10. Structura de date graf

- În problemele care apar în programare, matematică, inginerie în general şi în multe alte domenii, apare adeseori necesitatea reprezentării unor relaţii arbitrare între diferite obiecte, respectiv a interconexiunilor dintre acestea.

- Spre exemplu, dându-se **traseele aeriene ale unui stat** se cere să se precizeze drumul optim dintre două oraşe.

  - Criteriul de optimalitate poate fi spre exemplu timpul sau preţul, drumul optim putând să difere pentru cele două situaţii.

- **Circuitele electrice** sunt alte exemple evidente în care interconexiunile dintre obiecte joacă un rol central.

  - Piesele (tranzistoare, rezistenţe, condensatoare) sunt interconectate prin fire electrice.

  - Astfel de circuite pot fi reprezentate şi prelucrate de către un sistem de calcul în scopul rezolvării unor probleme simple cum ar fi: "Sunt toate piesele date conectate în acelaşi circuit?" sau a unor probleme mai complicate cum ar fi: "Este funcţional în anumit circuit electric?".

- Un al treilea exemplu îl reprezintă **planificarea activităţilor**, în care obiectele sunt task-uri (activităţi, procese) iar interconexiunile precizează care dintre activităţi trebuiesc finalizate înaintea altora.

  - Întrebarea la care trebuie să ofere un răspuns este: "Când trebuie planificată fiecare activitate?".

- Structurile de date care pot modela în mod natural situaţii de natura celor mai sus prezentate sunt cele derivate din conceptul matematic cunoscut sub denumirea de **graf**.

- Teoria grafurilor este o ramură majoră a matematicii combinatorii care în timp a fost şi este încă intens studiată.

  - Multe din proprietăţile importante şi utile ale grafurilor au fost demonstrate, altele cu un grad sporit de dificultate îşi aşteaptă încă rezolvarea.

- În cadrul capitolului de faţă vor fi prezentate doar câteva din proprietăţile fundamentale ale grafurilor în scopul înţelegerii algoritmilor fundamentali de prelucrare a structurilor de date graf.

  - Ca şi în multe alte domenii, studiul algoritmic al grafurilor respectiv al structurilor de date graf, este de dată relativ recentă astfel încât alături de algoritmii fundamentali cunoscuţi de mai multă vreme, mulţi dintre algoritmii de mare interes au fost descoperiţi în ultimii ani [Se88].

10.1. Definiţii

- Un graf, în cea mai largă accepţiune a termenului, poate fi definit ca fiid o colecţie

de **noduri** şi **arce**.

- Un **nod** este un obiect care poate avea un nume şi eventual alte proprietăţi asociate
- Un **arc** este o conexiune neorientată între două noduri.

- Notând cu `N` mulţimea nodurilor şi cu `A` mulţimea arcelor, un graf `G` poate fi precizat formal prin enunţul `G=(N,A)`.

- În figura 10.1.a. (a),(b) apar două exemple de grafuri.

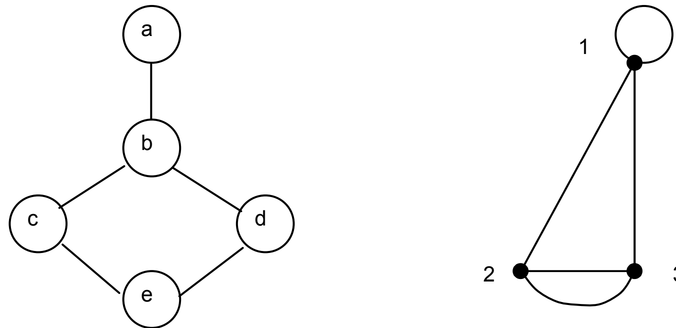


Fig.10.1.a. Exemple de grafuri

- **Ordinul** unui graf este numărul de noduri pe care acesta le conţine şi se notează cu `|G|`.

- Arcele definesc o **relaţie de incidenţă** între perechile de noduri.

- Două noduri conectate printr-un arc se numesc **adiacente**, altfel ele sunt **independente**.

- Dacă `a` este un arc care leagă nodurile `x` şi `y (a~(x,y))`, atunci `a` este **incident** cu x,y.

    - Singura proprietate presupusă pentru această relaţie este simetria [10.1.a]

`(x,y)ЄA =>(y,x)ЄA` unde `A` este mulţimea arcelor. `[10.1.a]`

- Un graf poate fi definit astfel încât să aibă un arc `a~(x,x)`.

    - Un astfel de arc se numeşte **buclă** ("**loop**").

    - Dacă relaţia de incidenţă este reflexivă, atunci fiecare nod conţine o astfel de buclă.

- Există şi posibilitatea ca să existe mai multe arce care conectează aceeaşi pereche de noduri.

    - Într-un astfel de caz se spune că cele două noduri sunt conectate printr-un **arc multiplu**.

- În figura 10.1.a (b) este reprezentat un graf cu buclă şi arc multiplu.

- Grafurile în care nu sunt acceptate arce multiple se numesc grafuri **simple**.

- Numărul de arce incidente unui nod reprezintă **gradul** nodului respectiv.

    - Se numeşte **graf regulat** acel graf în care toate nodurile sunt de acelaşi grad.

- Într-un graf **complet** de ordinul n notat cu K_n, fiecare pereche de noduri este adiacentă.

    - În figura 10.1.b. apar reprezentate grafurile complete până la ordinul 6.

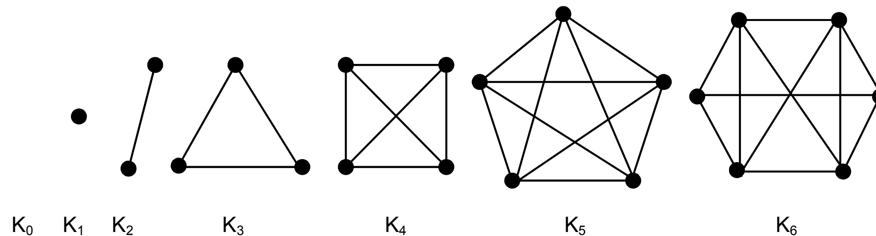


Fig.10.1.b. Exemple de grafuri complete

- Un graf se numeşte **planar** dacă el poate fi astfel reprezentat într-un plan, încât oricare două arce ale sale se intersectează numai în noduri [GG78].

- O teoremă demonstrată de Kuratowski precizează că orice **graf neplanar** conține cel puțin unul din următoarele grafuri de bază:

    1. 5-graful complet (K_5), sau
    2. Graful utilitar în care există două mulțimi de câte trei noduri, fiecare nod fiind conectat nodurile din cealaltă mulțime.

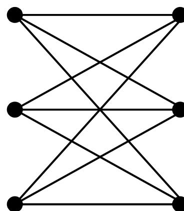


Fig.10.1.c. Graf utilitar

- Un graf se numeşte **bipartit** dacă nodurile sale pot fi partiționate în două mulțimi distincte N_1 şi N_2 astfel încât orice arc al său conectează un nod din N_1 cu un nod din N_2.

    - Spre exemplu graful utilitar este un în acelaşi timp un graf bipartit.

- Fie $G=(N,A)$ un graf cu mulțimea nodurilor N şi cu mulțimea arcelor A. Un **subgraf** al lui G este graful $G' = (N',A')$ unde:

1. `N'`este o submulțime a lui `N`;
2. `A'` constă din arce `(x,y)` ale lui `A`, astfel încât `x` şi `y` aparțin lui `N'`.

- În figura 10.1.d apare un exemplu de graf (a) şi un subgraf al său (b).

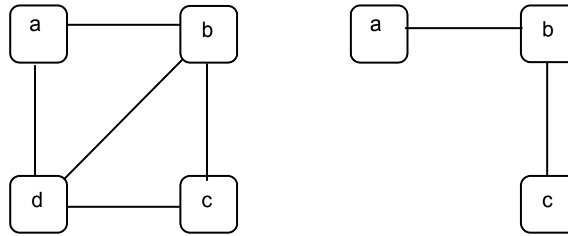


Fig.10.1 d. Graf şi un subgraf al său

- Dacă mulțime de arce `A'` conține toate arcele `(x,y)` ale lui `A` pentru care atât `x` cât şi `y` sunt în `N'`, atunci `G'` se numeşte **subgraf indus** al lui `G` [AH85].

- Un graf poate fi reprezentat în manieră grafică marcând nodurile sale şi trasând linii care materializează arcele.

    - În acelaşi timp însă, un graf poate fi conceput ca şi un tip de date abstracte, independent de o anumită reprezentare.

    - Spre exemplu fig.10.1.e (a) respectiv (b) reprezintă unul şi acelaşi graf.

    - Un graf, poate fi definit spre exemplu precizând doar mulțimea nodurilor şi mulțimea arcelor sale.

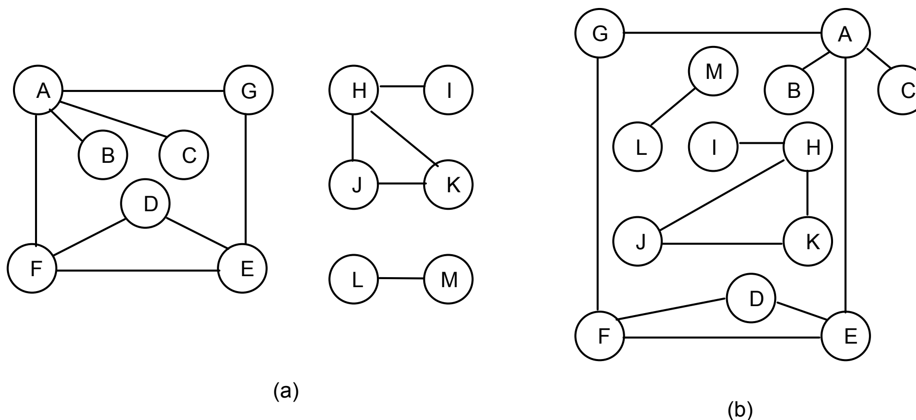


Fig.10.1.e. Reprezentări echivalente ale unui graf

- În anumite aplicații, cum ar fi exemplul cu traseele aeriene, poziția nodurilor (oraşelor) este precizată fizic prin amplasarea lor pe harta reală a statului, rearanjarea structurii fiind lipsită de sens.

- În alte aplicații însă, cum ar fi planificarea activităților, sunt importante nodurile şi arcele ca atare independent de dispunerea lor geometrică.

- În cadrul capitolului de față vor fi abordați algoritmi generali, care prelucrează colecții de noduri şi arce, făcând abstracție de dispunerea lor geometrică, cu alte cuvinte făcând abstracție de topologia grafurilor.

- Se numeşte **drum** ("**path**") de la nodul `x` la nodul `y`, o secvență de noduri n_1`,n2,...,nj` în care nodurile succesive sunt conectate prin arce aparținând grafului.

    - **Lungimea** unui drum este egală cu numărul de arce care compun drumul.

    - La limită, un singur nod precizează un drum la el însuşi de lungime zero.

- Un drum se numeşte **simplu** dacă toate nodurile sale, exceptând eventual primul şi ultimul sunt distincte.

