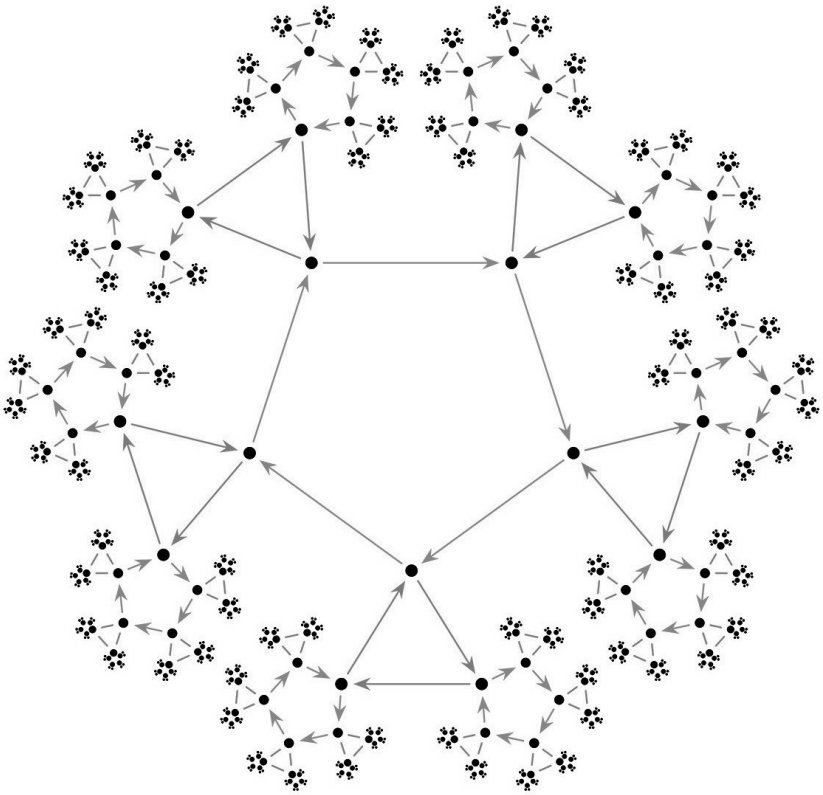
INTRODUCERE ÎN TEORIA GRAFURILOR

CÂTEVA CONSIDERAȚII ASUPRA CONCEPTELOR ESENȚIALE CARE STAU LA BAZA TEORIEI GRAFURILOR ȘI ALGORITMILOR FUNDAMENTALI UTILIZAȚI ÎN REZOLVAREA PROBLEMELOR COMPUTAȚIONALE



Un graf Cayley pentru $C_3 * C_5$.

DE

SÎRBU MATEI-DAN

hello@msirbu.eu

BRAȘOV, DECEMBRIE 2020

Cuprins

Capitolul 1

Despre conceptul de *graf*

1.1 Terminologie

Definim în mod informal un graf ca fiind o colecție de „noduri" unite prin „muchii", ca în exemplul [1] următor:

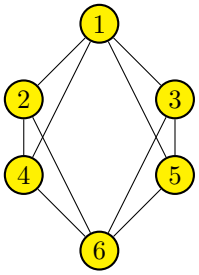


Figura 1.1: Un graf neorientat oarecare.

Definiția 1. *Un* ***graf*** *este o pereche* (V, E)*, unde* V *este o mulțime finită de elemente, numite noduri (**V**ertices), iar* E *este o mulțime finită de perechi de noduri, numite muchii (**E**dges).*

Dacă perechile din mulțimea E sunt ordonate, atunci spunem că graful este ***orientat***, sau ***digraf***; în caz contrar, graful este ***neorientat***. De asemenea, două noduri unite de o muchie se numesc ***adiacente***. Conceptul analog muchiilor aplicabil grafurilor orientate este ***arcul***.

1.2 Dimensiunile unui graf

De obicei, notăm cu n numărul de noduri ale unui graf; mai precis, $n = |V|$. Cu m vom nota numărul de muchii, adică $m = |E|$. În graful din figura 1.1, n este 6, iar m este 10.

Teorema 1. *Dacă un graf neorientat are* n *noduri, atunci* ***numărul total de grafuri neorientate*** [2] *care se pot forma cu aceste noduri este* $g = 2^{C_n^2}$.

Teorema 2. *Graful* ***complet****, graful care are toate muchiile posibile, conține* $m = \frac{n(n-1)}{2}$ *muchii* [2]*, dacă este neorientat.*

Spunem despre un nod că este ***izolat*** dacă nu aparține niciunei muchii. Întrucât nodurile izolate sunt inutile în majoritatea aplicațiilor, presupunem că nu există astfel de noduri; în acest caz, știm despre numărul de muchii că este $m \geq \frac{n}{2}$.

