12. Grafuri orientate

- După cum s-a mai precizat, **grafurile orientate** sunt grafuri în care arcele care conectează nodurile au un singur sens.
    - Această restricţie face mult mai dificilă evidenţierea şi exploatarea diverselor proprietăţi ale grafurilor de acest tip.
    - Prelucrarea unui astfel de graf este identică cu o călătorie cu maşina într-un oraş cu foarte multe străzi cu sens unic sau cu o călătorie cu avionul într-o ţară în care liniile aeriene nu sunt dus-întors.
        - În astfel de situaţii a ajunge dintr-un loc în altul poate reprezenta o adevărată problemă.
- De multe ori arcele orientate reflectă anumite tipuri de **relaţii de precedenţă** în aplicaţia pe care o modelează.
    - Spre **exemplu** un **graf orientat** poate fi utilizat ca model pentru o **linie de fabricaţie:**
        - Nodurile corespund diverselor operaţii care trebuiesc executate.
        - Arcele orientate ordonează temporal structura. Astfel, un arc de la nodul `x` la nodul y precizează faptul că operaţiunea corespunzătoare nodului `x` trebuie executată **înaintea** celei corespunzătoare nodului `y`.
        - În acest context, problema care se pune este aceea de a executa toate aceste operaţii, fără a eluda niciuna dintre relaţiile de precedenţă impuse.
- După cum s-a menţionat în capitolul 10, **reprezentările grafurilor orientate** sunt simple extensii (de fapt simplificări) ale reprezentărilor grafurilor neorientate. Astfel:
    - În reprezentarea bazată pe **liste de adiacenţe**, **fiecare arc** apare **o singură dată**: arcul de la nodul `x` la `y` este reprezentat prin prezenţa nodului `y` în lista de adiacenţe a lui `x`.
    - În reprezentarea bazată pe **matrice de adiacenţe**, arcul de la nodul `x` la `y` se marchează tot **o singură dată**, prin valoarea `"adevărat"` a elementului matricii de adiacenţe situat în linia `x`, coloana `y` (nu şi a elementului situat în linia `y`, coloana `x`).
- În figura 12.1.a apare un exemplu de **graf orientat**
    - Graful constă din arcele `AG,AB,CA,LM,JM,JL,JK,ED,DF,NI,FE,AF, GE,GC,HG,GJ,LG,IH, ML`.
    - **Ordinea** în care nodurile apar la specificarea arcelor este **semnificativă**.
        - Notaţia `AG` precizând un arc care îşi are sursa în nodul `A` şi destinaţia în nodul `G`.

- Este însă posibil ca între două noduri să existe două arce având sensuri opuse (exemplu `HI` şi `IH`, respectiv `LM` şi `ML`).

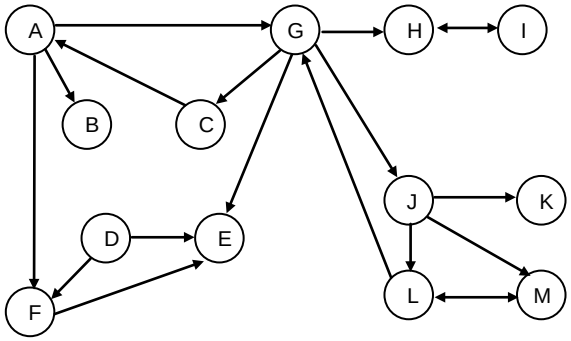


Fig.12.1.a. Exemplu de graf orientat.

- Pornind de la această precizare este uşor de observat faptul că un **graf neorientat** poate fi asimilat cu **graful orientat** identic ca şi structură, care conţine pentru fiecare arc al grafului neorientat două arce opuse.
    - În acest context unii dintre algoritmii dezvoltaţi în acest capitol, pot fi consideraţi **generalizări** ale **algoritmilor** din capitolele anterioare.
- În cadrul acestui capitol vor fi prezentate unele dintre problemele specifice acestei categorii de grafuri precum şi o serie de algoritmi consacraţi care le soluţionează.
- Este vorba despre:
    - (1) Problema drumurilor minime cu origine unică.
    - (2) Problema drumurilor minime corespunzătoare tuturor perechilor de noduri.
    - (3) Închiderea tranzitivă.
    - (4) Grafuri orientate aciclice.
    - (5) Componente conectate în grafuri orientate.
    - (6) Problema reţelelor de curgere.
    - (7) Problema potrivirilor.

12.1. Problema drumurilor minime cu origine unică ("Single-Source Shortest Path Problem")

- O primă problemă care se abordează în contextul grafurilor orientate este următoarea:
    - Se consideră un graf orientat `G=(N,A)` în care fiecare arc are **o pondere pozitivă** şi pentru care se precizează un nod pe post de "**origine**".
    - Se cere să se determine **costul** celui mai scurt drum de la origine la oricare alt nod al grafului.
        - Conform celor deja precizate, **costul** unui drum este **suma ponderilor arcelor** care formează drumul.

    - Această problemă este cunoscută sub numele de **problema drumurilor minime cu origine unică** (“**single source shortest path problem**”).

- Desigur o abordare mai naturală, ar fi aceea care-şi propune să determine **cel mai scurt drum de la o origine la o destinaţie precizată**.

    - Această problemă are acelaşi grad de dificultate ca şi problema anterioară care de fapt o **generalizează**.

    - Astfel, pentru a rezolva această ultimă problemă, este suficient ca execuţia algoritmului care determină drumurile minime cu origine unică să fie oprită în momentul în care se ajunge la destinaţia precizată.

- Exact ca şi în cazul grafurilor ponderate neorientate, ponderea poate avea semnificaţia fizică a unui cost, a unei valori, a unei lungimi, a unui timp, etc.

- Intuitiv, graful `G` poate fi asimilat spre exemplu, cu o hartă a traseelor aeriene ale unui stat în care:

    - Fiecare nod reprezintă un oraş.

    - Fiecare arc `(x,y)` reprezintă o legătură aeriană de la oraşul `x` la oraşul `y`.

    - Ponderea unui arc `(x,y)` poate fi timpul de zbor sau preţul biletului.

- Desigur, ca model ar putea fi utilizat şi un graf neorientat deoarece ponderea arcului `(x,y)` trebuie să fie în principiu aceeaşi cu cea a arcului `(y,x)`.

    - În realitate pe de o parte **nu** toate traseele aeriene sunt dus-întors, iar pe de altă parte dacă ele sunt dus-întors, timpul de zbor s-ar putea să difere în cele două sensuri.

    - În orice caz, considerând arcele `(x,y)` şi `(y,x)` cu ponderi identice, aceasta nu conduce la o simplificare a rezolvării problemei.

12.1.1. Algoritmul lui Dijkstra

- Pentru a rezolva problema drumurilor minime cu origine unică, o manieră clasică de abordare o reprezintă utilizarea unui algoritm bazat pe tehnica “**greedy**” adesea cunoscut sub denumirea de “**algoritmul lui Dijkstra**”

    - Acest algoritm a fost publicat de către **E.W. Dijkstra** în anul 1959.

- Algoritmul se bazează pe o **structură de date mulţime** `M` care conţine nodurile pentru care cea mai scurtă distanţă la nodul origine este deja cunoscută.

    - Iniţial `M` conţine numai nodul **origine**.

    - În fiecare pas al execuţiei algoritmului, se adaugă mulţimii `M` un nod `x` care **nu** aparţine încă mulţimii şi a cărui distanţă de la origine este cât mai scurtă posibil.

    - Presupunând că toate arcele au ponderi pozitive, întotdeauna se poate găsi un **drum minim** care leagă originea de nodul `x`, drum care trece **numai** prin nodurile conţinute de `M`.

        - Un astfel de drum se numeşte “**drum special**”.

    - Pentru înregistrarea **lungimii drumurilor speciale** corespunzătoare nodurilor grafului se utilizează un tablou `D` care este actualizat în fiecare pas al algoritmului.

- În momentul în care `M` include toate nodurile grafului, toate drumurile sunt speciale şi în conscinţă, tabloul `D` memorează cea mai scurtă distanţă de la origine la fiecare nod al grafului.

- Schiţa de principiu a **algoritmului lui Dijkstra** apare în secvenţa [12.1.1.a].

```
-----------------------------------------------------------
procedura Dijkstra;

/*Determină costurile celor mai scurte drumuri care
conectează nodul 1, considerat drept origine, cu toate
celelalte noduri ale unui graf orientat. Distanţele se
pastrează în tabloul D*/

[1] M=[1]; /*nodul origine*/
    /*iniţializarea tabloului de distanţe D*/
[2] pentru (i=2 la n)
[3]   D[i]=COST[1,i];                          /*[12.1.1.a]*/
    /*determinarea drumurilor minime*/
[4] pentru (i=1 la n-1)
[5]  |*alege un nod x aparţinând mulţimii N-M astfel
     | încât D[x] să fie minim;
[6]  |*adaugă pe x lui M;
     |/*actualizare distanţe*/
[7]  |pentru (fiecare nod y din mulţimea N-M)
[8]  |  D[y]= min(D[y], D[x]+COST[x,y])
     |□ /*pentru*/
/*Dijkstra*/
-----------------------------------------------------------
```

- Referitor la algoritmul lui **Dijkstra**, se presupune că:
  - Se dă un **graf orientat ponderat** `G=(N,A)` unde `N={1,2,3,...,n}`.
  - Nodul `1` este considerat drept origine.
  - `COST` este un tablou cu două dimensiuni unde `COST[i,j]` reprezintă costul deplasării de la nodul `i` la nodul `j` pe arcul `(i,j)`.
    - Dacă arcul `(i,j)` nu există, se presupune că `C[i,j]` este ∞, respectiv are o valoare **mai mare decât orice cost**.
  - La fiecare iteraţie a algoritmului, tabloul `D[i]` conţine lungimea drumului special minim curent de la origine la nodul `i`.

- Pentru exemplificare, se prezintă execuţia algoritmului lui Dijkstra pentru graful orientat din figura 12.1.1.a.

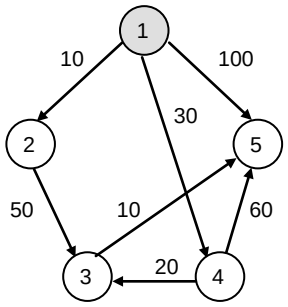
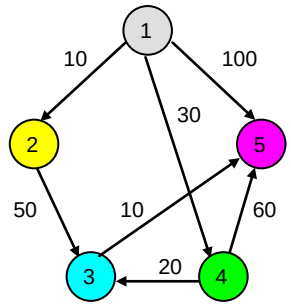


Fig.12.1.1.a. Graf orientat ponderat

- Iniţial `M={1}`, `D[2]=10`, `D[3]=∞`, `D[4]=30` şi `D[5]=100`.
- În prima iteraţie a buclei **`pentru`** din liniile `[4]-[8]`, se selectează `x=2` ca fiind nodul de distanţă minimă din `D`.
- În continuare se face `D[3]=min(∞,10+50)=60`.
  - `D[4]` şi `D[5]` **nu** se modifică deoarece în ambele cazuri, drumul direct care conectează nodurile respective cu originea este mai scurt decât drumul care trece prin nodul `2`.
- În continuare, modul în care se modifică tabloul `D` după fiecare iteraţie a buclei **`pentru`** mai sus precizate apare în figura 12.1.1.b.

Iteratia	M	x	D[2]	D[3]	D[4]	D[5]
Initial	{1}	-	10	∞	30	100
(1)	{1,2}	2	10	60	30	100
(2)	{1,2,4}	4	10	50	30	90
(3)	{1,2,4,3}	3	10	50	30	60
(4)	{1,2,4,3,5}	5	10	50	30	60



x	1	2	3	4	5
Parinte[x]	0	1	4	1	3

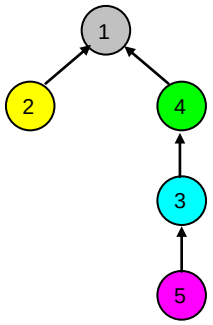


Fig.12.1.1.b. Determinarea drumurilor minime cu origine unică în baza algoritmului lui Dijkstra

- În vederea reconstrucţiei drumurilor minime de la origine la fiecare nod al grafului, se utilizează un alt tablou denumit `Părinte`.
  - În acest tablou, `Părinte[x]` memorează nodul care precede nodul `x` în cadrul drumului minim, adică memorează tatăl nodului `x`.
  - Tabloul `Părinte` se iniţializează cu `Părinte[i]=1` pentru toate valorile lui `i≠1`.
  - Tabloul `Părinte` poate fi actualizat după linia `[8]` în procedura `Dijkstra` din secvenţa [12.1.1.a].
    - Astfel, dacă în linia `[8]`, `D[x]+COST[x,y]<D[y]`, atunci se face `Părinte[y]=x`.

- După terminarea procedurii, drumul la fiecare nod poate fi reconstituit în sens invers mergând pe înlănţuirile indicate de tabloul `Părinte`.
- Astfel, pentru graful orientat din fig. 12.1.1.a, tabloul `Părinte` conţine valorile precizate în figura 12.1.1.b.
    - Pentru a găsi spre exemplu, drumul minim de la nodul `1` la nodul `5` al grafului, se determină predecesorii (părinţii) în ordine inversă începând cu nodul `5`.
    - Astfel se determină `3` ca predecesorul lui `5`, `4` ca predecesor al lui `3`, `1` ca predecesor al lui `4`.
    - În consecinţă drumul cel mai scurt de la nodul `1` la nodul `5` este `1,4,3,5` iar lungimea sa `60` este memorată în `D[5]`.
- În aceeaşi figură apare reprezentat şi **arborele de acoperire corespunzător drumurilor minime cu origine unică** ataşat grafului din figura 12.1.1.a.

12.1.2. Demonstrarea funcţionalităţii algoritmului lui Dijkstra

- Algoritmul lui Dijkstra este un exemplu de tehnică "**greedy**" în sensul că ceea ce apare local ca fiind cea mai bună activitate de realizat la un moment dat, reprezintă de fapt cel mai bun lucru de realizat la nivelul întregului la acel moment.
    - În acest caz, cel mai bun lucru de realizat la nivel local este acela de a determina distanţa la acel nod `x` care este situat în afara mulţimii `M` şi care are **cel mai scurt drum special** la momentul considerat.
- Pentru a demonstra ca în această situaţie **nu** poate exista un alt drum nespecial mai scurt de la nodul `x` la origine decât drumul special care cuprinde **numai** noduri din `M`, se va utiliza **metoda reducerii la absurd**.
    - Se presupune că **există** un drum ipotetic nespecial, mai scurt decât drumul de la origine la nodul `x`, care întâi părăseşte mulţimea `M`, atinge nodul `z`, iar apoi (probabil) intră şi iese din `M` de câteva ori înainte de a ajunge la `x` (fig.12.1.2.a. (a)).

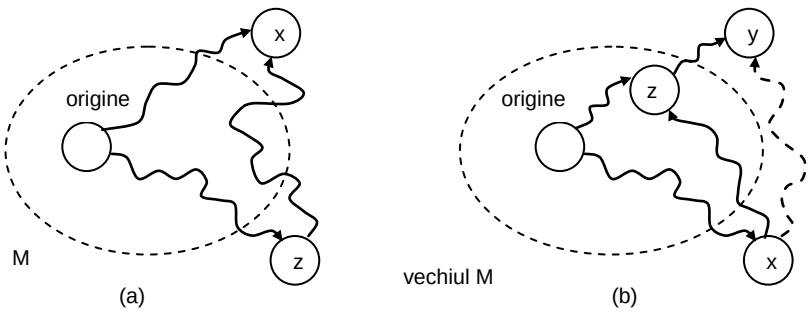


Fig.12.1.2.a. Drum minim ipotetic şi drum special minim imposibil

- Dacă acest drum este mai scurt decât drumul special care leagă originea de `x`, atunci segementul iniţial al drumului care leagă originea de `z`, este un drum special la `z`, mai scurt decât cel mai scurt drum special la `x`.
    - Trebuie subliniată în acest context importanţa faptului că valorile ponderilor sunt **pozitive**.

- Fără această restricţie, presupunerea avansată poate fi incorectă după cum întreaga funcţionare a algoritmului lui Dijkstra poate fi incorectă.

- În acest caz însă, în linia [5] a algoritmului din secvenţa [12.1.1.a] ar fi fost ales nodul z şi nu nodul x, deoarece D[z]<D[x] conform celor presupuse.

- Întrucât acest lucru **nu** s-a întâmplat, rezultă că **nu** există un astfel de nod z şi că ipoteza formulată conduce la o contradicţie.

- Pentru a completa demonstraţia funcţionalităţii algoritmului lui Dijkstra, mai trebuie verificat dacă în fiecare moment, D[y] reprezintă într-adevăr **cea mai scurtă distanţă** a unui drum special la nodul y.

- Esenţa acestei argumentaţii rezidă în observaţia că atunci când se adaugă un nou nod x mulţimii M în linia [6] a secvenţei [12.1.1.a], liniile [7] şi [8] ajustează pe D luând în considerare posibilitatea de a exista în acel moment un drum special mai scurt spre y trecând prin nodul x.

    - Dacă există, acest drum trece prin vechiul M, ajunge la x şi de aici imediat la y, şi costul său, D[x]+COST[x,y], va fi comparat cu D[y] în linia [8].

    - Dacă noul drum special este mai scurt, D[y] va fi redus în consecinţă.

    - Această verificare se realizează pentru fiecare nod y încă neselectat, adică aparţinând mulţimii N-M

- Singura **altă** posibilitate pentru un **drum mai scurt** apare în figura 12.1.2.a. (b), în care un astfel de drum înaintează spre x, revine în vechiul M la un membru oarecare z al acestuia, iar apoi atinge pe y.

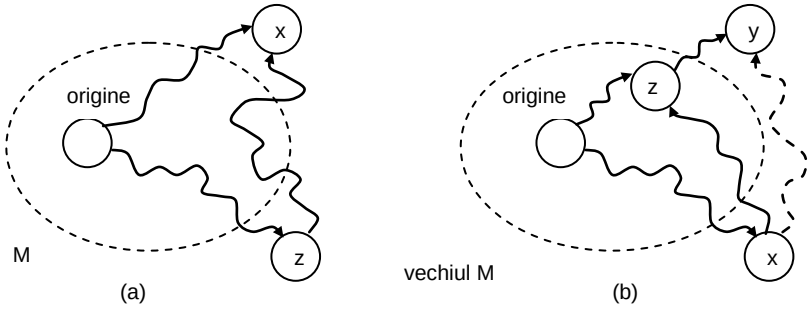


Fig.12.1.2.a. Drum minim ipotetic şi drum special minim imposibil (reluare)

- În realitate un astfel de drum **nu poate exista**.

- Deoarece z a fost plasat în M înaintea lui x, cel mai scurt dintre toate drumurile care conectează originea cu z trece obligatoriu prin vechiul M.

- Deci drumul la z trecând prin x, prezentat în figura 12.1.2.a. (b), **nu** este mai scurt decât drumul direct la z prin M.

- În consecinţă, lungimea drumului din figura 12.1.2.a. (b), de la origine la y, trecând prin x şi z **nu este mai mică** decât vechea valoare D[y], de vreme ce D[y] **nu a fost mai mare** decât lungimea celui mai scurt drum la z prin M şi apoi direct la y la momentul selecţiei nodului z.

