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.
        - 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 restricţii) 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

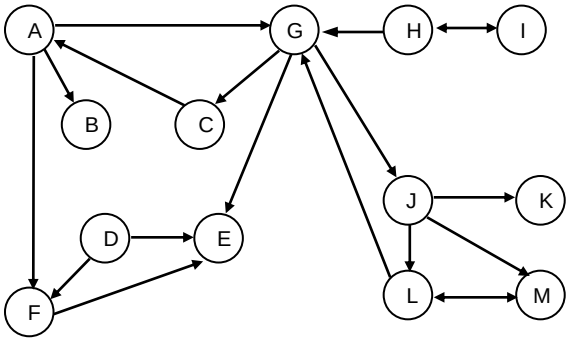


Fig.12.1.a. Exemplu de graf orientat.

- Graful constă din arcele `AG,AB,CA,LM,JM,JL,JK,ED,DF,HI,FE,AF,GE,GC,HG,GJ,LG,IH,ML`.

- **Ordinea** în care nodurile apar la specificarea arcelor este **semnificativă** întucât precizează **sursa** respectiv **destinaţia** arcului.

    - 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`).

- 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.

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 orice semnificaţie fizică: un cost, o valoare, o lungime, un timp, etc.
- Intuitiv, graful `G` poate fi asimilat 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.
        - De exemplu se zboară cu companii diferite sau cu avioane diferite.
    - 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].

```
-------------------------------------------------------------
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 {iniţializarea lui D}
[3]   D[i]:= COST[1,i];                          [12.1.1.a]
[4] FOR i:= 1 TO n-1 DO
      BEGIN
[5]     *alege un nod x aparţinând mulţimii 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] + 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,…,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.

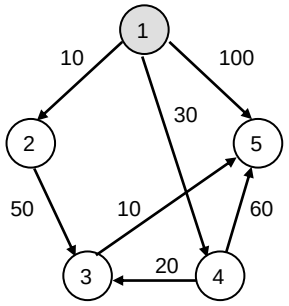
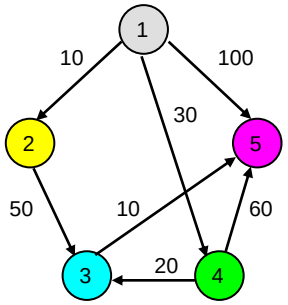


Fig.12.1.1.a. Graf orientat ponderat

- Iniţial `M={1}`, `D[2]=10`, `D[3]=∞`, `D[4]=30` şi `D[5]=100`.
- În prima iteraţie a buclei **`FOR`** 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 **`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

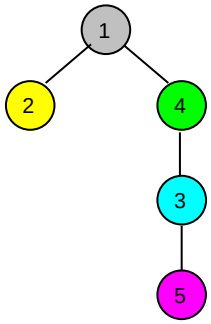


Fig.12.1.1.b. Determinarea drumurilor minime cu origine unică în baza algoritmului lui Dijkstra

- În vederea reconstrucţiei 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 părintele 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.

12.1.3. 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.