Astfel, deducem, utilizând simbolul O al lui Landau (notația *big-O*), că în general, $m = O(n^2)$, iar în majoritatea aplicațiilor, $m = \Omega(n)$, limite care sunt aplicabile și digrafurilor. Din punct de vedere terminologic, grafurile cu $m = \Theta(n)$ se numesc ***rare***, iar cele cu $m = \Theta(n^2)$ sunt ***dense*** [1].

1.3 Conexitate

Definiția 2. *Un* ***subgraf*** $G' = (V', E')$ *este un subgraf al lui* $G = (V, E)$ *dacă* $V' \subseteq V$ *și* $E' \subseteq E$.

Definiția 3. *Un* ***lanț*** *este o succesiune de noduri* $v_0, v_1, \ldots, v_l$, $l \geq 0$, *cu* $(v_i, v_{i+1}) \in E$ *pentru* $i = \overline{0,\ l-1}$. *Analog, în cazul grafurilor orientate, această succesiune de noduri se numește* ***drum****.*

Un lanț este ***simplu*** dacă nu trece de două ori prin aceeași muchie. În caz contrar, se numește lanț ***compus***. Este de remarcat faptul că un lanț poate avea lungime nulă.

Definiția 4. *Un* ***ciclu*** *este un drum cu* $l \geq 3$, $v_0 = v_l$ *cu toate nodurile și muchiile distincte.*

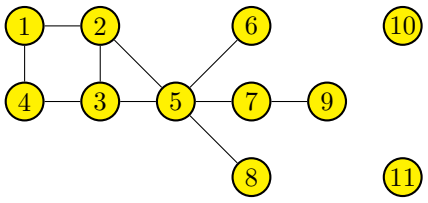
Definiția 5. ***Gradul*** *unui nod* v_k *al grafului* G *este egal cu numărul muchiilor incidente cu nodul și se notează cu* $d(v_k)$.

Teorema 3. *Dacă graful* G *are* m *muchii și* n *noduri, atunci între* ***gradul nodurilor*** *și* ***numărul de muchii*** *există relația* [2] $\sum_{i=1}^{n} d(v_i) = 2m$.

În funcție de gradul nodurilor putem distinge câteva cazuri particulare [2]. Un ***nod terminal*** este incident cu o singură muchie, adică $d(v_k) = 1$. Un ***nod izolat*** nu este adiacent cu nici un alt nod al grafului, adică nu se găsește în extremitatea niciunei muchii; altfel spus, $d(v_k) = 0$. Un exemplu [2]:

Graful $G = (V, E)$ din figură este definit astfel:

- $V = \{1, 2, 3, \ldots, 11\}$
- $E = \{(1,2), (1,4), (2,3), (2,5), (3,4), (3,5), (5,6), (5,7), (5,8), (7,9)\}$



Despre graful G putem spune că:

- $d(v_5) = 5$, deoarece (5) are 5 muchii incidente: $(2,5), (3,5), (5,6), (5,7)$ și $(5,8)$.

- $d(v_9) = 1$, adică (9) este *nod terminal*, deoarece are o singură muchie incidentă: $(7, 9)$.
- $d(v_{10}) = 0$, adică (10) este *nod izolat*, deoarece nu are muchii incidente.

Definiția 6. *Un graf neorientat este* ***conex*** *dacă are un lanț care unește oricare două noduri.*

Definiția 7. *Un* ***arbore*** *este un graf neorientat conex fără cicluri.*

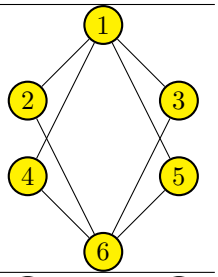
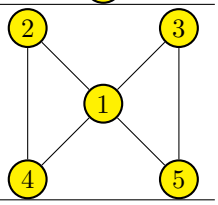
Definiția 8. *Un* ***arbore de acoperire*** [1] *este un subgraf al unui graf neorientat conex care conține toate nodurile din graf.*

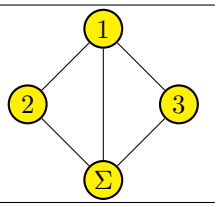
Figura 1.2: Arborele din fig. 1.1, respectiv arborele de acoperire al acestuia.

Într-un graf orientat, un arc (u, v) pornește din (u) și ajunge în (v); (v) se numește ***capul*** arcului, iar (u) este ***coada***. (v) este accesibil din (u) dacă și numai dacă există un drum de la (u) la (v). O remarcă importantă este aceea că ***buclele***, (u, u), sunt permise în grafurile orientate, dar nu și în cele neorientate.