- Un **ciclu** (**buclă**) este un drum simplu de lungime cel puțin 1, care începe şi se sfârşeşte în acelaşi nod.

- Dacă există un drum de la nodul `x` la nodul `y` se spune că acel drum **conectează** cele două noduri, respectiv nodurile `x` şi `y` sunt **conectate**.

- Un graf se numeşte **conex**, dacă de la fiecare nod al său există un drum spre oricare alt nod al grafului, respectiv dacă oricare pereche de noduri aparținând grafului este conectată.

    - Intuitiv, dacă nodurile se consideră obiecte fizice, iar conexiunile fire care le leagă, atunci un graf conex rămâne unitar, indiferent de care nod ar fi "suspendat în aer".

- Un graf care nu este conex este format din **componente conexe.**

    - Spre exemplu**,** graful din fig.10.1.a este format din trei componente conexe.

- O componentă conexă a unui graf G este de fapt un **subgraf indus maximal conectat** al său [AH 85].

- Un graf se numeşte **ciclic** dacă conține cel puțin un ciclu.

    - Un ciclu care include toate arcele grafului o singură dată se numeşte **ciclu eulerian** (**hamiltonian**).

    - Este uşor de observat că un asemenea ciclu există numai dacă graful este conex şi gradul fiecărui nod este par.

- Un **graf conex aciclic** se mai numeşte şi **arbore liber**.

    - În fig.10.1.f apare un graf constând din două componente conexe în care fiecare componentă conexă este un arbore liber.

    - Se remarcă în acest sens, observația ca arborii sunt de fapt cazuri particulare ale grafurilor.

    - Un grup de arbori neconectați formează o **pădure** ("**forest**").

* Un **arbore de acoperire** ("**spanning tree**") al unui graf, este un subgraf care conține toate nodurile grafului inițial, dar dintre conexiuni numai atâtea câte sunt necesare formării unui arbore.

    * Se face precizarea că termenul de "acoperire" în acest context are sensul termenului "**cuprindere**".

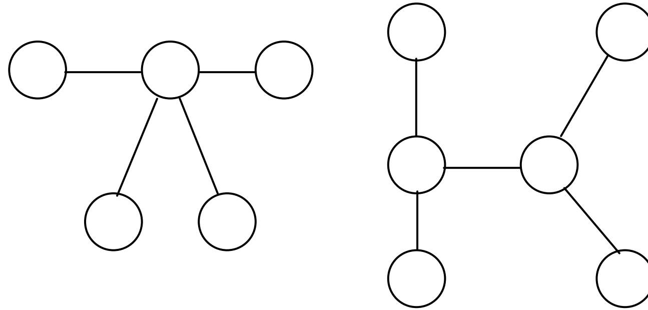


Fig.10.1.f. Graf aciclic format din două componente conexe

* În figura 10.1.g este prezentat un graf (a) şi un arbore de acoperire al grafului (b).

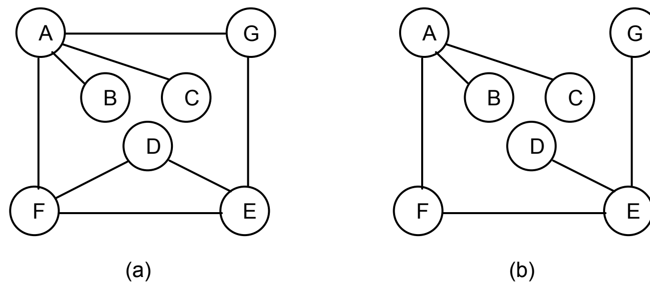


Fig.10.1.g. Graf şi un arbore de acoperire al grafului

* Un arbore liber poate fi transformat într-un arbore ordinar dacă se "suspendă" arborele de un nod considerat drept rădăcină şi se orientează arcele spre rădăcină.

* Arborii liberi au două proprietăți importante:

    1. Orice arbore liber cu `n` noduri conține exact `n-1` arce (câte un arc la fiecare nod, mai puțin rădăcina);

    2. Dacă unui arbore liber i se adaugă un arc el devine obligatoriu un graf ciclic.

* De aici rezultă două consecințe importante şi anume:

    1. Un graf cu `n` noduri şi mai puțin de `n-1` arce nu poate fi conex;

    2. Pot exista grafuri cu `n` noduri şi `n-1` arce care nu sunt arbori liberi. (Spre exemplu dacă au mai multe componente conexe).

* Notând cu `n` numărul de noduri ale unui graf şi cu `a` numărul de arce, atunci `a` poate lua orice valoare între `0` şi `(1/2)n(n-1)`.

- Graful care conține toate arcele posibile este graful complet de ordinul n (K_n).

- Graful care are relativ puține arce (spre exemplu $a<n\log_2 n$) se numeşte **graf rar** (“**sparse**”)

- Graful cu un număr de arce apropiat de graful complet se numeşte **graf dens**.

- Dependența fundamentală a topologiei unui graf de doi parametri (n şi a), face ca studiul comparativ al algoritmilor utilizați în prelucrarea grafurilor să devină mai complicat din cauza posibilităților multiple care pot să apară.

  - Astfel, presupunând că un algoritm de prelucrare a unui graf necesită un efort de calcul de ordinul $O(n^2)$ în timp ce un alt algoritm care rezolvă aceeaşi problemă necesită un efort de ordinul $O((n+a)\log_2 n)$ paşi, atunci, în cazul unui graf cu n noduri şi a arce, este de preferat primul algoritm dacă graful este dens, respectiv al doilea dacă graful este rar.

- Grafurile prezentate până în prezent se numesc şi **grafuri neorientate** şi ele reprezintă cea mai simplă categorie de grafuri.

- Prin asocierea de informații suplimentare nodurilor şi arcelor, se pot obține categorii de grafuri mai complicate.

- Astfel, într-un **graf ponderat** (“**weighted graph**”), fiecărui arc i se asociază o valoare (de regulă întreagă) numită **pondere** care poate reprezenta spre exemplu o distanță sau un cost.

- În cadrul **grafurilor orientate** (“**directed graphs**”), arcele sunt orientate, având un sens precizat, de la x la y spre exemplu.

  - In acest caz x se numeşte coada sau sursa arcului iar y vârful sau destinația sa.

  - Pentru reprezentarea arcelor orientate se utilizează săgeți sau segmente direcționate (fig.10.1.h. (a), (b),(c)).

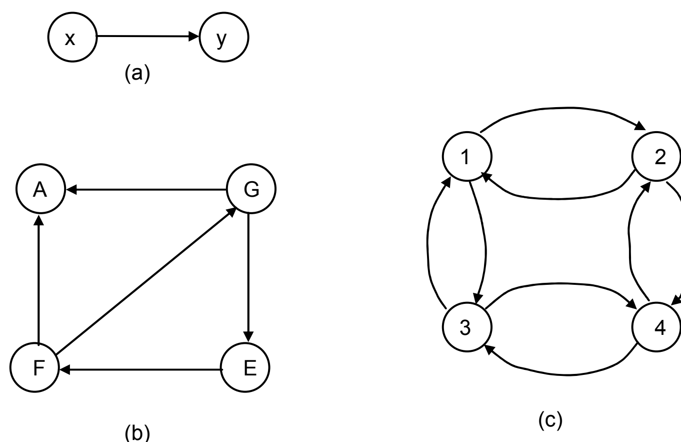


Fig.10.1.h. Grafuri orientate

- Grafurile orientate ponderate se mai numesc şi **rețele** (“**networks**”).

- Informațiile suplimentare referitoare la noduri și arce nuanțează și în același timp complică manipularea grafurilor care le conțin.

10.2. Tipul de date abstract graf

- Pentru a defini **tipul de date abstract** (TDA) **graf** este necesară:
    - (1) Precizarea modelului matematic care conturează conceptul de graf prezentat în paragraful anterior
    - (2) Precizarea operațiilor definite pe acest model. În cele ce urmează se prezintă două variante, una extinsă și alta redusă, de seturi de operatori aferenți unui TDA graf.

10.2.1. TDA graf. Varianta 1 (Shiflet)

- În cadrul variantei Shiflet [Sh90] un graf este considerat ca și o structură de noduri și arce.
    - Fiecare nod are o cheie care identifică în mod univoc nodul.
    - Modelul matematic, notațiile utilizate și setul de operatori preconizat pentru această variantă apar în [10.2.1.a]:

```
------------------------------------------------------------
TDA Graf    (Varianta 1 - Shiflet)                [10.2.1.a]

Modelul matematic: graful definit în sens matematic.

Notații:
     TipGraf      - tipul de date abstract graf;
     TipElement   - tipul asociat porțiunii element a nodului
     TipCheie     - tipul asociat porțiunii cheie a unui
                  element
     TipInfo      - tipul corespunzator porțiunii de
                   informație a unui element
     TipIndicNod - tip referință la structura unui nod
     TipIndicArc - tip referință la structura unui arc

Operatori:

  1. InitGraf(g: TipGraf); - procedură care creează graful
     vid g;

  2. GrafVid(g: TipGraf):boolean; - operator care returnează
     true dacă graful este vid respectiv false în caz
     contrar;

  3. GrafPlin(g: TipGraf):boolean; - operator boolean care
     returnează true dacă graful este plin. Se precizează
     faptul că această funcție este legată direct de maniera
     de implementare a grafului. Valoarea ei adevărată
     presupune faptul că zona de memorie alocată structurii
     a fost epuizată și în consecință nu se mai pot adăuga
     noi noduri.
```

4. ***CheieElemGraf****(g: TipGraf, e: TipElement): TipCheie;* - operator care returnează cheia elementului *e* aparţinând grafului *g*.

5. ***CautaCheieGraf****(g: TipGraf; k: TipCheie);* - operator boolean care returnează valoarea adevărat dacă cheia *k* este găsită în graful *g*.

6. ***IndicaNod****(g: TipGraf; k: TipCheie; var indicNod: TipIndicNod);* - operator care face ca *IndicNod* să indice acel nod din *g* care are cheia *k*, presupunând că un astfel de nod există.

7. ***IndicaArc****(g: TipGraf; k1,k2: TipCheie;* **var** *indicArc: TipIndicArc);*- operator care face ca *indicArc* să indice arcul care conectează nodurile cu cheile k_1 şi k_2 din graful *g*, presupunând că arcul există. În caz contrar *indicArc* ia valoarea indicatorului vid.

8. ***ArcVid****(g: TipGraf; indicArc: TipIndicArc):boolean;* - operator boolean care returnează valoarea adevărat dacă arcul indicat de *indicArc* este vid.

9. ***InserNod****(***var*** g: TipGraf; e: TipElement);* - operator care inserează un nod *e* în graful *g* ca un nod izolat (fără conexiuni). Se presupune că înaintea inserției în *g* nu există nici un nod care are cheia identică cu cheia lui *e*.

10. ***InserArc****(***var*** g: TipGraf; k1,k2: TipCheie);*- operator care inserează în *g* un arc incident nodurilor având cheile k_1 şi k_2. Se presupune că cele două noduri există şi că arcul respectiv nu există înaintea inserției.

