Grafuri

1. **Generalitati**
2. **Reprezentarea grafurilor**
3. **Explorarea sistematica a grafurilor**

1. Generalitati

Un graf este un obiect G = (V, E) unde V este o multime finita (|V|=n), iar E o submultime a produsului cartezian VxV (|E|=m). V se numeste multimea nodurilor (varfurilor) si E multimea muchiilor (arcelor). Fara a restrange generalitatea se poate presupune ca V={1,2,...,n}. Rezulta ca E este o submultime {(i,j) / i,j din V }.

Daca (i,j) = (j,i) pentru oricare pereche (i,j) din E, graful se numeste neorientat sau simplu: graf. Altfel, graful se numeste orientat (digraf).

Observatie: Un arbore este un caz particular de graf.

Fie G=(V,E) un digraf si un arc (i,j) din E:
- i se numeste predecesor al lui j;
- j se numeste succesor al lui i;
- (i, j) este incident cu i si j; nodurile i, j sunt adiacente;
- E_i = { k / (k,i) in E} multimea de vecini la intrare;
- E'_i = { j / (i,j) in E} multimea de vecini la iesire;
- E_i E'_i multimea vecinilor.
- grad_intrare(i) = in_degree(i) = $|E_i|$
- grad_iesire(i) = out_degree(i) = $|E'_i|$

Pentru un graf neorientat (G - neorientat), $E_i = E'_i$ si degree(i)= $|E_i|$.

Se numeste *drum orientat* intr-un graf de la x la y secventa de noduri
P = (i(1) = x, i(2), ... , i(n) = y), cu (i(k),i(k+1)) in E.

Daca x=y, P se numeste *ciclu*.

Un graf G se numeste *conex* daca oricare ar fi x, y din V, exista P un drum de la x la y.
Un digraf D se numeste tare conex daca oricare ar fi x, y din V, exista P un drum orientat de la x la y si un drum orientat P' de la y la x.

Un graf G este complet daca fiecare nod este conectat cu oricare din celelalte noduri:

$$E = VxV \setminus \{(i,i)/\ i \text{ in } V\}$$

Daca fiecare muchie a grafului are asociata o pondere, atunci graful se numeste *ponderat*.

2. Reprezentarea grafurilor

Un graf G=(V,E) este caracterizat de multimea nodurilor V, de dimensiune n si multimea arcelor E, de dimensiune m. Aceste informatii trebuie memorate intr-o structura care sa raspunda urmatoarelor cerinte:

- Sa fie usor de manevrat de catre algoritmii care prelucreaza graful. Principalele doua operatii cu care se opereaza in grafuri sunt:
  - Testul daca intre doua noduri i si j exista arc.
  - Parcurgerea tuturor nodurilor adiacente unui nod dat i.
- Sa foloseasca eficient spatiul de memorare.

Exercitiu: Desenati pe hartie graful reprezentat de urmatoarele date:

n=7
m=12
V={1,2,3,4,5,6,7}
E={(1,2), (1,3), (2,4), (2,5), (3,5), (4,1), (4,7), (5,4), (5,6), (5,7), (6,7), (7,5) }

Principalele doua metode de reprezentare a grafurilor sunt:

[A] Matricea de adiacenta

Matricea de adiacenta a grafului este o matrice de dimensiune n x n cu elemente 0 sau 1, definita astfel:

A[i,j] = 1, daca exista arcul (i,j)
A[i,j] = 0, daca nu exista arcul (i,j)

Pentru testul daca intre doua noduri exista arc, in cazul matricii de adiacenta este suficienta testarea valorii elementului A[i,j].

Parcurgerea tuturor elementelor adiacente unui nod dat i inseamna parcurgerea liniei

A[i,k] cu k=1,n

din matrice, selectand elementele egale cu 1.

Spatiul de memorare ocupat este de dimensiune n x n si nu depinde de numarul de arce din graf. Ca urmare, atunci cand numarul de arce este mult mai mic decat n x n (caz foarte frecvent), acesta reprezentare este ineficienta.

In cazul grafurilor ponderate, valorile din matricea de adiacenta sunt:

$A[i,j] = c_{ij}$, daca exista arcul (i,j)

A[i,j] = 0, daca nu exista arcul (i,j)

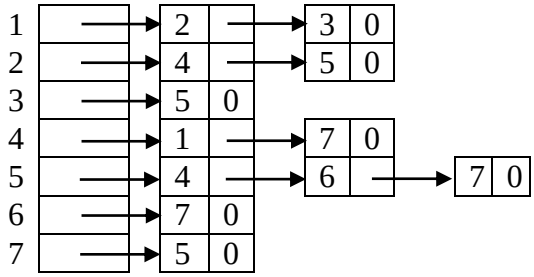
unde c_{ij} indica costul arcului (i,j).

