

12. Grafuri orientate

- După cum s-a mai precizat, **grafurile orientate** sunt grafuri în care arcele care conectează nodurile au **un singur sens**.
 - Această restricție face mult mai dificilă evidențierea și exploatarea diverselor proprietăți ale grafurilor de acest tip.
 - Prelucrarea unui astfel de graf este identică cu o călătorie cu mașina într-un oraș cu foarte multe străzi cu sens unic sau cu o călătorie cu avionul într-o țară în care liniile aeriene nu sunt dus-întors.
 - În astfel de situații a ajunge dintr-un loc în altul poate reprezenta o adevărată problemă.
- De multe ori arcele orientate reflectă anumite tipuri de **relații de precedență** în aplicația pe care o modelează.
 - Spre **exemplu** un graf orientat poate fi utilizat ca model pentru o linie de fabricație:
 - Nodurile corespund diverselor operații care trebuie executate
 - Un arc de la nodul x la nodul y precizează faptul că operațiunea corespunzătoare nodului x trebuie executată **înaintea** celei corespunzătoare nodului y .
 - În acest context, problema care se pune este aceea de a executa toate aceste operații, fără a eluda niciuna dintre relațiile de precedență impuse.
- După cum s-a menționat în capitolul 10, **reprezentările grafurilor orientate** sunt simple extensii (de fapt restricții) ale reprezentărilor grafurilor neorientate. Astfel:
 - În reprezentarea bazată pe **liste de adiacențe**, fiecare arc apare o singură dată: arcul de la nodul x la y este reprezentat prin prezența nodului y în lista de adiacențe a lui x .
 - În reprezentarea bazată pe matrice de adiacențe, arcul de la nodul x la y se marchează tot o singură dată, prin valoarea "adevărat" a elementului matricii de adiacențe situat în linia x , coloana y (nu și 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
 - Notația AG precizând un arc care își are sursa în nodul A și destinația în nodul G .
- Este însă posibil ca între două noduri să existe două arce având sensuri opuse (exemplu HI și IH , respectiv LM și ML).

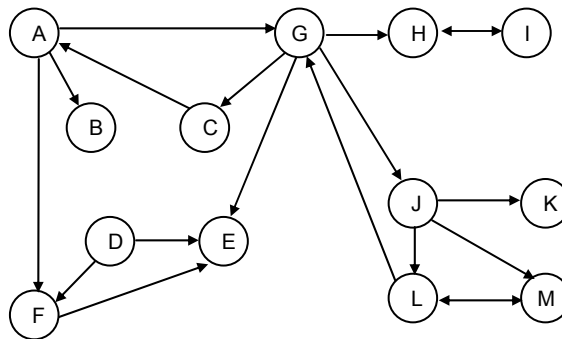


Fig.12.1.a. Exemplu de graf orientat.

- Pornind de la această precizare este ușor de observat faptul că un **graf neorientat** poate fi asimilat cu **graful orientat** identic ca și structură, care conține pentru fiecare arc al grafului neorientat două arce opuse.
 - În acest context unii dintre algoritmi dezvoltati în acest capitol, pot fi considerați **generalizări** ale algoritmilor din capitolele anterioare.
- În cadrul acestui capitol vor fi prezentate unele dintre problemele specifice acestei categorii de grafuri precum și o serie de algoritmi consacrați care le soluționează.
- 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

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** și 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-și propune să determine cel mai scurt drum de la o origine la o destinație precizată.
 - Această problemă are același grad de dificultate ca și problema anterioară care de fapt o generalizează.
 - Astfel, pentru a rezolva această ultimă problemă, este suficient ca execuția algoritmului care determină drumurile minime cu origine unică să fie oprită în momentul în care se ajunge la destinația precizată.
- Exact ca și în cazul grafurilor ponderate neorientate, ponderea poate avea semnificația fizică a unui cost, a unei valori, a unei lungimi, a unui timp, etc.
- Intuitiv, graful G poate fi asimilat cu o hartă a traseelor aeriene ale unui stat în care:
 - Fiecare nod reprezintă un oraș
 - Fiecare arc (x, y) reprezintă o legătură aeriană de la orașul x la orașul y .
 - Ponderea unui arc (x, y) poate fi timpul de zbor sau prețul biletului.
- Desigur, ca model ar putea fi utilizat și un graf neorientat deoarece ponderea arcului (x, y) trebuie să fie în principiu aceeași 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) și (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 mulțime** M care conține nodurile pentru care cea mai scurtă distanță la nodul origine este deja cunoscută.
 - Inițial M conține numai nodul **origine**.
 - În fiecare pas al execuției algoritmului, se adaugă mulțimii M un nod x care **nu** aparține încă mulțimii și a cărei distanță 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 conținute de M .
 - Un astfel de drum se numește “**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 și în conștiință, tabloul D memorează cea mai scurtă distanță de la origine la fiecare nod al grafului.
- Schița de principiu a algoritmului lui Dijkstra apare în secvența [12.1.1.a].

PROCEDURE Dijkstra;

{Determină costurile celor mai scurte drumuri care conectează nodul 1, considerat drept origine, cu toate celelalte noduri ale unui graf orientat}

BEGIN

[1] $M := [1];$ {nodul origine}

[2] **FOR** $i := 2$ **TO** n **DO** {inițializarea lui D }

[3] $D[i] := \text{COST}[1, i];$ [12.1.1.a]

[4] **FOR** $i := 1$ **TO** $n-1$ **DO**

BEGIN

[5] *alege un nod x aparținând mulțimii $N - M$ astfel încât $D[x]$ să fie minim;

[6] *adaugă pe x lui M ;

[7] **FOR** fiecare nod y din $N - M$ **DO**

[8] $D[y] := \min(D[y], D[x] + \text{COST}[x, y])$

END

END; {Dijkstra}

- Referitor la algoritmul lui Dijkstra, se presupune că:
 - Se dă un graf orientat $G = (N, A)$ unde $N = \{1, 2, 3, \dots, n\}$

- 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 iterație a algoritmului, tabloul $D[i]$ conține lungimea drumului special minim curent de la origine la nodul i .
- Pentru exemplificare, se prezintă execuția algoritmului lui Dijkstra pentru graful orientat din figura 12.1.1.a.