11. ***SuprimNod****(***var*** g: TipGraf; indicNod: TipIndicNod);* - operator care suprimă din *g* nodul precizat de *indicNod*, împreună cu toate arcele incidente. Se presupune că înaintea suprimării, un astfel de nod există.

12. ***SuprimArc****(***var*** g: TipGraf; indicArc: TipIndicArc);* - operator care suprimă din *g*, arcul precizat de *indicArc*. Se presupune că înaintea suprimării un astfel de arc există.

13. ***ActualizNod****(***var*** g: TipGraf; indicNod: TipIndicNod: x: TipInfo);* - operator care plasează valoarea lui x în porțiunea "informație" a nodului indicat de *indicNod* din graful *g*. Se presupune că *indicNod* precizează un nod al grafului.

14. ***FurnizeazaNod****(g:TipGraf; indicNod: TipIndicNod): TipElement;* - operator care returnează valoarea elementului memorat în nodul indicat de *indicNod* în graful *g*.

```
15.TraversGraf(var g: TipGraf; Vizită (ListăArgumente)) -
   operator care realizează traversarea grafului g,
   executând pentru fiecare element al acestuia procedura
   Vizită(ListaArgumente), unde Vizită este o procedură
   specificată de utilizator iar ListăArgumente este lista
   de parametri a acesteia.
------------------------------------------------------------
```

10.3. Tehnici de implementare a tipului de date abstract graf

- În vederea prelucrării grafurilor concepute ca şi tipuri de date abstracte (TDA) cu ajutorul sistemelor de calcul, este necesară la primul rând stabilirea modului lor de reprezentare.

- Această activitate constă de fapt din desemnarea unei structuri de date concrete care să materializeze în situaţia respectivă tipul de date abstract graf.

- În cadrul paragrafului de faţă se prezintă mai multe posibilităţi, alegerea depinzând, ca şi pentru marea majoritate a tipurilor de date deja studiate:

  - (1) De natura grafurilor de implementat

  - (1) De natura şi frecvenţa operaţiilor care se execută asupra lor.

- În esenţă se cunosc **două modalităţi majore** de implementare a grafurilor: una bazată pe matrici de adiacenţe iar celalată pe structuri de adiacenţe.

10.3.1. Implementarea grafurilor cu ajutorul matricilor de adiacenţe.

- Cel mai direct mod de reprezentare al unui tip de date abstract graf îl constituie **matricea de adiacenţe** (“**adjacency matrix**”).

- Dacă se consideră graful `G=(N,A)` cu mulţimea nodurilor `N={1,2,...,n}`, atunci **matricea de adiacenţe** asociată grafului `G`, este o matrice `A[n,n]` de elemente booleene, unde `A[x,y]` este adevărat dacă şi numai dacă în graf există un arc de la nodul `x` la nodul `y`.

- Adesea elementelor booleene ale matricii sunt înlocuite cu întregii 1 (adevărat), respectiv 0 (fals).

- Primul pas în reprezentarea unui graf printr-o matrice de adiacenţe constă în stabilirea unei corespondenţe între numele nodurilor şi mulţimea indicilor matricei.

  - Această corespondenţă poate fi realizată:

    - (1) În mod **implicit** prin alegerea corespunzătoare a tipului de bază al mulţimii N

    - (2) În mod **explicit** prin precizarea unei asocieri definite pe mulţimea nodurilor cu valori în mulţimea indicilor matricei.

- În cazul **corespondenței implicite** cel mai simplu mod de implementare constă în “a denumi” nodurile cu numere întregi care coincid cu indicii de acces în matricea de adiacențe.

    - Numele nodurilor pot fi de asemenea litere consecutive ale alfabetului sau în cazul limbajului Pascal, constante ale unui tip enumerare definit de utilizator, în ambele situații existând posibilitatea conversiei directe a tipurilor respective în tipul întreg prin funcții specifice de limbaj.

- În cazul **corespondenței explicite**, pentru implementarea asocierii pot fi utilizate:

    - Tehnici specifice simple cum ar fi cele bazate pe tablouri sau liste sau

    - Tehnici mai sofisticate bazate spre exemplu pe arbori binari sau pe metoda dispersiei.

- Pentru o urmărire facilă a algoritmilor, în cadrul capitolului de față, se va utiliza o metodă implicită conform căreia nodurile vor avea numele format dintr-o singură literă.

- În figura 10.3.1.a. apare reprezentarea bazată pe matrice de adiacențe (b) a grafului (a).

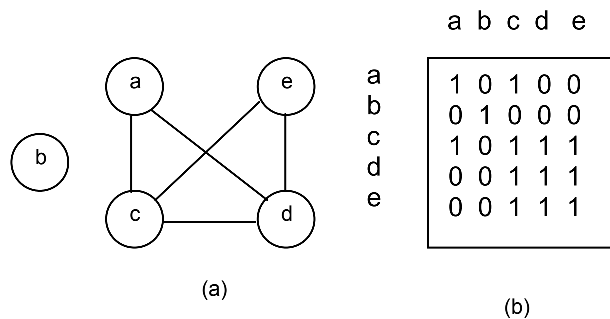


Fig.10.3.1.a. Graf (a) reprezentat prin matrice de adiacențe (b).

- Se observă faptul că reprezentarea are un caracter simetric întrucât fiind vorba despre un graf neorientat, arcul care conectează nodul `x` cu nodul `y` este reprezentat prin două valori în matrice: `A[x,y]` respectiv `A[y,x]`.

    - În astfel de situații, deoarece matricea de adiacențe este simetrică, ea poate fi memorată pe jumătate, element care pe lângă avantaje evidente are şi dezavantaje.

    - Astfel, pe de-o parte nu toate limbajele de programare sunt propice unei astfel de implementări,

    - Pe de altă parte algoritmii care prelucrează astfel de matrici sunt oarecum mai complicați decât cei care prelucrează matrici integrale.

- În prelucrarea grafurilor se poate face presupunerea că un nod este conectat cu el însuşi, element care se reflectă în valoarea "adevărat" memorată în toate elementele situate pe diagonala principală a matricei de adiacențe.
    - Acest lucru nu este însă obligatoriu şi poate fi reconsiderat de la caz la caz.
- În continuare se prezintă două studii de caz pentru implementarea TDA graf cu ajutorul matricilor de adiacențe.

10.3.1.1. Studiu de caz 1.

- După cum s-a precizat, un graf este definit prin mulțimea nodurilor şi prin mulțimea arcelor sale.
- În vederea prelucrării, un astfel de graf trebuie furnizat drept dată de intrare algoritmului care realizează această activitate.
- În acest scop, este necesar a se preciza modul în care se vor introduce în memoria sistemului de calcul elementele celor două mulțimi.
    - (1) O posibilitate în acest sens o reprezintă **citirea directă**, ca dată de intrare a matricii de adiacențe, metodă care nu convine în cazul matricilor rare.
    - (2) O altă posibilitate o reprezintă următoarea:
        - În prima etapă se citesc **numele nodurilor** în vederea asocierii acestora cu indicii matricei de adiacențe
        - În etapa următoare, se citesc perechile de nume de noduri care definesc arce în cadrul grafului.
        - Pornind de la aceste perechi se generează matricea de adiacențe.
        - Se face precizarea că prima etapă poate să lipsească dacă în implementarea asocierii se utilizează o metodă implicită.
- În secvența [10.3.1.1.a] apare un exemplu de program pentru **crearea** unei matrice de adiacențe.
    - Corespondența nume nod-indice este realizată implicit prin funcția "index", care are drept parametru numele nodului şi returnează indicele acestuia.
    - Din acest motiv, prima etapă se reduce la citirea valorilor `n` şi `a` care reprezintă numărul de noduri, respectiv numărul de arce ale grafului.
    - Ordinea în care se furnizează perechile de noduri în etapa a doua nu este relevantă, întrucât matricea de adiacențe nu este în nici un mod influențată de această ordine.

```
------------------------------------------------------------
{Cazul 1. Implementarea TDA Graf utilizand matrici de
adiacente}

const maxN = 50;
```

```
var j,x,y,N,A: integer;
    n1,n2: TipNod;
    graf: array[1..maxN,1..maxN] of boolean;
           {N = nr.de noduri, A = nr.de arce}
begin
  readln(N,A);                                   [10.3.1.1.a]
  for x := 1 to N do
    for y := 1 to N do graf[x,y]:= false;
  for x := 1 to N do graf[x,x]:= true;
  for j := 1 to A do
    begin
      readln(n1,n2);
      x:= index(n1); y := index(n2);
      graf[x,y]:= true; graf[y,x]:= true
    end
end. {Creare matrice de adiacente}
------------------------------------------------------------
```

- După cum se observă în cadrul secvenței, tipul variabilelor `n1` şi `n2` este `TipNod` care nu este precizat

- De asemenea nu este precizat nici codul aferent funcției ***index***, acestea depinzând direct de maniera de reprezentare a nodurilor.

    - Spre exemplu `n1` şi `n2` pot fi de tip caracter iar funcția ***index*** o expresie de forma `ord(n1)-ord (`a´).`

10.3.1.2. Studiu de caz 2

- Studiul de caz 2 prezintă o metodă mai elaborată de implementare a unui TDA graf, utilizând drept suport limbajul PASCAL.

- Reprezentarea presupune definirea tipurilor şi structurilor de date în conformitate cu secvența [10.3.1.2.a].

```
------------------------------------------------------------
{Cazul 2. Implementarea TDA Graf utilizand matrici de
adiacente}

const NumarNoduri = .......;

type  TipCheie     = .......;
      TipInfo      = .......;

      TipElement = record
                     Cheie: TipCheie;
                     Info : TipInfo
                   end;

      TipContor       = 0..NumarNoduri;
      TipIndex        = 1..NumarNoduri;
      TipTablouElem   = array[TipIndex] of TipElement;
      TipMatrAdj      = array[TipIndex,TipIndex] of boolean;

      TipGraf = record
                  Contor : TipContor;          [10.3.1.2.a]
                  Noduri : TipTablouElem;
```

```
                Arce     : TipMatrAdj
               end;

      TipArc = record
                 linie,
                 coloana : TipIndex
               end;

var   g          : TipGraf;
      k,k1,k2    : TipCheie;
      e          : TipElement;
      indicNod   : TipIndex;
      indicArc   : TipArc;
------------------------------------------------------------
```

- În accepţiunea acestei reprezentări, graful din fig. 10.3.1.2.a.(a) va fi implementat prin următoarele elemente:
    - (1) `contor` – care precizează numărul de noduri,
    - (2) `noduri` – tabloul care păstrează nodurile propriu-zise
    - (3) `arce` – matricea de adiacenţe a grafului (fig.10.3.1.2.a.(b)).