```
-------------------------------------------------------------
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 {iniţializarea lui D}
[3]   D[i]:= COST[1,i];                          [12.1.1.a]
[4] FOR i:= 1 TO n-1 DO
      BEGIN
[5]     *alege un nod x aparţinând mulţimii 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] + COST[x,y])
      END
  END; {Dijkstra}
-------------------------------------------------------------
```

- (1) Dacă pentru reprezentarea grafului se utilizează o **matrice de adiacenţe**, atunci bucla **FOR** din liniile `[7]-[8]` necesită un efort de calcul proporţional cu O*(n)*.

  - Întrucât această buclă este executată de `n-1` ori, (bucla **FOR** 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 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 n)*, astfel încât timpul total consumat în liniile `[7]-[8]` va fi proporţional cu O*(a log n)* şi nu cu O*(n^2)*.

- În mod evident liniile `[2]-[3]` necesită asemeni liniilor `[4]-[6]` un efort de calcul proporţional cu O*(n)*.

- Utilizând în reprezentarea mulţimii `N-M` aceeaşi **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 n)* unităţi de timp.

- În consecinţă, **timpul total** consumat de această variantă a algoritmului lui Dijkstra este limitat la O*(a log 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 `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 **distanţa 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 ponderat G=(N,A) are 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.
    - Iniţial se face A[i,j]=COST[i,j] pentru toţi $i \neq j$.
        - Dacă **nu** există niciun arc de la nodul i la nodul j, se presupune că COST[i,j]=∞.
    - Elementele diagonalei matricei A se pun toate pe 0.
- **Algoritmul lui Floyd** execută n iteraţii asupra matricii A.
    - După cea de-a k-a iteraţie, locaţia 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 are numărul mai mic sau cel mult egal cu k.
    - În cea de-a k iteraţie se utilizează următoarea formulă în calculul lui A (formula [12.2.1.a]):

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

    - 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 sugerată de figura 12.2.1.a.

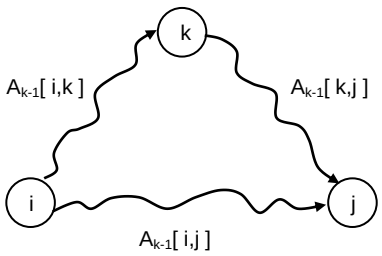


Fig.12.2.1.a. Selecţia drumului minim de la nodul `i` la nodul `j` la momentul `k`

- Pentru calculul lui $A_k[i,j]$ 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`, 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`.

- 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.

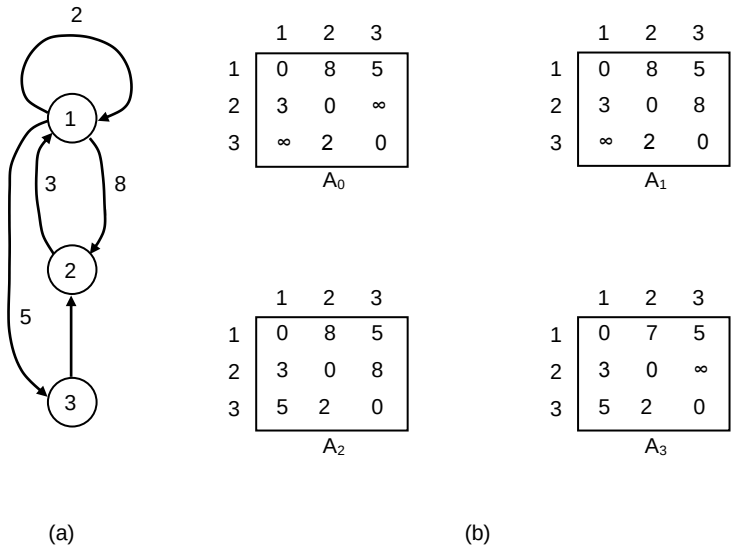


Fig.12.2.1.b. Calculul lungimii drumurilor minime corespunzătoare tuturor perechilor de noduri ale unui graf utilizând algoritmul lui Floyd

- **Observaţie importantă:**

  - 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(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 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 **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 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 drept consecinţă grafurile sunt reprezentate prin **liste 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 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 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 cele mai 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 `Drum`.

  - În matricea `Drum`, o locaţie `Drum[i,j]` memorează acel 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].

```
-------------------------------------------------------------
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 primeşte matricea COST şi determină matricea A
care memorează lungimile drumurilor minime şi 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 colecţie de **arbori corespunzători drumurilor minime**, respectiv câte un arbore pentru fiecare nod al grafului original.

- 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]).

