8 Grafuri - parcurgere DFS

8.1 Obiective

Scopul acestui laborator este de a prezenta parcurgerea grafurilor în adancime.

8.2 Noțiuni teoretice

8.2.1 Explorarea (căutarea) în adancime

Explorarea (căutarea) în adancime (DFS, *Depth-First Search*) poate fi utilizată pentru rezolvarea mai multor probleme care implică grafuri, dupa cum vom vedea.

Explorarea în adancime pornește de la un vârf al grafului și explorează primul vecin nevizitat. Apoi pentru nodul curent se va vizita din nou primul vecin nevizitat. Atunci cand se ajunge in punctul in care nu se mai gasesc vecini de vizitat din nodul curent, algoritmul cauta un vecin de vizitat parcurgand inapoi nodurile vizitate (backtracking).

Algoritmul DFS se aplică atât pentru grafuri orientate cât și pentru cele neorientate. Pentru grafuri neorientate, DFS vizitează toate nodurile din aceeași componentă conexă cu nodul de start. Pentru grafuri orientate, nodurile vizitate pornind de la nodul de start sunt cele la care se poate ajunge din nodul de start, dar ele nu formează neapărat o componentă tare conexă.

Pentru a vizita toate nodurile grafului, se aplică algoritmul DFS repetat pentru noduri sursă care nu au fost încă vizitate, până când nu mai rămân noduri nevizitate.

Pentru un graf G cu n vârfuri și m muchii, traversarea DFS durează $O(n+m)$. Ca urmare, există algoritmi $O(n+m)$ bazați pe DFS pentru următoarele probleme:

- gasirea unui drum intre doua noduri
- gasirea unui ciclu intr-un graf
- testarea daca un graf este sau nu bipartit
- aflarea componentelor tare conexe
- sortarea topologica

In practica se foloseste atat implementarea iterativa bazata pe stiva, cat si implementarea recursiva. In acest laborator se vor prezenta cate o varianta pentru ambele strategii.

8.2.2 DFS iterativ

Pasii de realizat in algoritmul iterativ de parcurge in adancime sunt:

1. Se trece într-o stiva vidă nodul (vârful) de pornire;
2. Se trece extrage din stiva câte un nod, care este prelucrat (vizitat) și apoi se adaugă toate nodurile adiacente lui care încă nu au fost vizitate;
3. Se repetă pasul 2 până când stiva devine vidă.

Algoritmul iterativ (pseudocod) este următorul:

```
enum {WHITE, GRAY, BLACK};

DFS_ITERATIVE(v) // v - start node
            Stack S; // node stack, initially empty
            push(S, v); // push the start node onto the stack

            while S is not empty
                v = pop(S);      // pop an element from the stack, v
                if color[v] == WHITE
                        color[v] = GRAY;        // if v hasn't been discovered, mark it
                        process node v; // process the node, e.g. print its label, etc.
                        for ( each node w adjacent to v )  // traverse its neigbours
                                if ( color[ w ] == WHITE ) // for each unvisited node, w
                                        push(S, w); // push w onto the stack to await processing
```

O trasare a pașilor relevanți pentru DFS iterativ este prezentată în figura 8.1. Presupunand ca nodurile apar in listele de adiacenta in ordine lexicografica, in ce ordine se vor parcurge vecinii unui nod, prin aceasta strategie?

Ex. 1 — Implementati algoritmul DFS iterativ folosind ca metoda de reprezentare a grafurilor listele de adiacenta. Realizati apoi modificarile necesare pentru ca vecinii unui nod sa fie parcursi in ordinea in care apar in lista de adiacenta (si nu invers, asa cum se parcurg in varianta din pseudocodul dat). Totodata, modificati codul astfel incat in parcurgere sa se construiasca in mod corect si arborele de parcurgere in adancime (i.e. adaugati partea care seteaza parintele fiecarui nod in parcurgere - i.e. popularea vectorului π).