1.4 Operații pe grafuri

În tabelul următor [1] vor fi enumerate operațiile de bază care pot fi executate pe grafuri, și rezultatul aplicării acestora pe graful din figura 1.1, notat aici cu G.

Operația pe graf	**Graful rezultat**
Ștergerea unei muchii din G înseamnă formarea grafului $G - e$. **Exemplu:** $e = \{(2, 4), (3, 5)\}$	
Ștergerea unei nod din G determină formarea grafului $G - v$; presupune păstrarea tuturor nodurilor și muchiilor din G cu excepția nodurilor din v și a muchiilor incidente acestora **Exemplu:** $v = \{6\}$; formează un „*graf papion*”	

Operația pe graf	**Graful rezultat**
Contractarea unei mulțimi de noduri S determină formarea grafului $G - S$, unde mulțimea S este înlocuită de un nou nod (Σ), adiacent tuturor vecinilor nodurilor din S **Exemplu:** $\Sigma = \{4, 5, 6\}$	

1.5 Reprezentarea grafurilor

Există mai multe moduri de reprezentare la nivel logic a unui graf [2], care pot fi implementate în memoria unui calculator, folosind diverse tipuri de structuri de date. Aceste reprezentări pot fi folosite în algoritmii care prelucrează grafuri și, implicit, în programele prin care vor fi implementați în calculator acești algoritmi. Printre modurile de reprezentare a unui graf se numără:

- reprezentarea prin ***matricea de adiacență***;
- reprezentarea prin ***lista muchiilor (arcelor)***;
- reprezentarea prin ***lista de adiacență (listele vecinilor)***;
- reprezentarea prin ***matricea costurilor***.

Fiecare reprezentare prezintă avantaje în ceea ce privește utilizarea eficientă a memoriei interne, în funcție de tipul grafului (cu noduri puține, dar cu muchii multe - sau cu noduri multe, dar cu muchii puține) și din punct de vedere al eficienței algoritmilor de prelucrare (în funcție de aplicație). În următoarele reprezentări se consideră că graful $G = (V, E)$ are n noduri și m muchii.

1.5.1 Matricea de adiacență

Definiția 9. ***Matricea de adiacență*** [2] *a unui graf este o matrice pătratică binară de ordinul n* $(A_{n,n})$*, ale cărei elemente* $a_{i,j}$ *sunt definite astfel:*

$$a_{i,j} = \begin{cases} 1, & \text{dacă } [i,j] \in E \\ 0, & \text{dacă } [i,j] \notin E \end{cases}.$$

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

Figura 1.3: Matricea de adiacență a grafului din fig. 1.1

În cazul grafurilor orientate putem diferenția gradul unui nod:

Definiția 10. ***Gradul intern*** *al unui nod v_i al grafului G este egal cu numărul arcelor care intră în nodul v_i și se notează cu $d^-(v)$.*

Definiția 11. ***Gradul extern*** *al unui nod v_i al grafului G este egal cu numărul arcelor care ies din nodul v_i și se notează cu $d^+(v)$.*

Pentru a calcula gradul extern al unui digraf reprezentat ca o matrice de adiacență vom utiliza următorul algoritm [1], care are complexitatea de timp $\Theta(n^2)$:

```
procedure GRAD-EXTERN(A, v)
    for v ← 1 to n do
        d[v] ← 0
        for w ← 1 to n do
            d[v] ← d[v] + A[v, w]
        end for
    end for
end procedure
```

Proprietăți

- Elementele de pe diagonala principală au valoarea 0. Din definiția grafului rezultă că orice muchie/arc (i, j) trebuie să respecte condiția $i \neq j$.

Pentru grafurile neorientate:

- Matricea de adiacență este o matrice simetrică față de diagonala principală, deoarece, dacă există muchia (i, j), atunci există și muchia (j, i).
- Suma elementelor matricei de adiacență este egală cu $2m$.
- Gradul unui nod i este egal cu suma elementelor de pe linia i.
- Nodurile adiacente nodului i sunt nodurile j pentru care elementele din linia i sunt egale cu 1. Mai pot fi definite ca nodurile j pentru care elementele din coloana i sunt egale cu 1.
- Numărul de vecini ai nodului i este egal cu gradul nodului.
- Muchia (i, j) a grafului reprezintă un element al matricei de adiacență care îndeplinește condiția $a_{i,j} = a_{j,i} = 1$

Pentru grafurile orientate:

- Suma elementelor matricei de adiacență ete egală cu m.
- Gradul extern al nodului i este egal cu suma elementelor de pe linia i.
- Gradul intern al nodului i este egal cu suma elementelor de pe coloana i.
- Succesorii nodului i sunt nodurile j pentru care elementele din linia i sunt egale cu 1.