- Astfel `D[y]` **nu** va fi modificat în linia `[8]` a secvenţei [12.1.1.a] de un drum trecând prin `x` şi `z` (fig.12.1.2.a (b)), motiv pentru care lungimea unui astfel de drum **nu** este luată în considerare în cadrul algoritmului.

12.1.3. Analiza performanţei algoritmului lui Dijkstra

- Se presupune că algoritmul din secvenţa [12.1.1.a] reluată mai jos, operează asupra unui graf orientat cu `n` noduri şi `a` arce.

```
-------------------------------------------------------------
procedura Dijkstra;

/*Determină costurile celor mai scurte drumuri care
conectează nodul 1, considerat drept origine, cu toate
celelalte noduri ale unui graf orientat. Distanţele se
pastrează în tabloul D*/

[1] M=[1]; /*nodul origine*/
    /*iniţializarea tabloului de distanţe D*/
[2] pentru (i=2 la n)
[3]   D[i]=COST[1,i];                          /*[12.1.1.a]*/
    /*determinarea drumurilor minime*/
[4] pentru (i=1 la n-1)
[5]  |*alege un nod x aparţinând mulţimii N-M astfel
     |  încât D[x] să fie minim;
[6]  |*adaugă pe x lui M;
     |/*actualizare distanţe*/
[7]  |pentru (fiecare nod y din mulţimea N-M)
[8]  |  D[y]= min(D[y], D[x]+COST[x,y])
     |□ /*pentru*/
/*Dijkstra*/
-------------------------------------------------------------
```

- (1) Dacă pentru reprezentarea grafului se utilizează o **matrice de adiacenţe**, atunci bucla **`pentru`** din liniile `[7]-[8]` necesită un efort de calcul proporţional cu $O(n)$.
  - Întrucât această buclă este executată de `n-1` ori, (bucla **`pentru`** din linia `[4]`), timpul total de execuţie va fi proporţional cu $O(n^2)$.
  - Este uşor de observat că restul algoritmului **nu** necesită timpi superiori acestei valori.
- (2) Dacă `a` este mult mai mic decât n^2, este mai indicat ca în reprezentarea grafului să se utilizeze **liste de adiacenţe**, respectiv o **coadă bazată pe prioritate** implementată ca un ansamblu (arbore binar parţial ordonat), pentru a păstra distanţele la nodurile mulţimii `N-M`.
  - În aceste condiţii, bucla **`pentru`** din liniile `[7]-[8]` poate fi implementată ca şi o traversare a **listei de adiacenţe** a lui `x` cu actualizarea distanţelor nodurilor din **coada bazată pe prioritate**.
  - În total, pe întreaga durată a execuţiei procedurii, se vor realiza `a` actualizări, fiecare cu un efort proporţional cu $O(\log_2 n)$, astfel încât timpul total consumat în liniile `[7]-[8]` va fi proporţional cu $O(a \log_2 n)$ şi nu cu $O(n^2)$.
  - În mod evident liniile `[2]-[3]` necesită asemenea liniilor `[4]-[6]` un efort de calcul proporţional cu $O(n)$.

- Utilizând pentru reprezentarea mulţimii `N-M` o **coadă bazată pe priorităţi**, linia `[5]` implementează operaţia de extragere a elementului cu prioritatea minimă din coadă, fiecare din cele `n-1` iteraţii ale acestei linii necesitând $O(log_2\ n)$ unităţi de timp, în total $O(n\ log_2\ n)$.
- În consecinţă, **timpul total** consumat de această variantă a algoritmului lui Dijkstra este limitat la $O(a\ log_2\ n)$ `(a≥n-1)`, performaţă care este considerabil mai bună decât $O(n^2)$.
- Se reaminteşte faptul că această performanţă se poate obţine numai dacă `a` este mult mai mic în raport cu n^2.

12.2 Problema drumurilor minime corespunzătoare tuturor perechilor de noduri ("All-Pairs Shortest Path Problem")

- Se presupune că există un **graf orientat ponderat** conţinând timpii de zbor pentru anumite trasee aeriene care conectează oraşe şi că se doreşte construcţia unei tabele care furnizează **cei mai scurţi timpi de zbor** între **oricare două oraşe**.
  - Aceasta este o instanţă a **problemei drumurilor minime corespunzătoare tuturor perechilor de noduri** ("**all-pairs shortest paths problem**").
- Pentru a formula problema în **termeni formali**, se presupune că se dă un **graf orientat ponderat** `G=(N,A)`, în care fiecare arc `(x,y)` are o pondere pozitivă `COST[x,y]`.
  - Rezolvarea **problemei drumurilor minime corespunzătoare tuturor perechilor de noduri** pentru graful `G`, presupune determinarea pentru fiecare pereche ordonată de noduri `(x,y)`, unde `x,y ∈ N`, a **lungimii drumului minim** care conectează nodul `x` cu nodul `y`.
- Această problemă poate fi rezolvată cu ajutorul **algoritmului lui Dijkstra**, considerând pe rând, fiecare nod al grafului `G` drept origine.
  - De fapt rezultatul execuţiei algoritmului lui Dijkstra este un **tablou** care conţine distanţele de la origine la restul nodurilor grafului.
  - În cazul de faţă, deoarece este necesară determinarea distanţelor pentru toate perechile de noduri, avem nevoie de o **matrice de elemente** în care fiecare **rând** se poate determina în baza algoritmului lui Dijkstra.
- O rezolvare mai directă a **problemei drumurilor minime corespunzătoare tuturor perechilor de noduri ale unui graf**, este datorată **algoritmului lui R.W. Floyd** datând din anul 1962.

12.2.1. Algoritmul lui Floyd

- Fie **graful orientat ponderat** `G=(N,A)` care are ataşată matricea de ponderi `COST`.
  - Pentru convenienţă, se consideră că nodurile mulţimii `N` sunt numerotate de la `1` la `n`.
- **Algoritmul lui Floyd** utilizează o matrice `A` de dimensiuni `nXn` cu ajutorul căreia determină **lungimile drumurilor minime** între toate perechile de noduri.
  - Iniţial se face `A[i,j]=COST[i,j]` pentru toţi `i≠j`.

- Dacă nu există niciun arc de la nodul `i` la nodul `j`, se presupune că `COST[i,j]`= ∞.
- Elementele diagonalei principale a matricei `A` se pun toate pe 0.

- **Algoritmul lui Floyd** execută `n` iteraţii asupra matricii `A`.
  - După cea de-a `k`-a iteraţie, `A[i,j]` va conţine lungimea minimă a unui drum de la nodul `i` la nodul j, care **nu** trece prin nici un nod cu număr mai mare ca şi `k`.
  - Cu alte cuvinte, `i` şi `j` pot fi oricare două noduri ale grafului care delimitează începutul şi sfârşitul drumului, dar orice nod intermediar care intră în alcătuirea drumului trebuie să aibă numărul mai mic sau cel mult egal cu `k`.
  - În cea de-a `k` iteraţie, se examinează nodul `k` şi se utilizează următoarea **formulă** în calculul lui `A` (formula [12.2.1.a]):

$$A_k[i,j] = min\begin{cases} A_{k-1}[i,j] \\ A_{k-1}[i,k] + A_{k-1}[k,j] \end{cases} \qquad \texttt{[12.2.1.a]}$$

  - Se face precizarea că indicele `k` semnifică **momentul de timp** la care se realizează calculul matricei `A` (în cadrul celei de-a `k` iteraţii) şi **nu** faptul că ar exista `n` matrici `A`, fiind utilizat pentru o înţelegere mai facilă a funcţionării algoritmului.
  - Formula de mai sus are interpretarea simplă precizată în figura 12.2.1.a.

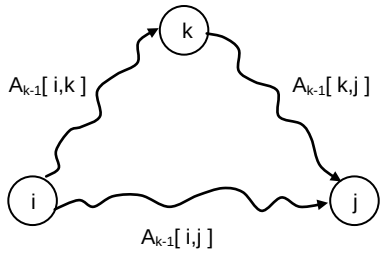


Fig.12.2.1.a. Selecţia drumului minim de la nodul `i` la nodul `j` la momentul `k`

  - Pentru calculul lui A_k`[i,j]` la momentul curent, se compară A_{k-1}`[i,j]` - adică costul drumului de la `i` la `j` fără a trece prin nodul `k` sau oricare alt nod cu număr mai mare ca şi `k`, la momentul anterior, cu A_{k-1}`[i,k]`+A_{k-1}`[k,j]` - adică costul drumului de la `i` la `k` însumat cu costul drumului de la `k` la `j` fără a trece prin nici un nod cu număr mai mare ca şi `k`, tot la momentul anterior.
  - Dacă drumul din urmă se dovedeşte a fi mai scurt, atunci el este atribuit valorii A_k`[i,j]`, altfel aceasta rămâne identică cu valoarea anterioară.

- Pentru graful ponderat din figura 12.2.1.b. (a), se prezintă în cadrul aceleiaşi figuri modul în care se modifică matricea A pornind de la conţinutul ei iniţial A_0 şi terminând cu conţinutul său final A_3.

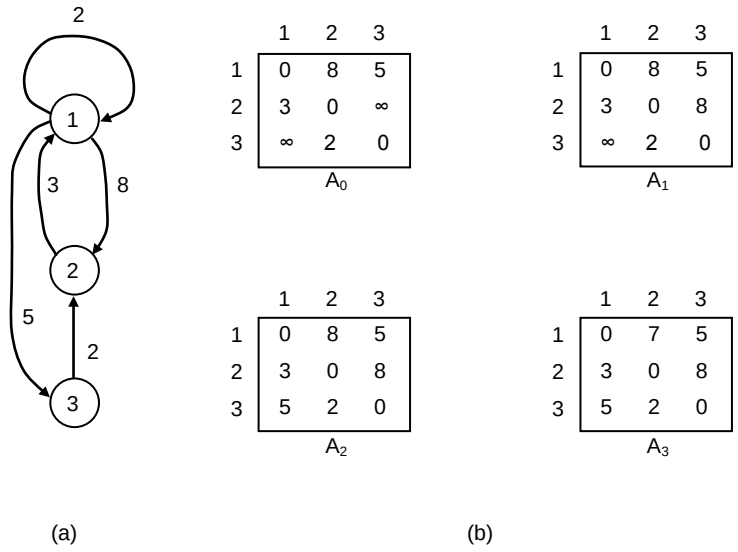


Fig.12.2.1.b. Calculul lungimii drumurilor minime corespunzătoare tuturor perechilor de noduri ale unui graf utilizând algoritmul lui Floyd

- Deoarece $A_k[i,k]=A_{k-1}[i,k]$ şi $A_k[k,j]=A_{k-1}[k,j]$, nici o intrare a matricii A în care intervine pe post de indice valoarea k, **nu** se modifică în timpul celei de-a k-a iteraţii.
    - Cu alte cuvinte, matricea A este **invariantă** în raport cu indicele k.
- În consecinţă se poate utiliza **o singură copie a matricii** A.
- Structura de principiu a procedurii care implementează **algoritmul lui Floyd** în baza considerentelor mai sus enunţate apare în secvenţa [12.2.1.a].

```
-------------------------------------------------------------
PROCEDURE Floyd(float A[n,n], float COST[n,n])

/*Procedura determină matricea drumurilor minime A pornind
de la matricea ponderilor COST*/

    int i,j k;
    /*matricea A se iniţializează cu matricea COST*/
[1] pentru (i=1 la n)
[2]   pentru (j=1 la n)
[3]     A[i,j]=COST[i,j];                    /*[12.2.1.a]*/
    /*diagonala principală a lui A pe zero*/
[4] pentru (i=1 la n)
[5]   A[i,i]=0;
    /*determinarea matricei lungimilor drumurilor minime A*/
[6] pentru (k=1 la n)
[7]   pentru (i=1 la n)
[8]     pentru (j=1 la n)
[9]       daca (A[i,k]+A[k,j]<A[i,j])
[10]        A[i,j]=A[i,k]+A[k,j];
```

```
/*Floyd*/
------------------------------------------------------------
```

- Timpul de execuţie al acestei proceduri este în mod clar proporţional cu $O(n^3)$ deoarece la baza sa stă o buclă triplă încuibată.
- Verificarea corectitudinii funcţionării algoritmului se poate realiza simplu prin metoda inducţiei.
    - Astfel este uşor de demonstrat că după ce `k` trece prin bucla triplă **`pentru`**, `A[i,j]` memorează lungimea celui mai scurt drum de la nodul `i` la nodul `j` care în niciun caz nu trece printr-un nod cu număr mai mare ca şi `k` [AH85].

12.2.2. Comparaţie între algoritmul lui Floyd şi algoritmul lui Dijkstra

- Pentru grafuri reprezentate prin **matrici de adiacenţe:**
    - (1) Versiunea **algoritmului Dijkstra** determină drumurile minime specifice unui nod precizat cu performanţa $O(n^2)$.
    - (2) **Algoritmul lui Floyd** determină toate drumurile minime cu performanţa $O(n^3)$.
        - Constantele de proporţionalitate reale depind de natura compilatorului, de sistemul de calcul şi de implementarea propriu-zisă.
        - De fapt activitatea experimentală şi măsurătorile reale reprezintă cele mai bune criterii de apreciere a performanţelor unui algoritm.
- În **condiţiile** în care numărul de arce `a` este **mult mai redus** decât n^2 şi grafurile sunt reprezentate prin **structuri de adiacenţe**:
    - (1) **Algoritmul lui Floyd** îşi păstrează performanţa stabilită $O(n^3)$.
    - (2) Este de aşteptat ca **algoritmul lui Dijkstra** să se comporte mai bine.
        - Astfel după cum s-a precizat, acest algoritm determină drumurile minime corespunzătoare unui nod precizat (origine) cu un efort de calcul proporţional cu $O(a \log_2 n)$.
        - În consecinţă utilizând această variantă de algoritm, problema drumurilor minime corespunzătoare tuturor perechilor de noduri se poate rezolva aplicând algoritmul lui Dijkstra tuturor nodurilor grafului cu performanţa $O(na \log_2 n)$.
        - Se face din nou precizarea că această performanţă se poate obţine atunci când $a<<n^2$ respectiv în cazul **grafurilor rare de mari dimensiuni**.

12.2.3. Determinarea traseelor drumurilor minime

- Algoritmii dezvoltaţi în cadrul acestui paragraf determină de fapt **costurile drumurilor minime**.
    - De multe ori, în practică este însă foarte util a se cunoaşte şi **traseul** acestor drumuri minime.
- Pentru a rezolva această problemă, în contextul **algoritmului lui Floyd** este necesară utilizarea unei alte matrici numite `Drum`.
    - În matricea `Drum`, o locaţie `Drum[i,j]` memorează indicele acelui nod `k` care conduce în cadrul algoritmului la cea mai mică valoare pentru `A[i,j]`.
    - Dacă `Drum[i,j]=0` atunci cel mai scurt drum este cel direct de la nodul `i` la nodul `j`.
- Varianta modificată a **algoritmului lui Floyd** care determină şi matricea `Drum` apare în secvenţa [12.2.3.a].

```
-------------------------------------------------------------
/*definirea matricii care memorează traseele drumurilor
minime*/
float Drum[n,n];

PROCEDURE Floyd(float A[n,n], float COST[n,n])

/*Procedura determină matricea A care memorează lungimile
drumurilor minime pornind de la matricea ponderilor COST. În
plus determină traseul drumurilor minime în matricea Drum*/

    int i,j k;
    /*iniţializarea matricilor A şi Drum*/
[1] pentru (i=1 la n)
[2]   pentru (j=1 la n)
[3]     |A[i,j]=COST[i,j];
[4]     |Drum[i,j]=0;
        |□                                    /*[12.2.3.a]*/
    /*diagonala principală a lui A pe zero*/
[5] pentru (i=1 la n)
[6]   A[i,i]= 0;
    /*determinarea lungimii drumurilor minime în matricea A
      şi a traseelor acestor drumuri în matricea Drum*/
[7] pentru (k=1 la n)
[8]   pentru (i=1 la n)
[9]     pentru (j=1 la n)
[10]      daca (A[i,k]+A[k,j]<A[i,j])
[11]        |A[i,j]=A[i,k]+A[k,j];
[12]        |Drum[i,j]=k;
            |□
/*Floyd*/
-------------------------------------------------------------
```

- Matricea `Drum` este de fapt o **colecţie de arbori corespunzători drumurilor minime**, câte unul pentru fiecare nod al grafului original, considerat drept origine,.

- Afişarea **traseului drumului minim** de la nodul `i` la nodul `j`, prin evidenţierea nodurilor intermediare se poate realiza cu ajutorul procedurii recursive **`Traseu`** (secvenţa [12.2.3.b]).

```
-------------------------------------------------------------
PROCEDURE Traseu(int i,j)

/*Afişeaza traseul drumului minim de la nodul i la nodul j*/

  int k;
                                                  /*[12.2.3.b]*/
  k=Drum[i,j];
  daca (k!=0)
    |Traseu(i,k);
    |*scrie(k);
    |Traseu(k,j);
    |□
/*Traseu*/
-------------------------------------------------------------
```

- Procedura **`Traseu`** aplicată unei matrici oarecare ar putea cicla la infinit.
  - Acest lucru **nu** se întâmplă în situaţia în care ea este aplicată matricii `Drum`, deoarece este imposibil ca un nod oarecare `k` să apară în drumul minim de la nodul `i` la nodul `j` şi în acelaşi timp `j` să apară în drumul minim de la `i` la `k`.
  - Se reaminteşte din nou importanţa crucială a valorilor **pozitive** a ponderilor arcelor grafului.
- În figura 12.2.3.a apare conţinutul final al matricii `Drum` (b) pentru graful orientat din aceeaşi figura (a).

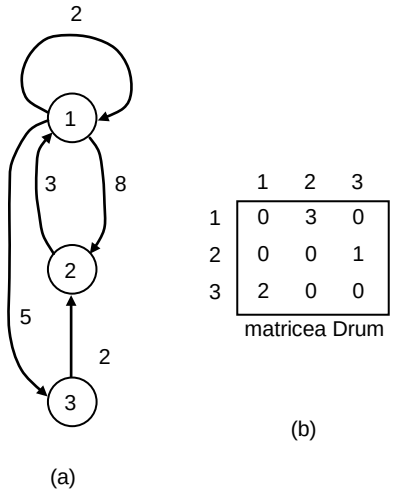


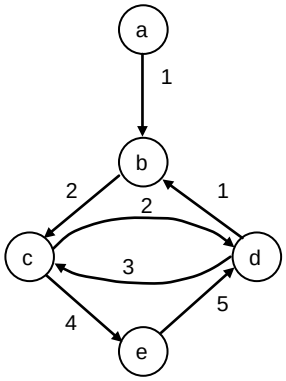
Fig.12.2.3.a. Matricea traseelor drumurilor minime (b) corespunzătoare grafului (a).