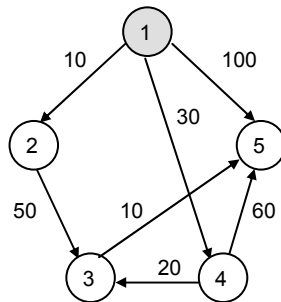


Fig.12.1.1.a. Graf orientat ponderat

- Inițial $M=\{1\}$, $D[2]=10$, $D[3]=\infty$, $D[4]=30$ și $D[5]=100$.
- În prima iterație a buclei **FOR** din liniile [4] – [8], se selectează $x=2$ ca fiind nodul de distanță minimă din D .
- În continuare se face $D[3]=\min(\infty, 10+50)=60$.
 - $D[4]$ și $D[5]$ 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 iterație a buclei **FOR** 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

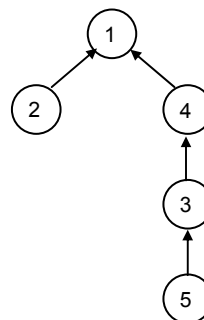


Fig.12.1.1.b. Determinarea drumurilor minime cu origine unică în baza algoritmului lui Dijkstra

- În vederea reconstrucției drumurilor minime de la origine la fiecare nod al grafului, se utilizează un alt tablou *Părinte*
 - În acest tablou, *Părinte[x]* memorează nodul care precede nodul *x* în cadrul drumului minim, adică memorează tatăl nodului *x*.
 - Tabloul *Părinte* se inițializează cu *Părinte[i]=1* pentru toate valorile lui *i*≠1.
 - Tabloul *Părinte* poate fi actualizat după linia [8] în procedura *Dijkstra* din secvența [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 înlănțuirile indicate de tabloul *Părinte*.
- Astfel, pentru graful orientat din fig. 12.1.1.a, tabloul *Părinte* conține 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 (părinții) 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 consecință 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 aceeași figură apare reprezentat și arborele de acoperire corespunzător drumurilor

12.1.3. Analiza performanței algoritmului lui Dijkstra

- Se presupune că algoritmul din secvența [12.1.1.a], operează asupra unui graf orientat cu n noduri și a arce.
- Dacă pentru reprezentarea grafului se utilizează o **matrice de adiacențe**, atunci bucla **FOR** din liniile [7] – [8] necesită un efort de calcul proporțional cu $O(n)$.
 - Întrucât această buclă este executată de $n-1$ ori, (bucla **FOR** din linia [4]), timpul total de execuție va fi proporțional cu $O(n^2)$.
 - Este ușor 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 adiacențe**, respectiv o **coadă bazată pe prioritate** implementată ca un ansamblu (arbore binar parțial ordonat), pentru a păstra distanțele la nodurile mulțimii $N-M$.
 - În aceste condiții, bucla din liniile [7] – [8] poate fi implementată ca și o traversare a listei de adiacențe a lui x cu actualizarea distanțelor nodurilor din coada bazată pe prioritate.
 - În total, pe întreaga durată a execuției procedurii, se vor realiza a actualizări, fiecare cu un efort proporțional cu $O(\log n)$, astfel încât timpul total consumat în liniile [7] – [8] va fi proporțional cu $O(a \log n)$ și nu cu $O(n^2)$.
 - În mod evident liniile [2] – [3] necesită asemeni liniilor [4] – [6] un efort de calcul proporțional cu $O(n)$.
 - Utilizând în reprezentarea mulțimii $N-M$ o coadă bazată pe priorități, linia [5] implementează operația de extragere a elementului cu prioritatea minimă din coadă, fiecare din cele $n-1$ iterații ale acestei linii necesitând $O(\log n)$ unități de timp.
 - În consecință, **timpul total** consumat de această variantă a algoritmului lui Dijkstra este limitat la $O(a \log n)$ ($a \geq n-1$), performanță care este considerabil mai bună decât $O(n^2)$.
 - Se reamintește faptul că această performanță se poate obține 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 conținând timpii de zbor pentru anumite trasee aeriene care conectează orașe și că se dorește construcția unei tabelă care furnizează **cei mai scurți timpi de zbor** între oricare două orașe.
 - Aceasta este o instanță 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 $G = (N, A)$, 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 $x, y \in N$, 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 execuției algoritmului lui Dijkstra este un tablou care conține distanța de la origine la restul nodurilor grafului.
 - În cazul de față, deoarece este necesară determinarea distanțelor pentru toate perechile de noduri, avem nevoie de o matrice de elemente în care fiecare rând se poate determina în baza algoritmului lui Dijkstra.
- 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 ponderat $G = (N, A)$ are atașată matricea de ponderi $COST$.
 - Pentru conveniență, se consideră că nodurile mulțimii N sunt numerotate de la 1 la n .
- **Algoritmul lui Floyd** utilizează o matrice A de dimensiuni $n \times n$ cu ajutorul căreia determină lungimile drumurilor minime.
 - Inițial se face $A[i, j] = COST[i, j]$ pentru toți $i \neq j$.
 - Dacă nu există niciun arc de la nodul i la nodul j , se presupune că $COST[i, j] = \infty$.
 - Elementele diagonalei matricei A se pun toate pe 0.
- Algoritmul lui Floyd execută n iterații asupra matricii A .
 - După cea de-a k -a iterație, $A[i, j]$ va conține lungimea minimă a unui drum de la nodul i la nodul j , care nu trece prin nici un nod cu număr mai mare ca și k .

- Cu alte cuvinte, i și j pot fi oricare două noduri ale grafului care delimitează începutul și sfârșitul 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 iterație se utilizează următoarea formulă în calculul lui A (formula [12.2.1.a]):

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

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

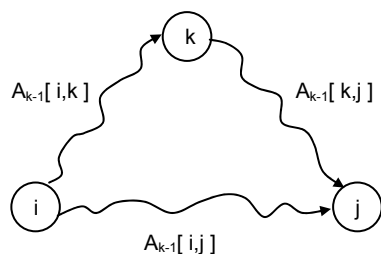
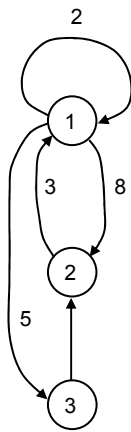


Fig.12.2.1.a. Selecția drumului minim de la nodul i la nodul j la momentul k

- Pentru calculul lui $A_k[i, j]$ 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 și k , 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 și k .
- Dacă drumul din urmă se dovedește 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 aceleiași figuri modul în care se modifică matricea A pornind de la conținutul ei inițial A_0 și terminând cu conținutul său final A_3 .