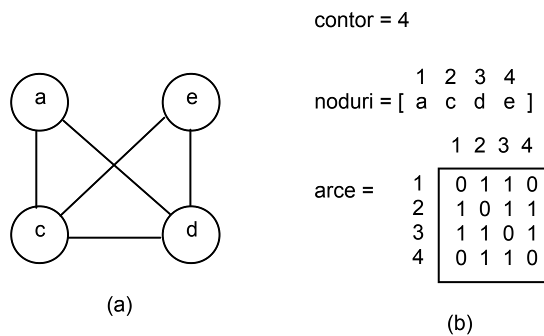


Fig.10.3.1.2.a. Reprezentarea elaborată a unui graf utilizând matrice de adiacenţe

- În anumite situaţii, pentru simplificare, nodurile nu conţin alte informaţii în afara cheii, caz în care `TipElement` = `TipCheie`.
- Alteori nodurile nu conţin nici un fel de informaţii (nici măcar cheia) situaţie în care interesează numai numele nodurilor în vederea identificării lor în cadrul reprezentării.
- În continuare se fac unele **consideraţii** referitoare la implementarea în acest context a setului de operatori extins (Varianta 1, (Shiflet)).
- (1) Operatorii ***InitGraf***, ***GrafVid***, ***GrafPlin***, împreună cu ***InserNod*** şi ***SuprimNod*** se referă în regim de consultare sau modificare la contorul care păstrează numărul nodurilor grafului: `g.contor`.

* (2) Informația conținută de tabloul `noduri` poate fi ordonată sau neordonată.

  * Dacă tabloul `noduri` este **ordonat**, localizarea unei chei în cadrul operatorilor ***CautăCheieGraf*** sau ***IndicaNod*** se poate realiza prin **tehnică căutării binare**

  * Dacă tabloul `noduri` este **neordonat** localizarea unei chei se poate realiza prin **tehnica căutării liniară**.

  * Operatorul ***CautăCheieGraf*** indică numai dacă cheia este prezentă sau nu

  * Operatorul ***IndicaNod****(g:TipGraf; k:TipCheie; var indicNod:TipIndicNod)* asignează lui ***IndicNod*** indexul nodului din graful `g`, care are cheiea egală cu `k`.

  * Operatorul ***IndicaArc****(g: TipGraf; k1,k2:TipCheie;* ***var*** *indicArc:TipArc)* se comportă într-o manieră similară, returnând indexul nodului `k1` în variabila de ieşire `indicArc.linie` şi indexul nodului `k2` în `indicArc.coloană`.

* (3) Inserția unui nod depinde de asemenea de maniera de organizarea datelor în cadrul tabloului `noduri`.

  * (a) Dacă `noduri` este un tablou **neordonat**, se incrementează `g.contor` şi se memorează nodul de inserat în `g.noduri[g.contor]`.

    * După cum rezultă din fig.10.3.1.2.b, care reprezintă inserția nodului `b` în graful din figura 10.3.1.2.a, nodul nou introdus este izolat, adică în matricea de adiacențe se introduce valoarea fals pe linia `g.contor` şi pe coloana `g.contor`.

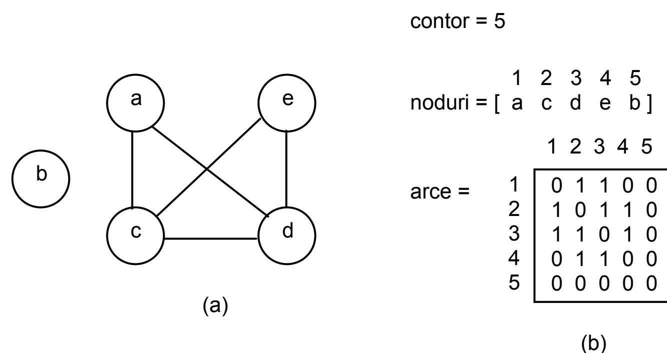


Fig.10.3.1.2.b. Inserția unui nod într-un graf

    * Procedura efectivă de inserție a unui nod nou în acest context apare în secvența [10.3.1.2.b].

```
------------------------------------------------------------
{Insertia unui nod. (Tabloul noduri neordonat)}

procedure InserNod(var g: graf; e: TipElement);
  var i,j: TipIndex;
```

```
begin                                             [10.3.1.2.b]
  g.contor:= g.contor + 1;
  g.noduri[g.contor]:= e;   {se plasează nodul nou}
  for i:= 1 to g.contor do
    g.arce[i,g.contor]:= false; {se inițializează matricea
  for j:= 1 to g.contor do        de adiacente pt. nodul
    g.arce[g.contor,j]:= false    nou}
end; {InserNod}
-------------------------------------------------------------
```

- (b) Dacă în tabloul `noduri` informațiile sunt **ordonate**, atunci:

    - În prealabil trebuie determinat locul în care se va realiza inserția

    - După acesta, trebuiesc realizate mutări de elemente în tabloul `g.noduri`

    - Urmate de mutări de linii şi de coloane în matricea de adiacențe `g.arce`.

    - În final se realizează inserția propriu-zisă şi se compleazā matricea de adiacențe.

- (4) Într-o manieră similară, la suprimarea nodului cu indexul `indicNod`, trebuiesc efectuate mişcări în tablourile `g.noduri şi g.arce`.
    - Figura 10.3.1.2.c ilustrează suprimarea nodului `c` din structura de graf din figura 10.3.1.2.b.

    - În vederea suprimării, `indicNod` are valoarea `2` precizând nodul `c`, iar suprimarea propriu-zisă presupune ştergerea nodului din `g.Noduri` şi modificarea matricei de adiacențe prin excluderea arcelor conexe nodului `c`.

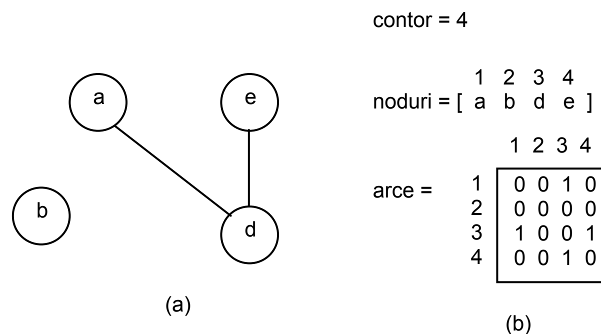


Fig.10.3.1.2.c. Suprimarea unui nod dintr-o structură graf

- (a) Dacă tabloul `noduri` este **neordonat**, stergerea lui `c` se poate realiza prin mutarea ultimului element al tabloului în locul său.

    - Pentru păstrarea corectitudinii reprezentării, este necesară ştergerea arcelor conexe lui `c` din matricea de adiacențe,

        - Pentru aceasta se copiază linia şi coloana corespunzătoare ultimului nod din matricea `g.arce`, adică nodul `b` peste linia şi coloana nodului care a fost suprimat.

- În final se decrementează variabila `g.contor`.

- Procedura care implementează aceste activități se numește ***SuprimNod*** și apare în secvența [10.3.1.2.c].

```
------------------------------------------------------------
{Suprimarea unui nod. (Tabloul noduri neordonat)}

procedure SuprimNod(var g: Graf; indicNod: TipIndex);
  var i,j: TipIndex;
  begin
    g.noduri[indicNod]:= g.noduri[g.contor];    [10.3.1.2.c]
    for j:= 1 to g.contor do
      g.arce[indicNod,j]:= g.arce[g.contor,j];
    for i:= 1 to g.contor do
      g.arce[i,indicNod] := g.arce[i,g.contor];
    g.contor := g.contor - 1
  end; {SuprimNod}
------------------------------------------------------------
```

- (b) Dacă tabloul `noduri` este **sortat**, atunci suprimarea presupune:

  - Mutarea cu o poziție a tuturor nodurilor care au indexul mai mare ca `indicNod` din tabloul `noduri`

  - Mutarea liniilor și coloanelor corespunzătoare lor din matricea de adiacențe.

- După cum se observă suprimarea unui nod presupune implicit și suprimarea arcelor conexe lui.

- Există însă posibilitatea de a șterge arce fără a modifica mulțimea nodurilor.

- În acest scop se utilizează procedura ***SuprimArc***`(g: TipGraf; indicArc: TipArc)` secvența [10.3.1.2.d].

  - Datorită simetriei reprezentării stergerea unui arc presupune două modificări în matricea de adiacențe.

```
------------------------------------------------------------
{Suprimarea unui arc}                         [10.3.1.2.d]

procedure SuprimArc(var g: Graf; indicArc: TipArc);
  begin
    g.Arc[indicArc.linie,indicArc.coloana]:= false;
    g.Arc[indicArc.coloana,indicArc.linie]:= false
  end; {SuprimArc}
------------------------------------------------------------
```

- În concluzie în studiul de caz 2, crearea unei structuri pentru un TDA graf presupune două etape:

- (1) Precizarea nodurilor grafului, implementată printr-o suită de apeluri ale procedurii ***InserNod*** (câte un apel pentru fiecare nod al grafului);

- (2) Conectarea nodurilor grafului, implementată printr-o suită de apeluri ale procedurii ***InserArc*** (câte un apel pentru fiecare arc al grafului).

- În general reprezentarea bazată pe matrice de adiacențe este eficientă în cazul grafurilor dense.

  - Din punctul de vedere al spațiului de memorie necesar reprezentării, matricea de adiacențe necesită n^2 locații de memorie pentru un graf cu `n` noduri.

  - În plus mai sunt necesare locații de memorie pentru memorarea informațiilor aferente celor `n` noduri.

  - Crearea grafului necesită un efort proporțional cu $O(n)$ pentru noduri şi aproximativ $O(n^2)$ paşi pentru arce, mai precis $O(a)$.

- În consecință de regulă, utilizarea reprezentării bazate pe **matrice de adiacențe** conduce la algoritmi care necesită un efort de calcul de ordinul $O(n^2)$.

10.3.2. Implementarea grafurilor cu ajutorul structurilor de adiacențe

- O altă manieră de reprezentare a TDA graf o **constituie structurile de adiacențe** ("**adjacency-structures**").

  - În cadrul acestei reprezentări, fiecărui nod al grafului i se asociază **o listă de adiacențe** în care sunt înlănțuite toate nodurile cu care acesta este conectat.

- În continuare se prezintă trei studii de caz pentru implementarea grafurilor cu ajutorul structurilor de adiacență.

10.3.2.1. Studiu de caz 1

- Implementarea structurii de adiacențe se bazează în Cazul 1 de studiu pe **liste înlănțuite simple**.

  - Începuturile listelor de adiacențe sunt păstrate într-un tablou `Stradj` indexat prin intermediul nodurilor.

  - Inițial în acest tablou se introduc înlănțuiri vide, urmând ca inserțiile în liste să fie de tipul "la începutul listei".

  - Adăugarea unui arc care conectează nodul `x` cu nodul `y` în cadrul acestui mod de reprezentare presupune în cazul grafurilor neorientate, inserția nodului `x` în lista de adiacențe a lui `y` şi inserția lui `y` în lista de adiacențe a nodului `x`.