12.2.4. Aplicaţie. Determinarea centrului unui graf orientat ponderat

- Se presupune că se dă graful orientat ponderat `G=(N,A)` şi se cere să se determine **cel mai central nod** al său adică **centrul grafului**.
- Pentru a defini termenul de "**cel mai central nod**" se defineşte pentru început noţiunea de **excentricitate a unui nod** aparţinând unui graf ponderat.
    - **Excentricitatea** unui nod `x∈N` este precizată de formula [12.2.4.a]:

$$Excentricitate[x] = \max_{x \in N} \{ \min_{y \in N} \{ \text{drumurile de la } x \text{ la } y \} \} \qquad [12.2.4.a]$$

- Cu alte cuvinte, **excentricitatea** unui nod al unui graf ponderat este **valoarea maximă** dintre lungimile drumurilor minime de la nodul respectiv la toate celelalte noduri ale grafului.
- **Centrul unui graf** `G` este nodul a cărui excentricitate este **minimă**.
    - Spre exemplu pentru graful din figura 12.2.4.a. (a), valorile **excentricităţilor** nodurilor apar în tabloul (c) din aceeaşi figură.
    - Conform definiţiei anterioare, centrul grafului `G` este nodul `d`.
- Utilizând **algoritmul lui Floyd**, determinarea **centrului unui graf** `G` se poate realiza simplu.
- Presupunând că **ponderilor arcelor grafului** sunt memorate în matricea `COST`, se procedează astfel:
    - (1) Se determină cu ajutorul **procedurii Floyd** matricea drumurilor minime corespunzătoare tuturor perechilor de noduri (fig.12.2.4.a. (b)).
    - (2) Se determină **excentricităţile nodurilor** `i`, `(1≤i≤N)` găsind valoarea maximă de pe fiecare coloană `i` a matricei.
    - (3) Se caută nodul cu **excentricitatea minimă**. Acesta este centrul grafului `G`.
- În figura 12.2.4.a. (b) apare matricea valorilor drumurilor minime pentru graful din aceeaşi figură (a).
    - Determinarea nodului central al grafului este evidentă.



(a)

	a	b	c	d	e
a	0	1	3	5	7
b	∞	0	2	4	6
c	∞	3	0	2	4
d	∞	1	3	0	7
e	∞	6	8	5	0
max	∞	6	8	5	7

(b)

nod	excentricitate
a	∞
b	6
c	8
d	5
e	7

(c)

Fig.12.2.4.a. Determinarea centrului unui graf ponderat

12.3. Închiderea tranzitivă

- În grafurile neorientate, nodurile la care se poate ajunge pornind de la un nod precizat, cu alte cuvinte conexiunile unui nod la grafului, pot fi determinate simplu aplicând **proprietăţile de conectivitate** ale grafurilor.
    - Pur şi simplu, toate nodurile la care se poate ajunge în procesul de căutare aparţin unei aceleiaşi **componente conexe a grafului**.
- Această observaţie este valabilă în principiu şi în cazul **grafurilor orientate** cu precizarea însă că în această situaţie, rezolvarea este mai complexă şi ea **nu** poate fi redusă la simpla determinare a componentelor conexe.
- O modalitate de a rezolva problemele de conectivitate în cazul **grafurilor orientate** este următoarea:
    - Se **completează** graful iniţial cu arce pe baza următoarei metode: dacă în graful iniţial se poate ajunge într-un mod oarecare de la nodul `x` la nodul `y` parcurgând arcele orientate ale grafului, atunci se adaugă grafului arcul `(x,y)`.
- Graful care se obţine adăugând **toate arcele** de această natură se numeşte **închiderea tranzitivă** a grafului iniţial.
    - Deoarece este de aşteptat să fie adăugate un număr mare de arce, deci graful obţinut să fie dens, se apreciază că pentru **închiderea tranzitivă**, cea mai potrivită metodă de reprezentare este cea bazată pe **matrice de adiacenţe**.
- Odată determinată închiderea tranzitivă a unui graf orientat, răspunsul la întrebarea: "Există un drum în graful orientat de la nodul `x` la nodul `y`? " este imediat.
- **Algoritmul lui Floyd** poate fi specializat astfel încât să determine **închiderea tranzitivă** a unui graf `G`.
    - Algoritmul care rezultă se numeşte **algoritmul lui Warshall** şi care deşi a apărut tot în anul 1962 este anterior ca dată algoritmului lui Floyd.

12.3.1. Algoritmul lui Warshal

- **Algoritmul lui Warshall** se bazează pe observaţia simplă că, dacă într-un graf orientat există o modalitate de a ajunge de la nodul `i` la nodul `k` şi o modalitate de a ajunge de la nodul `k` la nodul `j`, parcurgând arce ale grafului, atunci cu siguranţă există un drum care conectează nodul `i` cu nodul `j`.
    - Problema constă de fapt în a exploata de asemenea manieră această observaţie încât calculul să se realizeze la **o singură trecere prin matrice**.
- Acest lucru este posibil în baza următoarei interpretări sugerate de **Warshall**:
    - "Dacă există o posibilitate de a ajunge de la nodul `i` la nodul `k` utilizând numai noduri cu indici mai mici decât `k`, şi o posibilitate de a ajunge de la

nodul k la nodul j în aceleaşi condiţii, atunci există un drum de la nodul i la nodul j care străbate numai noduri care cu indicele mai mic ca şi k+1".

- În baza acestei interpretări, dându-se graful ordonat G şi matricea sa de adiacenţe A, **algoritmul lui Warshall** determină matricea T care reprezintă **închiderea tranzitivă** a grafului G.
  - În această matrice T[i,j]=true, dacă există posibilitatea de a ajunge de la nodul i la nodul j, altfel T[i,j]= false.
- Spre exemplu în figura 12.3.a. (b) apare închiderea tranzitivă în forma matricii T a grafului orientat din figura 12.3.a. (a).

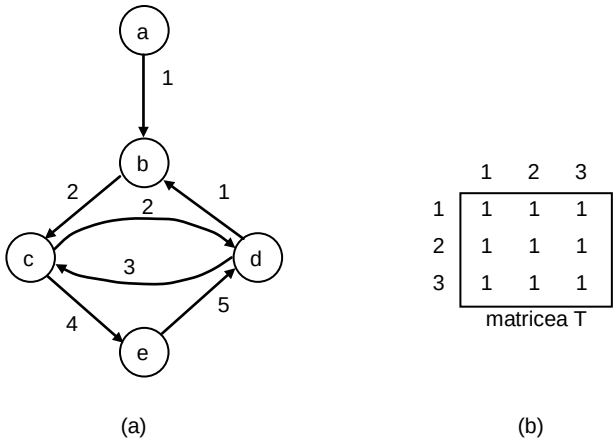


Fig.12.3.a. Închiderea tranzitivă a unui graf orientat

- **Închiderea tranzitivă** poate fi determinată aplicând o procedură similară procedurii **Floyd**, utilizând însă următoarea formulă de calcul în cea de-a k-a trecere prin matricea T (formula [12.3.a]):

$$T_k[i,j] = T_{k-1}[i,j] \text{ OR } (T_{k-1}[i,k] \text{ AND } T_{k-1}[k,j]) \qquad [12.3.a]$$

- Această formulă precizează că există un drum de la nodul i la nodul j care nu trece printr-un nod cu număr mai mare ca şi k dacă:
  - (1) Există deja un drum de la i la j care nu trece prin nici un nod cu număr mai mare decât k-1, **sau**
  - (2) Există un drum de la i la k care nu trece prin nici un nod cu număr mai mare decât k-1 **şi** există un drum de la k la j, care nu trece prin niciun nod cu număr mai mare decât k-1.
- Ca şi în cazul algoritmului lui Floyd, sunt valabile formulele $T_k[i,k]=T_{k-1}[i,k]$ şi $T_k[k,j]= T_{k-1}[k,j]$, adică matricea T **este invariantă** în raport cu k, motiv pentru care în calcul se poate utiliza o singură instanţă a matricii T.
- Structura de principiu a procedurii care implementează **algoritmul lui Warshall** apare în secvenţa [12.3.a].

```
/*Algoritmul lui Warshall – varianta C*/
```

```
void Warshall(Boolean A[n][n], Boolean T[n][n])

/*Procedura construieşte în T închiderea tranzitivă a lui
A*/
{
      int i,j,k;

      /*iniţializarea matricei T*/
  [1] for (i=0;i<n;i++)
  [2]   for (j=0;j<n;j++)
  [3]     T[i,j]=A[i,j];                          /*[12.3.a]*/

      /*determinarea matricei T*/
  [4] for (k=0;k<n;k++)
  [5]   for (i=0;i<n;i++)
  [6]     for (j=0;j<n;j++)
  [7]       if (T[i,j]==false)
  [8]         T[i,j]= T[i,k] && T[k,j];
}
/*Warshall*/
```

12.3.2. Analiza performaţei algoritmului lui Warshal

- În urma unei analize sumare a codului, performanţa algoritmului rezultă imediat ca fiind egală cu $O(n^3)$.
- După alţi autori, performanţa poate fi stabilită şi astfel:
  - Fiecare din cele `a` arce conduce la o iteraţie cu `n` paşi în bucla **`for`** cea mai interioară.
  - În plus, sunt testate şi eventual actualizate toate cele n^2 locaţii ale matricii `T`.
  - Rezultă un timp de execuţie proporţional cu $O(an + n^2)$ [Se 88].

12.4. Traversarea grafurilor orientate

- Tehnicile fundamentate de traversare a grafurilor dezvoltate în cadrul capitolului 10, au un **caracter universal** şi ele pot fi aplicate, cu particularizări specifice, oricărui tip de graf.
  - Astfel în cazul **grafurilor orientate**, în timpul traversării se ţine cont efectiv de **orientarea arcelor** examinate, lucru care în contextul grafurilor neorientate **nu** este necesar.
  - Din acest motiv şi arborii de acoperire rezultaţi în urma procesului de traversare a grafurilor orientate au o structură mai complicată.
- În cele ce urmează vor fi abordate unele aspecte legate de **traversarea grafurilor orientate** prin tehnica "**căutării în adâncime**".
  - Opţiunea este motivată de faptul că această manieră de traversare a grafurilor orientate stă la baza rezolvării a numeroase probleme practice care se pun în legătură cu această categorie de grafuri.

12.4.1. Traversarea grafurilor orientate prin tehnica căutării "în adâncime"

- Principiul **traversării în adâncime** este deja cunoscut.
  - Se presupune că există un graf orientat `G` în care toate nodurile sunt marcate iniţial cu “nevizitat”.
  - Căutarea în adâncime acţionează iniţial selectând un nod `x` al lui `G` ca şi nod de start, nod care este marcat cu “vizitat”.
  - În continuare, fiecare nod nevizitat adiacent lui `x` este "căutat" pe rând, utilizând aceeaşi tehnică în manieră **recursivă**.
  - În momentul în care toate nodurile la care se poate ajunge pornind de la `x` au fost vizitate, traversarea este încheiată.
  - Dacă totuşi mai rămân noduri nevizitate, se selectează unul dintre acestea drept următorul nod de start şi se relansează căutarea recursivă.
  - Acest proces se repetă până când sunt parcurse toate nodurile grafului `G`.
- Pentru implementare se consideră că graful `G` care se doreşte a fi traversat este reprezentat cu ajutorul **listelor de adiacenţe**.
  - Se notează cu `L(x)` lista de adiacenţe a nodului `x`.
  - De asemenea se mai utilizează tabloul `marc`, ale cărui elemente pot lua valorile "`nevizitat`" respectiv "`vizitat`", tablou care este utilizat în menţinerea evidenţei stării nodurilor grafului.
- Structura de principiu a procedurii recursive de **căutare în adâncime** apare în secvenţa [12.4.1.a].
  - Această procedură trebuie apelată însă dintr-un context mai general care asigură iniţializarea tabloului marc şi selectarea nodurilor încă nevizitate ale grafului (secvenţa [12.4.1.b]).

```
-----------------------------------------------------------
/*Căutare în adâncime - varianta pseudocod*/

procedura CautInAdâncime(tip_nod x)

/*varianta pseudocod CautInAdacime*/

    tip_nod k;

[1] marc[x]= vizitat;
[2] Prelucrare(x);
[3] pentru (fiecare nod k din L(x))          /*[12.4.1.a]*/
[4]   daca (marc[k] este nevizitat)
[5]     CautInAdâncime(k);
  /*CautInAdâncime*/
-----------------------------------------------------------
/*Programul principal*/

  /*iniţializare tablou mark*/
  pentru (x=1 la n)
    marc[x]= nevizitat;
  /*determinare componente conexe*/
  pentru (x= 1 la n)                         /*[12.4.1.b]*/
```

```
    daca (marc[x] este nevizitat)
      CautInAdâncime(x);
-------------------------------------------------------------
```

- Se face precizarea, că procedura **`CautInAdâncime` nu** realizează nici o prelucrare efectivă a nodurilor vizitate, aceasta fiind sugerată generic prin apelativul **`Prelucrare(x)`** în linia `[2]` a secvenţei [12.4.1.a].

- După cum s-a precizat, această tehnică stă la baza rezolvării mai multor probleme specifice grafurilor orientate.
  - În consecinţă, prelucrarea nodurilor vizitate va îmbrăca o formă specifică funcţie de problema care se doreşte a fi soluţionată în baza acestei tehnici de parcurgere a grafurilor orientate.
- **Performanţa procedurii `CautInAdâncime`**, analizată în contextul parcurgerii unui graf orientat cu `a` arce, cu `n≤a`, în reprezentarea bazată pe liste de adiacenţe este *O(a)*, după cum s-a evidenţiat în capitolul 10.
- Pentru **exemplificarea** procesului de traversare a grafurilor orientate prin metoda căutării în adâncime:
  - Se presupune că procedura din secvenţa [12.4.1.b] este aplicată grafului orientat din figura 12.4.1.a. (a), considerând că nodul de pornire este nodul `a` `(x=a)`.
  - Graful se consideră reprezentat printr-o **structură de adiacenţe** sugerată în aceeaşi figura (b).

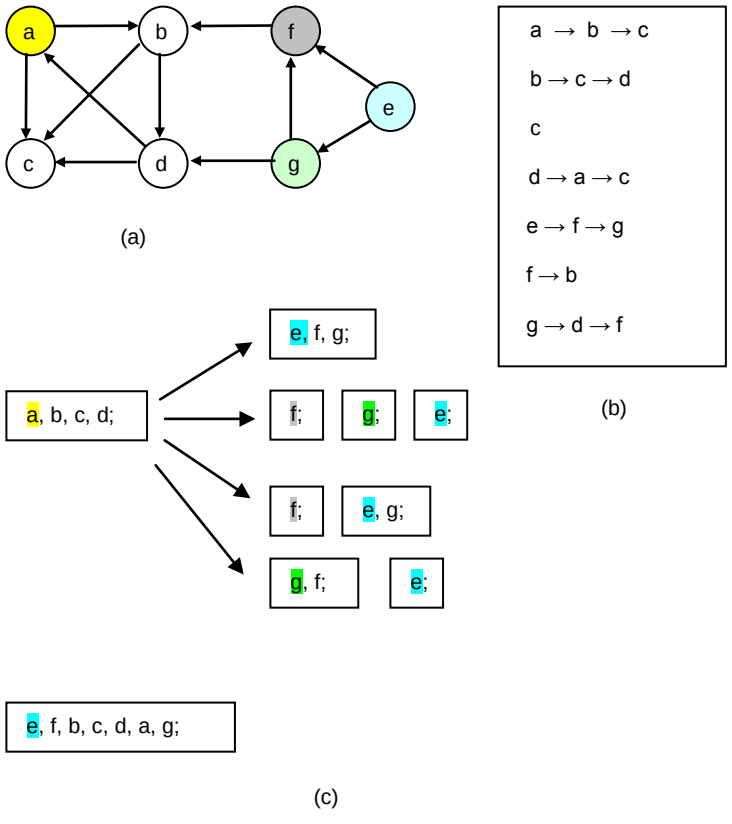


Fig.12.4.1.a. Traversarea unui graf orientat

- Algoritmul marchează nodul `a` cu vizitat şi selectează nodul `b`, primul din lista de adiacenţe a lui `a`.

- Deoarece nodul `b` este nevizitat, căutarea continuă prin realizarea unei apel **`CautInAdâncime`**`(b)`.
- În continuare se marchează `b` cu vizitat şi se selectează primul nod din lista sa de adiacenţe.
- Presupunând că nodul `c` apare înaintea lui `d` în această listă, aşa cum se prezintă în figură, se apelează **`CautInAdâncime`**`(c)`.
- Deoarece lista de adiacenţe a lui `c` este vidă, căutarea revine în lista lui `b` de unde se selectează nodul `d`.
- Se parcurge în continuare lista de adiacenţe a lui `d`.
- Întrucât nodurile `a` şi `c` care apar în lista de adiacenţe a lui `d` au fost deja vizitate, căutarea lui `d` se încheie şi se revine în lista lui `b` apoi în cea a lui `a`, care sunt şi ele epuizate.
- În acest moment apelul iniţial **`CautInAdâncime`**`(a)` s-a terminat.
- Totuşi, deoarece graful nu a fost încă parcurs în întregime întrucât nodurile `e,f` şi `g` sunt încă nevizitate, se realizează un nou apel al procedurii **`CautInAdâncime`** spre exemplu pentru nodul `e`, apel care asigură parcurgerea integrală a grafului.

- Se face următoarea **observaţie**:
  - Dacă graful din fig.12.4.1.a. (a) ar fi un **graf neorientat**, el ar reprezenta **o singură componentă conexă** şi ar putea fi parcurs integral, realizând un singur apel al procedurii **`CautInAdâncime`** din cadrul secvenţei [12.4.1.b] **indiferent** de nodul care este selectat drept **punct de start.**
- În cazul **grafurilor orientate** situaţia se schimbă.
  - Numărul de apeluri nerecursive ale procedurii **`CautInAdâncime`**, realizat în cadrul secvenţei [12.4.1.b] depinde în mod esenţial de ordinea în care sunt selectate nodurile care reprezintă puncte de start.
    - Spre exemplu în cazul mai sus prezentat, deoarece s-a ales iniţial nodul `a` şi apoi nodul `e` drept puncte de pornire, graful apare constituit din două componente conexe `(a,b,c,d)` şi `(e,f,g).`
    - Dacă în locul nodului `e`, la cel de-al doilea apel se selectează nodul `g` apoi `e`, rezultă trei componente conexe `(a,b,c,d),(g,f)` şi `(e).`
    - Dacă în locul nodului `e`, la cel de-al doilea apel se selectează nodurile `f`,`g` şi `e` în această ordine rezultă 4 componente conexe.
    - Dacă în locul nodului `e`, la cel de-al doilea apel se selectează nodurile `f`,`e` şi `g` în această ordine rezultă 3 componente conexe.
    - Dacă la primul apel, se alege nodul `e` drept nod de start, graful din figură conţine o singură componentă conexă (fig.12.4.1.a. (c)).
  - Se putea însă alege iniţial oricare alt nod drept punct de pornire a parcurgerii, iar ulterior oricare dintre nodurile rămase, fiecare situaţie conducând la o structură diferită de componente conexe ale grafului.
- Se poate concluziona că **numărul de componente conexe ale unui graf orientat** depinde în mod esenţial de **ordinea în care sunt selectate nodurile** care reprezintă puncte de start ale componentelor.

- Această particularitate se reflectă direct în topologia **arborilor de acoperire** care rezultă în urma unor astfel de parcurgeri ale grafurilor orientate.

12.4.2. Păduri de arbori de căutare în adâncime pentru grafuri orientate

- În procesul de traversare al unui graf orientat, anumite arce conduc la noduri nevizitate.
    - Arcele care conduc la noduri nevizitate se numesc "**arce de arbore**" şi ele formează un **arbore** respectiv o **pădure de arbori de căutare în adâncime** pentru graful orientat dat.
- În figura 12.4.2.a apare o astfel de **pădure de arbori de căutare în adâncime** corespunzătoare grafului orientat din fig. 12.4.1.a. (a).

