt: $\{x_3,x_4,x_5,x_7\} \neq \emptyset$

$w\{x_3) = \min(w(x_2) + v(x_2,x_3)) = \min(4 + 7) = 11$

$w\{x_4) = \min(w(x_2) + v(x_2,x_4)) = \min(2 + 9) = 11$

$w\{x_5) = \min(w(x_1) + v(x_1,x_5)) = \min(0 + 5) = 5$

$w\{x_7) = \min(w(x_6) + v(x_6,x_7)) = \min(3 + 5) = 0$

$\min(w\{x_3),w\{x_4),w\{x_5),w\{x_7)) = w\{x_5) = 5$

pas4: $x_9 \notin A$

pas3: A = $\{x_1,x_2,x_5,x_6\}$ şi nodurile în care se poate ajunge prin arce directe din x_1, x_2, x_5, x_6 şi x_7 sunt: $\{x_3,x_4,x_7,x_8\} \neq \emptyset$

$w\{x_3) = \min(w(x_2) + v(x_2,x_3), w(x_5) + v(x_5,x_3)) = \min(4 + 7,5 + 2) = 7$

$w\{x_4) = \min(w(x_2) + v(x_2,x_4), w(x_5) + v(x_5,x_4)) = \min(4 + 9,5 + 7) = 12$

$w\{x_7) = \min(w(x_5) + v(x_5,x_7), w(x_6) + v(x_6,x_7)) = \min(5 + 2,3 + 5) = 7$

$w\{x_8) = \min(w(x_5) + v(x_5,x_8)) = \min(5 + 9) = 14$

$\min(w\{x_3),w\{x_4),w\{x_7),w\{x_8)) = w\{x_3) = w\{x_7) = 7$

pas4: $x_9 \notin A$

pas3: A = $\{x_1,x_2,x_3,x_5,x_6,x_7\}$ şi nodurile în care se poate ajunge prin arce directe din x_1, x_2, x_3, x_5, x_6, şi x_7 sunt: $\{x_4,x_8,x_9\} \neq \emptyset$

$w\{x_4) = \min(w(x_2) + v(x_2,x_4), w(x_3) + v(x_3,x_4),w(x_5) + v(x_5,x_4)) = \min(4 + 9,7 + 3,5 + 7) =10$

$w\{x_8) = \min(w(x_5) + v(x_5,x_8), w(x_7) + v(x_7,x_8)) = \min(5 + 9,7 + 6) = 13$

$w\{x_9) = \min(w(x_3) + v(x_3,x_9), w(x_7) + v(x_7,x_9)) = \min(7 + 9,7 + 8) = 15$

$\min(w\{x_4),w\{x_8),w\{x_9)) = w\{x_4) = 10$

pas4: $x_9 \notin A$

pas3: A = $\{x_1,x_2,x_3,x_4,x_5,x_6,x_7\}$ şi nodurile în care se poate ajunge prin arce directe din x_1, x_2, x_3, x_4, x_5, x_6, şi x_7 sunt: $\{x_8,x_9\} \neq \emptyset$

$w\{x_9) = \min(w(x_3) + v(x_3,x_9), w(x_4) + v(x_4,x_9), w(x_7) + v(x_7,x_9)) = \min(7 + 9,10 + 3,7+8)=13$

$w\{x_8) = \min(w(x_5) + v(x_5,x_8), w(x_7) + v(x_7,x_8)) = \min(5 + 9,7 + 6) = 13$

$\min(w\{x_8),w\{x_9)) = w\{x_8) = w\{x_9) = 13$

pas4: $x_9 \in A$ şi urmează să găsim drumul care are lungimea 13.

Avem succesiv:

$w(x_9) = w(x_4) + v(x_4,x_9)$

$w(x_4) = w(x_3) + v(x_3,x_4)$

$w(x_3) = w(x_5) + v(x_5,x_3)$

$w(x_5) = w(x_1) + v(x_1,x_5)$

deci drumul căutat este: $x_1 \rightarrow x_5 \rightarrow x_3 \rightarrow x_4 \rightarrow x_9$

9. Rețele de transport

Într-o mare varietate de situații concrete din practica economică se pune problema deplasării unei cantități de materie, energie, informație etc, din anumite locuri, numite **surse**, în alte locuri, numite **destinații.** Pentru realizarea acestui transport se folosesc o serie de trasee, numite **rute de legătură.** Unitățile indivizibile ale cantității Q, care se deplasează de-a lungul rutelor între surse şi destinații, se numesc **unități de flux,** iar ansamblul rutelor, surselor, destinațiilor şi, eventual, a altor **puncte intermediare** se numeşte rețea de transport.