(a)

	1	2	3
1	0	8	5
2	3	0	∞
3	∞	2	0

A_0

	1	2	3
1	0	8	5
2	3	0	8
3	5	2	0

A_2

	1	2	3
1	0	8	5
2	3	0	8
3	∞	2	0

A_1

	1	2	3
1	0	7	5
2	3	0	∞
3	5	2	0

A_3

(b)

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

- Deoarece $A_k[i, k] = A_{k-1}[i, k]$ și $A_k[k, j] = A_{k-1}[k, j]$, 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 iterații.
- Cu alte cuvinte, matricea A este invariantă în raport cu indicele k .
- În consecință se poate utiliza o singură copie a matricii A .
- Structura de principiu a procedurii care implementează algoritmul lui Floyd în baza considerentelor mai sus enunțate apare în secvența [12.2.1.a].

```

PROCEDURE Floyd(VAR A: ARRAY[1..n,1..n] OF real;
                  COST: ARRAY[1..n,1..n] OF real);

```

```

{Procedura determină matricea drumurilor minime A pornind de
la matricea ponderilor COST}

```

```

VAR i, j k: INTEGER;

```

```

BEGIN

```

```

[1] FOR i:= 1 TO n DO

```

```

[2]   FOR j:= 1 TO n DO

```

```

[3]     A[i, j] := COST[i, j];

```

[12.2.1.a]

```

[4] FOR i:= 1 TO n DO

```

```

[5]   A[i, i] := 0;

```

```

[6] FOR k:= 1 TO n DO

```

```

[7]   FOR i:= 1 TO n DO

```

```

[8]     FOR j:= 1 TO n DO

```

```

[9]       IF A[i, k] + A[k, j] < A[i, j] THEN

```

```

[10]        A[i, j] := A[i, k] + A[k, j]

```

```

END; {Floyd}

```

- Timpul de execuție al acestei proceduri este în mod clar proporțional cu $O(n^3)$ deoarece la baza sa stă o buclă triplă încuibată.
- Verificarea corectitudinii funcționării algoritmului se poate realiza simplu prin metoda inducției.
 - Astfel este ușor de demonstrat că după ce k trece prin bucla triplă **FOR**, $A[i, j]$ 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 și k [AH85].

12.2.2. Comparație între algoritmul lui Floyd și algoritmul lui Dijkstra

- Pentru grafuri reprezentate prin **matrici de adiacențe**:
 - (1) Versiunea **algoritmului Dijkstra** determină drumurile minime specifice unui nod precizat cu performanța $O(n^2)$,
 - (2) **Algoritmul lui Floyd** determină toate drumurile minime cu performanța $O(n^3)$.
 - Constantele de proporționalitate reale depind de natura compilatorului, de sistemul de calcul și de implementarea propriu-zisă.
 - De fapt activitatea experimentală și măsurătorile reale reprezintă cele mai bune criterii de apreciere a performanțelor unui algoritm.
- În condițiile în care numărul de arce a este mult mai redus decât n^2 și drept consecință grafurile sunt reprezentate prin **liste de adiacențe**:
 - (1) **Algoritmul lui Floyd** își păstrează performanța stabilită $O(n^3)$,
 - (2) Este de așteptat 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 proporțional cu $O(a \log n)$.
 - În consecință utilizând această variantă de algoritm, problema drumurilor minime corespunzătoare tuturor perechilor de noduri se poate rezolva aplicând algoritmul lui Dijkstra tuturor nodurilor grafului cu performanța $O(na \log n)$.
 - Se face din nou precizarea că această performanță se poate obține atunci când $a \ll n^2$ respectiv în cazul grafurilor rare de mari dimensiuni.

12.2.3. Determinarea traseelor drumurilor minime

- Algoritmii dezvoltați în cadrul acestui paragraf determină de fapt **costurile drumurilor minime**.
- De multe ori, în practică este însă foarte util a se cunoaște și **traseul** acestor drumuri minime.
- Pentru a rezolva această problemă, în contextul algoritmului lui Floyd este necesară utilizarea unei alte matrici Drum
 - În matricea Drum, o locație $\text{Drum}[i, j]$ memorează acel nod k care conduce în cadrul algoritmului la cea mai mică valoare pentru $A[i, j]$.
 - Dacă $\text{Drum}[i, j]=0$ atunci cel mai scurt drum este cel direct de la nodul i la nodul j .
- Varianta modificată a algoritmului lui Floyd care determină și matricea Drum apare în secvența [12.2.3.a].

```
-----
PROCEDURE Floyd1 (VAR A: ARRAY [1..n,1..n] OF real
                  COST: ARRAY [1..n,1..n] OF real
                  VAR Drum: ARRAY [1..n,1..n] OF integer);
```

```
{Procedura primește matricea COST și determină matricea A
care memorează lungimile drumurilor minime și matricea Drum
care memorează nodul intermediar al fiecărui drum de la i la
j (dacă există)}
```

```
VAR i, j, k: integer;
```

```
BEGIN
```

```
[1] FOR i:= 1 TO n DO
```

```
[2]   FOR j:= 1 TO n DO
```

```
       BEGIN
```

```
[3]         A[i, j] := C[i, j];
```

```
[12.2.3.a]
```

```
[4]         Drum[i, j] := 0
```

```
       END;
```

```
[5] FOR i:= 1 TO n DO
```

```
[6]   A[i, i] := 0;
```

```
[7]   FOR k:= 1 TO n DO
```

```
[8]     FOR i:= 1 TO n DO
```

```
[9]       FOR j:= 1 TO n DO
```

```
[10]        IF A[i, k] + A[k, j] < A[i, j] THEN
```

```
            BEGIN
```

```
[11]              A[i, j] := A[i, k] + A[k, j];
```

```
[12]              Drum[i, j] := k
```

```
            END
```

```
        END; {Floyd1}
```

```
-----
```

- Matricea Drum este de fapt o colecție de arbori corespunzători drumurilor minime, câte unul pentru fiecare nod al grafului original.
- Afișarea **traseului drumului minim** de la nodul i la nodul j , prin evidențierea nodurilor intermediare se poate realiza cu ajutorul procedurii Traseu (secvența [12.2.3.b]).

```

-----
PROCEDURE Traseu(i,j: integer);
{Afișează traseul drumului minim de la nodul i la nodul j}

VAR k: integer;

BEGIN
    k:= Drum[i,j];
    IF k <> 0 THEN
        BEGIN
            Traseu(i,k);
            writeln(k);
            Traseu(k,j)
        END
    END; {Traseu}
-----

```

- Procedura Traseu aplicată unei matrici oarecare ar putea cicla la infinit.
 - Acest lucru **nu** se întâmplă în situația 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 și în același timp j să apară în drumul minim de la i la k .
 - Se reamintește din nou importanța crucială a valorilor pozitive a ponderilor arcelor grafului.
- În figura 12.2.3.a apare conținutul final al matricii Drum pentru graful orientat din figura 12.2.1.b. (a).

	1	2	3
1	0	3	0
2	0	0	1
3	2	0	0

matricea Drum

Fig.12.2.3.a. Matricea traseelor drumurilor minime corespunzătoare grafului din figura 12.2.1.b. (a).

12.2.4. Aplicație. Determinarea centrului unui graf orientat ponderat

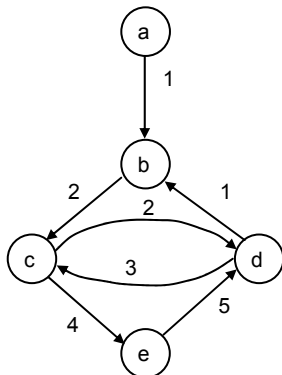
- Se presupune că se dă graful orientat ponderat $G=(N, A)$ și se cere să se determine **cel mai central nod** al său adică **centrul grafului**.