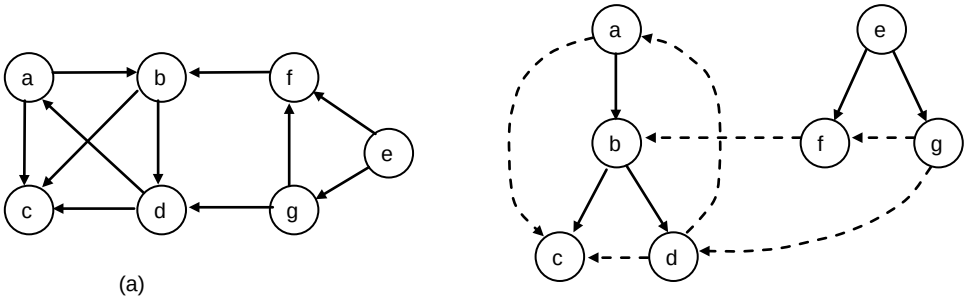


Fig.12.4.2.a. Pădure de arbori de căutare în adâncime pentru graful din fig. 12.4.1.a. (a)

- După cum se observă în această figură, în afara "**arcelor de arbore**" trasate cu linie continuă, mai apar şi alte categorii specifice de arce rezultate din parcurgerea grafurilor orientate prin tehnica căutării în adâncime.
    - Este vorba despre:
        - (1) Arce de tip "**înapoi**" ("**back**").
        - (2) Arce de tip "**înainte**" ("**forward**").
        - (3) Arce numite de "**trecere**" ("**cross**").
            - Toate aceste tipuri de arce apar trasate cu **linie întreruptă** în cadrul figurii.
- Se reaminteşte faptul că în contextul **grafurilor neorientate** pentru **arborii de căutare în adâncime** se definesc numai două tipuri de arce: arce "**de arbore**" şi arce "**de retur**".
- (1) În contextul **grafurilor orientate**, un arc ca şi arcul `(d,a)` se numeşte **arc de tip** "**înapoi**" deoarece el conectează un nod cu unul din **strămoşii săi** din cadrul arborelui de căutare.
    - Se precizează faptul că un arc care conectează un nod cu el însuşi este încadrat tot în această categorie.
- (2) Un arc care conectează un nod cu unul din **descendenţii săi proprii** se numeşte **arc de tip** "**înainte**".
    - În figura 12.4.2.a un astfel de arc este `(a,c)`.

- (3) Un arc de tip `(d,c)` sau `(g,d)` care conectează două noduri care **nu** sunt în relaţie **descendent** sau **strămoş** se numeşte **arc** "**de trecere**".
    - Se observă că toate arcele de trecere din figura 12.4.2.a sunt orientate de la dreapta la stânga, pe baza presupunerii că:
        - (a) Nodurile fii nou descoperite sunt adăugate în arborii de căutare în adâncime de la **stânga la dreapta**, în ordinea în care sunt vizitate.
        - (b) Noii arbori sunt adăugaţi pădurii tot de la **stânga la dreapta**.
    - Această presupunere nu este întâmplătoare şi ea este legată de observaţia formulată în paragraful anterior relativ la dependenţa structurii topologice a pădurii arborilor de căutare de ordinea de vizitare a nodurilor.
- **Problema** care se pune în continuare se referă la modul în care pot fi individualizate cele patru categorii de arce.
    - În primul rând este evident faptul că arcele "**de arbore**" sunt o categorie specială de arce, ele conducând la **noduri nevizitate** din cadrul grafului, motiv pentru care pot fi depistate cu uşurinţă.
- Pentru a depista celelelalte categorii de arce, care toate conduc la noduri care au fost deja vizitate, este necesar a se introduce o **numerotare a nodurilor** grafului orientat în ordinea în care acestea sunt selectate în procesul de **căutare în adâncime**.
    - Aceste numere pot fi memorate într-un tablou `OrdineInAdâncime` de valori întregi, tablou care conţine o locaţie pentru fiecare nod al grafului.
    - Tabloul poate fi completat pe baza secvenţei [12.4.2.a] care se introduce după linia `[1]` a procedurii **`CautInAdâncime`** din secvenţa [12.4.1.a].

```
-------------------------------------------------------------
        /*completare procedura CautareInAdancime*/

        contor= contor+1;
        OrdineInAdâncime[x]=contor;        /*[12.4.2.a]*/
-------------------------------------------------------------
```

- Se precizează faptul că această manieră de numerotare a mai fost utilizată atât pentru grafuri, cât şi pentru alte structuri de date ca de exemplu arborii, ca un element care cuantizează esenţa procesului de traversare a unei structuri de date.
    - Se reaminteşte de asemenea faptul că **esenţa procesului de traversare** constă în **transformarea structurii** supusă traversării într-o **structură listă liniară**.
- În consecinţă, în timpul traversării unui graf orientat, tuturor descendenţilor unui nod oarecare `x` li se vor atribui **numere** mai mari decât lui `x` în tabloul `OrdineInAdâncime`.
- În acest context este valabilă următoarea teoremă: `y` este un **descendent** al lui `x` dacă şi numai dacă este satisfăcută relaţia [12.4.2.b].

```
-------------------------------------------------------------
OrdineInAdâncime[x]≤OrdineInAdâncime[y]≤OrdineInAdâncime[x] +
"numărul de descendenţi ai lui x"                [12.4.2.b]
-------------------------------------------------------------
```

- În aceste condiţii:
    - (1) Un **arc de tip** "**înainte**" conectează un nod cu un număr de ordine mai mic cu un nod având un număr de ordine mai mare.

- (2) Un **arc de tip** "**înapoi**" conectează un nod cu număr mai mare cu un nod având numărul de ordine mai mic.
    - Se face precizarea că în ambele situaţii, cele două noduri trebuie să fie în **relaţia strămoş sau descendent**, respectiv pentru ele trebuie să fie satisfăcută **relaţia** [12.4.2.b].
- (3) **Arcele** "**de trecere**" conectează noduri cu numere de ordine mai mari cu noduri cu numere mai mici, care **nu** sunt în **relaţia strămoş sau descendent** deci pentru care relaţia [12.4.2.b] **nu** este valabilă.

- Pentru a demonstra această afirmaţie se procedează prin **reducere la absurd**.
    - Se presupune ca `(x,y)` este un **arc "de trecere"** şi că `OrdineInAdâncime[x]`≤`OrdineInAdâncime[y]`, adică `x` şi `y` sunt în relaţia strămoş-descendent.
    - Deoarece numărul lui `x` este mai mic sau egal cu al lui `y`, rezultă că nodul `x` este vizitat înaintea lui `y`.
    - În consecinţă, orice nod vizitat în timpul scurs între primul apel al procedurii **`CautInAdâncime`**`(x)` şi terminarea acestui apel, este un descendent al nodului `x` în procesul de căutare în adâncime.
    - Nodul `y` se află într-o astfel de situaţie.
        - Dacă `y` este un nod nevizitat în momentul explorării arcului `(x,y)`, atunci arcul `(x,y)` este un arc "**de arbore**".
        - Altfel, nodul `y` a fost cu siguranţă deja vizitat anterior momentului explorării arcului `(x,y)` şi în consecinţă arcul`(x,y)` este un arc de tip "**înainte**".
    - Întrucât concluzia la care s-a ajuns în ambele situaţii contrazice ipoteza iniţială, rezultă că **nu** poate exista un arc de trecere `(x,y)` astfel încât `OrdineInAdâncime[x]`≤`OrdineInAdâncime[y].`

- În următoarele două secţiuni, se prezintă câteva dintre modalităţile de utilizare a traversării grafurilor orientate prin tehnica căutării în adâncime la rezolvarea diverselor probleme legate de acest tip de grafuri.

12.5. Grafuri orientate aciclice

- Un **graf orientat aciclic** este un graf orientat care **nu** conţine cicluri.
- Apreciate în termenii relaţiilor pe care le modelează, **grafurile orientate aciclice** au un caracter de generalitate mai pronunţat decât **arborii** dar sunt mai puţin generale ca şi **grafurile orientate**.
- Ca şi **topologie**, astfel de grafuri pot conţine multe cicluri, dacă **nu** se ia în considerare direcţionarea arcelor.
    - Dacă însă se ia în considerare direcţionarea arcelor un astfel de graf **nu** conţine nici un ciclu.
- De fapt aceste grafuri pot fi considerate parţial **arbori**, parţial **grafuri orientate**, element care le conferă o serie de proprietăţi speciale.
    - Spre exemplu, **pădurea de arbori de căutare în adâncime** asociată unui graf

orientat aciclic, **nu** conţine arce de tip "înapoi".

- În figura 12.5.a. apare un exemplu de arbore (a), un exemplu de graf orientat aciclic (b) şi un exemplu de graf orientat cu un ciclu (c).

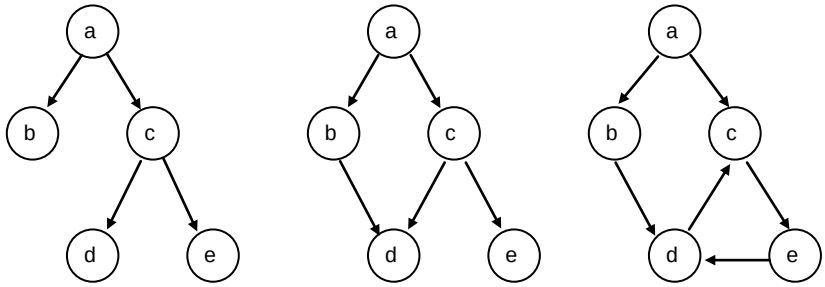


Fig.12.5.a. Tipuri de grafuri orientate

- Printre altele, **grafurile orientate aciclice** sunt utile în reprezentarea sintetică a structurii **expresiilor aritmetice** care conţin **subexpresii comune**.
  - Spre exemplu figura 12.5.b. evidenţiază un astfel de graf pentru expresia aritmetică:

```
------------------------------------------------------------
                ((a+b)*c+((a+b)+e)*(e+f))*((a+b)*c)
------------------------------------------------------------
```

  - Termenii `a+b` şi `(a+b)*c` reprezintă **subexpresii comune partajate**, motiv pentru care sunt reprezentate prin noduri care sunt destinaţia mai multor arce orientate.

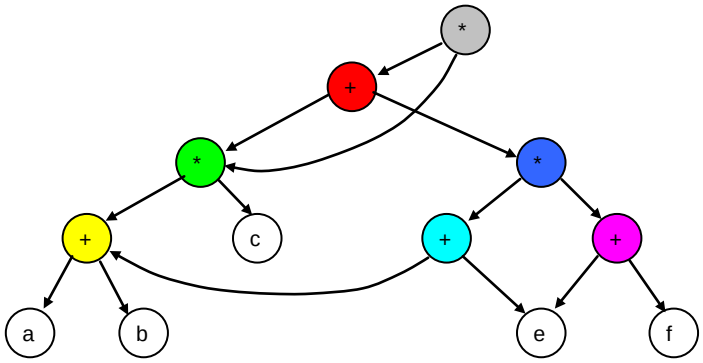


Fig.12.5.b. Graf orientat aciclic reprezentând o expresie aritmetică

- **Grafurile orientate aciclice** pot fi utilizate cu succes în modelarea **relaţiei de ordonare parţială**.
  - Se reaminteşte faptul că o **relaţie de ordonare parţială R** pe o mulţime `M`, este o **relaţie binară** care satisface următoarele proprietăţi:
    - (1) `aRa` este falsă pentru $\forall$ `a`$\in$`M` (**nereflexivitate**).
    - (2) Pentru $\forall$ `a,b,c`$\in$`M`, dacă `aRb` şi `bRc` sunt adevărate, atunci `aRc` este adevărată (**tranzitivitate**).
    - (3) Dacă `aRb` este adevărată şi `bRa` este adevărată, atunci `a=b` pentru $\forall$ `a,b`$\in$`M` (**antisimetrie**).

- Două exemple naturale de relaţii de ordonare parţială sunt:
  - (1) Relaţia "**mai mic decât**" (<) definită pe **mulţimea numerelor întregi.**
  - (2) Relaţia "**submulţime proprie a unei mulţimi**" ($\subset$) definită pentru **mulţimi de elemente.**
- Spre exemplu fie **mulţimea** `M={1,2,3}` şi fie `P(M)` **puterea mulţimii** `M`, adică mulţimea tuturor submulţimilor proprii ale mulţimii M.
  - În acest context `P(M)={{Ø},{1},{2},{3},{1,2},{1,3},{2,3},{1,2,3}}`.
  - Este simplu de verificat că relaţia "submulţime a mulţimii `M`" ($\subset$) este o **relaţie de ordonare** parţială pentru puterea mulţimii `M`.
- **Grafurile orientate aciclice** pot fi utilizate în **reprezentarea** unor astfel de relaţii.
- În acest, scop o **relaţie R** poate fi asimilată cu o **mulţime de perechi** (arce), care satisfac următoarea proprietate: perechea `(a,b)` aparţine mulţimii **R** dacă a**R**b este adevărată.
- În aceste condiţii, este valabilă următoarea **definiţie**:
  - Dacă **R** este o **relaţie de ordonare parţială** pe o mulţime `M`, atunci graful `G=(M,R)` este un **graf orientat aciclic**.
- **Reciproc**:
  - Se presupune că `G=(M,R)` este un **graf orientat aciclic.**
  - Se presupune de asemenea că $\mathbf{R}^+$ este o **relaţie** definită prin afirmaţia a$\mathbf{R}^+$b este adevărată dacă şi numai dacă există un drum de lungime mai mare sau egală cu `1` care conectează nodul `a` cu nodul `b` în graful `G`
  - În aceste condiţii, $\mathbf{R}^+$ reprezintă o **relaţie de ordonare parţială** pe `M`.
    - $\mathbf{R}^+$ este de fapt mulţimea relaţiilor care materializează **închiderea tranzitivă** a lui **R**.
- În figura 12.5.c apare graful orientat aciclic `G=(P(M),R)` unde `M={1,2,3}`.
  - Relaţia $\mathbf{R}^+$ se traduce prin **submulţime proprie** a puterii mulţimii `M`.

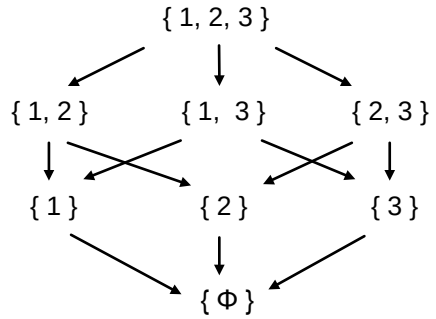


Fig.12.5.c. Graf orientat aciclic reprezentând submulţimile proprii ale unei mulţimi date

12.5.1. Determinarea aciclităţii unui graf orientat

- Se consideră un graf orientat G=(N,A) şi se cere să se stabilească dacă G este aciclic, cu alte cuvinte să se determine dacă G nu conţine cicluri.
- Pentru a rezolva această problemă poate fi utilizată cu succes **tehnica căutării în adâncime**.
    - Astfel, dacă pe parcursul traversării grafului G prin **tehnica căutării în adâncime** se întâlneşte cel puţin un arc de tip "**înapoi**", în mod evident graful conţine cel puţin un **ciclu**.
    - **Reciproc**, dacă un graf orientat conţine cel puţin un **ciclu**, atunci în orice traversare a grafului prin tehnica căutării în adâncime va apare cel puţin un arc de tip "**înapoi**".
- Pentru a **demonstra** acest lucru, se presupune că G este **ciclic**.
    - Realizând o traversare prin căutare în adâncime în G, va exista cu siguranţă în ciclu un nod x, al cărui număr de ordine la căutarea în adâncime este mai mic decât numărul corespunzător oricărui nod aparţinând ciclului.
    - Se consideră un arc (y,x) din ciclul care-l conţine pe x.
    - Deoarece y este în ciclu, el trebuie să fie un **descendent** al lui x în pădurea de arbori de căutare prin cuprindere.
    - Deci (y,x) **nu** poate fi un arc de "trecere".
    - Deoarece numărul de ordine la căutarea în adâncime a lui y este mai mare decât numărul lui x, (y,x) **nu** poate fi nici arc de tip "arbore" nici arc de tip "înainte".
    - În consecinţă (y,x) **nu** poate fi decât un arc de tip "înapoi".

12.5.2. Aplicaţie. Sortarea topologică

- **Sortarea topologică** a mai făcut obiectul unor prezentări pe parcursul acestei lucrări.
- În paragraful de faţă se prezintă o altă modalitate de soluţionare a **sortării topologice** utilizând drept structuri de date suport **grafurile orientate aciclice**.
- Se reaminteşte faptul că un **proiect de mari dimensiuni**, este adesea divizat într-o colecţie de activităţi (task-uri) mai mici, fiecare având un caracter mai mult sau mai puţin unitar.
    - În vederea finalizării proiectului, unele dintre activităţi trebuiesc realizate într-o anumită ordine specificată.
- Spre exemplu, o **programă universitară** poate conţine o serie precizată de cursuri dintre care unele presupun în mod obligatoriu absolvirea în prealabil a altora (pre requisite).
    - O astfel de situaţie poate fi modelată simplu cu ajutorul **grafurilor orientate aciclice.**
    - Spre exemplu dacă cursul D este un curs care nu poate fi urmat decât după absolvirea cursului C, în graf apare un arc (C,D).
- În figura 12.5.1.a apare un **graf orientat aciclic** care evidenţiază intercondiţionările de această natură impuse unui număr de 5 cursuri.

- Cursul `C3` spre exemplu, presupune absolvirea în prealabil a cursurilor `C1` şi `C2`.

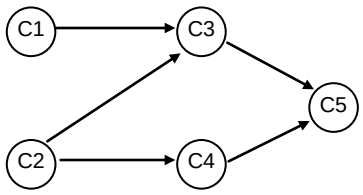


Fig.12.5.1.a. Graf orientat aciclic modelând intercondiţionări

- **Sortarea topologică** realizează o astfel de **ordonare liniară** a activităţilor proiectului încât niciuna dintre intercondiţionările impuse să **nu** fie încălcate.
    - Cu alte cuvinte, **sortarea topologică** este un **proces de ordonare liniară** a nodurilor unui **graf orientat aciclic**, astfel încât dacă există un arc de la nodul `i` la nodul `j`, atunci nodul `i` să apară înaintea nodului `j` în ordonarea liniară.
- Spre exemplu lista `C1,C2,C3,C4,C5` reprezintă o sortare topologică a grafului din figura 12.5.1.a.
- Sortarea topologică poate fi realizată simplu pornind de la algoritmul traversării unui graf prin **tehnica căutării în adâncime**.
    - Astfel, dacă în procedura **`CautInAdâncime`** din secvenţa [12.4.1.a] se adaugă linia `[5]` care conţine o instrucţiune de tipărire a lui `x`, se obţine procedura **`SortTopologic`** care afişează nodurile grafului aciclic **sortate topologic în ordine inversă** (secvenţa [12.5.1.a]).
    - Acest lucru se întâmplă deoarece procedura **`SortTopologic`** afişează un nod oarecare `x` al grafului aciclic după ce a terminat explorarea (afişarea) tuturor descendenţilor săi.

```
-------------------------------------------------------------
PROCEDURE SortTopologic(tip_nod x);

/*Afişează nodurile accesibile pornind de la nodul x, în
ordine topologică inversă*/

     tip_nod k;
                                              /*[12.5.1.a]*/
[1]  marc[x]=vizitat;
[2]  pentru fiecare nod k din L(x)
[3]   daca (marc[k] este nevizitat)
[4]     SortTopologic(k);
[5]  *scrie(x);
/*SortTopologic*/
-------------------------------------------------------------
```

- **Precizare**: realizarea sortării topologice a unui graf orientat, este echivalentă cu realizarea unei sortări topologice de tip **`SortTopologic`** în graful obţinut prin **inversarea sensului arcelor grafului** [Se 88].
- Este de asemenea evident faptul că, de regulă, ordonarea produsă de acest tip de sortare **nu** este unică.