- Predecesorii nodului i sunt nodurile j pentru care elementele din coloana i sunt egale cu 1.
- Nodurile adiacente nodului i sunt nodurile j pentru care elementele din linia i sau din coloana i sunt egale cu 1, sau reuniunea dintre mulțimea succesorilor și mulțimea predecesorilor nodului.
- Numărul de vecini ai nodului i este egal cu cardinalul mulțimii de noduroi adiacente nodului i.
- Arcul (i, j) al grafului reprezintă un element al matricei de adiacență care îndeplinește condiția $a_{i,j} = 1$.

1.5.2 Lista muchiilor (arcelor)

Definiția 12. ***Lista muchiilor*** [2] *unui graf este formată din m elemente care conțin, fiecare, câte o pereche de două noduri, v_i și v_j, care formează o muchie, adică pentru care $(v_i, v_j) \in E$.*

Lista de muchii care descrie graful din figura 1.1 este:

$$L = \{(1,2), (1,3), (1,4), (1,5), (2,4), (2,6), (3,5), (3,6), (4,6), (5,6)\}.$$

Proprietăți

Pentru grafurile neorientate:

- Graful unui nod i este egal cu numărul de apariții ale etichetei nodului în câmpurile vectorului de înregistrări.
- Nodurile adiacente nodului i sunt etichetele j din câmpul $L_{k,1}$, pentru care $L_{k,0} = i$, sau de pe al doilea câmp, pentru care $L_{k,2} = i$.
- Numărul de vecini ai nodului i este egal cu gradul nodului.

Pentru grafurile orientate:

- Gradul extern al nodului i este egal cu numărul de apariții ale etichetei nodului în primul câmp în vectorul de înregistrări.
- Gradul intern al nodului i este egal cu numărul de apariții ale etichetei nodului în al doilea câmp în vectorul de înregistrări.
- Succesorii nodului i sunt etichetele j din câmpul $L_{k,2}$ pentru care $L_{k,1} = i$.
- Predecesorii nodului i sunt etichetele j din câmpul $L_{k,1}$ pentru care $L_{k,2} = i$.
- Nodurile adiacente nodului i sunt date de reuniunea dintre mulțimea succesorilor și mulțimea predecesorilor nodului.

1.5.3 Lista de adiacență

Definiția 13. ***Lista de adiacență*** [2] *este formată din listele L_i $(1 \leq i \leq n)$ care conțin toți vecinii unui nod v_i la care se poate ajunge direct din nodul v_i, adică toate nodurile v_j pentru care $(v_i, v_j) \in E$.*

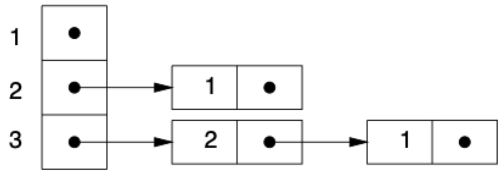


Figura 1.4: Listele de adiacență care definesc graful din fig. 1.1.

Proprietăți

Pentru grafurile neorientate:

- Numărul de elemente din toate listele simplu înlănțuite este egal cu $2m$.
- Lungimea listei de adiacență a nodului i este egală cu numărul de noduri ale listei ce are adresa nodului prim egală cu L_i.
- Graful unui nod i sunt nodurile a căror etichetă apare în lista ce are adresa nodului prim egală cu L_i.
- Numărul de vecini ai nodului i este egal cu gradul nodului.
- Muchia (i, j) a grafului reprezintă nodul i și un nod j din lista ce are adresa nodului prim egală cu L_i.

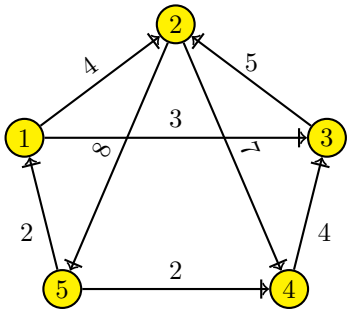
Pentru grafurile orientate:

- Numărul de elemente din toate listele simplu înlănțuite este egal cu m.
- Lungimea listei de adiacență i este egală cu numărul de noduri ale listei care are adresa nodului prim egală cu L_i.
- Gradul extern al nodului i este egal cu lungimea listei de adiacență a nodului.
- Gradul intern al nodului i este egal cu numărul de apariții ale etichetei nodului în toate listele simplu înlănțuite.
- Succesorii nodului i sunt nodurile a căror etichetă apare în lista ce are adresa nodului prim egală cu L_i.
- Predecesorii nodului i sunt nodurile j în ale căror liste, ce au adresa nodului prim egală cu L_j, apare nodului i.
- Nodurile adiacente nodului i sunt date de reuniunea dintre mulțimea succesorilor și mulțimea predecesorilor nodului.
- Numărul de vecini ai nodului i este egal cu cardinalul mulțimii de noduri adiacente nodului i.
- Arcul (i, j) al grafului reprezintă nodul i și un nod j din lista ce are adresa nodului prim egală cu L_i.

1.5.4 Matricea costurilor

Definiția 14. ***Graful ponderat****, este graful $G = (V, E)$ pentru care s-a definit o funcție $f : E \to \mathbb{R}_+$ care asociază fiecărei muchii/arc e un număr real pozitiv (care poate avea semnificația de cost, distanță, timp, durată), numită în general* ***costul muchiei****, sau* ***funcția cost****.*

Definiția 15. ***Matricea costurilor*** [2] *unui graf este o matrice pătratică de dimensiune n ($A_{n,n}$) ale cărei elemente $a_{i,j}$ sunt definite astfel încât să pună în evidență costul asociat fiecărei muchii/arc.*



	1	2	3	4	5
1	0	4	3	∞	∞
2	∞	0	∞	7	8
3	∞	5	0	∞	∞
4	∞	∞	4	0	∞
5	2	∞	∞	2	0

Figura 1.5: Un graf ponderat, respectiv matricea sa de costuri.

Pentru a crea matricea costurilor trebuie să se citească pentru fiecare muchie (arc) nodurile de la extremități și costul asociat fiecărei muchii (arc). Aceste informații se pot citi de la tastatură sau dintr-un fișier. Algoritmul pentru crearea matricii costurilor este:

1. Se inițializează matricea astfel: toate elementele de pe diagonala principală cu valoarea 0, iar restul elementelor cu valoarea corespunzătoare pentru ∞ ($-\infty$).

2. Se actualizează matricea cu informațiile despre costurile asociate muchiilor (arcelor) astfel: pentru fiecare muchie (arc) (i, j) cu costul c, elementului $a_{i,j}$ i se va atribui valoarea costului c.

Capitolul 2

Algoritmi esențiali

O mare varietate de probleme se formulează în termeni de grafuri. Pentru a le rezolva, de multe ori trebuie să explorăm un graf, adică să consultăm (vizităm) vârfurile sau muchiile grafului respectiv. Uneori trebuie să consultăm toate vârfurile sau muchiile, alteori trebuie să consultăm doar o parte din ele.

În acest capitol, introducem câteva tehnici care pot fi folosite atunci când nu este specificată o anumită ordine a consultărilor. Indiferent de obiectivul urmărit, explorarea se realizează pe baza unor algoritmi de parcurgere, care asigură consultarea sistematică a vârfurilor sau muchiilor grafului respectiv.

2.1 Depth-First Search

Fie $G = (V, E)$ un graf orientat sau neorientat, ale cărui vârfuri dorim să le consultăm. Presupunem că avem posibilitatea să marcăm vârfurile deja vizitate în tabloul global *marca*. Inițial, nici un vârf nu este marcat. Pentru a efectua o parcurgere în adâncime, alegem un vârf oarecare, $v \in V$, ca punct de plecare și îl marcăm. Dacă există un vârf w adiacent lui v (adică, dacă există muchia (v, w) în graful G) care nu a fost vizitat, alegem vârful w ca noul punct de plecare și apelăm recursiv procedura de parcurgere în adâncime. La întoarcerea din apelul recursiv, dacă există un alt vârf adiacent lui v care nu a fost vizitat, apelăm din nou procedura etc.

Când toate vârfurile adiacente lui v au fost marcate, se încheie consultarea începută în v. Dacă au rămas vârfuri în V care nu au fost vizitate, alegem unul din aceste vârfuri și apelăm procedura de parcurgere. Continuăm astfel, până când toate vârfurile din V au fost marcate.