Situația de mai sus poate fi reprezentată geometric printr-un graf finit, conex şi fără bucle.

Pentru ca o astfel de problemă să fie suficient de complexă pentru a necesita un studiu matematic riguros, trebuie ca fiecare sursă să poată aproviziona mai multe destinații şi orice destinație să poată fi aprovizionată de mai multe surse.

Aprovizionarea destinațiilor se poate face direct de la surse sau prin intermediul altor puncte, numite puncte intermediare. În cazul cel mai general pot exista de asemenea legături între surse şi/sau legături între destinații.

Aşa cum s-a văzut şi la problema de transport, situația de mai sus este un cadru extrem de larg, care permite existența unui număr foarte mare de tipuri de probleme posibile, diferite între ele prin informațiile suplimentare pe care le avem despre rețea şi prin obiectivele urmărite.

Una dintre acestea este problema determinării cantității maxime (minime) care poate fi transportată de la surse la destinații, în situația în care sursele dispun de cantități limitate (inferior sau superior), destinațiile au un necesar sau o putere de absorbție limitată inferior sau superior iar pe fiecare rută se poate transporta doar o cantitate cuprinsă între anumite limite.

Pentru studiul matematic al acestei situații vom da definițiile matematice ale obiectelor implicate în problemă şi ipotezele modelului.

Definiția 1: Se numeşte **rețea de transport standard** un graf finit, simplu, conex, fără bucle G = (X,U) care are următoarele proprietăți:

1. Există şi este unic $s \in X$ a.î. $U_s^+ \neq \emptyset$, $U_s^- = \emptyset$ (din care doar "ies" arce), numit **intrarea** rețelei de transport;
2. Există şi este unic $t \in X$ a.î. $U_t^+ = \emptyset$, $U_t^- \neq \emptyset$ (în care doar "intră" arce) numit **ieşirea** rețelei de transport;
3. S-a definit o funcție c: $U \to R^+$ care asociază fiecărui arc u un număr strict pozitiv c_u numit **capacitatea arcului**.

Observație: Este clar că exemplele obişnuite au doar rareori o singură sursă şi o singură destinație. Totuşi, printr-o tehnică foarte simplă, orice rețea de transport se poate aduce la forma standard:

1. Dacă sunt mai multe surse se introduce un nod suplimentar din care "pleacă" câte un arc spre fiecare sursă (şi numai spre acestea), iar capacitățile acestor arce vor fi egale cu disponibilurile surselor corespunzătoare;
2. Dacă sunt mai multe destinații se introduce un nod suplimentar spre care "pleacă" câte un arc din fiecare destinație (şi numai din acestea), iar capacitățile acestor arce vor fi egale cu necesarurile destinațiilor corespunzătoare;

Definiția 2: Se numeşte **flux într-o rețea de transport** R = (X,U) o funcție φ: $U \to R^+$ care are următoarele proprietățile:

P1. $0 \le \varphi_u \le c_u$ oricare ar fi u din U; valoarea φ_u se numeşte **flux al arcului u**

P2. $\sum_{u \in U_i^+} \varphi_u = \sum_{u \in U_i^-} \varphi_u$ oricare ar fi $i \neq s,t$ (suma fluxurilor arcelor care "intră" într-un nod i este egală cu suma fluxurilor arcelor care "ies" din acest nod, cu excepția nodului inițial şi al celui final.

Definiția 3: Se numeşte **valoare a fluxului** suma fluxurilor arcelor care "pleacă" din nodul inițial s şi se notează cu Φ.

Observație: Se poate demonstra uşor că această valoare este egală şi cu suma fluxurilor arcelor care "intră" în nodul final t. În concluzie avem:

$$\Phi = \sum_{u \in U_s^+} \varphi_u = \sum_{u \in U_t^-} \varphi_u$$

Valoarea fluxului reprezintă cantitatea care se transportă efectiv pe rețea de la surse la destinații.