- Un exemplu de program care construieşte o astfel de structură apare în secvența [10.3.2.1.a]

```
---------------------------------------------------------------
{Cazul 1. Construcţia unui graf utilizând structuri de
adiacenţe implementate cu ajutorul listelor înlănţuite simple}

const maxN = 100;
type RefTipNod = ^TipNod;
     TipNod = recod
                nume: integer;
                urm: RefTipNod
              end; {TipNod}

 var j,x,y,N,A: integer;
     v: RefTipNod;
     StrAdj: array[1..maxN] of RefTipNod;

 begin
   readln(N,A);                                     [10.3.2.1.a]
   for j:= 1 to N do StrAdj[j]:= nil;
   for j:= 1 to A do
     begin
       readln(n1,n2);
       x:= index(n1); y:= index(n2);
       NEW(v); v^.nume:= x; v^.urm:= StrAdj[y];
       StrAdj[y]:= v; {inserţie în faţă}
       NEW(v); v^.nume:= y; v^.urm:= StrAdj[x];
       StrAdj[x]:= v  {inserţie în faţă}
     end
 end;
---------------------------------------------------------------
```

- În figura 10.3.2.1.a se prezintă reprezentarea grafică a structurii construite pornind de la graful (a) din aceeaşi figură.

- Se face precizarea că datele de intrare (arcele) au fost furnizate în următoarea ordine: `(a,c)`, `(a,d)`, `(c,e)`, `(c,d)` şi `(d,e)`.

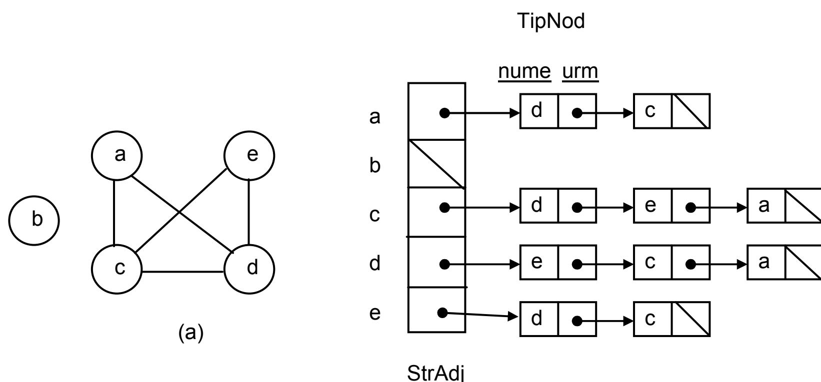


Fig.10.3.2.1.a. Graf şi structura sa de adiacenţe

- Se observă că un arc oarecare `(x,y)` este evidenţiat în două locuri în cadrul structurii, atât în lista de adiacenţe a lui `x`, cât şi în cea a lui `y`.

  - Acest mod redundant de evidenţiere îşi dovedeşte utilitatea în situaţia în care se cere să se determine într-o manieră eficientă care sunt nodurile conectate la un anumit nod x.

- Pentru acest mod de reprezentare, contează ordinea în care sunt prezentate arcele

respectiv perechile de noduri, la intrare.

- Astfel, un acelaşi graf poate fi reprezentat ca structură de adiacenţe în moduri diferite.

- Ordinea în care apar arcele în lista de adiacenţe, afectează la rândul ei ordinea în care sunt prelucrate arcele de către algoritm.

  - Funcţie de natura algoritmilor utilizaţi în prelucrare această ordine poate să influenţeze sau nu rezultatul prelucrării.

10.3.2.2. Studiu de caz 2

- În acest cazul 2 de studiu, implementarea structurilor de adiacenţe se bazează pe **structuri multilistă**.

  - Astfel, o structură de adiacenţe este de fapt o listă înlănţuită a nodurilor grafului.

  - Pentru fiecare nod al acestei liste se păstrează o listă a arcelor, adică o listă înlănţuită a cheilor nodurilor adiacente.

  - În consecinţă, fiecare nod al listei nodurilor va conţine două înlănţuiri, una indicând nodul următor, cealaltă, lista nodurilor adiacente.

- În figura 10.3.2.2.a apare structura de adiacenţe (b) a grafului (a).

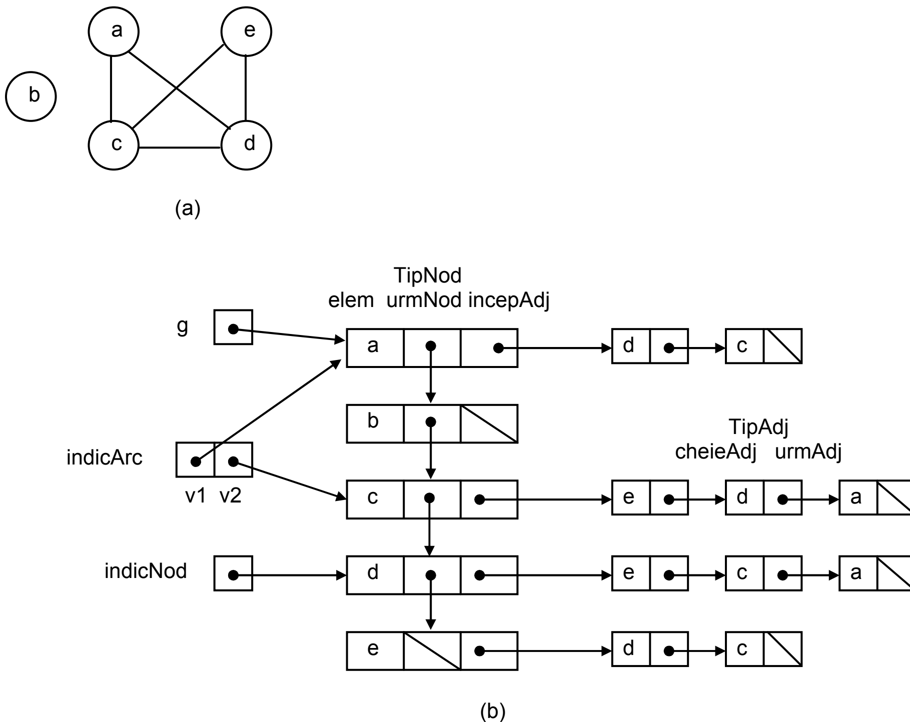


Fig.10.3.2.2.a. Reprezentarea unui graf ca şi o structură de adiacenţe utilizînd structuri multilistă

- Implementarea PASCAL a structurii multilistă apare în secvenţa [10.3.2.2.a].

```
{Cazul 2. Implementarea grafurilor utilizând structuri de
adiacenţe implementate cu ajutorul structurilor multilistă}

type TipCheie   = .....;
     TipInfo    = .....;

     TipElement = record
                    cheie: TipCheie;
                    info : TipInfo
                  end;

     RefTipAdj  = ^TipAdj;

     TipAdj     = record
                    cheieAdj: TipCheie;
                    urmAdj  : RefTipAdj
                  end;
     RefTipNod  = ^TipNod;
     TipGraf    = RefTipNod;
     TipNod     = record
                    elem    : TipElement;
                    urmNod  : RefTipNod;
                    incepAdj: RefTipAdj        [10.3.2.2.a]
                  end;
     TipArc     = record
                    v1,v2: RefTipNod
                  end;
var  g: TipGraf;
     indicNod: RefTipNod;
     indicArc: TipArc;
     k,k1,k2: TipCheie;
     e: TipElement;
-------------------------------------------------------------
```

- Se face precizarea că valorile aferente nodurilor sunt păstrate integral în lista de noduri, în lista de arce apărând numai cheile.

  - Este posibil ca câmpul `info` să lipsească şi deci `TipElement=TipCheie`.

- În figura 10.3.2.2.a apare reprezentarea schematică a unei structuri de adiacenţe cu precizarea câmpurilor aferente.

  - De asemenea sunt prezentate ca exemplu variabilele `g`, `indicNod` şi `indicArc`, evidenţiindu-se structura fiecăreia.

- În cadrul acestei structuri de date:

  - Operatorii ***CautăCheieGraf***, ***IndicăNod*** şi ***IndicăArc*** aparţinând setului de operatori extins (varianta 1) utilizează tehnica căutării liniare în determinarea unui nod a cărui cheie este cunoscută.

  - Inserţia unui nod nou se realizează simplu la începutul listei nodurilor.

  - Operatorul ***InserArc(g: TipGraf; ,k_1, k_2 : TipCheie)*** presupune inserţia lui k_1 în lista de adiacenţe a lui k_2 şi reciproc.

- Şi în acest caz inserţia se realizează cel mai simplu la începutul listei.

- Suprimarea unui arc precizat spre exemplu de indicatorul `indicArc` presupune extragerea a două noduri din două liste de adiacenţe diferite.

  - Astfel în figura 10.3.2.2.a, variabila `indicArc` conţine doi pointeri v_1 şi v_2, care indică cele două noduri conectate din lista de noduri.

  - În vederea suprimării arcului care le conectează este necesar ca fiecare nod în parte să fie suprimat din lista de adiacenţe a celuilalt.

  - În cazul ilustrat, pentru a suprima arcul `(a,c)=(c,a)` se scoate `a` din lista lui `c`, respectiv `c` din lista lui `a`.

- Procedura care realizează suprimarea în această manieră a unui arc apare în secvenţa [10.3.2.2.b]

  - Se face precizarea că procedura ***SuprimArc*** este redactată în termenii setului de operatori aplicabili obiectelor de tip listă [Vol 1. &6.2.1].

```
------------------------------------------------------------
{Suprimarea unui arc. În implementare se utilizează
operatorii definiţi pentru TDA Lista înlănţuită simplă}

procedure SuprimArc(g: TipGraf; indicArc: TipArc);
  var ik1,ik2: RefTipAdj;
  begin
    ik1:= Cauta(indicArc.v1^.elem.cheie,
                indicArc.v2^.incepAdj);
    ik2:= Cauta(indicArc.v2^.elem.cheie,
                indicArc.v1^.elem.cheie);     [10.3.2.2.b]
    Suprima(ik1,indicArc.v2^.incepAdj);
    Suprima(ik2,indicArc.v1^.incepAdj)
  end; {SuprimArc}
------------------------------------------------------------
```

- În figura 10.3.2.2.b apare structura de adiacenţe aferentă grafului din figura 10.3.2.2.a(a) după suprimarea arcului `(a,c)`.

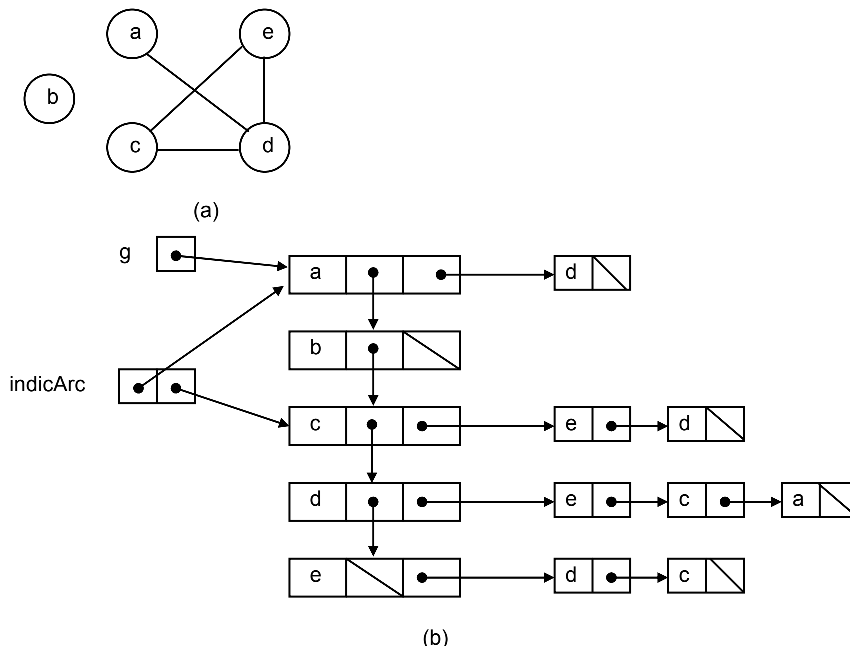


Fig.10.3.2.2.b. Structură de adiacenţe după suprimarea unui arc (a,c)

- Suprimarea unui nod dintr-o structură de graf, presupune nu numai suprimarea propriu-zisă a nodului respectiv ci în plus suprimarea tuturor arcelor incidente acestui nod.