- Tehnica utilizată este perfect funcţională deoarece la parcurgerea unui graf orientat aciclic **nu** apar arce de tip “înapoi”.
- Acest lucru se **demonstrează** după cum urmează:
  - Se consideră momentul la care căutarea în adâncime părăseşte nodul x.
  - Toate arcele care apar în pădurea de arbori de căutare în adâncime asociată grafului şi care provin din nodul x, sunt sau arce “de arbore”, sau arce de tip "înainte”, sau arce de "trecere".
  - Toate acest arce sunt însă direcţionate spre noduri a căror vizită s-a încheiat şi în consecinţă, evidenţierea lor precede evidenţierea lui x în cadrul ordinii care se construieşte.

12.6. Componente puternic conectate

- O **componentă puternic conectată** a unui **graf orientat** este o submulţime de noduri ale grafului în care există un drum de la **oricare** nod al mulţimii la **oricare** alt nod aparţinând aceleeaşi mulţimi.
- Fie $G=(N,A)$ un **graf orientat**.
- Se partiţionează mulţimea N a nodurilor grafului G în **clase de echivalenţă** $N_i, (1 \le i \le r)$, pe baza următoarei **definiţii a relaţiei de echivalenţă**:
  - Nodurile x şi y sunt **echivalente** dacă şi numai dacă există un drum de la nodul x la nodul y şi un drum de la nodul y la nodul x.
- Fie $A_i, (1 \le i \le r)$ mulţimea arcelor ale căror surse şi destinaţii aparţin mulţimii N_i.
- Grafurile $G_i=(N_i,A_i)$ se numesc **componentele puternic conectate** ale grafului G, sau mai simplu **componente puternice**.
- Un **graf orientat** care constă dintr-o singură componentă puternică se numeşte **puternic conectat**.
- În figura 12.6.a apare un graf orientat (a), care conţine două componente puternice (b).

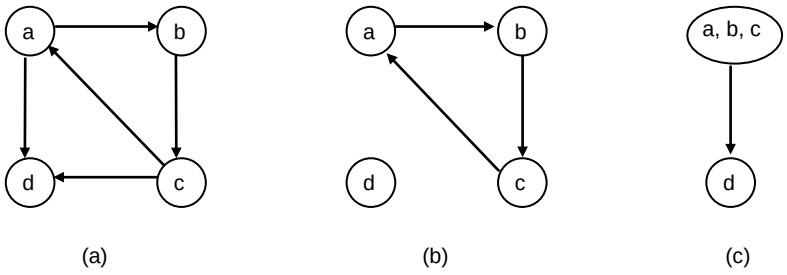


Fig.12.6.a. Componente puternic conectate şi graful redus al unui graf orientat

- Se face precizarea că fiecare **nod** al unui graf orientat aparţine unei **componente puternice** dar există **arce** care **nu** aparţin nici unei componente.
  - Astfel de arce se numesc **arce de “transfer”** şi ele conectează noduri aparţinând la **componente puternice** diferite.

- Considerând **componentele puternice** drept **noduri** şi reprezentând interconexiunile dintre componentele puternice ale unui graf `G`, respectiv reprezentând **arcele de transfer**, se obţine **graful redus** al grafului `G`.
    - Nodurile **grafului redus** sunt **componentele puternice** ale grafului orginal.
    - În **graful redus** apare un arc de la componenta puternică C_i la componenta puternică C_j dacă în graful original `G` există un arc ce leagă un nod aparţinând lui C_i cu un nod aparţinând lui C_j.
    - **Graful redus** este întotdeauna un **graf orientat aciclic**, deoarece dacă ar conţine vreun ciclu, atunci toate componentele din acel ciclu aparţin unei aceleeaşi componente puternic conectate, element ce ar evidenţia faptul că determinarea componentelor puternice pentru graful original s-a realizat defectuos.
- În figura 12.6.1. (c) apare graful redus al grafului orientat din fig. 12.6.1. (a).
- În continuare se vor prezenta doi algoritmi pentru determinarea **componentelor puternic conectate ale unui graf orientat**, ambii bazaţi pe **tehnica căutării în adâncime**.

12.6.1. Algoritmul lui Kosaraju-Sharir

- Algoritmul pentru determinarea **componentelor puternic conectate** ale unui **graf orientat** `G` a fost sugerat în anul 1978 de R.Kosaraju (nepublicat) şi publicat în anul 1981 de către Sharir, motiv pentru care a fost denumit **algoritmul Kosaraju–Sharir**.
- **Algoritmul Kosaraju–Sharir** constă din următorii paşi:
    - (1) Se realizează o **traversare prin căutare în adâncime** a grafului `G` şi **se numerotează nodurile** în ordinea terminării apelurilor recursive corespunzătoare lor.
        - Acest lucru se poate realiza, spre exemplu, numerotând nodul `x` după linia `[5]` a procedurii **`CautInAdancime`** din secvenţa [12.4.1.a].
    - (2) Se construieşte un nou **graf orientat** `Gr`, **inversând sensul** tuturor arcelor grafului original `G`.
    - (3) Se realizează o **traversare prin căutare în adâncime** în graful `Gr`, începând cu **nodul** care are **numărul cel mai mare** conform numerotării de la pasul 1.
    - (4) Dacă această traversare **nu** acoperă toate nodurile, se lansează următoarea traversare începând cu **nodul** care are **numărul cel mai mare** dintre nodurile neparcurse încă.
    - (5) Se continuă în aceeaşi manieră până la epuizarea tuturor nodurilor.
- Fiecare **arbore de căutare în adâncime** al **pădurii** care rezultă, este o **componentă puternic conectată** a grafului `G`.
- În figura 12.6.1.a este ilustrată aplicarea algoritmului Kosaraju-Sharir asupra grafului orientat (a).

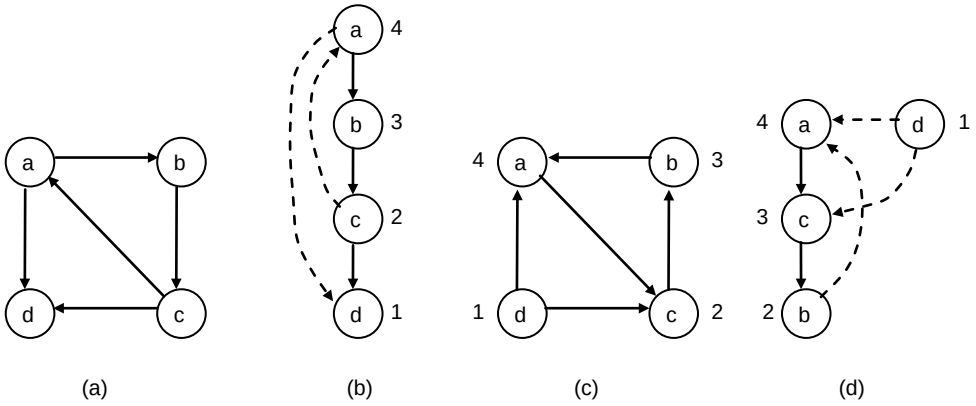


Fig.12.6.1.a. Determinarea componentelor puternic conectate ale unui graf orientat

- Astfel, după traversarea grafului începând cu nodul a şi continuând cu b, se obţine arborele de căutare în adâncime şi numerotarea nodurilor din fig. 12.6.1.a. (b).
- Inversând sensurile arcelor grafului original, rezultă graful Gr (fig.12.6.1.a. (c)).
- În continuare, realizând o traversare prin căutare în adâncime a lui Gr rezultă pădurea din aceeaşi figură (d).
- Într-adevăr, traversarea începe cu nodul a, considerat că rădăcină, deoarece a are cel mai mare număr.
    - Din a se ajunge la nodul c şi la nodul b.
    - Următorul arbore de căutare în adâncime are rădăcina d, deoarece d este nodul cu numărul cel mai mare dintre cele rămase neparcurse.
- Fiecare arbore al acestei păduri formează o **componentă puternic conectată** a grafului orientat original G.
- S-a afirmat că nodurile unei componente puternic conectate a unui graf, corespund exact nodurilor unui arbore al pădurii arborilor de acoperire rezultate în urma celei de-a doua traversări a grafului, respectiv a traversării grafului inversat.
- Pentru a **demonstra** acest lucru se porneşte de la următoarea observaţie:
    - Dacă x şi y sunt noduri aparţinând unei aceleeaşi componente puternice, atunci în G există un drum de la x la y şi un drum de la y la x.
    - În consecinţă şi în graful inversat Gr există un drum de la x la y şi un drum de la y la x.
- Se presupune că în traversarea lui Gr, căutarea începe cu o rădăcină r şi că se ajunge la x sau y.
    - Deoarece x şi y sunt conectaţi în ambele sensuri, atât x cât şi y vor aparţine arborelui de căutare în adâncime cu rădăcina r.
- Presupunând în continuare că x şi y aparţin unui aceluiaşi arbore de căutare în adâncime derivat din Gr, trebuie demonstrat că cele două noduri aparţin şi aceleeaşi componente puternic conectate.
    - Fie r rădăcina arborelui de căutare în adâncime în Gr care-i conţine pe x şi pe y.
    - Deoarece x este un descendent al lui r, rezultă că în Gr există un drum de la r la

x. În consecinţă în G există un drum de la x la r.

- Trebuie demonstrat în continuare că în G există şi un drum de la r la x.

- În construcţia pădurii de căutare în adâncime corespunzătoare lui Gr, nodul x era nevizitat în momentul în care se iniţia căutarea pentru r.

  - Acest lucru este motivat de faptul că r are un număr mai mare ca şi x, deoarece în căutarea în adâncime a lui G apelul recursiv al lui x se termină înaintea apelului recursiv a lui r.

- Se pune următoarea întrebare: "Ar fi putut, în căutarea în adâncime a lui G ca vizitarea nodului x să fie demarată înainte de vizitarea nodului r?"

  - Acest lucru este imposibil întrucât existenţa drumului în G de la x la r presupus anterior, ar implica faptul că vizitarea lui r să înceapă şi să se termine înaintea terminării vizitării lui x.

- În concluzie, în traversarea lui G, x este vizitat în timpul căutării lui r, astfel x este un descendent al lui r în arborele de căutare în adâncime al lui G.

  - Deci există un drum de la r la x în G şi în consecinţă nodurile x şi r aparţin unei aceleeaşi componente puternic conectate a grafului G.

- Printr-o argumentare identică se demonstrează că şi r şi y aparţin unei aceleeaşi componente puternic conectate.

  - Faptul că în ambele situaţii este vorba de una şi aceeaşi componentă este certificat de existenţa drumului x→r→y, respectiv de existenţa drumului y→r→x [AH 85].

12.6.2. Algoritmul lui Tarjan

- O altă metodă ingenioasă de determinare a **componentelor puternic conectate** ale unui graf orientat a fost publicată de R.E. Tarjan în anul 1972.
- Metoda se bazează tot pe **tehnica căutării în adâncime** şi o variantă de implementare a sa apare în secvenţa [12.6.2.a] în forma funcţiei **`Tarjan`**.
- În legătură cu această secvenţă se fac câteva precizări.
- (1) Graful se consideră reprezentat printr-o **structură de adiacenţe** implementată cu ajutorul listelor înlănţuite simple.
- (2) Funcţia **`Tarjan`** prezentată, utilizează variabila `min` pentru a determina **cel mai înalt nod** care poate fi atins prin intermediul unui arc de tip "înapoi", pentru orice descendent al nodului k pe care îl primeşte ca şi parametru.

  - Acest lucru este realizat într-o manieră similară funcţiei care determină **componentele biconexe** ale unui graf (secvenţa [10.5.2.3.a], capitolul 10).

```
-------------------------------------------------------------
int Tarjan(int k)

/*Determinarea componentelor puternice ale unui graf*/

   tip_legatura t;
   int m,min;
```

```
    id=id+1; marc[k]=id; min=id; /*marcare nod k*/
    Stiva[p]=k; /*introducere nod k în stiva*/
    p=p+1; /*avans indicator stiva*/
    t=Stradj[k]; /*iniţiere parcurgere lista de adiacenţe*/
    cat timp (t<>NULL)
     daca (marc[t^.nume]==0)
         m=Tarjan(t^.nume);
       altfel
         m=marc[t^.nume];              /*[12.6.2.a]*/
     daca (m<min)
       min=m;
     t=t^.urm;
     □ /*cat timp*/
    daca (min==marc[k]) /*evidenţiere componentă puternică*/
     repeta
      p=p-1;
      *scrie(Stiva[p]);
      marc[Stiva[p]]=N+1;  /*N este numărul de noduri*/
      pana cand (Stiva[p]==k);
      □ /*repeta*/
     *scrie_rand;
     □ /*daca*/
    return min;
/*Tarjan*/
```

- În plus însă, valoarea `min` determinată, este utilizată şi la evidenţierea **componentelor puternic conectate.**
- În acest scop se utilizează **o stivă** implementată ca un tablou `Stiva` controlată de indicele `p`.
  - În această stivă se introduc numele (numerele) nodurilor pentru care este apelată funcţia **`Tarjan`** imediat după intrarea în apel.
  - Nodurile care aparţin unei componente puternice, sunt afişate prin extragere din stivă, după vizitarea ultimului membru al componentei puternic conectate (bucla **`repeta`**).
- **Elementul esenţial** al determinării este verificarea `min=marc[k]` efectuată la terminarea apelurilor recursive:
  - Dacă acest test conduce la valoarea de adevăr, se afişează toate nodurile din stivă până la nodul `k` inclusiv, întrucât ele aparţin aceleeaşi componente puternic conectate ca şi nodul `k`.
- Acest algoritm poate fi însă adaptat să realizeze prelucrări mult mai sofisticate decât simpla afişare a componentelor puternic conectate.
- Fiind bazat pe tehnica căutării în adâncime, în grafuri reprezentate prin structuri de adiacenţe, algoritmul lui Tarjan obţine o **performanţă** de ordinul $O(n+a)$.
- Demonstraţia riguroasă a funcţionalităţii algoritmului este în afara scopului cursului, însă vor fi schiţate ideile care o fundamentează.
- **Metoda lui Tarjan** se bazează pe **două observaţii** care au fost subliniate şi cu alte prilejuri.

- (1) Prima observaţie se referă la faptul că la terminarea apelului funcţiei **`Tarjan`** pentru un nod `x`, **nu** mai pot fi întâlnite alte noduri care să aparţină aceleeaşi componente puternic conectate ca şi nodul respectiv deoarece toate nodurile la care se poate ajunge plecând de la `x` au fost deja parcurse.

- (2) A doua observaţie se referă la faptul că un arc de tip "înapoi" al arborelui de căutare în adâncime precizează un alt drum de la un nod la altul în cadrul arborelui şi el de fapt leagă elementele componentei puternice.

- Exact ca şi în cazul determinării **punctelor de articulaţie** ale unui graf şi în acest caz se păstrează urma **celui mai înalt ascendent**, la care se poate ajunge via un arc de tip "înapoi", pentru toţi descendenţii fiecărui nod.

- **Interpretarea rezultatului** algoritmului lui **`Tarjan`** este următoarea.

    - (1) Un nod `x` care nu are descendenţi sau legături de tip înapoi în arborele de căutare în adâncime, determină o **componentă puternic conectată**.

    - (2) Dacă un nod `x` are **un descendent** în arborele de căutare în adâncime din care porneşte un arc de tip "înapoi" spre `x` şi nu are niciun descendent din care să pornească vreun arc de tip "înapoi" spre vreun nod situat **deasupra** lui `x` în arborele de căutare în adâncime, atunci nodul `x` împreună cu toţi descendenţii săi (cu excepţia acelor noduri care satisfac situaţia (1), respectiv a nodurilor care satisfac situaţia (2) şi a descendenţilor lor), determină o **componentă puternic conectată.**

    - (3) Fiecare descendent `y` al nodului `x` care nu satisface nici una din cele două situaţii mai sus precizate, are descendenţi care sunt sursa unor arce de tip "înapoi" care ajung la noduri mai înalte ca `x` în arborele de căutare în adâncime.

        - Pentru un astfel de nod, există un drum direct de la `x` la `y` rezultat din parcurgerea arborelui de căutare în adâncime.

        - Mai există însă un drum de la nodul `y` la nodul `x` care poate fi determinat coborând în arbore de la `y` în jos până la nodul descendent de la care printr-un arc de tip "înapoi" se ajunge la un strămoş al lui `y` şi apoi continuând în aceeaşi manieră, se ajunge până la `x`.

        - Rezultă că nodul `y` aparţine aceleeaşi **componente puternic conectate** ca şi nodul `x`.

- Aceste situaţii pot fi urmărite în figura 12.6.2.a care reprezintă **pădurea de arbori de căutare în adâncime** (b) corespunzătoare grafului orientat (a).

    - Nodurile `B` şi `K` satisfac prima situaţie, astfel încât se constituie ele însele în componente puternice ale grafului.

    - Nodurile `F` (reprezentând pe `F,E` şi `D`), `H` (reprezentând pe `H` şi `I`) şi `A` (reprezentând pe `A,G,J,L,M` şi `C`) satisfac a doua situaţie.

        - Membrii componentei evidenţiate de `A` au fost determinaţi după eliminarea nodurilor `B,K,F` şi `H` şi a descendenţilor lor care apar în componentele puternice stabilite anterior.

    - Restul nodurilor se încadrează în situaţia a treia.

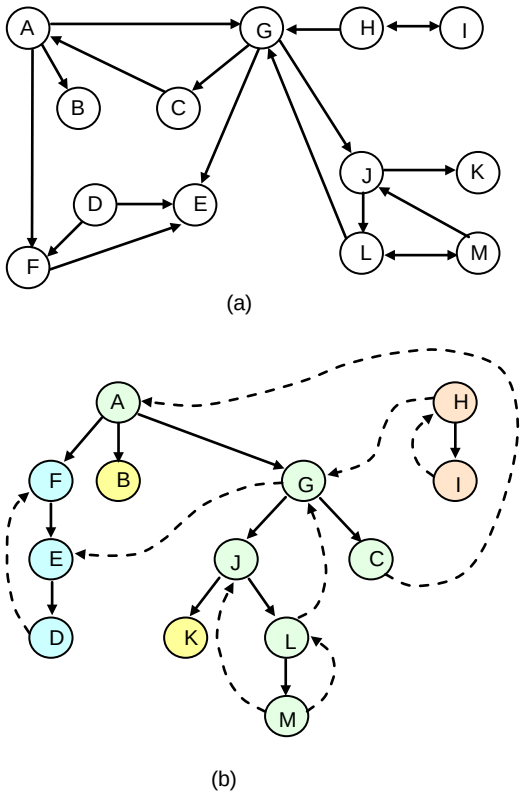


Fig.12.6.2.a. Graf orientat (a) şi pădure de arbori de căutare în adâncime asociată (b)

- O subliniere foarte importantă este aceea ca odată un nod parcurs, el primeşte în tabloul `marc` o valoare mare (`N+1`), astfel încât arcele de "trecere" spre aceste noduri sunt ignorate.