Definiția 4: Se numeşte **flux de valoare maximă într-o rețea** un flux φ în această rețea, cu proprietatea că, pentru orice alt flux φ' pe această rețea, avem $\Phi \geq \Phi'$.

Valoarea fluxului de valoare maximă reprezintă cea mai mare cantitate care se poate transporta efectiv pe rețea, de la surse la destinații.

Economic vorbind, ne interesează, referitor la o rețea, răspunsurile la următoarele întrebări:

1. Putem transporta întreaga cantitate necesară la destinații?
2. Dacă da, cum transportăm efectiv această cantitate de la surse la destinații?
3. Dacă nu, din ce motiv nu putem realiza acest transport?
4. Cum putem înlătura cu eforturi minime acest motiv?

Răspunsul la primele două întrebări se poate afla prin găsirea fluxului de valoare maximă şi compararea valorii lui cu suma necesarurilor destinațiilor. În plus, valoarea acestuia pe un arc reprezintă cantitatea care trebuie transportată pe ruta respectivă, pentru a obține această valoare a fluxului.

Răspunsul la ultimele două întrebări porneşte de la observația că cea mai mare cantitate care poate traversa rețeaua de la un cap la altul este egală cu dimensiunea celui mai îngust loc de trecere prin rețea. Dacă vrem, deci, să mărim fluxul va trebui să lărgim tocmai acest cel mai îngust loc de traversare al rețelei.

Pentru formalizarea considerațiilor de mai sus vom introduce noțiunea de **tăietură într-o rețea**:

Definiția 5: Dată o rețea de transport G(X,U) cu s = nodul inițial şi t = nodul final, se numeşte **tăietură în rețea** o partiție a mulțimii vârfurilor rețelei de transport, formată din două submulțimi V şi W ($V \cap W = \varnothing$, $V \cup W = X$) astfel încât $s \in V$ şi $t \in W$.

O tăietură poate fi privită, intuitiv, ca o secțiune a rețelei, care lasă nodul inițial cu o submulțime din noduri într-o parte, nodul final cu restul nodurilor în cealaltă parte şi retează toate arcele care trec dintr-o parte în cealaltă.

A cunoaşte o tăietură este echivalent cu a cunoaşte care sunt elementele celor două mulțimi, V şi W, care formează partiția.

Vom nota o tăietură prin T = (V,W), convenind ca mulțimea scrisă pe prima poziție să conțină nodul inițial s al rețelei iar cea scrisă pe a doua, nodul final t.

Definiția 6: Se numeşte **capacitate a unei tăieturi** T = (V,W) într-o rețea de transport G(X,U), notată C(T), suma capacităților tuturor arcelor care au extremitatea inițială în V şi cea finală în W.

$$C(T) = \sum_{\substack{u=(x_i,x_j) \\ x_i \in V \\ x_j \in W}} c_u$$

Pentru a nu exista nici o ambiguitate, insistăm asupra faptului că se vor lua în considerare doar arcele care trec de la mulțimea ce conține nodul inițial spre mulțimea care conține nodul final, adică în sensul normal de transport (surse → destinație).

Definiția 7: Se numeşte **tăietură de valoare minimă într-o rețea** o tăietură T în această rețea, cu proprietatea că, pentru orice altă tăietură T' în această rețea, avem $C(T) \le C(T')$.

Următoarele teoreme fac legătura matematică dintre fluxurile unei rețele şi tăieturile sale:

***Teorema* 1.** Dată o tăietură T = (V,W) şi un flux φ într-o rețea de transport avem:

$$\Phi = \sum_{\substack{u=(x_i,x_j) \\ x_i \in V \\ x_j \in W}} \varphi_u - \sum_{\substack{u=(x_i,x_j) \\ x_i \in W \\ x_j \in V}} \varphi_u$$

sau, altfel spus, valoarea unui flux oarecare este egală cu suma fluxurilor arcelor care trec de la V la W din care se scade suma fluxurilor arcelor care trec invers, de la W la V, oricare ar fi tăietura T = (V,W).