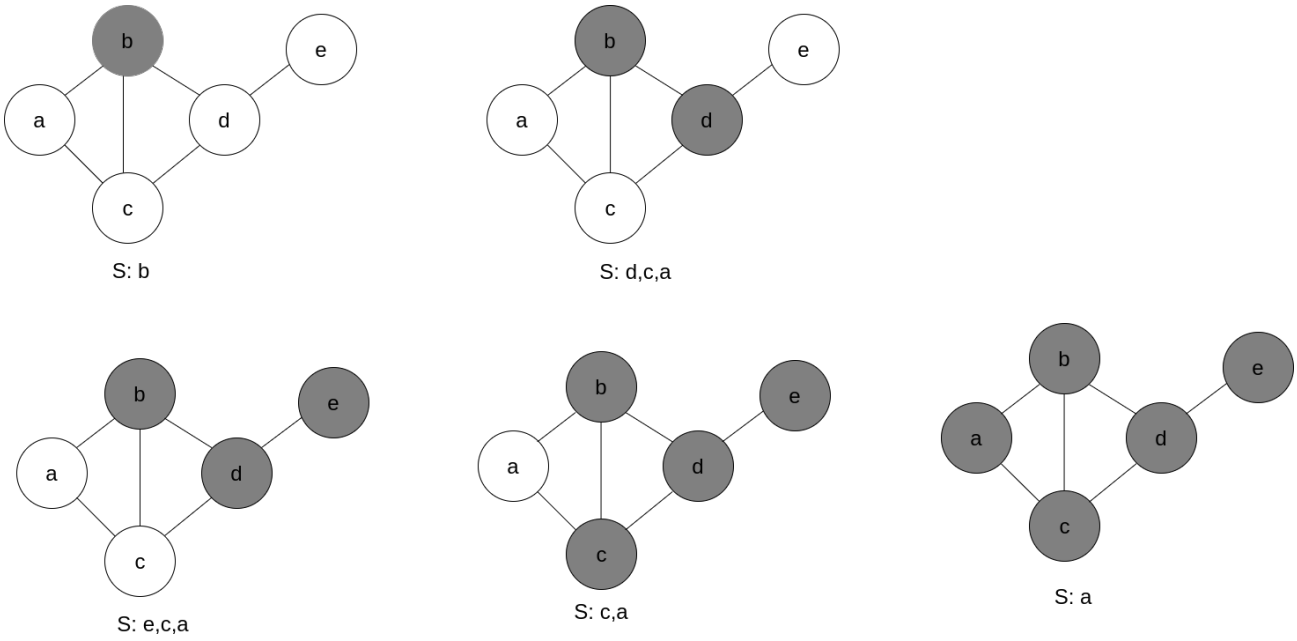

Figure 8.1: Cautarea în adâncime pe un graf: b, a, c, d, e

8.2.3 DFS recursiv (cu colorarea nodurilor si marcaje de timp)

La parcurgerea in adancime se pot inregistra o serie de informatii legate de ordinea de parcurgere a nodurilor - marcajele de timp (de descoperire/finalizare a unui nod). Aceste informatii pot fi utile in multi algoritmi bazati pe parcurgerea in adancime. Totodata, starea unui nod in cadrul parcurgerii poate fi de fapt caracterizata prin trei valori diferite: nedescoperit, in curs de parcurgere (vecinii sai sunt in curs de parcurgere) si finalizat. Aceste trei stari sunt marcate in cadrul parcurgerii prin culori. Culoarea unui nod poate fi:

- **ALB** - nodul nu a fost descoperit
- **GRI** - nodul a fost descoperit, si suntem in proces de parcurgere a vecinilor
- **NEGRU** - nodul a fost descoperit si toate nodurile la care se poate ajunge din nodul curent au fost descoperite (tot surabrorele sau a fost construit)

Asadar, in timpul parcurgerii, un nod va trece prin urmatoarele stari, in ordine: ALB->GRI->NEGRU.
Pentru a inregistra toate aceste informatii, vom avea urmatoarele siruri (de dimensiunea numarului de noduri din graf):

- π memoreaza arborele de parcurgere in adancime a componentei corespunzatoare nodului de pornire
- d memoreaza timpul la care a fost descoperit un nod (timpul la care nodul a devenit gri)
- f memoreaza momentul de timp cand un nod a fost explorat in totalitate (timpul la care nodul a devenit negru)
- *color* memoreaza culoarea curenta a nodurilor

Algoritmul recursiv de parcurgere, care inregistreaza toate aceste informatii, este:

```
enum {WHITE, GRAY, BLACK};

// the algorithm has as parameters: v = start node; t is a global time, initially 0

DFS_VISIT(v)
  time = time + 1
  d[v] = time
  color[v] = GRAY
  for ( each node w adjacent cu v)
      if color[w] == WHITE
        π[w] = v;          // mark v as w's parent
        DFS_VISIT(w)
  color[v] = BLACK
  time = time + 1
  f[v] = time
```

Ex. 2 — Implementati algoritmul DFS recursiv cu colorarea nodurilor si marcaje de timp folosind ca metoda de reprezentare a grafurilor listele de adiacenta.