12.7. Reţele de curgere ("Network-Flow")

- Grafurile orientate ponderate sunt modele foarte utile pentru anumite tipuri de aplicaţii inclusiv pentru **modelarea fluxului materialelor** în cadrul unei **reţele interconectate**.

- Se poate considera spre exemplu o **reţea de conducte de petrol** de diferite dimensiuni interconectate într-o manieră complexă, cu comutatoare controlând direcţia fluxului pe joncţiuni.

    - Se presupune în continuare că o astfel de reţea are o singură sursă (un câmp petrolier) şi o singură destinaţie (o mare rafinărie) la care în final sunt conectate toate conductele.

- Întrebarea care se pune este următoarea: "Ce configuraţie de poziţionare a comutatoarelor de control conduce la un **flux maxim** de la sursă la destinaţie?".

- Interacţiunile complexe care se manifestă la nivelul joncţiunilor fac ca **problema reţelelor de curgere** ("**network flow problem**") să **nu** fie una simplu soluţionabilă.

- Astfel de reţele sunt cunoscute şi sub denumirea de **reţele de transport** [FK 69].

- Acelaşi scenariu general, poate fi utilizat la:

- **(1)** Modelarea traficului rutier de-a lungul autostrăzilor unei reţele de drumuri
- **(2)** Modelarea fluxului materialelor într-o intreprindere
- **(3)** Modelarea fluxului datelor în reţelele de comunicaţii.

- Pentru rezolvarea acestei probleme au fost propuse mai multe soluţii, care sunt mai mult sau mai puţin satisfăcătoare funcţie de circumstanţe.
  - De fapt, această problemă este încă intens studiată şi soluţia cea mai bună nu a fost încă determinată.

- În cadrul capitolului de faţă se prezintă o soluţie clasică a problemei reţelelor de curgere, strâns legată de algoritmii referitori la structura de date graf, dezvoltaţi în cadrul capitolelor anterioare.
  - În acelaşi timp acesta este un exemplu de rezolvare a unei probleme utilizând unelte algoritmice deja consacrate.

12.7.1. Problema reţelelor de curgere

- În fig. 12.7.1.a apare o reprezentare idealizată a unei mici reţele de conducte de petrol.

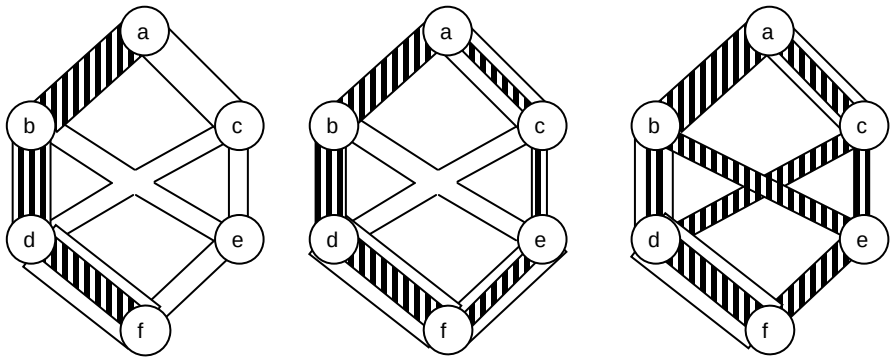


Fig.12.7.1.a. Diverse fluxuri printr-o reţea de curgere simplă

- Se fac următoarele specificaţii:
  - (1) Conductele sunt de **capacitate fixă**, proporţională cu grosimea lor şi pot fi parcurse într-un singur sens (în figură de sus în jos).
  - (2) Comutatoarele distribuitoare situate la fiecare joncţiune dirijează **cantitatea de petrol** care se scurge pe fiecare dintre direcţiile posibile.
  - (3) Indiferent de poziţionarea comutatoarelor, sistemul se găseşte într-o **stare de echilibru** atunci când:
    - **(3.1)** Cantitatea de petrol care intră în sistem la partea sa superioara,

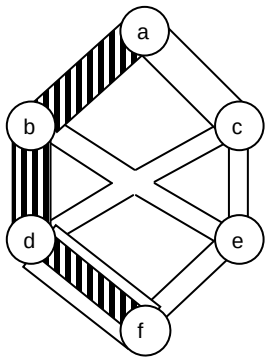
este egală cu cantitatea de petrol care părăseşte sistemul la partea sa inferioară. Aceasta este cantitatea pentru care trebuie determinată **valoarea maximă.**

- **(3.2)** Cantitatea de petrol care intră în fiecare joncţiune este egală cu cantitatea care iese din joncţiune.

- (4) Atât fluxul curent prin conductă, cât şi capacitatea sa maximă vor fi exprimate în valori întregi, spre exemplu litri pe secundă.

- La prima vedere **nu** pare evident faptul că poziţionarea comutatoarelor afectează fluxul maxim.
- În figura 12.7.1.a se demonstrează că acest lucru este posibil.

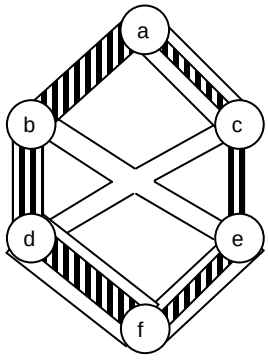
  - În situaţia (a), comutatoarele care controlează conductele `ab` şi `bd` sunt deschise integral, astfel încât acestea funcţionează la capacitatea maximă iar conducta `df` numai parţial.



  - În situaţia (b) se deschid în plus conducta `ac` (parţial), `ce` (total) şi `ef` (parţial).

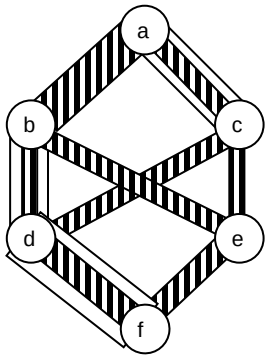
    - Fluxul prin reţea este mai mare acum întrucât atât conducta `bd` cât şi conducta `ce` funcţionează la capacitatea maximă.

    - Acest flux ar putea fi mărit deschizând traseul `acdf` astfel încât conducta `df` să funcţioneze şi ea la capacitatea maximă.



  - După cum se vede însă din situaţia (c) există o soluţie şi mai bună.

- Modificând comutatorul b şi permiţând în acest mod curgerea petrolului prin conducta be la capacitatea maximă a acesteia, se crează posibilitatea utilizării la capacitate maximă şi a conductei cd.



- Ca atare, după cum se observă, poziţionarea comutatoarelor distribuitoare influenţează în mod evident fluxul total prin reţea.

- **Problema reţelelor de curgere** constă în a determina acea poziţionare a comutatoarelor pentru orice reţea de acest tip, care conduce la **fluxul maxim**.

    - De asemenea trebuie dovedit faptul că nici o altă poziţionare **nu** poate obţine un flux mai mare.

- Această situaţie poate fi în mod evident modelată printr-un **graf orientat**.

- Astfel, o **reţea de curgere** se defineşte ca şi un **graf orientat ponderat** a cărui mulţime a nodurilor conţine în afara celor normale, două noduri speciale:

    - (1) Un nod sursă în care **nu** intră nici un arc

    - (2) Un nod destinaţie din care **nu** pleacă nici un arc.

- Ponderile arcelor se numesc **capacităţi** şi se presupun pozitive.

- **Fluxurile** (debitele) sunt definite printr-un alt set de ponderi asociate fiecărui arc.

    - Aceste ponderi sunt supuse la două restricţii:

        - **(1)** Fluxul într-un arc este mai mic sau cel mult egal cu capacitatea arcului respectiv

        - **(2)** Fluxul la intrare în oricare nod este egal cu fluxul de la ieşirea nodului respectiv.

    - Valoarea **fluxului reţelei** este fluxul din nodul sursă sau nodul destinaţie al reţelei.

- **Problema reţelelor de curgere** constă în determinarea **valorii maxime** a fluxului reţelei.

- Fiind modelate prin grafuri, reţelele pot fi reprezentate fie prin intermediul matricilor de adiacenţe fie prin intermediul structurilor de adiacenţe.

- În plus însă, în locul unei singure ponderi fiecărui arc îi sunt asociate **două ponderi** respectiv **capacitatea** şi **fluxul**.

- Această cerinţă poate fi rezolvată prin definirea a **două câmpuri specifice** în structura nodului unei liste de adiacenţe, sau prin utilizare a **două matrici** în reprezentarea bazată pe matrici de adiacenţe.

- De asemenea în ambele reprezentări poate fi utilizată ca şi alternativă o structură de tip articol cu două câmpuri corespunzătoare celor două ponderi.

- Chiar dacă reţelele sunt de fapt grafuri orientate, algoritmul care rezolvă problema reţelelor de curgere necesită traversarea grafului şi în sens invers sensului arcelor, motiv pentru care în reprezentare se va utiliza un **graf neorientat**.

  - Astfel, pentru fiecare arc de la `x` la `y` aparţinând reţelei, cu capacitatea `c` şi fluxul `f`, se va păstra şi imaginea arcului de la `y` la `x` având asociate ponderile `-c` şi `-f`.

  - În acest context, în reprezentarea bazată pe **structuri de adiacenţe**, este necesar să fie prevăzute legături care conectează nodurile aparţinând celor două liste de adiacenţe care reprezintă acelaşi arc, astfel încât modificarea fluxului într-una dintre reprezentări să poată fi imediat actualizată şi în cealaltă.

12.7.2 Metoda Ford-Fulkerson

- Metoda clasică de rezolvare a problemei reţelelor de curgere a fost dezvoltată de către L.R.Ford şi D.R.Fulkerson în anul l962.

  - Ei au furnizat o metodă care îmbunătăţeşte orice flux legal prin reţea, evident cu excepţia celui maxim.

- Practic se porneşte de la **valoarea zero a fluxului** şi se aplică metoda în mod repetat.

  - Atâta vreme cât metoda poate fi aplicată, ea conduce la obţinerea unor fluxuri crescătoare, iar când nu mai poate fi aplicată s-a obţinut fluxul maxim.

- De fapt fluxul maxim din figura 12.7.1.a a fost obţinut aplicând această metodă.

- În continuare se reexaminează aceeaşi situaţie în termenii grafurilor reprezentate în figura 12.7.2.a.

  - Pentru fiecare arc al grafului se indică într-o **elipsă** ponderile asociate în formatul **flux/capacitate** (`f/c`).

  - Pentru simplificare, săgeţile arcelor au fost omise, considerându-se că toate indică sensul “în jos”.

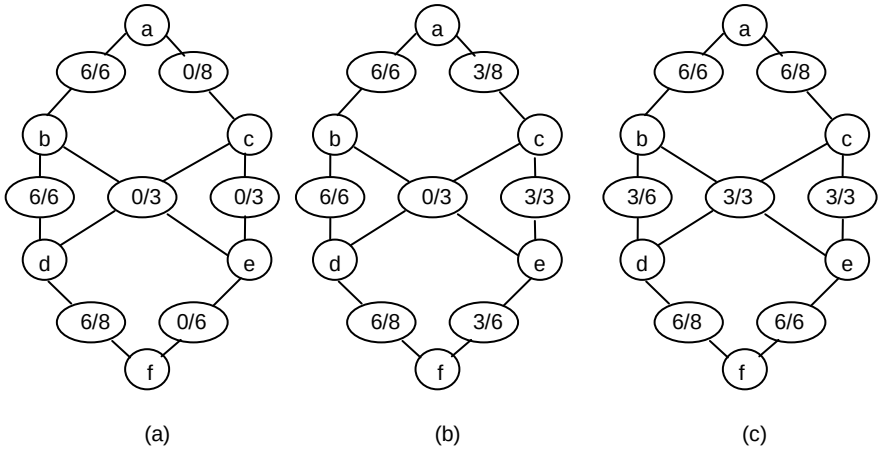


Fig.12.7.2.a. Determinarea fluxului maxim într-o reţea de curgere

- Se face însă precizarea că aplicabilitatea metodei care va fi dezvoltată în continuare **nu** este restrânsă numai la grafuri în care toate săgeţile indică aceeaşi direcţie.

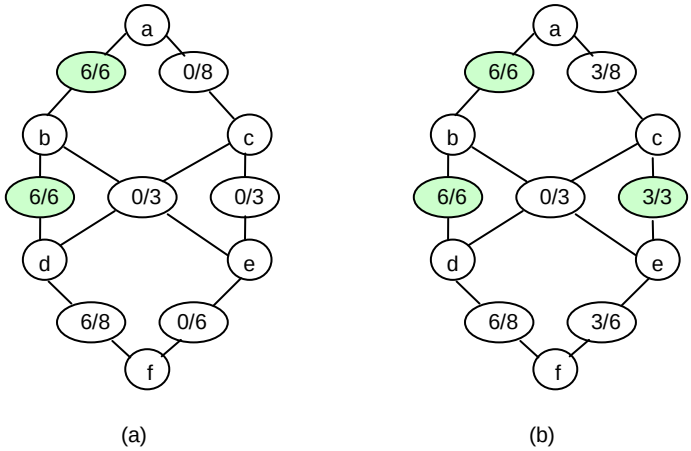
    - Pentru prezentarea metodei însă, s-a adoptat o astfel de soluţie deoarece aceasta este foarte intuitivă.

    - Cu această simplificare, practic fluxul într-o reţea de curgere poate fi exprimat natural în termenii unui lichid care străbate conductele reţelei.

- Se consideră **orice drum orientat** "în jos" prin reţea de la sursă la destinaţie.

    - Este evident faptul că pe orice drum fluxul poate fi crescut până la **capacitatea maximă a celei mai înguste conducte de pe traseu**, ajustând în mod corespunzător fluxul în toate arcele care compun drumul.

        - În figura 12.7.2.a a fost aplicată această regulă de-a lungul drumului `abdf` (situaţia (a)), respectiv şi de-a lungul drumului `acef` (situaţia (b)).



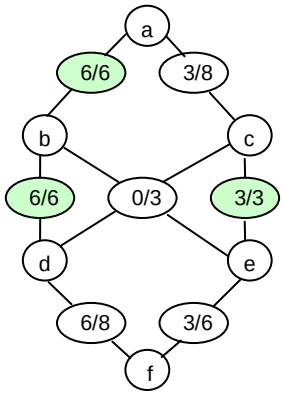
    - În continuare, regula poate fi aplicată şi altor drumuri, creând spre exemplu situaţia în care toate drumurile orientate din reţea au cel puţin un arc care funcţionează la capacitatea maximă.

- Există însă şi **o altă metodă** de creştere a fluxului.

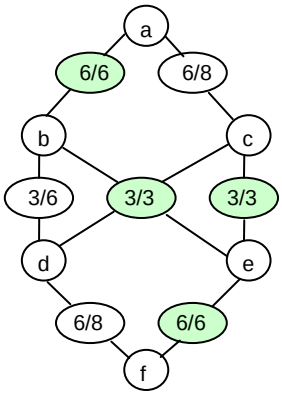
- Se consideră **drumuri arbitrare** prin reţea care pot conţine şi arce orientate în sens invers (respectiv de la destinaţie spre sursă).

- Fluxul poate fi crescut de-a lungul unui astfel de drum, crescând fluxul în arcele care au sensul de la sursă spre destinaţie şi diminuând cu aceeaşi valoare, fluxul în arcele direcţionate în sens invers.

- Spre exemplu, după cum se observă în figura 12.7.2.a (c), fluxul prin reţea poate fi crescut cu `3` unităţi de-a lungul drumului `acdbef` faţă de situaţia (b).



(b)



(c)

- Aceasta rezultă din suplimentarea fluxului prin `ac` şi `cd` cu câte `3` unităţi la fiecare şi prin reducerea fluxului cu `3` unităţi prin `bd` cu redirectarea acestuia pe traseul `be` şi `ef`.

- De fapt nu se pierde din fluxul de pe ramura `df` întrucât cele 3 unităţi care vedeau dinspre `bd` vin acum pe ramura `cd.`

- Pentru a simplifica terminologia:

  - Arcele direcţionate de la **sursă** spre **destinaţie** se numesc arce de tip "**înainte**".

  - Arcele orientate de la **destinaţie** spre **sursă** se numesc arce de tip "**înapoi**".

- Este important de subliniat faptul că valoarea cu care poate fi crescut fluxul este limitată de

  - (1) **Minimul** dintre capacităţile neutilizate ale arcelor de tip "înainte"

  - (2) **Minimul** fluxurilor în arcele de tip "înapoi" de pe traseu.

  - Cu alte cuvinte, în noul flux, cel puţin unul din arcele de tip înainte situate pe traseu funcţionează la capacitatea maximă sau cel puţin unul dintre arcele de tip "înapoi" situat de-a lungul traseului devine gol.

  - Ca atare fluxul **nu** mai poate fi crescut de-a lungul unui traseu care conţine un arc "înainte" plin sau un arc "înapoi" gol.

- Observaţiile formulate mai sus sugerează **principiul metodei** de creştere a fluxului într-o reţea de curgere.

    - Acesta se referă la determinarea acelor drumuri care conţin arce "înainte" care **nu** sunt pline sau arce "înapoi" care **nu** sunt goale.

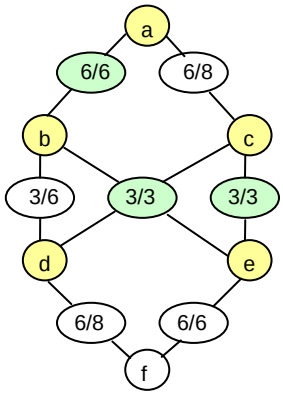
- **Elementul fundamental** al **metodei Ford-Fulkerson** constă în observaţia că dacă **fluxul în reţea este maxim**, în reţea **nu** există nici un astfel de drum.

    - Cu alte cuvinte, dacă **orice drum** de la sursă la destinaţie dintr-o reţea, conţine un arc "înainte" plin sau un arc "înapoi" gol, atunci **fluxul în reţea este maxim**.

- Pentru a demonstra acest fapt, în primul rând, se inspectează graful şi se determină **primul arc** "înainte" plin sau **primul arc** "înapoi" gol pe fiecare drum.

    - Mulţimea arcelor astfel determinate **taie** graful în două părţi.

        - Spre exemplu din figura 12.7.2.a (c) arcele `ab,cd` şi `ce` realizează această tăietură.



(c)

- Pentru orice tăietură a reţelei se poate determina fluxul prin tăietură.

    - Acesta este de fapt **fluxul total** prin arcele tăieturii care merg de la sursă spre destinaţie.

    - În general, arcele pot traversa în ambele sensuri tăietura.

    - Pentru determinarea efectivă a fluxului se adună fluxurile care trec spre destinaţie şi se scad fluxurile din arcele care traversează în sens invers tăietura.

        - În cazul prezentat, fluxul prin tăietură are valoarea `12` şi este egal cu fluxul prin reţea.

- Rezultă că, ori de câte ori fluxul prin tăietură egalează fluxul total, pe de o parte fluxul este **maxim** iar pe de altă parte tăietura este **minimă**, ceea ce înseamnă că orice altă tăietură are fluxul cel puţin la fel de mare.