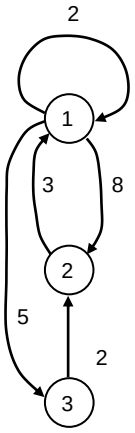
```
PROCEDURE Traseu(i,j: integer);
{Afişeaza traseul drumului minim de la nodul i la nodul j}

  VAR k: integer;

  BEGIN                                                [12.2.3.b]
    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 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).



(a)

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

matricea Drum

(b)

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`.

- **Determinarea centrului unui graf** `G` se poate realiza simplu utilizând **algoritmul lui Floyd**

- 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 **excentricităţile nodurilor** `i`,`(1≤i≤N)` găsind valoarea maximă de pe fiecare coloană `i` a matricii.

- (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.

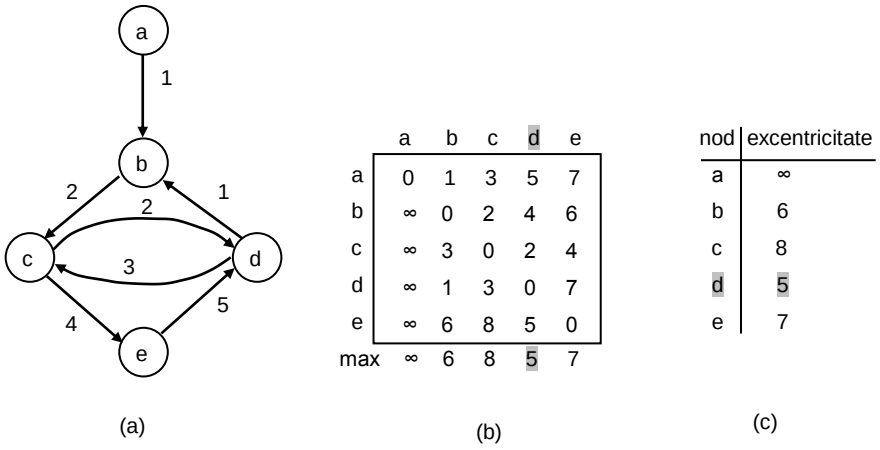


Fig.12.2.4.a. Determinarea centrului unui graf ponderat

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.

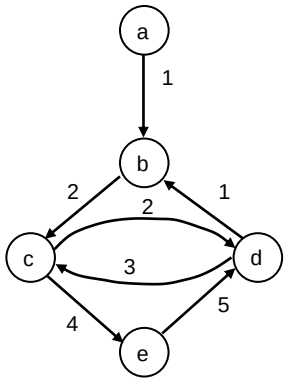
- Acest lucru este posibil în baza următoarei **interpretări** sugerate de **Warshall**:

    - "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).



(a)

	1	2	3
1	1	1	1
2	1	1	1
3	1	1	1

matricea T

(b)

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 $T_k[k,j]=T_{k-1}[k,j]$, fapt ce probează **invarianţa** lui `T` în raport cu valoarea curentă a lui `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].

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

{Procedura construieşte 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, performanţa algoritmului rezultă imediat, $O(n^3)$.

- 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 **algoritmului recursiv 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]).

```
-------------------------------------------------------------
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 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 iniţial este nodul `a (x=a)`.

    - Graful se consideră reprezentat printr-o structură de adiacenţe sugerată în aceeaşi figura (b).