- Pentru a defini termenul de “**cel mai central nod**” se definește pentru început noțiunea de **excentricitate** a unui nod aparținând unui graf ponderat.

- **Excentricitatea** unui nod $x \in \mathbb{N}$ este precizată de formula [12.2.4.a]:

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

- 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 excentricităților nodurilor apar în tabloul (c) din aceeași figură.
 - Conform definiției anterioare, centrul grafului G este nodul d .
- Utilizând **algoritmul lui Floyd**, determinarea centrului unui graf G se poate realiza simplu.
- Presupunând că $COST$ este matricea ponderilor arcelor grafului, 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 excentricitățile nodurilor $i, (1 \leq i \leq N)$ 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 aceeași figură (a).
 - Determinarea nodului central al grafului este evidentă.



	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

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

(a)

(b)

(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 proprietățile de conectivitate ale grafurilor.
 - Pur și simplu, toate nodurile la care se poate ajunge în procesul de căutare aparțin unei aceleiași componente conexe a grafului.
- Această observație este valabilă în principiu și în cazul grafurilor orientate cu precizarea însă că în această situație rezolvarea este mai complexă și 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 inițial cu arce pe baza următoarei metode: dacă în graful inițial 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 obține adăugând toate arcele de această natură se numește **închiderea tranzitivă** a grafului inițial.
 - Deoarece este de așteptat să fie adăugate un număr mare de arce, deci graful obținut să fie dens, se apreciază că pentru închiderea tranzitivă, cea mai potrivită metodă de reprezentare este cea bazată pe **matrice de adiacențe**.
- 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 numește algoritmul lui Warshall și care deși 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 observația simplă că, dacă într-un graf orientat există o modalitate de a ajunge de la nodul i la nodul k și o modalitate de a ajunge de la nodul k la nodul j , parcurgând arce ale grafului, atunci cu siguranță există un drum care conectează nodul i cu nodul j .

- Problema constă de fapt în a exploata de asemenea manieră această observație 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 , și o posibilitate de a ajunge de la nodul k la nodul j în aceleași condiții, atunci există un drum de la nodul i la nodul j care străbate numai noduri care cu indicele mai mic ca și $k+1$ ”.
- În baza acestei interpretări, dându-se graful ordonat G și matricea sa de adiacențe 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).

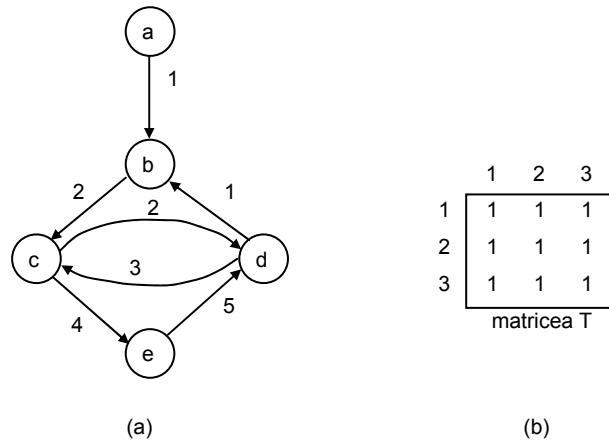


Fig.12.3.a. Închiderea tranzitivă a unui graf orientat

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

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

- 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 și 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 $k-1$ **și** există un drum de la k la j , care nu trece prin niciun nod cu număr mai mare decât $k-1$.

- Ca și în cazul algoritmului lui Floyd, sunt valabile formulele $T_k[i, k] = T_{k-1}[i, k]$ și $T_k[k, j] = T_{k-1}[k, j]$ motiv pentru care în calcul se poate utiliza o singură instanță a matricii T.
- Structura de principiu a procedurii care implementează algoritmul lui Warshall apare în secvența [12.3.a].

```

-----
PROCEDURE Warshall (A: ARRAY[1..n,1..n] OF boolean;
                    VAR T: ARRAY[1..n,1..n] OF boolean);

{Procedura construiește în T închiderea tranzitivă a lui A}

    VAR i, j, k: integer

    BEGIN
[1] FOR i:= 1 TO n DO
[2]     FOR j:= 1 TO n DO
[3]         T[i, j] := A[i, j];                                [12.3.a]
[4]     FOR k:= 1 TO n DO
[5]         FOR i:= 1 TO n DO
[6]             FOR j:= 1 TO n DO
[7]                 IF T[i, j] = false THEN
[8]                     T[i, j] := T[i, k] AND T[k, j]
    END; {Warshall}
-----

```

- În urma unei analize sumare a codului, performanța algoritmului rezultă imediat, $O(n^3)$.
- După alți autori, performanța poate fi stabilită și astfel.
 - Fiecare din cele a arce conduce la o iterație cu n pași în bucla **FOR** cea mai interioară.
 - În plus, sunt testate și eventual actualizate toate cele n^2 locații ale matricii T.
 - Rezultă un timp de execuție proporțional 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 și ele pot fi aplicate, cu particularizări specifice, oricărui tip de graf.
- Astfel în cazul grafurilor orientate, în timpul traversării se ține cont efectiv de orientarea arcelor examinate, lucru care în contextul grafurilor neorientate nu este necesar.

- Din acest motiv și arborii de acoperire rezultați 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”.
 - Opțiunea 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 inițial cu “nevizitat”.
 - Căutarea în adâncime acționează inițial selectând un nod x al lui G ca și nod de start, nod care este marcat cu “vizitat”.
 - În continuare, fiecare nod nevizitat adiacent lui x este "căutat" pe rând, utilizând aceeași 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 totuși mai rămân noduri nevizitate, se selectează unul dintre acestea drept următorul nod de start și 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 dorește a fi traversat este reprezentat cu ajutorul listelor de adiacențe.
 - Se notează cu $L(x)$ lista de adiacențe a nodului x .
 - De asemenea se mai utilizează tabloul `marc`, ale cărui elemente pot lua valorile "nevizitat" respectiv "vizitat", tablou care este utilizat în menținerea evidenței stării nodurilor grafului.
- Structura de principiu a procedurii recursive de căutare în adâncime apare în secvența [12.4.1.a].
 - Această procedură trebuie apelată însă dintr-un context mai general care asigură inițializarea tabloului `marc` și selectarea nodurilor încă nevizitate ale grafului (secvența [12.4.1.b]).