Demonstrație: Avem succesiv:

$$\Phi = \sum_{\substack{u=(s,x_j) \\ x_j \in X}} \varphi_u = \sum_{\substack{u=(s,x_j) \\ x_j \in X}} \varphi_u + \sum_{\substack{x_i \in V \\ x_i \ne s}} \left(\sum_{u \in U^+_{x_i}} \varphi_u - \sum_{u \in U^-_{x_i}} \varphi_u \right) =$$

$$= \sum_{\substack{u=(x_i,x_j) \\ x_i \in V \\ x_j \in V}} (\varphi_u - \varphi_u) + \sum_{\substack{u=(x_i,x_j) \\ x_i \in V \\ x_j \in W}} \varphi_u - \sum_{\substack{u=(x_i,x_j) \\ x_i \in W \\ x_j \in V}} \varphi_u = \sum_{\substack{u=(x_i,x_j) \\ x_i \in V \\ x_j \in W}} \varphi_u - \sum_{\substack{u=(x_i,x_j) \\ x_i \in W \\ x_j \in V}} \varphi_u$$

***Corolar*:** Într-o rețea de transport valoarea oricărui flux este mai mică sau egală decât valoarea oricărei tăieturi.

Demonstrație: Fie T o tăietură oarecare şi φ un flux oarecare. Avem succesiv:

$$\Phi = \sum_{\substack{u=(x_i,x_j) \\ x_i \in V \\ x_j \in W}} \varphi_u - \sum_{\substack{u=(x_i,x_j) \\ x_i \in W \\ x_j \in V}} \varphi_u \le \sum_{\substack{u=(x_i,x_j) \\ x_i \in V \\ x_j \in W}} \varphi_u \le \sum_{\substack{u=(x_i,x_j) \\ x_i \in V \\ x_j \in W}} c_u = C(T)$$

***Corolar*:** Într-o rețea de transport valoarea fluxului maxim este mai mică sau egală decât valoarea tăieturii minime.

Demonstrația e evidentă. În plus, din cele de mai sus se vede că egalitatea are loc numai dacă, pentru tăietura minimă, există un flux pentru care toate arcele de la V la W sunt folosite la maxim (fluxul e egal cu capacitatea arcelor) iar pe toate arcele de la W la V nu se transportă nimic.

Teorema lui Ford-Fulkerson Dacă fluxul φ este maximal atunci există o tăietură de capacitate egală cu valoarea fluxului.

***Demonstrație*:** Fie φ un flux maximal. Introducem următorul procedeu de marcare a vârfurilor:

Pasul 1. Se marchează nodul inițial s cu 0(zero);

Pasul 2. Pentru fiecare vârf marcat x_i se marchează cu:

- $[+x_i]$ toate vârfurile nemarcate x_j pentru care există arcul (x_i,x_j) şi $\varphi(x_i,x_j) < c(x_i,x_j)$ (adica nodurile spre care mai e loc pentru a se transporta ceva din x_i);
- $[-x_i]$ toate vârfurile nemarcate x_j pentru care există arcul (x_j,x_i) şi $\varphi(x_j,x_i) > 0$ (adică toate nodurile spre care pleacă deja ceva din x_i);

Pasul 3. Se repetă pasul 2 până nu mai poate fi marcat nici un vârf.

Dacă vârful final t ar fi marcat, atunci începând de la acesta, am putea construi lanțul $x_{k_1}, x_{k_2}, \ldots, x_{k_r}$ unde $x_{k_1} = s$, $x_{k_r} = t$ şi marcajul oricărui vârf $x_{k_{i+1}}$ este $+x_{k_i}$ sau $-x_{k_i}$. Adăugând la fluxul fiecărui arc al lanțului de tipul $(x_{k_i}, x_{k_{i+1}})$ valoarea:

$$\Delta = \min\Big(\min_{(x_{k_i},x_{k_{i+1}})}\big(c(x_{k_i},x_{k_{i+1}}) - \varphi(x_{k_i},x_{k_{i+1}})\big), \min_{(x_{k_{i+1}},x_{k_i})}\varphi(x_{k_i},x_{k_{i+1}})\Big)$$

şi scăzând din fluxul fiecărui arc de tipul $(x_{k_{i+1}}, x_{k_i})$ aceeaşi valoare Δ, obținem un flux de valoare Φ + Δ, deci fluxul φ nu ar fi maximal.