    - Aceasta este teorema "**flux maxim – tăietură minimă**" numită şi **teorema lui Ford-Fulkerson.**

- Obs: Prin **tăietură minimă** se înţelege capacitate neutilizată minimă.

- Drept urmare:

  - **(1)** Fluxul nu poate fi oricât de mare - altfel tăietura ar putea fi şi ea mai mică.

  - **(2)** Nu există tăieturi mai mici - altfel şi fluxul ar putea de asemenea să aibă valori mai mari [Se 88].

- **Teorema Ford-Fulkerson** se mai poate enunţa şi în următoarea formă:

  - Într-o reţea de curgere dată, **valoarea maximă** a unui **flux** este egală cu **capacitatea minimă** a unei **tăieturi** [FK 69].

12.7.3. Implementarea metodei Ford-Fulkerson. Căutarea în reţea

- Metoda Ford-Fulkerson descrisă anterior poate fi rezumată după cum urmează:

  - (1) Se porneşte cu flux zero peste tot şi se creşte fluxul de-a lungul oricărui drum de la sursă la destinaţie care **nu** conţine **arce înainte pline** sau **arce înapoi goale**.

  - (2) Operaţiunea continuă până când **nu** mai există nici un astfel de drum în reţea.

- După cum se observă însă acesta **nu** este un algoritm în sensul strict al cuvântului deoarece metoda de determinare a drumurilor **nu** este specificată, practic putând fi selectată orice succesiune de drumuri.

  - Spre exemplu la baza selecţiei ar putea sta observaţia că cu cât drumul este mai lung, cu atât reţeaua va fi mai repede umplută, astfel încât sunt de preferat drumurile cele mai lungi.

  - Indiferent însă de strategia abordată, maniera de selectare a drumurilor are o influenţă decisivă asupra performanţei metodei.

- În anul 1972 **Edmonds şi Karp** au demonstrat următoarea **proprietate**.

  - Dacă în metoda Ford-Fulkerson se utilizează **cel mai scurt drum curent disponibil** de la sursă la destinaţie, atunci numărul de paşi necesari determinării fluxului maxim într-o reţea cu n noduri şi a arce este mai mic decât produsul na.

    - Demonstraţia acestei proprietăţi depăşeşte scopul cursului de faţă.

- În consecinţă, o posibilitate de a aborda rezolvarea acestei probleme o reprezintă utilizarea **tehnicii de căutare "prin cuprindere"** pentru găsirea **drumurilor minime**.

- Proprietatea anterior enunţată precizează o margine superioară pentru performanţa acestui algoritm.

  - În realitate, pentru reţelele obişnuite sunt necesari mult mai puţini paşi.

- Pornind însă de la traversarea grafurilor prin **tehnica căutării bazate pe prioritate**

abordată în capitolul anterior se poate implementa şi o altă metodă sugerată tot de Edmonds şi Karp şi anume:

- Găsirea acelui drum din reţea care determină creşterea fluxului cu cea mai mare valoare posibilă.

- Aceast obiectiv poate fi atins simplu utilizând pentru **prioritate**, în cadrul metodei dezvoltate în secvenţa [11.4.a] din capitolul &11, o valoare corespunzătoare.

```
------------------------------------------------------------
PROCEDURE CautPrioritar;
VAR k,min,t: integer;

  BEGIN
    FOR k:= 1 TO N DO
      BEGIN
        marc[k]:= -nevazut;                          [11.4.a]
        parinte[k]:= 0
      END;
    marc[0]:= -(nevazut + 1);
    min:= 1;
    REPEAT
      k:= min; marc[k]:= -marc[k]; min:= 0;
      IF marc[k] = nevazut THEN marc[k]:= 0;
      FOR t:= 1 TO N DO
        IF marc[t] < 0 THEN
          BEGIN
            IF(A[k,t] <> 0) AND (marc[t] < -prioritate) THEN
              BEGIN
                marc[t]:= -prioritate;
                parinte[t]:= k
              END;
            IF marc[t] > marc[min] THEN min:= t
          END
    UNTIL min = 0
  END; {CautPrioritar}
------------------------------------------------------------
```

- Spre exemplu, pentru grafurile reprezentate prin matrice de adiacenţe, **determinarea valorii priorităţii** se face pe baza următorului calcul [12.7.3.a]:

```
------------------------------------------------------------
IF capacitate[k,t] > 0 THEN
    prioritate:= capacitate[k,t] - flux[k,t];
  ELSE                                           [12.7.3.a]
    prioritate:= -flux[k,t];
IF prioritate > marc[k] THEN prioritate:= marc[k];
------------------------------------------------------------
```

- În continuare, pentru a adapta procedura `CautPrioritar` din secvenţa [11.4.a] noului scop, trebuiesc operate unele modificări.

  - **(1)** Deoarece se doreşte selecţia nodului cu cea mai mare prioritate, este necesară:

    - (a) Fie modificarea mecanismului cozii bazate pe prioritate astfel încât

acesta să returneze maximul în locul minimului

- (b) Fie utilizarea pentru prioritate a expresiei `maxint-1-prioritate` (`maxint` fiind întregul maxim reprezentabil).

- **(2)** Este necesar a fi modificată procedura de căutare bazată pe prioritate astfel încât să primească drept date de intrare nodul sursă şi nodul destinaţie.

  - În acest fel orice căutare porneşte de la nodul sursă şi se termină când se determină un drum până la nodul destinaţie.

  - Dacă **nu** se găseşte un astfel de drum, arborele de căutare parţial care a fost determinat defineşte o **tăietură minimă** pentru reţea.

  - Dacă se găseşte un astfel de drum, atunci fluxul prin reţea poate fi crescut în continuare.

- **(3)** Valoarea iniţială din tabloul `marc` corespunzător nodului sursă trebuie poziţionată pe `maxint` pentru a preciza faptul că practic de la sursă se poate obţine orice flux.

  - În realitate această valoare este imediat restrânsă la capacitatea maximă a tuturor conductelor care părăsesc sursa.

- Implementând în procedura `CautPrioritar` (secvenţa [11.4.a]), modificările descrise anterior, determinarea fluxului maxim poate fi realizată conform secvenţei [12.7.3.b].

```
-------------------------------------------------------------
REPEAT
  CautPrioritar(1,N);
  y:= N; x:= parinte[N];
  WHILE x <> 0 DO
    BEGIN                                          [12.7.3.b]
      flux[x,y]:= flux[x,y] + marc[N];
      flux[y,x]:= -flux[x,y];
      y:= x; x:= parinte[y]
    END
UNTIL marc[N] = 1 - maxint;
-------------------------------------------------------------
```

- În cadrul secvenţei se presupune că reţeaua este reprezentată printr-o **matrice de adiacenţe**.

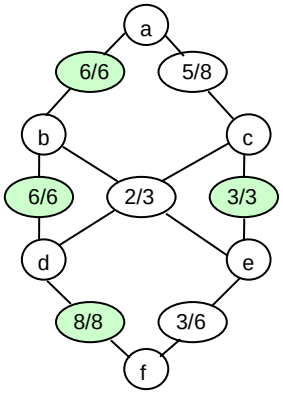
  - Atâta vreme cât `CautPrioritar` poate determina un drum care creşte fluxul cu cantitatea maximă posibilă, se revine de-a lungul drumului utilizând tabloul `părinte` construit de procedura `CautPrioritar` şi se creşte fluxul după cum rezultă din secvenţa [12.7.3.b], bucla **`WHILE`**.

  - Dacă `N` (nodul destinaţie) nu poate fi atins după câteva apeluri ale procedurii `CautPrioritar`, atunci s-a determinat o tăietură minimă şi algoritmul se termină.

- După cum s-a observat din fig.12.7.2.a, algoritmul creşte mai întâi fluxul de-a lungul

drumului `abdf`, apoi de-a lungul drumului `acef` şi în cele din urmă de-a lungul lui `acdbef`.

- De notat faptul că traseul `acdf` nu este ales pentru al treilea pas deoarece el ar creşte fluxul cu două unităţi şi nu cu trei unităţi, după cum se realizează în pasul selectat.



- Deşi acest algoritm este simplu de implementat şi relativ uşor de aplicat în practică pentru reţelele utilizate în mod curent, analiza sa este complicată.

  - În primul rând, `CautPrioritar` necesită un efort proporţional cu $O(n^2)$ în cel mai defavorabil caz.

  - Ca şi alternativă poate fi utilizat algoritmul `ArboreMinim` (secvenţa [11.2.2.a]), dezvoltat pentru reprezentarea grafurilor ca şi structuri de adiacenţe, care necesită un timp proporţional cu $O((a+n) \log n)$ pentru fiecare iteraţie.

- Oricum este de aşteptat că algoritmul din secvenţa [12.7.3.b] să dureze mai puţin decât procedura `ArboreMinim` (secvenţa [11.2.2.a]) deoarece spre deosebire de cea din urmă care determină toate drumurile care aparţin arborelui de acoperire minim, algoritmul de faţă se opreşte la determinarea drumului care conduce la destinaţie.

- Problema care se pune în continuare se referă la numărul necesar de iteraţii.

- O margine superioară pentru acest număr este furnizată de următoarea **proprietate** anunţată tot de Edmonds şi Karp.

  - Dacă în metoda Ford-Fulkerson se foloseşte drumul care creşte fluxul cu cantitatea maximă posibilă, atunci numărul de paşi necesari determinării fluxului maxim într-o reţea este mai mic decât $1 - \log_{M/M-1} f^*$ unde f^* reprezintă costul fluxului şi M numărul maxim de arce dintr-o tăietură a reţelei.

- Această proprietate a fost enunţată **nu** atât pentru precizarea limitei numărului de iteraţii cât mai ales pentru a evidenţia complexitatea analizei.

- De fapt această problemă a fost intens studiată şi au fost dezvoltaţi algoritmi complecşi care obţin performanţe mult mai bune decât algoritmul prezentat.

- Cu toate acestea algoritmul lui Edmonds şi Karp este foarte greu de depăşit în performanţă pentru reţelele care apar în mod obişnuit în practică.

- Problema reţelelor de curgere (de transport) poate fi extinsă în diferite moduri şi multe din variantele sale au fost studiate în detaliu datorită importanţei pe care le prezintă pentru aplicaţiile curente.

    - Spre exemplu **problema fluxului de mărfuri multiple** ("**multicommodity flux problem**") presupune introducerea în reţea a mai multor destinaţii şi a mai multor tipuri de materiale care parcurg reţeaua.

- Aceste generalizări fac ca problema fluxului maxim să devină mult mai dificilă şi rezolvarea ei presupune algoritmi mult mai complicaţi.

    - Spre exemplu, până la ora actuală **nu** se cunoaşte pentru acest caz o teoremă analoagă teoremei flux maxim – tăietură minimă.

- Alte extensii ale problemei reţelelor de curgere se bazează pe următoarele abordări:

    - Luarea în considerare a restricţiilor de capacitate din noduri - uşor de rezolvat prin introducerea unor arce artificiale care tratează aceste capacităţi

    - Utilizarea conductelor cu dublu sens - uşor de rezolvat prin înlocuirea arcului neorientat cu o pereche de arce orientate

    - Introducerea unei limite inferioare a fluxului în arce - dificil de rezolvat

- Dacă se ia în considerare faptul real că conductele presupun în afară de capacităţi de transport şi costuri atunci problema **costului minim** al unei reţele pentru un flux precizat, devine o problemă complicată de cercetare.

12.8. Problema potrivirilor ("Matching")

- O problemă care apare destul de frecvent în practică este problema **perechilor stabile**.

- Ea se referă la asociarea în perechi a unor elemente în conformitate cu preferinţele acestora, evitând conflictele.

    - Spre exemplu, în S.U.A. a fost implementat un sistem destul de complicat de plasare a studenţilor din anii superiori de la universităţile de medicină în spitale pentru efectuarea stagiului.

        - Fiecare student prezintă o listă cu spitale ordonate în ordinea preferinţelor şi fiecare spital prezintă o listă cu studenţi de asemenea ordonaţi în ordinea preferinţelor.

        - Problema complicată care trebuie rezolvată este aceea de a asocia fiecărui student un loc într-un spital, într-o manieră corectă respectând toate preferinţele formulate.

- Algoritmul care realizează această asociere este foarte sofisticat deoarece este de aşteptat ca cei mai buni studenţi să fie solicitaţi de mai multe spitale, iar spitalele cele mai bune să fie solicitate de mai mulţi studenţi.

- Este însă evident faptul că nu în toate situaţiile preferinţele reciproce pot fi satisfăcute prin asocieri corespunzătoare şi în mod efectiv deşi algoritmul găseşte soluţia optimă în condiţiile date, în multe situaţii apar nemulţumiri.

- **Problema perechilor (relaţiilor) stabile** este un caz special al **problemei potrivirilor**, care la rândul ei este o problemă fundamentală a **teoriei grafurilor**, problemă care a fost foarte intens studiată.

- Fiind dat un graf, o **potrivire** este **o submulţime a arcelor grafului**, selectată astfel astfel încât nici unul din nodurile conectate de arcele aparţinând acestei submulţimi **nu** pare mai mult decât odată.

    - Aceasta înseamnă că nodurile conectate de aceste arce sunt asociate în **perechi**.

    - **Nu** este necesar să fie implicate toate nodurile.

    - Chiar dacă se încearcă ca potrivirea să acopere cât mai multe noduri posibile, diferitele posibilităţi de asociere conduc de regulă la numere diferite de noduri neutilizate.

- De un interes particular se bucură **potrivirea maximă** care conţine numărul maxim posibil de arce respectiv numărul minim de noduri neutilizate.

    - Cazul ideal este acela al grafului cu $2n$ noduri şi n arce care conectează noduri care apar o singură dată.

- O **potrivire completă** este potrivirea în care fiecare nod al grafului este conectat prin arce aparţinând potrivirii.

    - Este evident că orice potrivire completă este în acelaşi timp şi maximă [AH 85].

- În figura 12.8.a se prezintă o potrivire maximă pentru graful reprezentat (mulţimea arcelor îngroşate).

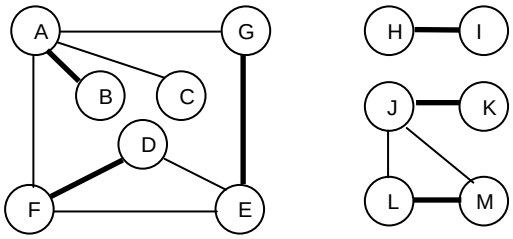


Fig.12.8.a. Potrivire maximă pentru un graf

- Determinarea unei astfel de potriviri este însă o problemă dificilă, chiar în cazul grafului simplu din figură.

    - Spre exemplu, o metodă de determinare a potrivirilor ar putea consta în selecţia arcelor eligibile în ordinea în care acestea apar la traversarea grafului prin **tehnica căutării în adâncime**.

    - În cazul grafului din figură această metodă conduce la selecţia arcelor `AF, EG, HI, JK` şi `LM` care reprezintă o potrivire dar **nu** cea maximă.

- Problema repartizării studenţilor la spitale pentru satisfacerea stagiului, descrisă anterior, poate fi modelată cu ajutorul unui graf în care studenţii şi spitalele corespund nodurilor grafului iar arcele reprezintă preferinţele.

- Există posibilitatea de a atribui **ponderi** preferinţelor (de exemplu valori cuprinse între 1 şi 10), situaţie care prefigurează **problema potrivirilor ponderate**.

- **Problema potrivirilor ponderate** se poate formula astfel.

    - Dându-se un graf ponderat, problema potrivirilor ponderate constă în determinarea unei mulţimi de arce, astfel încât nici un nod nu apare mai mult decât o singură dată şi suma ponderilor arcelor este maximă [SE 88].

    - Există şi o altă variantă a acestei probleme în care se ţine cont de **ordinea** în care sunt formulate preferinţele, fără a mai fi necesară acordarea de ponderi.

- Problema potrivirilor suscită un interes sporit din partea matematicienilor şi informaticienilor din cauza naturii sale intuitive şi a largii aplicabilităţi.

- Soluţionarea ei în cazul general care presupune utilizarea într-o manieră complexă a matematicii combinatoriale depăşeşte cadrul prezentului volum.

- O variantă cunoscută a problemei potrivirilor ponderate o **reprezintă problema relaţiilor stabile** (“**stable marriage problem**”), formulată de regulă în cadrul extrem de delicat al selecţiei partenerilor de viaţă.

    - În volumul 1 al cărţii de faţă, &5.4.5 s-a prezintat o modalitate de rezolvare a unei variante a acestei probleme, rezolvare care în contextul unei abordări recursive bazată pe tehnica backtracking, determină toate soluţiile posibile ale problemei.

    - Inconvenientele sunt legate de eficienţa relativ scăzută datorată utilizării recursivităţii şi contextul rigid al formulării problemei.

- În cadrul subcapitolului de faţă, vor fi dezvoltate alte două metode eficiente de determinare a **potrivirii maxime** în contextul simplificat al **grafurilor bipartite**.

12.8.1. Grafuri bipartite

- Aşa cum s-a precizat, problema potrivirilor este o problemă reprezentativă pentru foarte multe situaţii care apar în practică.

- Alte situaţii în afara celor deja prezentate, specifice acestei probleme, sunt:

    - Repartizarea şomerilor pe posturile disponibile,

- Alcătuirea orarului prin repartizarea cursurilor poziţiilor orarului,

- Repartizarea membrilor parlamentului pe comisii, etc.

- Grafurile care modelează astfel de cazuri şi numesc **grafuri bipartite**.

- După cum s-a mai precizat un **graf bipartit** este acel graf ale cărui noduri pot fi divizate în două mulţimi disjuncte şi ale cărui arce conectează exclusiv noduri aparţinând unul, uneia dintre mulţimi iar celălalt celei de-a doua mulţimi (Capitolul 10, &10.1).

    - În figura 12.8.1.a apare un exemplu de graf bipartit.

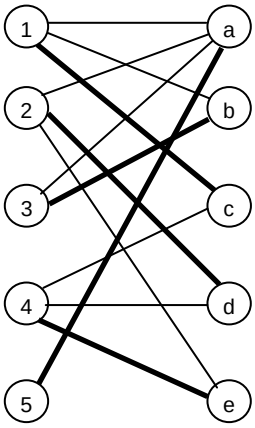


Fig.12.8.1.a. Graf bipartit şi o potrivire maximă a sa

- În ceea ce priveşte reprezentarea grafurilor bipartite rămân valabile modalităţile de reprezentare clasice definite în cazul grafurilor normale, adică matricile de adiacenţe respectiv structurile de adiacenţe.

- Revenind la exemplul din figura 12.8.1.a se observă că s-au notat cu litere mici elementele unei mulţimi de noduri şi cu cifre elementele celeilalte.

- În acest context, **problema potrivirii maxime** pentru grafurile bipartite poate fi formulată astfel.

    - Se cere să se determine cea mai mare submulţime a mulţimii perechilor literă-cifră care se bucură de proprietatea că oricare două perechi ale submulţimii **nu** conţin aceeaşi literă sau aceeaşi cifră.

- Rezolvarea eficientă a problemei potrivirii maxime este o sarcină dificilă.

- O metodă directă de rezolvare poate fi următoarea: se generează toate potrivirile posibile şi se selectează cea care conţine cel mai mare număr de arce.

    - Dezavantajul acestei metode este acela că timpul ei de execuţie este o funcţie exponenţială în raport cu numărul de arce.