PROCEDURE CautInAdâncime(x : Nod) ;

```

VAR k: Nod;
BEGIN
[1] marc[x]:= vizitat;
[2] Prelucrare(x):
[3] FOR fiecare nod k din L(x) DO [12.4.1.a]
[4]   IF marc[k] = nevizitat THEN
[5]     CautInAdâncime(k)
   END; {CautInAdâncime}

```

{Programul principal}

```

BEGIN
  FOR x:= 1 TO n DO
    marc[x]:= nevizitat;
  FOR x:= 1 TO n DO [12.4.1.b]
    IF marc[x] = nevizitat THEN
      CautInAdâncime(x)
END;

```

- 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 secvenței [12.4.1.a].
- După cum s-a precizat, această tehnică stă la baza rezolvării mai multor probleme specifice grafurilor orientate.
 - În consecință, prelucrarea nodurilor vizitate va îmbrăca o formă specifică funcție de problema care se dorește a fi soluționată în baza acestei tehnici de parcurgere a grafurilor orientate.
- **Performanța procedurii** CautInAdâncime, analizată în contextul parcurgerii unui graf orientat cu a arce, cu $n \leq a$, în reprezentarea bazată pe liste de adiacențe este $O(a)$, după cum s-a evidențiat în capitolul 10.
- Pentru **exemplificarea** procesului de traversare a grafurilor orientate prin metoda căutării în adâncime:
 - Se presupune că procedura din secvența [12.4.1.b] este aplicată grafului orientat din figura 12.4.1.a. (a), considerând că nodul de pornire inițial este nodul a ($x=a$).
 - Graful se consideră reprezentat printr-o structură de adiacențe sugerată în aceeași figură (b).

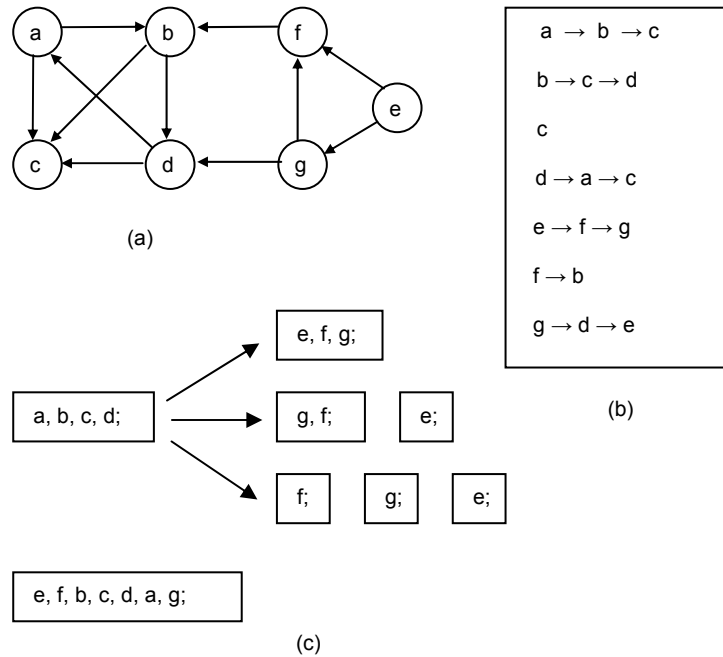


Fig.12.4.1.a. Traversarea unui graf orientat

- Algoritmul marchează nodul a cu vizitat și selectează nodul b, primul din lista de adiacențe a lui a.
- Deoarece nodul b este nevizitat, căutarea continuă prin realizarea unei apel `CautInAdâncime(b)`.
- În continuare se marchează b cu vizitat și se selectează primul nod din lista sa de adiacențe.
- Presupunând că nodul c apare înaintea lui d în această listă, așa cum se prezintă în figură, se apelează `CautInAdâncime(c)`.
- Deoarece lista de adiacențe a lui c este vidă, căutarea revine în lista lui b de unde se selectează nodul d.
- Se parcurge în continuare lista de adiacențe a lui d.
- Întrucât nodurile a și c care apar în lista de adiacențe a lui d au fost deja vizitate, căutarea lui d se încheie și se revine în lista lui b apoi în cea a lui a, care sunt și ele epuizate.
- În acest moment apelul inițial `CautInAdâncime(a)` s-a terminat.
- Totuși, deoarece graful nu a fost încă parcurs în întregime întrucât nodurile e, f și 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 observație.

- Dacă graful din fig.12.4.1.a. (a) ar fi un graf neorientat, el ar reprezenta o singură componentă conexă și ar putea fi parcurs integral, realizând un singur apel al procedurii `CautInAdâncime` din cadrul secvenței [12.4.1.b] indiferent de nodul care este selectat drept punct de start.
- În cazul grafurilor orientate situația se schimbă.
 - Numărul de apeluri nerecursive ale procedurii `CautInAdâncime`, realizat în cadrul secvenței [12.4.1.b] depinde în mod esențial de ordinea în care sunt selectate nodurile care reprezintă puncte de start.
 - Spre exemplu în cazul mai sus prezentat, graful apare constituit din două componente conexe (a, b, c, d) și (e, f, g) .
 - Dacă în locul nodului e , la cel de-al doilea apel ar fi fost selectat nodul g apoi e ar fi rezultat trei componente conexe (a, b, c, d) , (g, f) și (e)
 - Dacă ar fi fost selectate nodurile f, g și e în această ordine ar fi rezultat 4 componente conexe.
 - În sfârșit, dacă se alege inițial nodul e drept nod de start, graful din figură conține o singură componentă conexă (fig.12.4.1.a. (c)).
- Această particularitate se reflectă direct în topologia arborilor de acoperire care rezultă în urma unor astfel de parcurgeri ale grafurilor orientate.

12.5. Grafuri orientate aciclice

- Un **graf orientat aciclic** este un graf orientat care **nu** conține cicluri.
- Appreciate în termenii relațiilor pe care le modelează, grafurile orientate aciclice au un caracter de generalitate mai pronunțat decât arborii dar sunt mai puțin generale ca și grafurile orientate.
- Ca și topologie, astfel de grafuri pot conține multe cicluri, dacă **nu** se ia în considerare direcționarea arcelor.
 - Dacă această direcționare este însă luată în considerare un astfel de graf nu conține nici un ciclu.
- De fapt aceste grafuri pot fi considerate parțial arbori, parțial grafuri orientate, element care le conferă o serie de proprietăți speciale.
 - Spre exemplu, pădurea de arbori de căutare în adâncime asociată unui graf orientat aciclic, **nu** conține arce de tip “înapoi”.
 - În figura 12.5.a. apare un exemplu de arbore (a), un exemplu de graf orientat aciclic (b) și un exemplu de graf orientat cu un ciclu (c).