Algoritmul de parcurgere (DFS) [3] a grafurilor în adâncime este prezentat mai jos:

```
procedure PARCURGE(G)                                  ▷ Parcurgerea în adâncime
    for fiecare v ∈ V do
        marca[v] ← nevizitat
    end for
    for fiecare v ∈ V do
        if marca[v] = nevizitat then
            ADÂNCIME(v)
        end if
    end for
end procedure

procedure ADÂNCIME(v)
    marca[v] ← vizitat          ▷ Vârful v nu a fost vizitat, acum se marchează
    for fiecare vârf w adiacent lui v do
        if marca[w] = nevizitat then
            ADÂNCIME(w)
        end if
    end for
end procedure
```

Dacă reprezentăm graful prin liste de adiacență, adică prin atașarea la fiecare vârf a listei de vârfuri adiacente lui, atunci numărul total de testări este m, dacă graful este orientat, și $2m$, dacă graful este neorientat. Algoritmul necesită un timp [3] în $\Theta(n)$ pentru apelurile procedurii ADÂNCIME și un timp în $\Theta(m)$ pentru inspectarea mărcilor. Timpul de execuție este deci în $\Theta(\max(m, n)) = \Theta(m + n)$. Dacă reprezentăm graful printr-o matrice de adiacență, se obține un timp de execuție în $\Theta(n^2)$.

2.2 Breadth-First Search

Procedura de parcurgere în adâncime (BFS) [3], atunci când ajunge la un vârf v oarecare, explorează prima dată un vârf w adiacent lui v, apoi un vârf adiacent lui w etc. Pentru a efectua o parcurgere *în lățime* a unui graf, aplicăm următorul principiu: atunci când ajungem într-un vârf oarecare v nevizitat, îl marcăm și vizităm apoi toate vârfurile nevizitate adiacente lui v, apoi toate vârfurile nevizitate adiacente vârfurilor adiacente lui v etc. Spre deosebire de parcurgerea în adâncime, parcurgerea în lățime nu este în mod natural recursivă.

Pentru a putea compara aceste două tehnici de parcurgere, vom da pentru început o versiune nerecursivă pentru procedura ADÂNCIME. Versiunea se bazează pe utilizarea unei stive. Presupunem că avem funcția TOP care returnează ultimul vârf inserat în stivă, fără să îl șteargă. Folosim și funcțiile PUSH și POP specifice operațiilor pe stivă.

```
procedure ADÂNCIME-ITERATIV(v)
    S ← stivă vidă
    marca[v] ← vizitat
    PUSH(v, S)
    while S nu este vidă do
        while ∃ un vârf w adiacent lui TOP(S) a.î. marca[w] = nevizitat do
            marca[w] ← vizitat
            PUSH(w, S)
        end while
        PUSH(w, S)
    end while
end procedure
```

Pentru parcurgerea în lățime, vom utiliza o coadă și funcțiile INSERT-QUEUE și DELETE-QUEUE. Iată acum algoritmul:

```
procedure LĂȚIME(v)
    C ← coadă vidă
    marca[v] ← vizitat
    INSERT-QUEUE(v, C)
    while C nu este vidă do
        u ← DELETE-QUEUE(C)
        for fiecare vârf w adiacent lui u do
            if marca[w] = nevizitat then
                marca[w] ← vizitat
                INSERT-QUEUE(w, C)
            end if
        end for
    end while
end procedure
procedure PARCURGE(G)
    for fiecare v ∈ V do
        marca[v] ← nevizitat
    end for
    for fiecare v ∈ V do
        if marca[v] = nevizitat then
            ADÂNCIME-ITERATIV(v) sau LĂȚIME(v)
        end if
    end for
end procedure
```

Analiza eficienței [3] algoritmului de parcurgere în lățime se face la fel ca pentru parcurgerea în adâncime. Pentru a parcurge un graf cu n vârfuri și m muchii timpul este în $\Theta(n+m)$, dacă reprezentăm graful prin liste de adiacență, sau $\Theta(n^2)$, dacă reprezentăm graful printr-o matrice de adiacență. Parcurgerea în lățime este folosită de obicei atunci când se explorează parțial anumite grafuri infinite, sau când se caută cel mai scurt drum dintre două vârfuri.

2.3 Determinarea costului minim/maxim

Pentru determinarea drumului cu costul minim (maxim) [2] între două noduri ale unui graf se pot folosi:

- algoritmul Roy-Floyd;
- algoritmul Dijkstra.

Ambii algoritmi folosesc principiul enunțat prin *teorema lui Bellman*: drumul optim, cel cu costul minim, respectiv maxim, între două noduri oarecare i și j conține numai drumuri parțiale optime, cu costuri minime, respectiv maxime, care trec prin alte noduri ale grafului. Altfel spus, dacă drumul optim dintre două noduri oarecare i și j trece printr-un nod k, atunci și drumurile de la i la k și de la k la j sunt optime. Cei doi algoritmi diferă prin modul în care se identifică nodurile intermediare k.