- În continuare se prezintă două metode mai eficiente care rezolvă această problemă.

12.8.2. Determinarea potrivirii maxime prin tehnica drumurilor augmentate

- Unul din algoritmii eficienţi de determinare a potrivirii maxime utilizează tehnica **drumurilor augmentate** (“**augmenting path**”).

- Fie P o potrivire în graful G.

- Un nod x este “**potrivit**”, adică constituit într-o pereche, dacă este extremitatea unui arc aparţinând lui P.

- Un drum care conectează două noduri care nu au fost încă “potrivite” şi în care arcele alternative aparţin lui P, se numeşte **drum augmentat** relativ la P.

- Se observă că un **drum augmentat** trebuie să fie de lungime impară şi trebuie să înceapă şi să se termine cu arce care nu aparţin lui P.

- Se subliniază de asemenea faptul că fiind dat un drum argmentat D se poate găsi întotodeauna o potrivire mai mare în graful G, extrăgând din P acele arce care sunt în D şi apoi adăugând lui P, arcele din D care iniţial nu au fost în P.

- Această nouă potrivire este P ¤ D, unde semnul ¤ reprezintă operatorul “sau exclusiv” aplicat mulţimilor.

- De fapt noua potrivire conţine acele arce din care sunt ori în P ori în D dar nu în amândouă.

- În figura 12.8.2.a (a) apare **un graf bipartit** şi o **potrivire** P constând din arcele îngroşate (1,a),(3,b) şi (4,c).

    - Drumul 2,a,1,c,4,d din fig. 12.8.2.a (b) este un **drum augmentat** relativ la P.

    - În aceeaşi figură desenul (c) materializează potrivirea mai mare (1,c),(2,a),(3,b),(4,d) obţinută prin extragerea din P a arcelor ce aparţin drumului augmentat(a,1),(c,4) şi apoi adăugând lui P celelalte arce ale drumului (2,a),(1,c) şi (4,d).

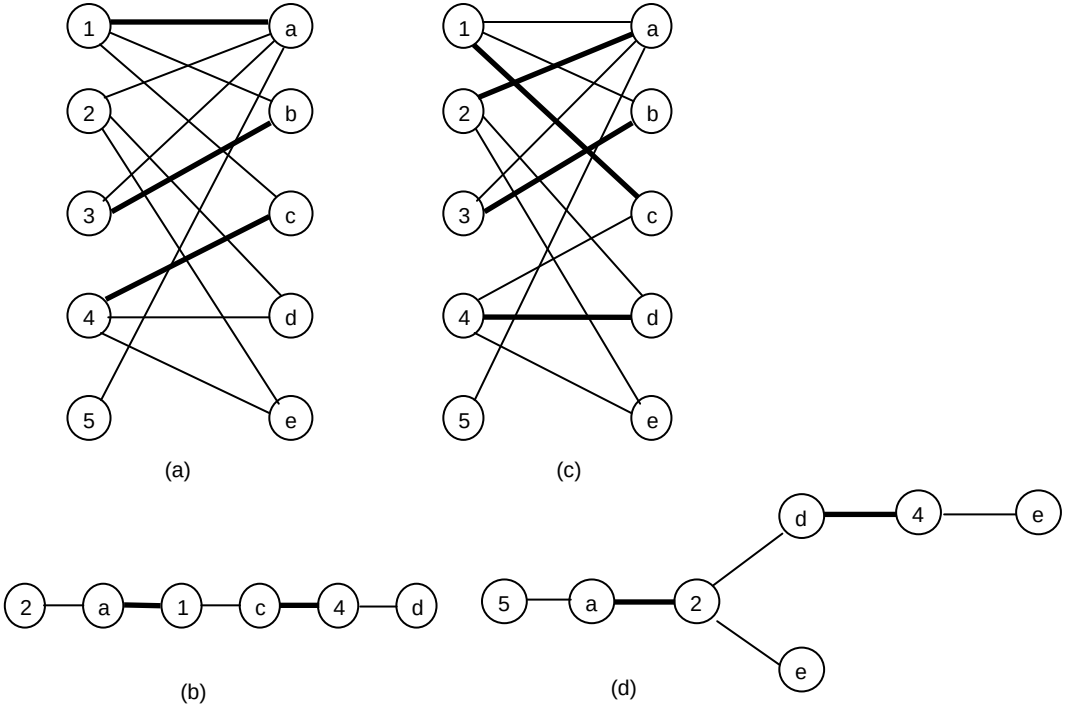


Fig.12.8.2.a. Potriviri şi drumuri augmentate într-un graf bipartit

- O **observaţie extrem de importantă** este următoarea: P este **potrivirea maximă**, dacă şi numai dacă, **nu** mai există nici un drum augmentat relativ la P.

  - Se presupune că P şi Q sunt două potriviri ale lui G

  - Se presupune de asemenea că |P|<|Q| (|P|precizează numărul de arce din P).

  - Pentru a vedea dacă P ¤ Q conţine un drum augmentat relativ la P, se consideră graful G'=(N,P ¤ Q).

  - Deoarece atât P cât şi Q sunt ambele potriviri în G, fiecare nod aparţinând lui N este o extremitate pentru cel mult un arc al lui P şi o extremitate pentru cel mult un arc al lui Q.

  - Astfel, fiecare **componentă conexă** a lui G' formează un drum simplu (posibil un ciclu) care conţine arce **alternând** între P şi Q.

  - Fiecare drum care nu este un ciclu, este fie un drum augmentat relativ la P fie un drum augmentat relativ la Q, după cum conţine mai multe arce din P respectiv din Q.

  - Fiecare ciclu are un număr egal de arce din P şi Q.

  - Deoarece |P|<|Q|, P ¤ Q are mai multe arce din Q decât din P şi astfel conţine cel puţin un **drum augmentat** relativ la P [AH 85].

- În continuare, se poate formula algoritmul care determină potrivirea maximă P pentru un

graf G=(N,A).

- **(1)** Se poneşte cu P=Ø;
- **(2)** Se determină un drum augmentat D relativ la P şi se înlocuieşte P cu P ¤ D;
- **(3)** Se repetă pasul (2) până când nu mai există nici un drum augmentat.
- **(4)** În acest moment P reprezintă potrivirea maximă.

- Mai rămâne de precizat modul în care poate fi determinat un **drum augmentat** relativ la o potrivire P.
- Acest lucru se va realiza în contextul unui **graf bipartit** G, ale cărui noduri sunt partiţionate în două mulţimi N_1 şi N_2.
- În continuare, utilizând o procedură similară "căutării prin cuprindere", se construieşte un graf al drumului augmentat pentru G relativ la potrivirea P, parcurgând succesiv nivelurile i=0,1,2,…,k.
- Nivelul 0 constă din toate nodurile "nepotrivite" ale lui N_1.
  - La **nivelul impar** i se adaugă noile noduri care sunt adiacente nodurilor nivelului i−1 prin arce care **nu aparţin potrivirii** şi care vor fi adăugate lui P.
  - La **nivelul par** i se adaugă noile noduri care sunt adiacente nivelului i-1 prin arce care **aparţin potrivirii** P.
- Se continuă construcţia grafului drumului augmentat nivel cu nivel până când un nod "nepotrivit" este adăugat unui nivel impar sau până când nu mai poate fi adăugat nici un nod.
  - Dacă există un drum augmentat relativ la P, un nod "nepotrivit" x va fi adăugat în consecinţă unui nivel impar.
  - Drumul de la x la oricare nod al nivelului 0 este un **drum augmentat** relativ la P.
- În figura 12.8.2.a (d) se prezintă un drum augmentat pentru graful (a), relativ la potrivirea (c).
  - În acest context N_1={1,2,3,4,5} iar N_2={a,b,c,d,e}.
  - La momentul considerat, la nivelul 0, mulţimea nodurilor "nepotrivite" ale lui N_1 conţine un singur element, nodul 5, cu care se porneşte construcţia grafului drumului augmentat.
  - La nivelul 1 se adaugă arcul care **nu aparţine** potrivirii (5,a).

- La nivelul 2 se adaugă arcul (a,2) care **aparţine** potrivirii.

- La nivelul 3 se poate adăuga fie arcul (2,d) fie arcul (2,e) nici unul dintre ele neaparţinând potrivirii.

- Dacă se adaugă arcul (2,d), se poate adăuga în continuare arcul (d,4)la nivelul 4 (aparţine potrivirii), după care se poate adăuga arcul (4,e)la nivelul 5 (nu aparşine potrivirii)

- Deoarece nodul e este "nepotrivit" din punctul de vedere al grafului drumului augmentat, construcţia acestui graf se termină în ambele cazuri după adăugarea nodului e.

- După cum se observă s-au constrit două drumuri augmentate după cum se remarcă in figura 12.8.2.a (d)

- Ambele drumuri e,4,d,2,a,5 şi e,2,a,5 sunt **drumuri augmentate** relativ la potrivirea din fig. 12.8.2.a (c).

- Spre exemplu dacă se utilizează primul drum şi se extrag din potrivirea P reprezentată la (c) arcele arcele din drum care aparţin lui P (4,d)şi(2,d), respectiv se adaugă arcele din drumul augmentat care nu aparţin lui P (e,4),(d,2),(a,5)se obţine potrivirea maximă din figura 12.8.1.a.

- Se presupune că G are n noduri şi a arce.

- Construcţia grafului drumului augmentat pentru o potrivire dată, necesită O*(a)* paşi de execuţie dacă se utilizează reprezentarea bazată pe structuri de adiacenţe.

    - Deci fiecare drum augmentat nou necesită un efort de calcul proporţional cu O*(a)*.

- Pentru determinarea potrivirii maxime este necesară determinarea a cel mult n/2 drumuri augmentate, deoarece fiecare dintre ele măreşte potrivirea curentă cu cel puţin un arc.

    - În consecinţă, pentru grafuri bipartite, potrivirea maximă poate fi determinată în O*(n*a)* unităţi de timp de execuţie.

12.8.3. Determinarea potrivirii maxime prin metoda căutării în reţele de curgere

- O altă soluţie eficientă pentru determinarea potrivirii maxime poate fi formulată după cum urmează.

- Pentru a rezolva o instanţă particulară a problemei potrivirilor, se va construi o instanţă corespunzătoare a **problemei reţelelor de curgere** a cărei soluţie va fi determinată utilizând metoda prezentată în secţiunea anterioară.

- Utilizând această soluţie, în continuare se poate rezolva şi problema potrivirilor în acelaşi context.

- De fapt în această manieră, problema potrivirilor se reduce la problema reţelelor de curgere.

- Construcţia reţelei este directă.

    - (1) Dându-se o instanţă a unei potriviri pentru un graf bipartit, se construieşte o instanţă a **reţelei de curgere** adăugând un **nod sursă** din care pornesc arce spre toate nodurile uneia dintre mulţimile de noduri ale grafului bipartit.

    - (2) Se adaugă în continuare toate arcele grafului bipartit care conectează nodurile primei mulţimi cu nodurile aparţinând celei de-a doua.

    - (3) La sfârşit se adaugă un nod destinaţie şi un set de arce care conectează nodurile celei de-a doua mulţimi a grafului bipartit cu acest nod.

    - (4) Toate arcele care apar în reţea se consideră de capacitate unu.

- În figura 12.8.3.b se prezintă modul de construcţie al reţelei de curgere pentru graful bipartit din fig. 12.8.3.a împreună cu paşii rezolvării problemei reţelei şi implicit a determinării potrivirii maxime.

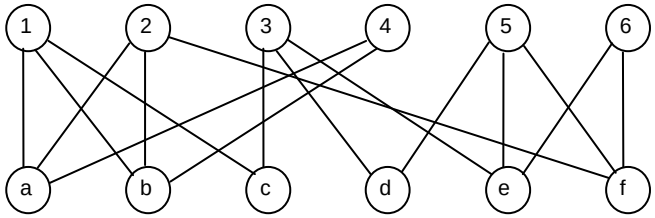


Fig.2.8.3.a. Graf bipartit

- Din reprezentare rezultă în mod evident partajarea nodurilor grafurilor bipartit, direcţia fluxurilor precum şi faptul că deoarece toate capacităţile sunt 1, fiecare drum prin reţea corespunde unui arc al potrivirii.

- Astfel în exemplul din figură, desenele (a), (b), (c) şi (d) materializează primii patru paşi care conduc la determinarea arcelor corepunzătoare unei potriviri parţiale: `a1,b2,c3` şi `d5`.

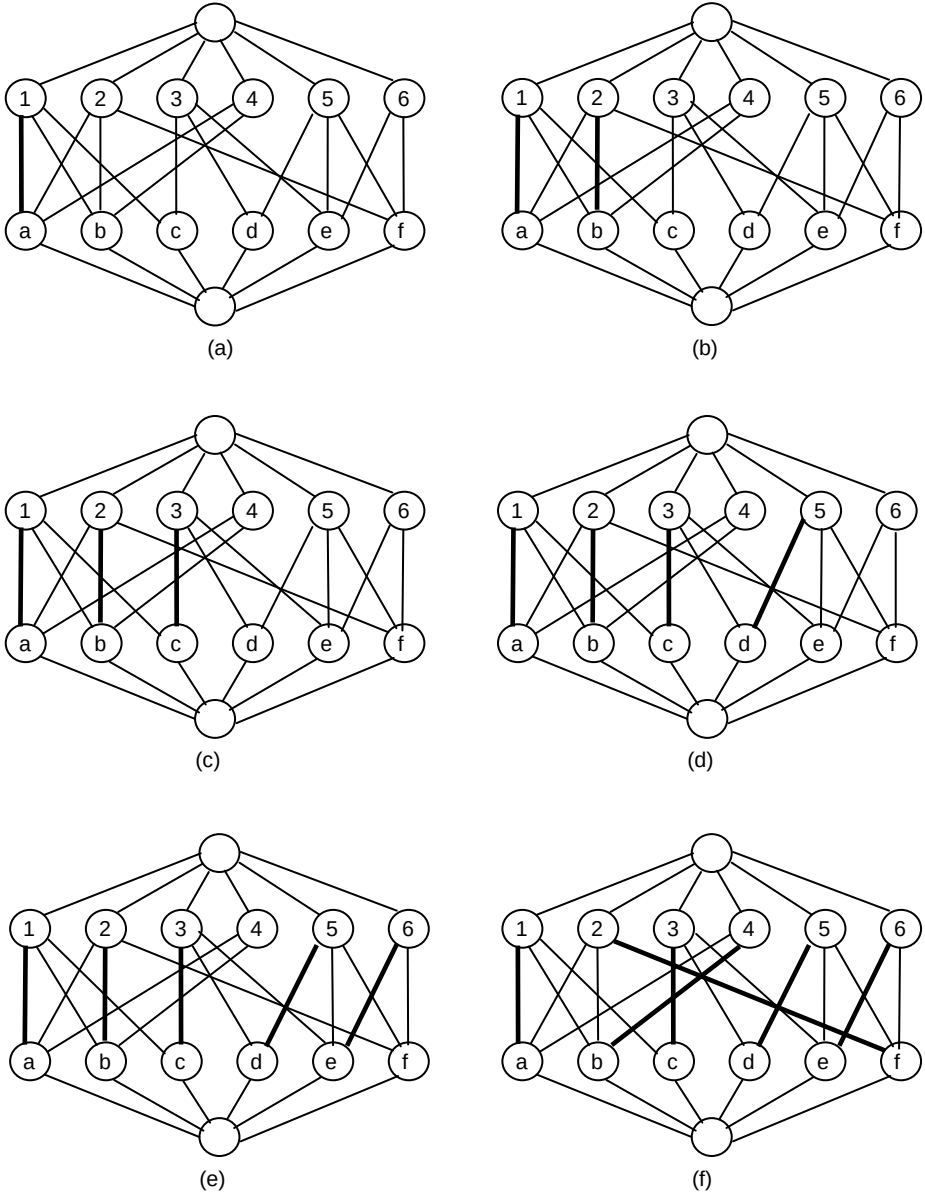


Fig.12.8.3.b. Utilizarea reţelelor de curgere în determinarea potrivirii maxime pentru un graf bipartit

- Pentru determinarea acestor drumuri se utilizează secvenţa [12.7.3.b].

- Fiecare apel al procedurii `CautPrioritar` determină fie găsirea unui drum care creşte fluxul total cu o unitate fie termină căutarea.

- În pasul (e) toate drumurile directe prin reţea au fost selectate şi algoritmul trebuie să utilizeze arce de tip "înapoi".

- Drumul găsit în acest pas este `4,b,2,f`.

    - Acest drum, în mod evident determină creşterea fluxului în reţea după cum s-a văzut în secţiunea anterioară.

- În contextul de faţă, un drum poate fi conceput ca şi un set de paşi prin care se crează o nouă potrivire parţială (utilizând unul sau mai multe arce) pornind de la potrivirea curentă.

- În situaţia reprezentată în desen (e), această construcţie decurge în mod natural din traversarea arcelor din reţea în ordinea:
    - (1) "`4b`" ceea ce presupune adăugarea arcului `(b,4)` potrivirii,
    - (2) "`b2`" ceea ce înseamnă extragerea arcului `(b,2)`
    - (3) "`2f`" ceea ce presupune adăugarea arcului `(f,2)` potrivirii.
- Astfel, după ce drumul e construit, se ajunge la potrivirea `a1,b4, c3,d5,e6,f2`.
- În mod echivalent fluxul prin reţea este produs de conductele pline care conectează nodurile precizate şi este maxim în situaţia dată.
- Demonstraţia faptului că potrivirea conţine exact arcele care umplute la capacitate determină fluxul maxim prin reţea este imediată.
    - (1) În primul rând, reţeaua de curgere conduce întotdeauna la o potrivire legală element care decurge din următorul raţionament:
        - Întrucât fiecare nod al reţelei are un arc de capacitate unu care îl traversează (intră sau iese din el), rezultă că prin fiecare nod poate trece o singură unitate de flux ceea ce implică faptul că fiecare nod va fi inclus o singură dată în potrivire.
    - (2) În al doilea rând, nici o potrivire nu poate avea mai multe arce, deoarece o astfel de potrivire ar conduce la un flux superior fluxului maxim determinat de algoritm.
- Astfel, pentru a determina potrivirea maximă pentru un graf bipartit, se transformă graful într-o reţea de curgere potrivită utilizării algoritmului dezvoltat în secţiunea precedentă.
- Este evident faptul că reţeaua care rezultă în acest caz este mai simplă decât în cazul grafurilor oarecare pentru care a fost conceput algoritmul.
    - Din acest motiv, algoritmul dezvoltat este oarecum mai eficient în acest caz particular.
- În consecinţă, luând în considerare performanţele algoritmului de căutare în reţele, potrivirea maximă pentru un graf bipartit poate fi determinată în:
    - O(n^3) paşi în cazul grafurilor dense reprezentate prin matrice de adiacenţe
    - O$(n(n+a)\ log\ n)$ paşi în cazul grafurilor rare reprezentate prin structuri de adiacenţe.
- Demonstraţia este imediată:
    - Construcţia preconizată asigură faptul că fiecare apel al procedurii `CautăPrioritar` adaugă cel puţin un arc potrivirii deci vor fi necesare cel mult `n/2` apeluri ale procedurii pe parcursul execuţiei algoritmului.

- Astfel timpul total necesitat de execuţia algoritmului este proporţional cu produsul dintre factorul n şi performanţa algoritmului de căutare în reţele utilizat.