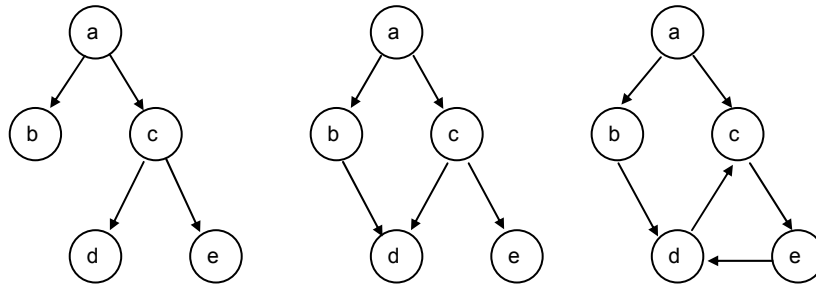


Fig.12.5.a. Tipuri de grafuri orientate

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

$$((a+b) * c + ((a+b) + e) * (e+f)) * ((a+b) * c)$$

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

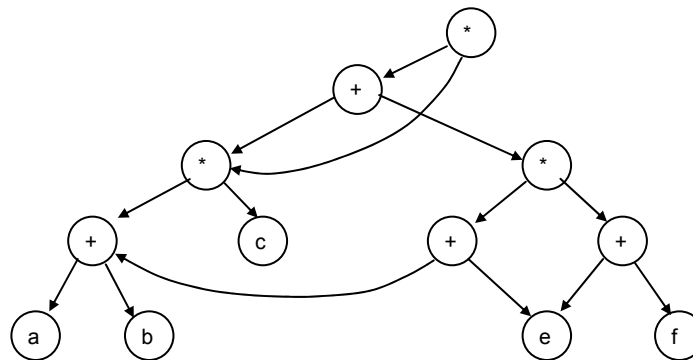


Fig.12.5.b. Graf orientat aciclic reprezentând o expresie aritmetică

- Grafurile orientate aciclice pot fi utilizate cu succes în reprezentarea **relației de ordonare parțială**.
 - Se reamintește faptul că o **relație de ordonare parțială** R pe o mulțime M , este o relație binară care satisface următoarele proprietăți:
 - (1) aRa este falsă pentru $\forall a \in M$ (**nerflexivitate**);
 - (2) Pentru $\forall a, b, c \in M$, dacă aRb și aRc sunt adevărate, atunci aRc este adevărată (**tranzitivitate**);
 - (3) Dacă aRb este adevărată și bRa este adevărată, atunci $a=b$ pentru \forall

$a, b \in M$ (**antisimetrie**).

- Două exemple naturale de relații de ordonare parțială sunt:
 - (1) Relația “mai mic decât” ($<$) definită pe mulțimea numerelor întregi
 - (2) Relația “submulțime proprie a unei mulțimi” (\subset) definită pentru mulțimi de elemente.
- Spre exemplu fie mulțimea $M = \{1, 2, 3\}$ și fie $P(M)$ puterea mulțimii M , adică mulțimea tuturor submulțimilor proprii ale mulțimii M .
 - În acest context $P(M) = \{\{\emptyset\}, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$.
 - Este simplu de verificat că relația “submulțime a mulțimii M ” (\subset) este o relație de ordonare parțială pentru puterea mulțimii M .
- Grafurile orientate aciclice pot fi utilizate în reprezentarea unei astfel de relații.
 - În acest scop o relație R poate fi asimilată cu o mulțime de perechi (arce), care satisfac următoarea proprietate: perechea (a, b) aparține mulțimii dacă aRb este adevărată.
- În aceste condiții, este valabilă următoarea **definiție**: dacă R este o relație de ordonare parțială pe o mulțime M , atunci graful $G = (M, R)$ este un graf ordonat aciclic.
- Reciproc, se presupune că $G = (M, R)$ este un graf ordonat aciclic și că R^+ este o relație definită prin afirmația aR^+b este adevărată dacă și numai dacă există un drum de lungime mai mare sau egală cu 1 care conectează nodul a cu nodul b în graful G .
 - R^+ este de fapt mulțimea relațiilor care materializează închiderea tranzitivă a lui R .
 - În aceste condiții, R^+ reprezintă o relație de ordonare parțială pe M .
- În figura 12.5.c apare graful orientat aciclic $G = (P(M), R)$ unde $M = \{1, 2, 3\}$.
 - Relația R^+ se traduce prin “submulțime proprie” a puterii mulțimii M .

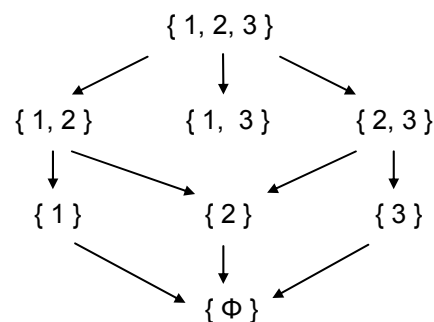


Fig.12.5.c. Graf orientat aciclic reprezentând submulțimile proprii ale unei mulțimi date

12.5.1. Determinarea aciclității unui graf orientat

- Se consideră un graf orientat $G = (N, A)$ și se cere să se stabilească dacă G este aciclic, cu alte cuvinte să se determine dacă G nu conține 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 întâlnește cel puțin un arc de tip “înapoi”, în mod evident graful conține cel puțin un ciclu.
 - Reciproc, dacă un graf orientat conține cel puțin un ciclu, atunci în orice traversare a grafului prin tehnica căutării în adâncime va apărea cel puțin 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 siguranță în ciclu un nod x , al cărui număr de ordine la căutarea în adâncime este mai mic decât numărul corespunzător oricărui nod aparținând ciclului.
 - Se consideră un arc (y, x) din ciclul care-l conține 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 consecință (y, x) **nu** poate fi decât un arc de tip “înapoi”.

12.5.2. Aplicație. Sortarea topologică

- Sortarea topologică a mai făcut obiectul unor prezentări pe parcursul acestei lucrări.
- În paragraful de față se prezintă o altă modalitate de soluționare a sortării topologice utilizând drept structuri de date suport grafurile orientate aciclice.
- Se reamintește faptul că un proiect de mari dimensiuni, este adesea divizat într-o colecție de activități (taskuri) mai mici, fiecare având un caracter mai mult sau mai puțin unitar.
 - În vederea finalizării proiectului, unele dintre activități trebuie realizate într-o anumită ordine specificată.
- Spre exemplu, o programă universală poate conține o serie precizată de cursuri dintre care unele presupun în mod obligatoriu absolvirea în prealabil a altora.

- O astfel de situație 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 evidențiază intercondiționările de această natură impuse unui număr de 5 cursuri.
 - Cursul C3 spre exemplu, presupune absolvirea în prealabil a cursurilor C1 și C2.