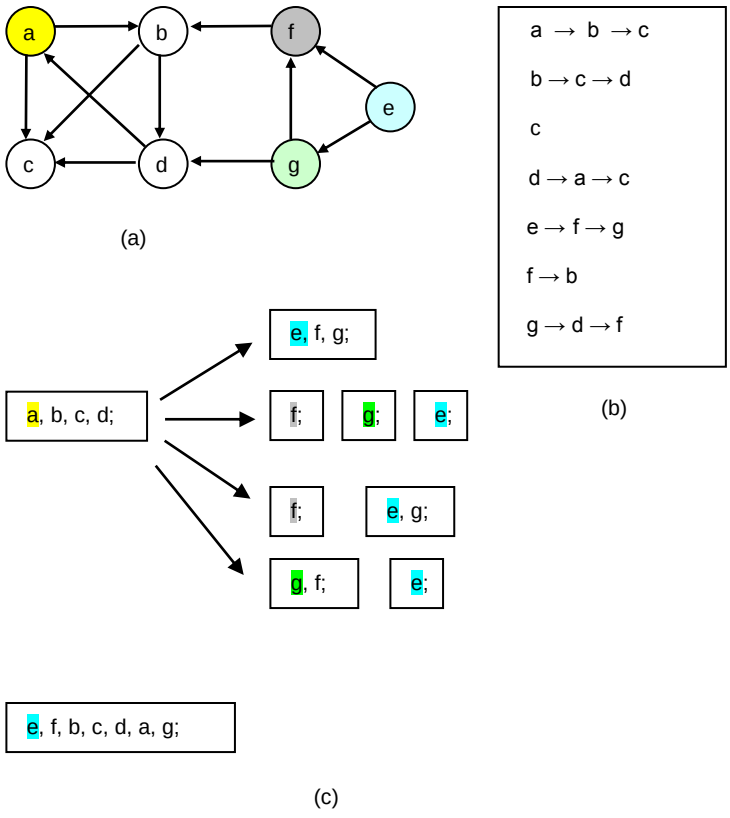


Fig.12.4.1.a. Traversarea unui graf orientat

- 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 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 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 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 conluziona 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 prin căutare în adâncime** 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).

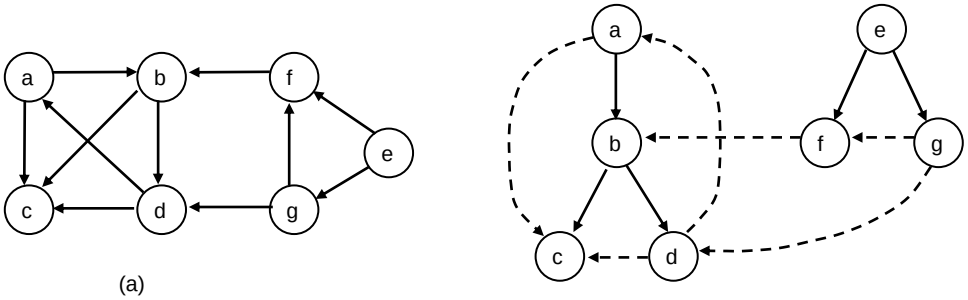


Fig.12.4.2.a. Pădure de arbori de căutare în adâncime pentru graful din fig. 12.4.1.a. (a)