[B] Liste de adiacenta

Se foloseste un vector V, de dimensiune n, de liste, continand, pentru fiecare nod i, nodurile adiacente, adica:

V[i] este o lista care contine toate nodurile j pentru care exista arc (i,j).

Pentru exemplul nostru:



Pentru testul daca intre doua noduri i si j exista arc, in cazul listelor de adiacenta, trebuie cautat nodul j in lista de adiacenta a nodului i.

Parcurgerea tuturor elementelor adiacente unui nod dat i insemna parcurgerea listei de adiacenta a nodului i.

Spatiul de memorare ocupat este:

n x sizeOf(pointer) + m x sizeOf(element de lista)

De obicei spatiul de memorie ocupat este mult mai mic decat in cazul matricii de adiacenta.

In cazul grafurilor ponderate, structura unui nod j din lista nodului i va contine si un camp care sa indice costul drumului de la i la j.

Observatie: Ordinea in care apar nodurile in listele de adiacenta nu este esentiala. Pentru a putea testa programele si pentru a verifica corectidudinea rezultatelor, vom prefera ca nodurile sa fie in ordine crescatoare in fiecare din listele de adiacenta.

3. Explorarea sistematica a unui graf

Se pune problema generarii listei de noduri accesibile dintr-un nod i al grafului. Pentru a evita revenirea intr-un nod vizitat, se asociaza acestuia o eticheta care sa indice acest lucru. Metodele de explorare a grafului sunt bazate pe acest principiu, deosebindu-le doar modul in care stocheaza si revin asupra unei directii necercetate.
Exista doua metode de explorare :

- Explorarea in adancime (DFS – Depth First Search)
- Explorarea in latime (BFS – Breadth First Search)

a) Explorarea in adancime

Cand este vizitat un nod v, aceasta parcurgere urmareste un drum in graf pana la sfarsitul acestuia, iar apoi se va intoarce si va urmari o alta ramura, s.a.m.d..

Corespunde unei traversari in preordine a unui arbore binar.

Parcurgerea DFS a arborelui din exemplul precedent, plecand din nodul i=1 va produce urmatoarea secventa de noduri:

1, 3, 5, 6, 7, 4, 2

Algoritmul nerecursiv :

```
DFS_iterativ(A, L, M , i)
//A – matricea de adiacenta
//L – lista nodurilor care formeaza parcurgerea din i
//M – vector de etichete vizitat/nevizitat – initial 0
//i – nodul de start
//S – stiva
push(S,i)
while S ≠ empty do
      j := pop(S)
      if M[j] = 0 then
            insert(L, j)
            M[j] := 1
            for k = 1 to n do
                  if A[j,k] = 1 then
                        push(S,k)
                  end-if
            end-for
      end-if
end-while
```

Algoritmul recursiv :

```
DFS_recursiv(A, L, M, i)
//A – matricea de adiacenta
//L – lista nodurilor care formeaza parcurgerea din i
//M – vector de etichete vizitat/nevizitat – initial 0
//i – nodul de start
insert(L, i)
M[i] := 1
for k = 1 to n do
     if A[i,k] = 1 then
          if M[k] = 0 then
               DFS_recursiv(A, L, M, k)
          end-if
     end-if
end-for
```

b) Explorarea in latime

Cand este vizitat un nod v, aceasta parcurgere urmareste in graf toti vecinii directi ai acestui nod, apoi vecinii vecinilor, s.a.m.d..

Parcurgerea BFS a arborelui din exemplul precedent, plecand din nodul i=1 va produce urmatoarea secventa de noduri:

1, 2, 3, 4, 5, 7, 6

Implementarea iterativa este similara cu cea a parcurgerii in adancime, doar ca in loc de stiva se foloseste o coada.

TEMA

1. Se da un graf orientat ponderat prin urmatoarele informatii :

   ```
   n m
   i1 j1 cost_i1_j1
   i2 j2 cost_i2_j2
   …
   ```

Sa se construiasca reprezentarea grafului prin matricea de adiacenta.

Indicatii:

- Se citeste `n` (numarul de noduri din graf)
- Se aloca o matrice de dimensiune `n` x `n` si se initializeaza cu valori 0

- Se citesc pe rand cele `m` muchii; pentru fiecare muchie `(i,j)` se scrie valoarea `cost_i_j` de pe pozitia `(i,j)` din matricea de adiacenta

2. Sa se scrie o functie recursiva pentru afisarea nodurilor unui graf in ordinea DFS.

Indicatii:

- Functia de explorare DFS va primi ca argumente: matricea de adiacenta ce reprezinta graful, n – numarul de noduri din graf, i – nodul de start, M – vectorul de marcaje, L – lista nodurilor ce formeaza parcurgerea din i

3. Sa se scrie functia iterativa pentru parcurgerea DFS.

4. Sa se scrie functiile pentru parcurgerea BFS, iterativ, respectiv recursiv.

NOTARE
1 – 2p
2 – 2p
3 – 2p
4 – 3p/3p