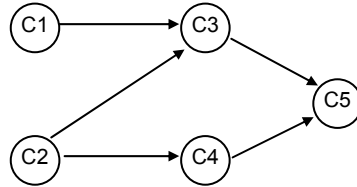


Fig.12.5.1.a. Graf orientat aciclic modelând intercondiționări

- Sortarea topologică realizează o astfel de ordonare liniară a activităților proiectului încât niciuna dintre intercondiționările impuse să **nu** fie violate.
 - 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 secvența [12.4.1.a] se adaugă după linia [5] o instrucțiune de tipărire a lui x , se obține o procedură care afișează nodurile grafului aciclic sortate topologic în ordine inversă (secvența [12.5.1.a]).
 - Acest lucru se întâmplă deoarece procedura `SortTopologic` afișează un nod oarecare x al grafului aciclic după ce a terminat explorarea (afișarea) tuturor descendenților săi.
- Se impune în acest moment următoarea precizare: realizarea sortării topologice în ordine inversă pentru un graf, este echivalentă cu realizarea unei sortări topologice, normale în graful obținut prin inversarea sensului arcelor [Se 88].
- Este de asemenea evident faptul că de regulă ordonarea produsă de acest tip de sortare **nu** este unică.

PROCEDURE SortTopologic(x: Nod);

```
{Afișează nodurile accesibile din x, în ordine topologică  
inversă}
```

```
VAR k: Nod;
```

[12.5.1.a]

```
BEGIN
```

```
  marc[x]:= vizitat;
```

```
  FOR fiecare nod k din L(x) DO
```

```
    IF marc[k] = nevizitat THEN
```

```
      SortTopologic(k);
```

```
    writeln(x)
```

```
  END; {SortTopologic}
```

-
- Tehnica utilizată este perfect funcțională deoarece la parcurgerea unui graf orientat aciclic **nu** apar arce de tip “înapoi”.
 - Se consideră momentul la care căutarea în adâncime părăsește nodul x .
 - Toate arcele care apar în pădurea de arbori de căutare în adâncime asociată grafului și care provin din nodul x , sunt sau arce “de arbore”, sau arce de tip “înainte”, sau arce de “trecere”.
 - Toate aceste arce sunt însă direcționate spre noduri a căror vizită s-a încheiat și în consecință, evidențierea lor precede evidențierea lui x în cadrul ordinii care se construiește.

12.6. Componente puternic conectate

- O **componentă puternic conectată** a unui graf orientat este o submulțime de noduri ale grafului în care există un drum de la **oricare** nod al mulțimii la **oricare** alt nod aparținând aceleiași mulțimi.
- Fie $G = (N, A)$ un graf orientat.
- Se partiționează mulțimea N a nodurilor grafului G în clase de echivalență $N_i, (1 \leq i \leq r)$, pe baza următoarei definiții a relației de echivalență:
 - Nodurile x și y sunt **echivalente** dacă și numai dacă există un drum de la nodul x la nodul y și un drum de la nodul y la nodul x .
- Fie $A_i, (1 \leq i \leq r)$ mulțimea arcelor ale căror surse și destinații aparțin mulțimii 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 numește **puternic conectat**.
- În figura 12.6.a apare un graf orientat (a), care conține două componente puternice (b).

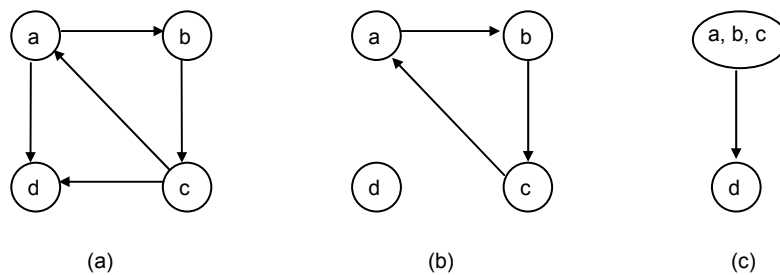


Fig.12.6.a. Componente puternic conectate și graful redus al unui graf orientat

- Se face precizarea că fiecare nod al unui graf orientat aparține unei componente puternice dar există arce care **nu** aparțin nici unei componente.
 - Astfel de arce se numesc **arce de “transfer”** și ele conectează noduri aparținând la componente puternice diferite.
- Considerând componentele puternice drept noduri și reprezentând interconexiunile dintre componentele puternice ale unui graf G , respectiv reprezentând arcele de transfer, se obține **graful redus** al grafului G .
 - Nodurile grafului redus sunt componentele puternice ale grafului original.
 - În graful redus apare un arc de la componenta C_i la componenta C_j dacă în graful origine G există un arc ce leagă un nod aparținând lui C_i cu un nod aparținând lui C_j .
 - Graful redus este întotdeauna un **graf orientat aciclic**, deoarece dacă ar conține vreun ciclu, atunci toate componentele din acel ciclu aparțin unei aceleiași componente puternic conectate, element ce ar evidenția 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 bazați pe tehnica căutării în adâncime.

12.6.1. Algoritmul lui Kosaraju-Sharir

- Algoritmul pentru determinarea componentelor puternic conectate ale unui graf orientat G care va fi prezentat în continuare, a fost sugerat în anul 1978 de R.Kosaraju (nepublicat) și publicat în anul 1981 de către Sharir, motiv pentru care va fi denumit **algoritmul Kosaraju-Sharir**.
- Algoritmul Kosaraju-Sharir constă din următorii pași:
 - (1) Se realizează o traversare prin căutare în adâncime a grafului G și 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 secvența [12.4.1.a].

- (2) Se construiește un nou graf orientat G_r , inversând sensul tuturor arcelor grafului original G .
- (3) Se realizează o traversare prin căutare în adâncime în graful G_r , începând cu nodul care are numărul cel mai mare conform numerotării de la pasul 1.
 - Dacă această traversare **nu** acoperă toate nodurile, se lansează următoarea traversare începând cu nodul care are numărul cel mai mare dintre nodurile neparcuse încă.
 - Se continuă în aceeași manieră până la epuizarea tuturor nodurilor.
- (4) 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).