2.3.1 Algoritmul Roy-Floyd

Algoritmul Roy-Floyd [2] folosește următorul principiu: găsirea drumului optim între două noduri oarecare i și j prin descoperirea drumurilor optime care îl compun și care trec prin nodurile l se face prin transformarea matricei costurilor. Matricea trece prin n transformări, în urma cărora fiecare element $a_{i,j}$ va memora costul drumului minim dintre nodurile i și j.

---

procedure ROY-FLOYD(a, n)
  for $k \in \overline{1,n}$ **do**
    for $(i,j), i \in \overline{1,n}, j \in \overline{1,n}$ **do**
      if $a_{i,k} + a_{k,j} < a_{i,j}$ **then**
        $a_{i,j} \leftarrow a_{i,k} + a_{k,j}$
      end if
    end for
  end for
end procedure

Pentru graful din figura 1.5, matricea costurilor suferă următoarele cinci transformări. La fiecare transformare, dacă drumul de la nodul i la nodul j are costul mai mare decât costul drumurilr care trec prin nodul intermediar k, atunci elementului $a_{i,j}$ i se va atribui valoarea $a_{i,k} + a_{k,j}$.

$k = 1$

	1	2	3	4	5
1	0	4	3	∞	∞
2	∞	0	∞	7	8
3	∞	5	0	∞	∞
4	∞	∞	4	0	∞
5	2	6	5	2	0

$k = 2$

	1	2	3	4	5
1	0	4	3	11	12
2	∞	0	∞	7	8
3	∞	5	0	12	13
4	∞	∞	4	0	∞
5	2	6	5	2	0

$k = 3$

	1	2	3	4	5
1	0	4	3	11	12
2	∞	0	∞	7	8
3	∞	5	0	12	13
4	∞	9	4	0	17
5	2	6	5	2	0

$k = 4$

	1	2	3	4	5
1	0	4	3	11	12
2	∞	0	11	7	8
3	∞	5	0	12	13
4	∞	9	4	0	17
5	2	6	5	2	0

$k = 5$

	1	2	3	4	5
1	0	4	3	11	12
2	10	0	11	7	8
3	15	5	0	12	13
4	19	9	4	0	17
5	2	6	5	2	0

Informațiile din matricea costurilor transformată prin algoritmul Roy-Floyd se ăpt folosi pentru a verifica dacă există drum cu costul minim între două noduri ale grafului, iar în caz afirmativ, se poate afișa lungimea lui și se poate descoperi drumul.

Algoritmul de transformare a matricei costurilor are ordinul de complexitate [2] $O(n \cdot n \cdot n) = O(n^3)$, deoarece fiecare structură repetitivă **for** se execută de n ori, iar structurile **for** sunt imbricate. Algoritmul de determinare a drumurilor cu costul minim din matricea costurilor transformată are ordinul de complexitate al algoritmului *divide et impera*: $O(n \log n)$. Ordinul algoritmului este $O(n^3) + O(n \log n) = O(n^3 + n \log n) = O(n^3)$.

2.3.2 Algoritmul Dijkstra

Algoritmul lui Dijsktra [2] construiește drumurile cu costul minim care pornesc de la un nod oarecare x, nodul sursă, până la fiecare nod din graful $G = (V, E)$, nodul destinație. Algoritmul întreține o mulțime cu nodurile care au fost deja selectate, S, și o coadă de priorități Q cu nodurile care nu au fost selectate încă: $Q = V - S$, astfel:

- Un nod y este declarat selectat atunci când s-a determinat costul final al drumului cu costul minim de la nodul sursă x la el. Selectarea unui nod nu este echivalentă cu găsirea drumului cu costul minim deoarece este posibil ca în urma calculării costului să rezulte că nu există drum de la nodul x la acel nod.
- În coada Q prioritatea cea mai mare o are nodul pentru care costul drumului are valoarea cea mai mică dintre toate costurile de drumuri care

pornesc de la nodul x la celelalte noduri selectate încă. La fiecare extragere a unui nod din coada de priorități Q, nodul este adăugat la mulțimea S, iar coada de priorități este reorganizată în funcție de acest nod. Pentru calcularea drumurilor de lungime minimă se întreține o mulțime D în care se memorează costul drumurilor de la nodul x la nodurile neselectate, costuri care se recalculează la fiecare extragere de nod.

Pentru implementarea algoritmului se folosesc trei vectori:

1. Vectorul s pentru mulțimea nodurilor selectate, definit astfel:

$$s_i = \begin{cases} 0, & \text{dacă nodul } i \text{ nu a fost selectat} \\ 1, & \text{dacă nodul } i \text{ a fost selectat} \end{cases}.$$

   Inițial, elementele vectorului s au valoarea 0, cu excepția elementului s_x care are valoarea 1. La terminarea execuției algoritmului, toate elementele din vectorul s vor avea valoarea 1. Nodurile i pentru care $s_i = 0$ se consideră că fac parte din coada de priorități Q.