În concluzie vârful t nu va fi marcat. Fie tăietura T = (V,W), unde V este formată din mulțimea nodurilor marcate iar W din cele nemarcate. În acest caz, pentru fiecare arc (x_i,x_j) care "traversează" tăietura avem:

- dacă $x_i \in V$ atunci $\varphi(x_i,x_j) = c(x_i,x_j)$ deoarece nodul x_j nu e marcat
- dacă $x_i \in W$ atunci $\varphi(x_i,x_j) = 0$ deoarece nodul x_i nu e marcat

În acest caz avem:

$$C(T) = \sum_{\substack{u=(x_i,x_j)\\ x_i\in V\\ x_j\in W}} c_u = \sum_{\substack{u=(x_i,x_j)\\ x_i\in V\\ x_j\in W}} \varphi_u - \sum_{\substack{u=(x_i,x_j)\\ x_i\in W\\ x_j\in V}} \varphi_u = \Phi$$

şi teorema e demonstrată.

Teorema lui Ford-Fulkerson poate stabili doar valoarea fluxului maxim dar nu dă o metodă de găsire a acestuia. Pentru a rezolva problema găsirii fluxului de valoare maximă se poate folosi algoritmul lui Ford-Fulkerson.

Pentru expunerea acestuia vom introduce şi noțiunile de:

arc saturat = un arc pe care fluxul este egal cu capacitatea;
drum complet = un drum de la nodul inițial s la nodul final t care conține cel puțin un arc saturat;
flux complet = un flux pentru care orice drum de la nodul inițial s la nodul final t este complet.

Algoritmul lui Ford-Fulkerson

ETAPA I Se determină un flux complet.

Pasul 1. Se numerotează nodurile rețelei de transport astfel încât $x_1 = s$ şi $x_n = t$;

Pasul 2. Se asociază grafului fluxul nul ($\varphi_u = 0$ pentru orice arc u din graf);

Pasul 3. În ordine lexicografică, se ia pe rând fiecare drum D de la nodul inițial la cel final, se calculează valoarea $\Delta_D = \min_{u\in D}(c_u - \varphi_u)$ şi se adaugă la fluxul de pe fiecare arc al

drumului. Arcul(arcele) unui drum D pentru care s-a obținut valoarea minimă Δ_D va fi după această adăugare, în mod evident, saturat şi deci drumul D va fi complet.

După epuizarea tuturor drumurilor se obține un flux complet, de valoare $\Phi = \sum_D \Delta_D$.

Deoarece alegerea drumurilor în ordine lexicografică nu ține cont de structura rețelei, așa cum se poate vedea pe un exemplu, acest procedeu nu asigură întotdeauna găsirea fluxului maxim. Acest impediment poate fi depăşit fie prin găsirea unei ordini de parcurgere a tuturor drumurilor, care să dea pentru fiecare rețea fluxul maxim, în urma procedeului de mai sus, fie prin redistribuirea judicioasă a fluxului găsit la etapa I. A doua variantă este cea care se aplică la etapa II.

ETAPA II Se determină fluxul maxim

Pasul 4. Se marchează nodul inițial s cu 0(zero);

Pasul 5. Pentru fiecare vârf marcat x_i se marchează cu:

- $[+x_i]$ toate vârfurile nemarcate x_j pentru care există arcul (x_i,x_j) şi $\varphi(x_i,x_j) < c(x_i,x_j)$ (adica nodurile spre care mai e loc pentru a se transporta ceva din x_i);
- $[-x_i]$ toate vârfurile nemarcate x_j pentru care există arcul (x_j,x_i) şi $\varphi(x_j,x_i) > 0$ (adică toate nodurile spre care pleacă deja ceva din x_i);

Pasul 6. Se repetă pasul 5 până este marcat nodul final sau până când nu mai poate fi marcat nici un vârf;

Pasul 7. Dacă nodul final a fost marcat atunci fluxul este maxim şi algoritmul se opreşte, în caz contrar trecându-se la pasul 8;