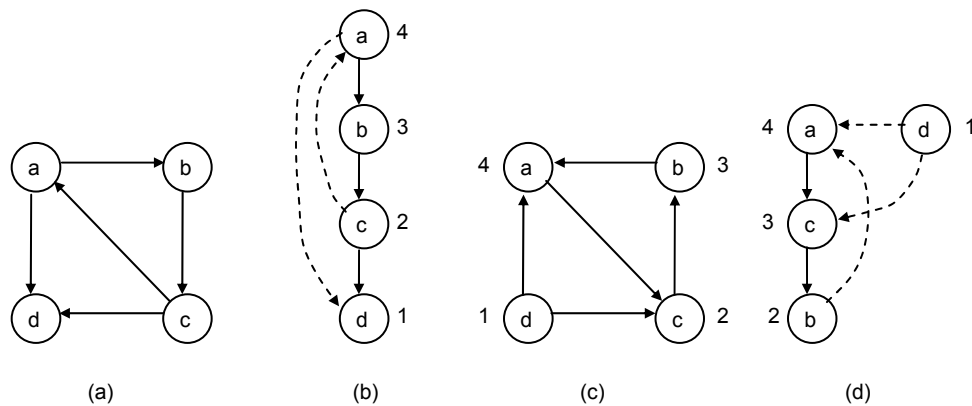


Fig.12.6.1.a. Determinarea componentelor puternic conectate ale unui graf orientat

- Astfel, după traversarea grafului începând cu nodul a și continuând cu b , se obține arborele de căutare în adâncime și numerotarea nodurilor din fig. 12.6.1.a. (b).
- Inversând sensurile arcelor grafului original, rezultă graful G_r (fig.12.6.1.a. (c)).
- În continuare, realizând o traversare prin căutare în adâncime a lui G_r rezultă pădurea din aceeași 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 și la nodul b .
 - Următorul arbore de căutare în adâncime are rădăcina d , deoarece d este nodul cu numărul cel mai mare dintre cele rămase neparcuse.
- Fiecare arbore al acestei păduri formează o componentă puternic conectată a grafului orientat original G .

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** și o variantă de implementare a sa apare în secvența [12.6.2.a] în forma funcției Tarjan.
- În legătură cu această secvență se fac câteva precizări.
 - (1) Graficul se consideră reprezentat printr-o structură de adiacențe implementată cu ajutorul listelor înlănțuite simple.
 - (2) Funcția 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 primește ca și parametru.
 - Acest lucru este realizat într-o manieră similară funcției care determină componentele biconexe ale unui graf (secvența [10.5.2.3.a], capitolul 10).

```
FUNCTION Tarjan(k: integer): integer;
```

```
{Determinarea componentelor puternice }
```

```
VAR t: legatura;  
      m,min: integer;
```

```
BEGIN
```

```
  id:= id + 1; marc[k]:= id; min:= id;
```

```
  Stiva[p]:= k; p:= p + 1;
```

```
  t:= Stradj[k];
```

```
  WHILE t <> z DO
```

```
    BEGIN [12.6.2.a]
```

```
      IF marc[t^.nume] = 0 THEN
```

```
        m:= Tarjan(t^.nume);
```

```
      ELSE
```

```
        m:= marc[t^.nume];
```

```
      IF m < min THEN
```

```
        min:= m;
```

```
      t:= t^.urm
```

```
    END;
```

```
  IF min = marc[k] THEN
```

```
    BEGIN
```

```
      REPEAT
```

```
        p:= p - 1; write(nume(Stiva[p]));
```

```
        marc[Stiva[p]] = N+1
```

```
      UNTIL Stiva[p] = k;
```

```
      writeln
```

```
    END;
```

```
  Tarjan:= min;
```

```
END; {Tarjan}
```

- În plus însă, valoarea \min determinată, este utilizată și la evidențierea 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 nodurilor pentru care este apelată funcția `Tarjan` imediat după intrarea în apel, acestea urmând a fi afișate prin extragere din stivă după vizitarea ultimului membru al componenteii puternic conectate determinate.
- Elementul esențial al determinării este verificarea $\min = \text{marc}[k]$ efectuată la terminarea apelurilor recursive;
 - Dacă testul conduce la valoarea de adevăr, se afișează toate nodurile din stivă întrucât ele aparțin aceleiași componenteii puternic conectate ca și nodul k .
- Acest algoritm poate fi însă adaptat să realizeze prelucrări mult mai sofisticate decât simpla afișare a componentelor puternic conectate.
- Fiind bazat pe tehnica căutării în adâncime, în grafuri reprezentate prin structuri de adiacențe, algoritmul lui Tarjan obține o **performanță** de ordinul $O(n+a)$.
- Interpretarea execuției algoritmului lui Tarjan este următoarea.
 - (1) Un nod x care nu are descendenți 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 pornește un arc de tip "înapoi" spre x și nu are nici un 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 toți descendenții săi (cu excepția acelor noduri care satisfac proprietatea 1^o, a nodurilor care satisfac proprietatea 2^o și a descendenților lor), determină o componentă puternic conectată.
 - (3) Fiecare descendent y al nodului x care nu satisface nici una din cele două situații mai sus precizate, are descendenți 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 strămoș al lui y și apoi continuând în aceeași manieră, se ajunge până la x .
 - Rezultă că nodul y aparține aceleiași componenteii puternic conectate ca și nodul x .
- Aceste situații pot fi urmărite în figura 12.6.2.a care reprezintă pădurea de arbori de

căutare în adâncime (b) corespunzătoare grafului orientat (a).

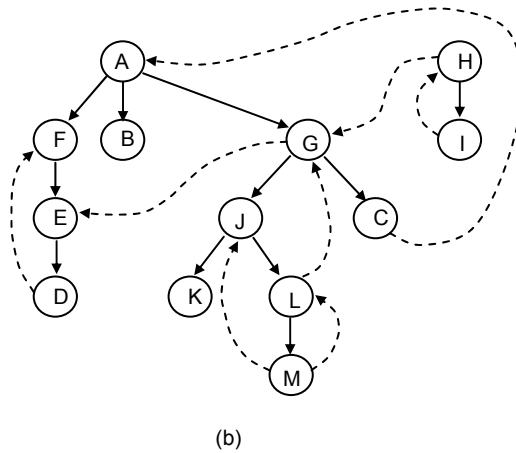
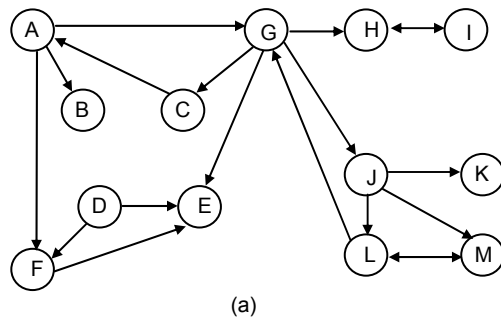


Fig.12.6.2.a. Graf orientat (a) și pădure de arbori de căutare în adâncime asociată (b)

- Nodurile B și K satisfac prima situație, astfel încât se constituie ele însele în componente puternice ale grafului.
- Nodurile F (reprezentând pe F, E și D), H (reprezentând pe H și I) și A (reprezentând pe A, G, J, L, M și C) satisfac a doua situație.
 - Membrii componentei evidențiate de A au fost determinați după eliminarea nodurilor B, K și F și a descendenților lor care apar în componentele puternice stabilite anterior.
- Restul nodurilor se încadrează în situația a treia.
- O subliniere foarte importantă este aceea ca odată un nod parcurs, el primește în tabloul `marc` o valoare mare ($N+1$), astfel încât arcele de “trecere” spre aceste noduri sunt ignorate.