2. Vectorul d conține costul drumurilor, astfel: d_i este costul drumului minim găsit la un moment dat de la nodul x la nodul i, cu $i \in \overline{1,n}$. Inițial $d_i = a_{x,i}$. La terminarea algoritmului, d_i este costul minim al drumului de la nodul x la nodul i.

3. Vectorul t memorează drumurile găsite între nodul x și celelalte noduri i ale grafului. Memorarea drumului se face prin legătura cu predecesorul are este definită astfel: p_i memorează nodul j care este predecesorul nodului i pe drumul de la x, cu excepția nodului sursă pentru care $p_x = 0$. Inițial, pentru toate nodurile i care nu au costul infinit (pentru care există un arc de la nodul x la nodul i), $p_i = x$; altfel $p_i = 0$.

Nodul i care se extrage din coada de priorități Q trebuie să îndeplinească următoarele condiții:

- $s_i = 0$
- $d_i = \min(d_j | 1 \leq j \leq n; s_j = 0)$.

Pentru graful din figura 1.5, considerând $x = 1$, algoritmul se execută astfel:
Inițial:

Vectorii	1	2	3	4	5
s	1	0	0	0	0
d	0	4	3	∞	∞
p	0	1	1	0	0

Drumul cu costul cel mai mic este cu nodul 3: $d_3 = 3$. Nodul 3 se va extrage din coada Q. Se analizează nodurile care rămân în coada de priorități.
Nodul 2: $d_3 + a_{3,2} = 3 + 5 = 8 > 4$. Nu se modifică nimic.
Nodul 4: $d_3 + a_{3,4} = 3 + \infty = \infty > 4$. Nu se modifică nimic.
Nodul 5: $d_3 + a_{3,5} = 3 + \infty = \infty > 4$. Nu se modifică nimic.

Vectorii	1	2	3	4	5
s	1	0	1	0	0
d	0	4	3	∞	∞
p	0	1	1	0	0

Drumul cu costul cel mai mic este cu nodul 2: $d_2 = 4$. Nodul 2 se va extrage din coada Q. Se analizează nodurile care rămân în coada de priorități.

Nodul 4: $d_2 + a_{2,4} = 4 + 7 = 11 < \infty$. Se modifică: $d_4 = 11$ și $p_4 = 2$.

Nodul 5: $d_2 + a_{2,5} = 4 + 8 = 12 < \infty$. Se modifică: $d_5 = 12$ și $p_5 = 2$.

Vectorii	1	2	3	4	5
s	1	1	1	0	0
d	0	4	3	11	12
p	0	1	1	2	2

Drumul cu costul cel mai mic este cu nodul 4: $d_4 = 11$. Nodul 4 se va extrage din coada Q. Se analizează nodurile care rămân în coada de priorități.

Nodul 5: $d_4 + a_{4,5} = 11 + \infty = \infty > 12$. Nu se modifică nimic.

Vectorii	1	2	3	4	5
s	1	1	1	1	0
d	0	4	3	11	12
p	0	1	1	2	2

Drumul cu costul cel mai mic este cu nodul 5: $d_5 = 15$. Nodul 5 se va extrage din coada Q. Coada este vidă și se termină execuția algoritmului.

Final:

Vectorii	1	2	3	4	5
s	1	1	1	1	1
d	0	4	3	11	12
p	0	1	1	2	2

Din datele care se găsesc în vectorii d și p la terrminarea algoritmului se obțin următoarele informații:

- d_i reprezintă costul minim al drumului de la nodul x la nodul i. De exemplu, pentru nodul 4 costul minim este 11.
- Din vectorul predecesorilor se reconstituie drumul cu costul minim de la nodul x la nodul i. De exemplu, pentru nodul 4: $p_4 = 2$, iar $p_2 = 1$. Drumul este $1 \to 2 \to 4$.

Bibliografie

[1] Harold N. Gabow. *Graph Theory Definitions*. The Department of Computer Science at the University of Colorado Boulder, 2008.

[2] Mariana Miloșescu. *Informatică intensiv: C++: manual pentru clasa a XI-a, ed. a 3-a*. Editura Didactică și Pedagogică, 2012.

[3] Răzvan Andonie, Ilie Gârbacea. *Algoritmi fundamentali. O perspectivă C++*. Editura Libris, Cluj-Napoca, 2012.