- În acest scop, se determină cheia `k1` a nodului de suprimat.

- În continuare, atât timp cât mai există elemente în lista sa de adiacente se realizează următoarea secvenţă de operaţii:

  - (1) Se determină cheia `k2` a primului element din lista de adiacenţe a lui `k1`;

  - (2) Se suprimă arcul `(k1,k2)` suprimând pe `k2` din lista lui `k1` şi pe `k1` din lista lui `k2`;

- În final se suprimă nodul `k1` din lista nodurilor grafului.

- Procedura care realizează suprimarea unui nod apare în secvenţa [10.3.2.2.c]

```
------------------------------------------------------------
{Suprimarea unui nod. În implementare se utilizează
operatorii definiţi pentru TDA Listă înlanţuită simplă}

procedure SuprimNod(g: TipGraf; indicNod: RefTipNod);

  var k1,k2       : TipCheie;
      indicNod2 : REfTipNod;
      ik1,curent: PtrAdj;                          [10.3.2.2.c]

  begin
    k1:= indicNod^.elem.cheie; curent:= indicNod^.incepAdj;
    WHILE not Fin(curent) do
      begin
        k2:= (Primul(curent))^.cheieAdj;
        Suprima(Primul(curent),curent);
         {Se presupune ca operatorul Suprima actualizează în
          mod corespunzator valoarea variabilei "curent"
          nemaifiind necesară trecerea explicită la
          elementul urmator al listei indicate de "curent".
          Se precizează faptul ca în această situaţie este
          vorba despre o suprimare a primului element al
          listei}
        indicNod2  := Cauta(k2,g);
        ik1:= Cauta(k1,indicNod2^.incepAdj);
        Suprima(ik1,indicNod2^.incepAdj);
      end
    Suprima(indicNod,g)
  end;
{SuprimNod}
------------------------------------------------------------
```

- În fig.10.3.2.2.c apare reprezentată structura de adiacențe rezultată în urma suprimării nodului `d` din graful din figura 10.3.2.2.b.

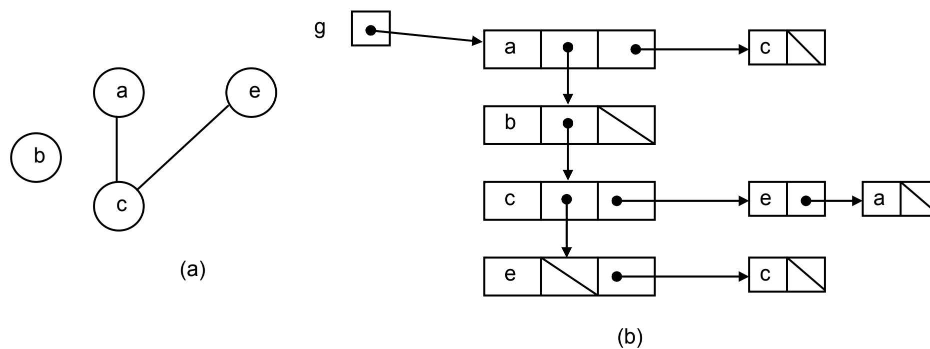


Fig.10.3.2.2.c. Structură de adiacențe după suprimarea unui nod (d).

10.4. Tehnici fundamentale de traversare a grafurilor

- Rezolvarea eficientă a problemelor curente referitoare la grafuri, presupune de regulă, **traversarea** (vizitarea sau parcurgerea) nodurilor şi arcelor acestora într-o manieră sistematică.

- În acest scop s-au dezvoltat două tehnici fundamentale, una bazată pe **căutarea în adâncime**, cealaltă bazată pe **căutarea prin cuprindere**.

10.4.1. Traversarea grafurilor prin tehnica căutării "în adâncime" ("Depth-First Search")

- Căutarea în adâncime este una dintre tehnicile fundamentale de traversare a grafurilor.

- Această tehnică este similară traversării în preordine a arborilor şi ea constituie nucleul în jurul căruia pot fi dezvoltați numeroşi algoritmi eficienți de prelucrare a grafurilor.

- Principiul căutării "**în adâncime**" într-un graf `G` este următorul:

  - Se marchează inițial toate nodurile grafului `G` cu marca "`nevizitat`".

  - Căutarea debutează cu selecția unui nod `n` a lui `G` pe post de nod de pornire şi cu marcarea acestuia cu "`vizitat`".

  - În continuare, fiecare nod nevizitat adiacent lui `n` este "căutat" la rândul său, aplicând în mod recursiv aceeaşi "căutare în adâncime".

  - Odată ce toate nodurile la care se poate ajunge pornind de la `n` au fost vizitate în maniera mai sus precizată, cercetarea lui `n` este terminată.

  - Dacă în graf au rămas noduri nevizitate, se selectează unul dintre ele drept nod nou de pornire şi procesul se repetă până când toate nodurile grafului au fost vizitate.

- Această tehnică se numeşte căutare “**în adâncime**” ("**depth-first**") deoarece parcurgerea grafului se realizează înaintând “în adâncime” pe o direcție aleasă atâta timp cât acest lucru este posibil.

  - Spre exemplu, presupunând că x este ultimul nod vizitat, căutarea în adâncime selectează un arc neexplorat conectat la nodul x.

  - Fie y nodul corespunzător acestui arc.

    - Dacă nodul y a fost deja vizitat, se caută un alt arc neexplorat conectat la x.

    - Dacă y nu a fost vizitat anterior, el este marcat “vizitat” şi se inițiază o nouă căutare începând cu nodul y.

  - În momentul în care se epuizează căutarea pe toate drumurile posibile pornind de la y, se revine la nodul x, (principiul recursivității) şi se continuă în aceeaşi manieră selecția arcelor neexplorate ale acestui nod, până când sunt epuizate toate posibilitățile care derivă din x.

  - Se observă clar tendința inițială de adâncire, de îndepărtare față de sursă, urmată de o revenire pe măsura epuizării tuturor posibilităților de traversare.

- Considerând o structură graf într-o reprezentare, oarecare şi un tablou “marc” ale cărui elemente corespunzând nodurilor grafului, marchează faptul că un nod a fost sau nu vizitat, schița de principiu a algoritmului de căutare în adâncime apare în secvența [10.4.1.a].

```
------------------------------------------------------------
{Cautarea "în adancime". Schița de principiu a algoritmului.
    Varianta 1}

procedure CautaInAdincime(x: TipNod);
  var y: TipNod;
  begin
    marc[x] := vizitat;                               [10.4.1.a]
    for fiecare nod y adiacent lui x do
      if marc[y] = nevizitat then
        CautaInAdincime(y)
  end; {CautaInAdincime}
------------------------------------------------------------
```

10.4.1.1. Căutarea "în adâncime", varianta CLR

- O variantă mai elaborată a traversării în adâncime este cea propusă de Cormen, Leiserson şi Rivest [CLR92].

- Conform acesteia, pe parcursul traversării nodurilor sunt **colorate** pentru a marca stările prin care trec.

- Din aceste motive traversarea grafurilor este cunoscută şi sub denumirea de "**colorare a grafurilor**".

  - Culorile utilizate sunt alb, gri şi negru.

    - Toate nodurile sunt inițial colorate în alb, în timpul traversării devin gri iar la terminarea traversării sunt colorate în negru.
    - Un nod alb care este descoperit prima dată în traversare este colorat.
    - În consecință, nodurile gri şi negre au fost deja întâlnite în procesul de traversare, ele marcând modul în care avansează traversarea.
    - Un nod este colorat în negru când toate nodurile adiacente lui au fost descoperite.
    - Un nod colorat în gri poate avea şi noduri adiacente albe.
    - Nodurile gri marchează frontiera între nodurile descoperite şi cele nedescoperite şi ele se păstrează de regulă în structura de date asociată procesului de traversare.

- În procesul de traversare se poate construi un **subgraf asociat traversării**, subgraf care include arcele parcurse în traversare şi care este de fapt un graf de precedențe.
    - Acest subgraf este un **graf conex aciclic**, adică un arbore, care poate fi simplu reprezentat prin tehnica "indicator spre părinte".
    - Tehnica de construcție a subgrafului este următoarea:
        - Atunci când în procesul de căutare se ajunge de la nodul `u` la nodul `v`, acest lucru se marchează în subgraful asociat `s` prin `s[v]=u`.
        - Graful de precedențe este descris formal conform următoarelor relații [10.4.1.1.a].

$$G_{pred} = (N, A_{pred})$$
$$A_{pred} = \{(s[v],v) : v \in A \ \& \ s[v] \neq nil\} \qquad [10.4.1.1.a]$$

        - Subgraful predecesorilor asociat căutării în adâncime într-un graf precizat, formează o pădure de arbori de căutare în adâncime.
    - Pe lângă crearea propriu-zisă a subgrafului predecesorilor fiecărui nod i se pot asocia două mărci de timp ("**timestamps"**):
        - (1) `i[v]` memorează momentul descoperirii nodului `v` (colorarea sa în gri),
        - (2) `f[v]` memorează momentul terminării explorării nodurilor adiacente lui `v` (colorarea se in negru).
    - Mărcile de timp sunt utilizate în multi algoritmi referitori la grafuri şi ele precizează în general comportamentul în timp.

- În cazul de față, pentru simplitate, timpul este conceput ca un întreg care ia valori între 1 şi 2|N|, întrucât este incrementat cu 1 la fiecare descoperire respectiv terminare de examinare a fiecăruia din cele |N| noduri ale grafului.