- 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 care conectează un nod cu unul din **strămoşii săi** din cadrul arborelui de căutare, se numeşte **arc de tip** "**înapoi**", spre exemplu arcul `(d,a)`.
  - 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 care conectează două noduri care **nu sunt în relaţie descendent sau strămoş** se numeşte **arc** "**de trecere**", spre exemplu arcele `(d,c)` sau `(g,d)`.
  - 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 ale arborilor de căutare în adâncime sunt adăugate 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].

```
-------------------------------------------------------------
        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 faptul că această esenţă constă în transformarea structurii supusă traversării într-o structură de 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`.

- De fapt `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 **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.

- 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 această direcţionare este însă luată în considerare 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).

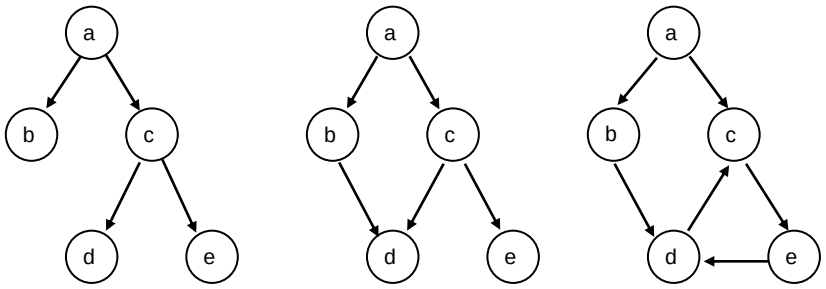


Fig.12.5.a. Tipuri de grafuri orientate

- Printre altele, **grafurile orientate aciclice** sunt utile în reprezentarea structurii sintetice a **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.

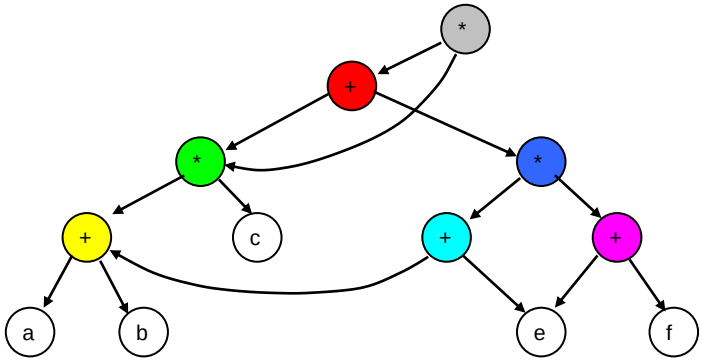


Fig.12.5.b. Graf orientat aciclic reprezentând o expresie aritmetică

- **Grafurile orientate aciclice** pot fi utilizate cu succes în reprezentarea **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 aRc 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 o **mulţime 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 unei astfel de **relaţii de ordonare parţială.**
    - În acest, scop **o relaţie** R poate fi asimilată cu **o mulţime de perechi orientate** (arce), care satisfac următoarea **proprietate**: perechea (a,b) aparţine mulţimii R dacă aRb 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 orientatat aciclic**.

- **Reciproc**, se presupune că G=(M,R) este un **graf ordonat aciclic** şi că R^+ este o relaţie definită prin afirmaţia aR^+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, R^+ reprezintă o **relaţie de ordonare parţială** pe M.
    - **Observaţie:** 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 R^+ se traduce prin "submulţime proprie" a puterii mulţimii M.

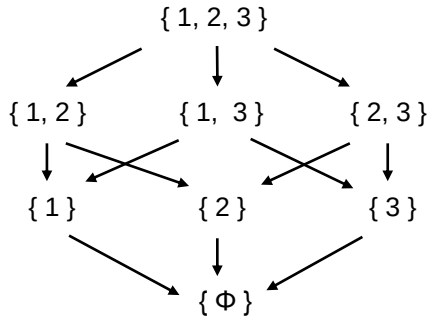


Fig.12.5.c. Graf orientat aciclic reprezentând submulţimile proprii ale unei mulţimi date

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**".

12.5.2. Aplicaţie. Sortarea topologică

- **Sortarea topologică** a mai făcut obiectul unor prezentări pe parcursul acestui curs.

- 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 (taskuri) mai mici, fiecare având un caracter mai mult sau mai puţin unitar.

    - În vederea finalizării corecte a 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.

    - 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`.

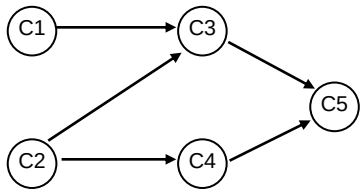


Fig.12.5.1.a. Graf orientat aciclic modelând intercondiţionări

- **Sortarea topologică** realizează o astfel de **ordonare liniară a activităţilor** proiectului încât niciuna dintre intercondiţionările impuse să **nu** fie 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 elementelor 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 după linia `[5]` o instrucţiune de afişare a lui `x`, se obţine o procedură 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.

  - Situaţia poate fi remediată simplu: se realizează o sortare topologică a grafului obţinut prin **inversarea sensului arcelor** grafului original [Se 88].

- Este de asemenea evident faptul că de regulă, **ordonarea** produsă de acest tip de sortare **nu** este unică.

```
-------------------------------------------------------------
PROCEDURE SortTopologic(x: Nod);
  {Afişează 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 funcţională deoarece la parcurgerea unui graf orientat aciclic **nu** apar arce de tip "înapoi".

    - 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 aceste 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).

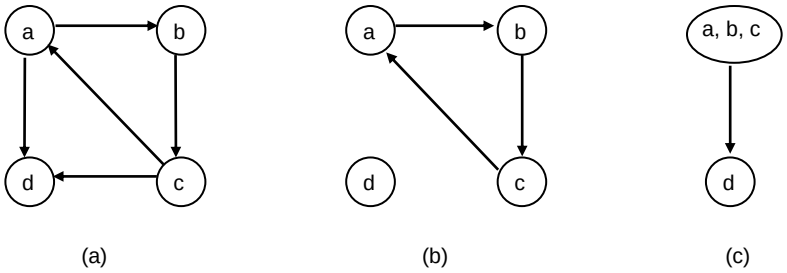


Fig.12.6.a. Componente puternic conectate şi graful redus al unui graf orientat

- 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 uni 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 C_i la componenta C_j dacă în graful origine `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` care va fi prezentat în continuare, a fost sugerat în anul 1978 de R.Kosaraju (nepublicat) şi publicat în anul 1981 de către Sharir, motiv pentru care va fi 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.