Pasul 8. Construim un lanțul $L = x_{k_1}, x_{k_2}, ..., x_{k_r}$ unde $x_{k_1} = s$, $x_{k_r} = t$ şi marcajul oricărui vârf $x_{k_{i+1}}$ este $+x_{k_i}$ sau $-x_{k_i}$. Se calculează:

$$\Delta_L = \min\left(\min_{(x_{k_i}, x_{k_{i+1}})}\left(c(x_{k_i}, x_{k_{i+1}}) - \varphi(x_{k_i}, x_{k_{i+1}})\right), \min_{(x_{k_{i+1}}, x_{k_i})}\varphi(x_{k_i}, x_{k_{i+1}})\right)$$

care se adaugă la fluxul fiecărui arc al lanțului de tipul $(x_{k_i}, x_{k_{i+1}})$ şi se scade din fluxul fiecărui arc de tipul $(x_{k_{i+1}}, x_{k_i})$.

Pasul 9. Se şterge marcajul şi se reia algoritmul de la pasul 4.

În final, tăietura de valoare minimă este cea în care V = mulțimea nodurilor marcate iar W = mulțimea nodurilor nemarcate.

Observația 1. Algoritmul nu asigură întotdeauna găsirea fluxului maxim, deoarece se poate ca creşterea fluxului la fiecare iterație să se facă cu cantități din ce în ce mai mici astfel încât suma lor să nu atingă niciodată marginea superioară dată de valoarea tăieturii minime, algoritmul având o infinitate de paşi. Teorema de mai jos dă o condiție suficientă pentru ca algoritmul să se termine într-un număr finit de paşi:

Teoremă Dacă toate capacitățile rutelor rețelei sunt numere raționale atunci algoritmul lui Ford-Fulkerson are un număr finit de paşi.

Demonstrație Prin înmulțirea tuturor acestor capacități cu cel mai mic multiplu comun al numitorilor se obține o rețea cu toate capacitățile numere naturale. Ținând cont de formula de calcul, la fiecare iterație cantitatea adăugată Δ va fi număr natural şi cum valoarea fluxului maxim este mărginită de capacitatea tăieturii minime C_{min}, care este de asemenea număr natural, algoritmul va avea nevoie de cel mult C_{min} paşi pentru a o atinge.

Observația 2. Teorema de mai sus asigură doar o limitare superioară a numărului de iterații ale algoritmului, față de capacitatea tăieturii minime. Această valoare poate fi însă, în anumite

cazuri, foarte mare şi, dacă nu se iau precauţii suplimentare, algoritmul nu va da soluţia în timp util. Depăşirea acestei situaţii este asigurată de următoarea teoremă:

Teoremă Dacă la fiecare iteraţie se alege drumul (lanţul) de lungime minimă atunci algoritmul va avea cel mult $\frac{1}{2}\cdot m\cdot n$ iteraţii, unde n = numărul de noduri iar m = numărul de muchii.

Observaţia 3. Există probleme în care se doreşte găsirea ***fluxului minim*** într-o reţea, valorile fluxului pe arce fiind limitate **inferior** de capacităţile acestora. În acest caz se aplică de asemenea algoritmul lui Ford-Fulkerson astfel:

Pasul 1. Se calculează M = maximul capacităţilor arcelor
Pasul 2. Se construieşte reţeaua R', care este fosta reţea, în care au fost modificate doar capacităţile arcelor, acestea devenind $c'_u = M - c_u$
Pasul 3. Se găseşte cu algoritmul Ford-Fulkerson fluxul φ, de valoare maximă, în această reţea
Pasul 4. Fluxul de valoare minimă în reţeaua iniţială va avea valorile $\varphi' = M - \varphi$

Observaţia 4. Există şi alte tipuri de probleme asemănătoare celor de mai sus. Astfel, se poate pune problema:

- găsirii capacităţilor minime ale arcelor cu care se poate asigura transportarea întregii cantităţi de la surse la destinaţii
- fluxului minim(maxim) într-o reţea în care capacităţile rutelor sunt limitate atât superior cât şi inferior;
- În cazul în care rutelor li se asociază şi costuri unitare de parcurgere, putem căuta fluxul maxim de cost minim;