- Varianta de căutare în adâncime propusă de Cormen, Lesiserson şi Rivest apare în secvența [10.4.1.1.b]

```
{Cautarea "în adancime". Schița de principiu a algoritmului.
    Varianta 2 (Cormen, Leiserson, Rivest}

procedure TraversareInAdincime(G: TipGraf);
[1]   for fiecare nod u ∊ N(G) do
        begin
[2]       culoare[u]<-alb;
[3]       sp[u]<-nil
        end
[4]   timp<-0;
[5]   for fiecare nod u ∊ N(G) do
[6]     if culoare[u]=alb then
[7]       CautareInAdincime(u);
                                                    [10.4.1.1.b]
procedure CautareInAdancime(u: TipNod);
[1]   culoare[u]<-gri;
[2]   timp<-timp+1; i[u]<-timp;
[3]   for fiecare v adiacent lui u do
[4]     if culoare[v]=alb then
          begin
[5]         sp[v]<-u;
[6]         CautareInAdancime(v)
          end
[7]   culoare[u]<-negru;
[8]   timp<-timp+1; f[u]<-timp;
```

- **Analiza algoritmului.**

  - Liniile 1-3 şi 5-7 ale lui ***TraversareInAdâncime*** necesită un timp proporțional cu O*(N)*, excluzând timpul necesar execuției apelurilor procedurii de căutare propriu-zise.

  - Procedura ***CăutareInAdâncime*** este apelată exact odată pentru fiecare nod v∈N, deoarece ea este invocată doar pentru noduri albe şi primul lucru pe care îl face este să coloreze respectivul nod în gri.

  - Liniile 3-6 se execută într-un interval de timp proporțional cu |Adj[v]| unde este valabilă formula [10.4.1.1.c].

$$\sum_{v \in N} |Adj[v]| = O(a) \qquad [10.4.1.1.c]$$

----

    * În consecință rezultă că costul total al execuției liniilor 2-5 ale procedurii **CautareInAdâncime** este O(*A*).

    * **Timpul total de execuție** al traversării prin căutare în adâncime este deci O(*N*+*A*).

* În continuare se detaliază această tehnică de căutare pentru diferite modalități de implementare a grafurilor.

10.4.1.2. Căutare ”în adâncime” în grafuri reprezentate prin structuri de adiacențe

* Procedura care implementează căutarea în adâncime în grafuri reprezentate prin structuri de adiacențe apare în secvența [10.4.1.2.a].

    * Procedura `Traversare1` completează tabloul `marc[1..maxN]` pe măsură ce sunt traversate (vizitate) nodurile grafului.

    * Tabloul `marc` este poziționat inițial pe zero, astfel încât `marc[x]=0` indică faptul că nodul `x` nu a fost încă vizitat.

    * Pe parcursul traversării câmpul `marc` corespunzător unui nod `x` se completează în momentul începerii vizitării cu valoarea “`id`”, valoare care se incrementează la fiecare nod vizitat şi care indică faptul că nodul `x` este cel de-al `id`-lea vizitat.

    * Procedura de traversare utilizează procedura recursivă `CautaInAdâncime` care realizează vizitarea tuturor nodurilor aparținătoare acelei componente conexe a grafului, căreia îi aparține nodul furnizat ca parametru.

    * Se face precizarea că procedura `Traversare1` din secvența [10.4.1.2.a] se referă la reprezentarea TDA graf bazată pe structuri de adiacență implementate cu ajutorul listelor înlănțuite simple.

```
------------------------------------------------------------
{Traversarea "în adancime" a grafurilor reprezentate prin
structuri de adiacențe implementate cu ajutorul listelor
înlănțuite simple.}

procedure Traversare1;
  var id,x: integer;
      marc: array[1..maxN] of integer;

  procedure CautaInAdincime(x: integer);
    var t: RefTipNod;
    begin
      id:= id + 1; marc[x]:= id; write(t^.nume);
      t:= Stradj[x];
      while t <> nil do
        begin
          if marc[t^.nume] = 0 then
            CautaInAdincime(t^.nume);
          t:= t^.urm
        end
```

```
    end; {CautaInAdincime}

  begin { Traversare1 }
    id:= 0;
    for x:= 1 to N do marc[x]:= 0;
    for x:= 1 to N do
      if marc[x] = 0 then
         begin
           CautaInAdincime(x); writeln
         end
  end; { Travesare1 }
------------------------------------------------------------
```

- **Vizitarea** unui nod presupune parcurgerea tuturor arcelor conexe lui, adică parcurgerea listei sale de adiacenţe şi verificarea pentru fiecare arc în parte, dacă el conduce la un nod care a fost sau nu vizitat.

- În caz că nodul este nevizitat procedura se apelează recursiv pentru acel nod.

- Procedura `Traversare1`, parcurge tabloul `marc` şi apelează procedura de `CautaInAdancime` pentru componentele nevizitate ale grafului, până la traversarea sa integrală.

    - Trebuie observat faptul că fiecare apel al procedurii `CautaInAdâncime` realizat din cadrul procedurii `Traversare1` asigură parcurgerea unei componente conexe a grafului şi anume a componentei conexe care conţine nodul selectat.

- În figura 10.4.1.2.a apare urma execuţiei algoritmului de căutare în adâncime pentru graful (a) din figură.

    - Structura de adiacenţe aferentă grafului apare în aceeaşi figură (b)

    - Evoluţia conţinutului stivei apare în în aceeaşi figură (c).

    - Se menţionează faptul că nodurile structurii de adiacenţe au ataşat un exponent fracţionar care la numărător precizează momentul la care este descoperit nodul, (adică momentul în care este introdus în stivă), iar la numitor, momentul terminării explorării nodului, (adică momentul în care este scos din stivă).

- Graful este **traversat** drept consecinţă a apelului `CautăInAdâncime(A)`, efectuat în ciclul **`for`** al procedurii `Traversare1`.

    - Ca atare nodul `A` este introdus în stivă la momentul 1.

    - Pentru nodul `A` se parcurge lista sa de adiacenţe, primul arc traversat fiind `AF`, deoarece `F` este primul nod din lista de adicenţe a lui `A`.

    - În continuare se apelează procedura `CautăInAdâncime` pentru nodul `F`, în consecinţă nodul `F` este introdus în stivă la momentul 2 şi se traversează arcul `FA`, `A` fiind primul nod din lista de adiacenţe a lui `F`.

- Întrucât nodul A a fost deja descoperit (intrarea sa în tabloul marc conține o valoare nenulă), se alege în continuare arcul FE, E fiind nodul următor în lista de adiacențe a lui F.

- Nodul E se introduce în stivă la momentul 3.

- Se traversează în continuare arcul EG (G se introduce în stivă la momentul 4) G fiind primul nod din lista de adiacențe a lui E.

- În continuare se traversează arcul GE respectiv GA (nodurile E respectiv A fiind deja descoperite), moment în care se termină parcurgerea listei de adiacențe a lui G, fapt realizat la timpul 5.

- Se elimină G din stivă, se revine în lista nodului E şi se continuă parcurgerea listei sale de adiacențe traversând arcele EF (nodul F a fost deja vizitat) şi ED.

- Ca atare nodul D este descoperit la momentul 6 şi în continuare vizita lui D presupune traversarea arcelor DE şi DF care niciunul nu conduce la un nod nou.

- Terminarea parcurgerii listei de adiacențe a lui D are drept consecuință finalizarea vizitării lui şi scoaterea din stivă la momentul 7.

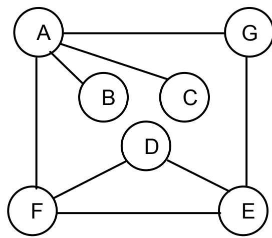
- Se revine în lista de adiacențe a lui E. Deoarece D a fost ultimul nod din lista de adiacențe a lui E, vizita lui E se încheie la momentul 8 şi revine în nodul F a cărui vizitare se încheie prin parcurgerea arcului FD (D deja vizitat).

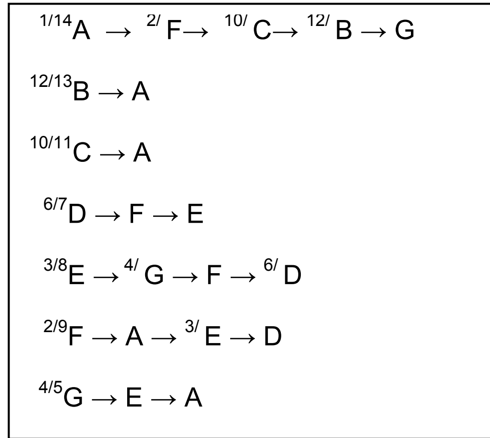
- Se elimină F din stivă la momentul 9, se revine în lista lui A şi se găseşte nodul C nevizitat încă.

- C se introduce în stivă la momentul 10, i se parcurge lista care îl conține doar pe A deja vizitat şi este extras din stivă la momentul 11.

- Se continuă parcurgerea listei de adiacențe a nodului A şi se găseşte nodul B care suferă un tratament similar lui C, la momentele 12 respectiv 13.

- În final se ajunge la sfârşitul listei lui A, A se scoate din stivă la momentul 14 şi procesul de traversare se încheie.





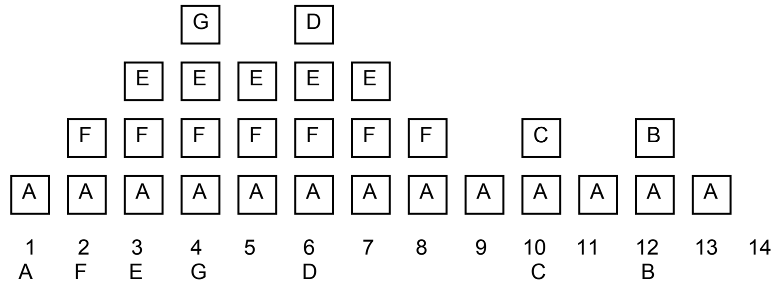


Fig.10.4.1.2.a. Urma execuției algoritmului recursiv de căutare în adâncime

- Un alt mod de a urmări desfăşurarea operației de căutare în adâncime este acela de a redesena graful traversat pornind de la apelurile recursive ale procedurii `CautInAdâncime`, ca în figura 10.4.2.1.b.

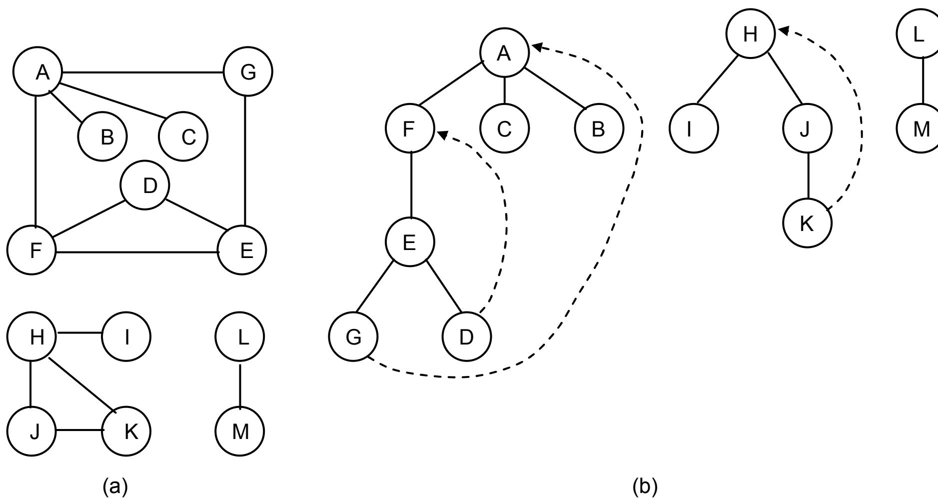


Fig.10.4.2.1.b. Arbori de căutare în adâncime

- În figura 10.4.2.1.b:

  - O **linie continuă** indică faptul că nodul aflat la extremitatea sa inferioară, a fost găsit în procesul de căutare în lista de adiacențe a nodului aflat la

extremitatea sa superioară şi nefiind vizitat la momentul considerat, s-a realizat pentru el un apel recursiv al procedurii de căutare.