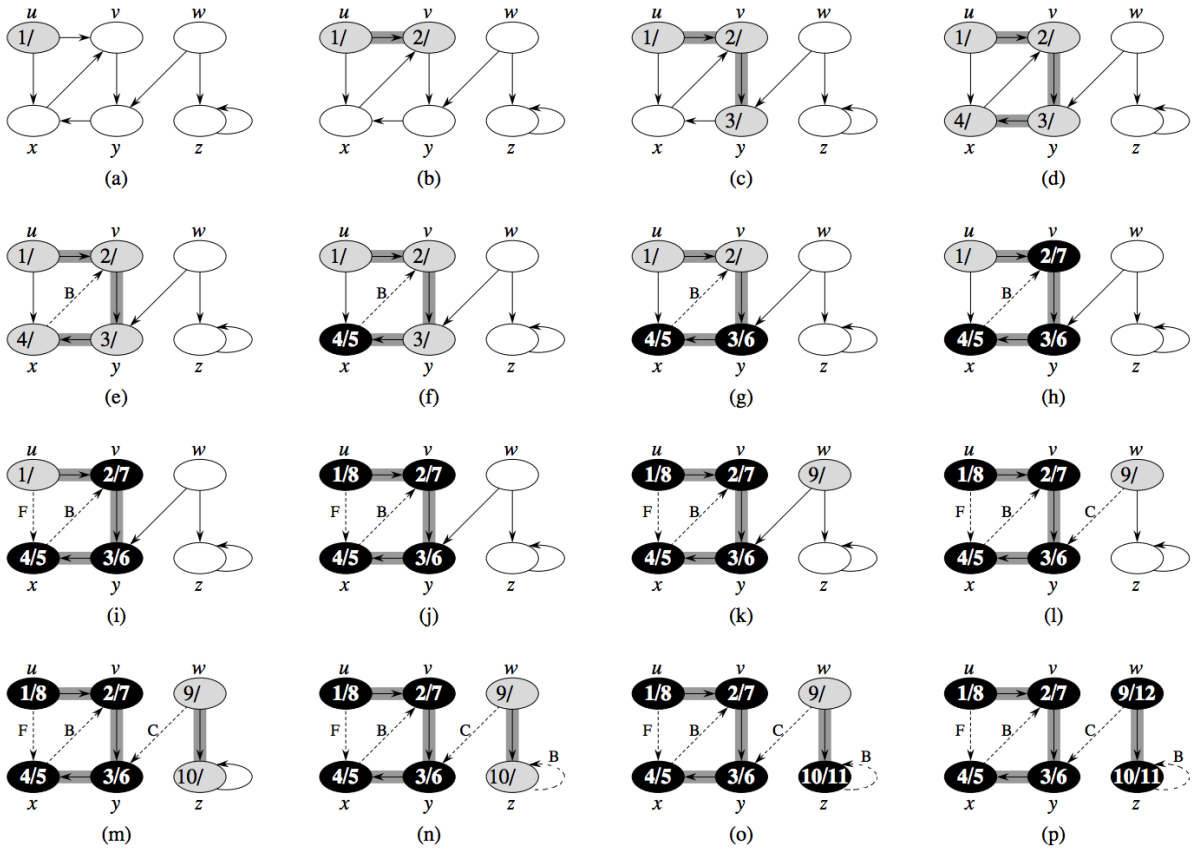

Figure 8.2: Cautarea în adâncime folosind colorare si marcaje de timp: x, y, v, u, z, w

8.2.4 Sortarea topologica

Sortarea topologica a unui graf directionat G = (V, E) este o ordonare lineara a tuturor nodurilor astfel incât daca G contine o muchie (u, v), atunci u apare înaintea lui v în ordonare. (daca graful nu este aciclic atunci aceasta ordonare este imposibila). O sortare topologica a unui graf poate fi vazuta ca o ordonare a nodurilor într-o linie orizontala astfel incât toate muchile directionate sa mearga de la stânga la dreapta.

```
SORTARE-TOPOLOGICA(G)
        apelam DFS(G) pentru a calcula timstamp-urile de final u.f pentru fiecare nod u
        cand un nod este explorat in totalitate, el este inserat intr-o lista inlantuita
        returnam lista inlantuita cand toate nodurile au fost explorate
```

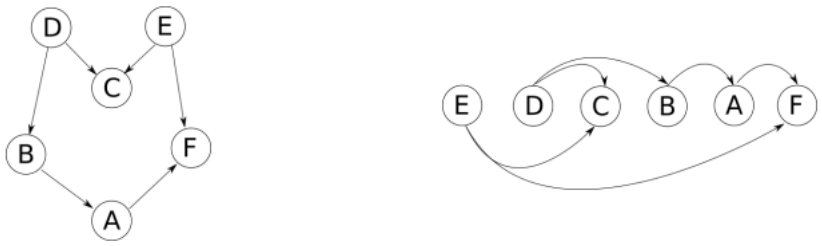

Figure 8.3: Sortatea topologica a unui graf

8.3 Mersul lucrării

8.3.1 Probleme opționale

1. *Sa se implementeze un algoritm care sa sorteze topologic nodurile unui graf orientat; daca o astfel de ordonare nu este posibila, sa semnaleze acest lucru.
2. Sa se implementeze un algoritm care sa determine daca un graf contine sau nu cicluri. Daca graful contine cicluri, sa se returneze primul ciclu gasit.
3. *Sa se implementeze un algoritm care sa gaseasca elementele puternic conexe a unui graf.
4. *Sa se implementeze un algoritm care sa gaseasca punctele de articulatie ale unui graf neorientat.