    - Dacă această traversare **nu** acoperă toate nodurile, se lansează următoarea traversare începând cu nodul care are numărul cel mai mare dintre nodurile neparcurse încă.

    - Se continuă în aceeaşi 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).

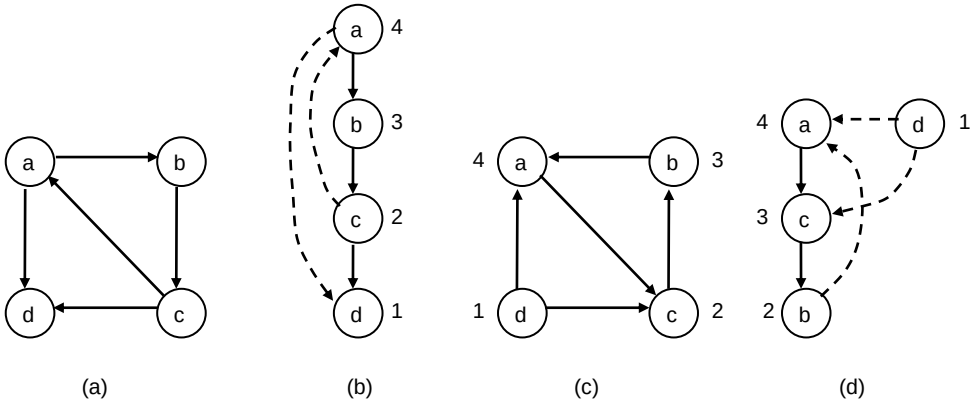


Fig.12.6.1.a. Determinarea componentelor puternic conectate ale unui graf orientat

- Astfel, după traversarea grafului începând cu nodul `a` 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.

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 şi este cunoscută sub denumirea de **algoritmul lui Tarjan**

- 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).

```
-------------------------------------------------------------
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 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 nodurilor pentru care este apelată funcţia **`Tarjan`** imediat după intrarea în apel, acestea urmând a fi afişate prin extragere din stivă după vizitarea ultimului membru al componentei puternic conectate determinate.

- Elementul esenţial al determinării este verificarea `min=marc[k]` efectuată la terminarea apelurilor recursive.

    - Dacă testul `min=marc[k]` conduce la valoarea de adevăr, se afişează toate nodurile din stivă 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)*.

- **Interpretarea** execuţiei 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 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 toţi descendenţii săi, cu excepţia acelor noduri care satisfac proprietatea 1^o, a nodurilor care satisfac proprietatea 2^o şi a descendenţilor acestora, 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).

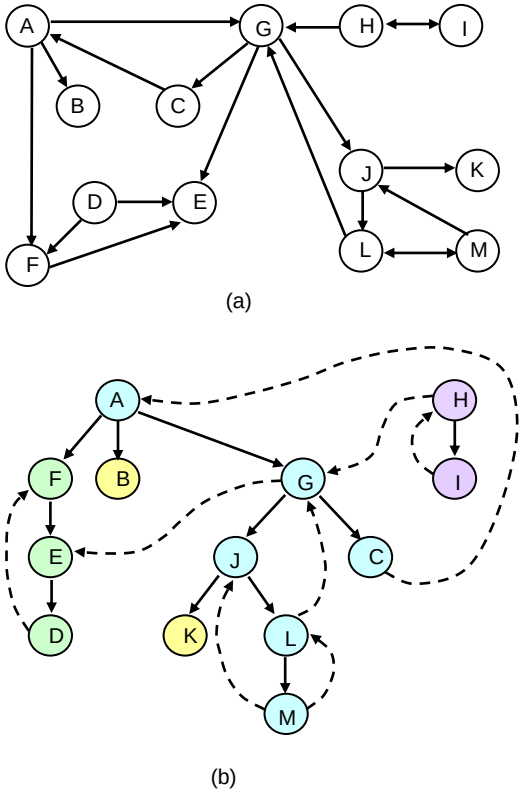


Fig.12.6.2.a. Graf orientat (a) şi pădure de arbori de căutare în adâncime asociată (b)

- 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.

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

- Restul nodurilor se încadrează în situaţia a treia.

- 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.