- O **linie punctată** indică un nod descoperit în lista de adiacențe a nodului sursă, pentru care apelul recursiv nu se realizează, deoarece nodul a fost deja vizitat sau este in curs de vizitare (este memorat în stiva asociată prelucrării).

  - Ca atare condiția din instrucția **`if`** a procedurii `CautăInAdâncime` nu este îndeplinită nodul fiind marcat cu vizitat în tabloul `marc` şi în consecință pentru acest nod nu se realizează un apel recursiv al procedurii.

- Aplicând această tehnică, pentru fiecare componentă conexă a unui graf se obține un **arbore de acoperire** ("**spanning tree**") numit şi **arbore de căutare în adâncime** al componentei.

  - La traversarea acestui arbore în preordine se obțin nodurile în ordinea în care sunt prima dată întâlnite în procesul de căutare

  - La traversarea sa în postordine furnizează nodurile în ordinea în care cercetarea lor se încheie.

- Este important de subliniat faptul că întrucât ramurile arborilor de acoperire, materializează arcele grafului, mulțimea (pădurea) arborilor de căutare în adâncime asociați unui graf reprezintă o altă metodă de reprezentare grafică a grafului.

* O proprietate esențială a arborilor de căutare în adâncime pentru grafuri neorientate este aceea că liniile punctate indică întotdeauna un strămoş al nodului în cauză.

* În orice moment al execuției algoritmului, nodurile grafului se împart în trei clase:

  - (1) Clasa I - conține nodurile pentru care procesul de vizitare s-a terminat (colorate în negru)

  - (2) Clasa II - conține nodurile care sunt în curs de vizitare (colorate în gri)

  - (3) Clasa III - conține nodurile la care nu s-a ajuns încă (colorate în alb).

* (1) În ceea ce priveşte prima clasă de noduri, datorită modului de implementare a procedurii de căutare, nu va mai fi selectat nici un arc care indică vreun astfel de nod, motiv pentru care aceste arce nu se reprezintă în structura de arbore.

* (2) În ceea ce priveşte clasa a III-a de noduri, aceasta cuprinde nodurile pentru care se vor realiza apeluri recursive şi arcurile care conduc la ele vor fi marcate cu linie continuă în arbore.

* (3) Mai rămân nodurile din cea de-a doua clasă: acestea sunt nodurile care au apărut cu siguranță în drumul de la nodul curent la rădăcina arborelui, sunt colorate în gri şi sunt memorate în stiva asociată căutarii.

  - Ca atare, orice arc procesat care indică vreunul din aceste noduri apare reprezentat punctat în arborele de căutare în adâncime.

- În concluzie:
    - Arcele marcate continuu în figura 10.4.1.2.b se numesc **arce de arbore**
    - Arcele marcate punctat se numesc **arce de retur.**
- Din punct de vedere formal, dacă `x` şi `y` sunt noduri ale grafului ce urmează a fi traversat atunci:
    - (1) **Arcul de arbore** este acel arc `(x,y)` al grafului pentru care instanţa de apel a procedurii recursive `CautăInAdâncime(x)` apelează instanţa `CautăInAdâncime(y).`
        - Nodul `y` aparţine clasei III la momentul apelului (colorat în alb, adică nevizitat).
    - (2) **Arcul de retur** `(x,y)` este un arc al grafului întâlnit în procesul de vizitare al nodului `x,` adică întâlnit în lista lui de adiacenţe şi care conduce la un nod `y` aparţinând clasei I sau II,
        - Cu alte cuvinte `y` este un nod pentru care procesul de vizitare s-a terminat (colorat în negru) sau care se află în stiva asociată traversării (colorat în gri).
        - Arcul de retur `(x,y)` indică de fapt un nod strămoş al nodului x.

10.4.1.3. Căutare ”în adâncime” în grafuri reprezentate prin matrici de adiacenţe

- Procedura care implementează căutarea în adâncime în grafuri reprezentate prin matrici de adiacenţe apare în secvenţa [10.4.1.3.a].
    - Traversarea listei de adiacenţe a unui nod din cazul anterior, se transformă în parcurgerea liniei corespunzătoare nodului din matricea de adiacenţe, căutând valori adevărate (care marchează arce).
    - Ca şi în cazul anterior, selecţia unui arc care conduce la un nod nevizitat este urmată de un apel recursiv al procedurii de căutare pentru nodul respectiv.
    - Datorită modului diferit de reprezentare a grafului, arcele conectate la noduri sunt examinate într-o altă ordine, motiv pentru care arborii de căutare în adâncime care alcătuiesc pădurea corespunzătoare grafului diferă ca formă.

```
------------------------------------------------------------
{Căutare "în adâncime" în grafuri reprezentate prin matrici
de adiacenţe}

procedure CautaInAdincime1(x: integer);
  var t: integer;
  begin
    id:= id + 1; marc[x]:= id;
```

```
    write(x);                                 [10.4.1.3.a]
    for t:= 1 to N do
      if A[x,t] then
        if marc[t] = 0 then CautaInAdincime1(t)
  end; { CautaInAdincime1 }
------------------------------------------------------------
```

- Pentru graful din figura 10.4.1.3.a (a), a cărui matrice de adiacențe apare în aceeaşi figură (b), evoluția stivei asociate traversării apare în (c) şi (d) iar arborii de căutare în adâncime corespunzători apar în fig. 10.4.1.3.b.

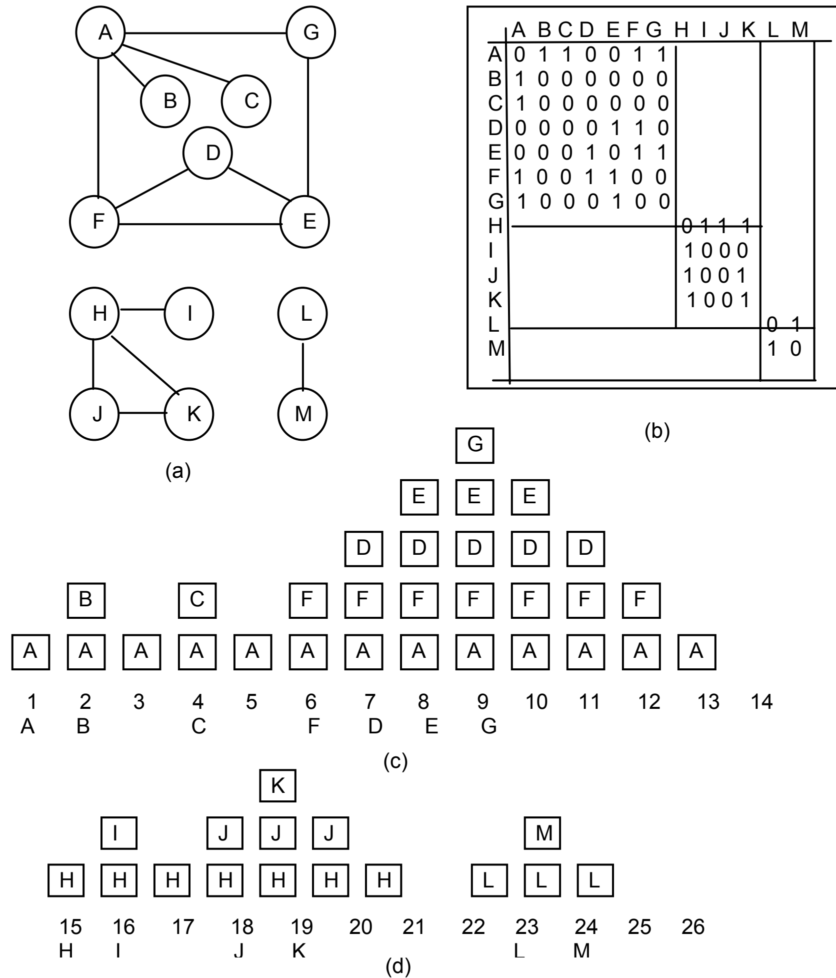


Fig.10.4.1.3.a.Căutare în adâncime în grafuri reprezentate prin matrice de adiacențe

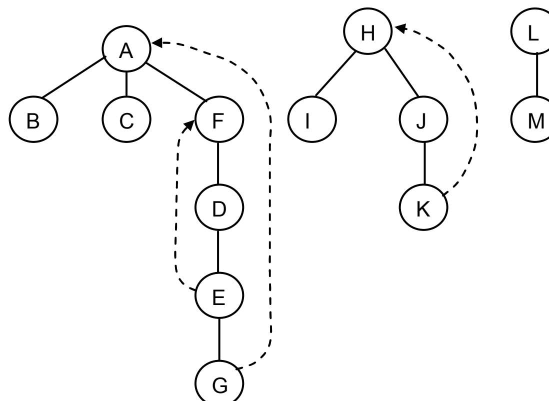


Fig.10.4.1.3.b. Pădure de arbori de căutare în adâncime în grafuri reprezentate prin matrice de adiacențe

- Se observă unele diferențe față de pădurea de arbori reprezentată în fig. 10.4.2.1.b, corespunzătoare aceluiaşi graf.

    - Prin aceasta se subliniază faptul că o pădure de arbori de căutare în adâncime nu este altceva decât o altă manieră de reprezentare a unui graf, a cărei alcătuire particulară depinde atât de metoda de traversare a grafului cât şi de reprezentarea internă utilizată.

- Din punct de vedere al eficienței, căutarea în adâncime în grafuri reprezentate prin matrice de adiacențe necesită un timp proporțional cu $O(n^2)$.

    - Acest lucru este evident întrucât în procesul de traversare este verificat fiecare element al matricei de adiacențe.

- Căutarea în adâncime rezolvă unele probleme fundamentale ale prelucrării grafurilor.

    - Astfel, deoarece procedura de parcurgere a unui graf se bazează pe traversarea pe rând a componentelor sale conexe, numărul componentelor conexe ale unui graf poate fi determinat simplu contorizând numărul de apeluri ale procedurii `CautăInAdâncime` efectuat din ultima linie a procedurii `Traversare1`.

- Căutarea în adâncime permite de asemenea verificarea simplă a existenței ciclurilor într-un graf.

    - Astfel, un graf conține un ciclu, dacă şi numai dacă procedura `CautăInAdâncime` descoperă o valoare diferită de zero în tabloul `marc`.

    - Această înseamnă că se parcurge un arc care conduce la un nod care a mai fost vizitat, deci graful conține un ciclu.

    - În cazul grafurilor neorientate trebuie însă să se țină cont de reprezentarea dublă a fiecărui arc, care poate produce confuzii.

    - Pentru reprezentarea grafurilor prin pădure de arbori de căutare, liniile punctate sunt acelea care închid ciclurile.