Grafuri. DFS. BFS

BUZATU GIULIAN & NIȚĂ ALEXANDROS

Grafuri neorientate

Graf neorientat: G = (V, E)

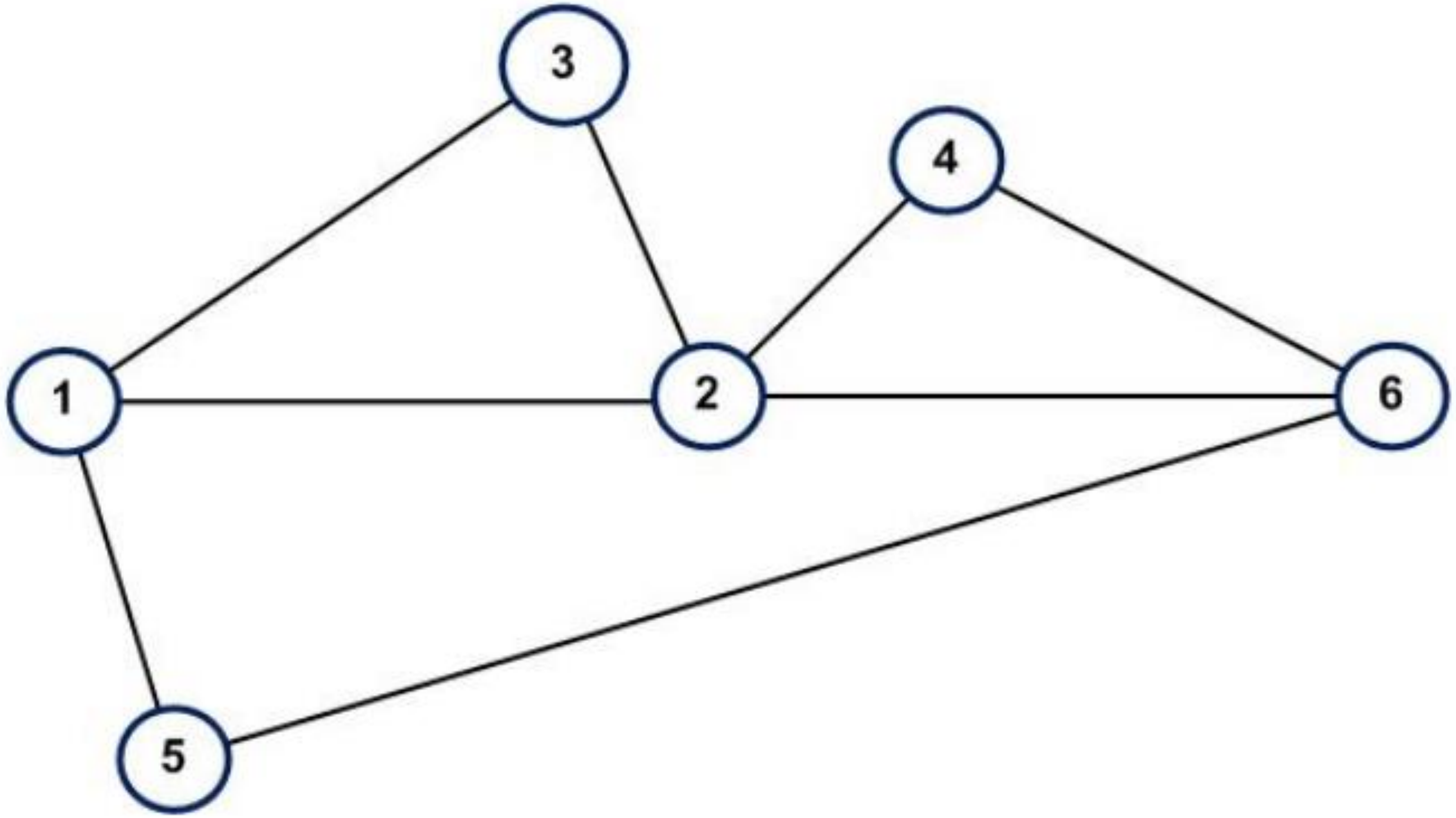
-V = mulțimea nodurilor (finită)

-E = mulțimea muchiilor (submulțime a produsului cartezian V $\times$ V)

-v $\in$ V = vârf/nod

-e = {u, v} = uv = muchie

-u, v = capetele/extremitățile muchiei uv


3
4
1
2
6
5

- **Vecinul** unui nod x este un nod y dacă există muchia xy.
- Doua noduri vecine se numesc adiacente.
- **Gradul** unui nod x este numărul de muchii care îl au ca extremitate pe x.

Teoremă: Suma gradelor tuturor nodurilor este dublul numărului de muchii.

$$2|E| = \sum_{v \in V} d(v)$$

Grafuri orientate

Graf orientat: G = (V, E)

-V = mulțimea nodurilor (finită)

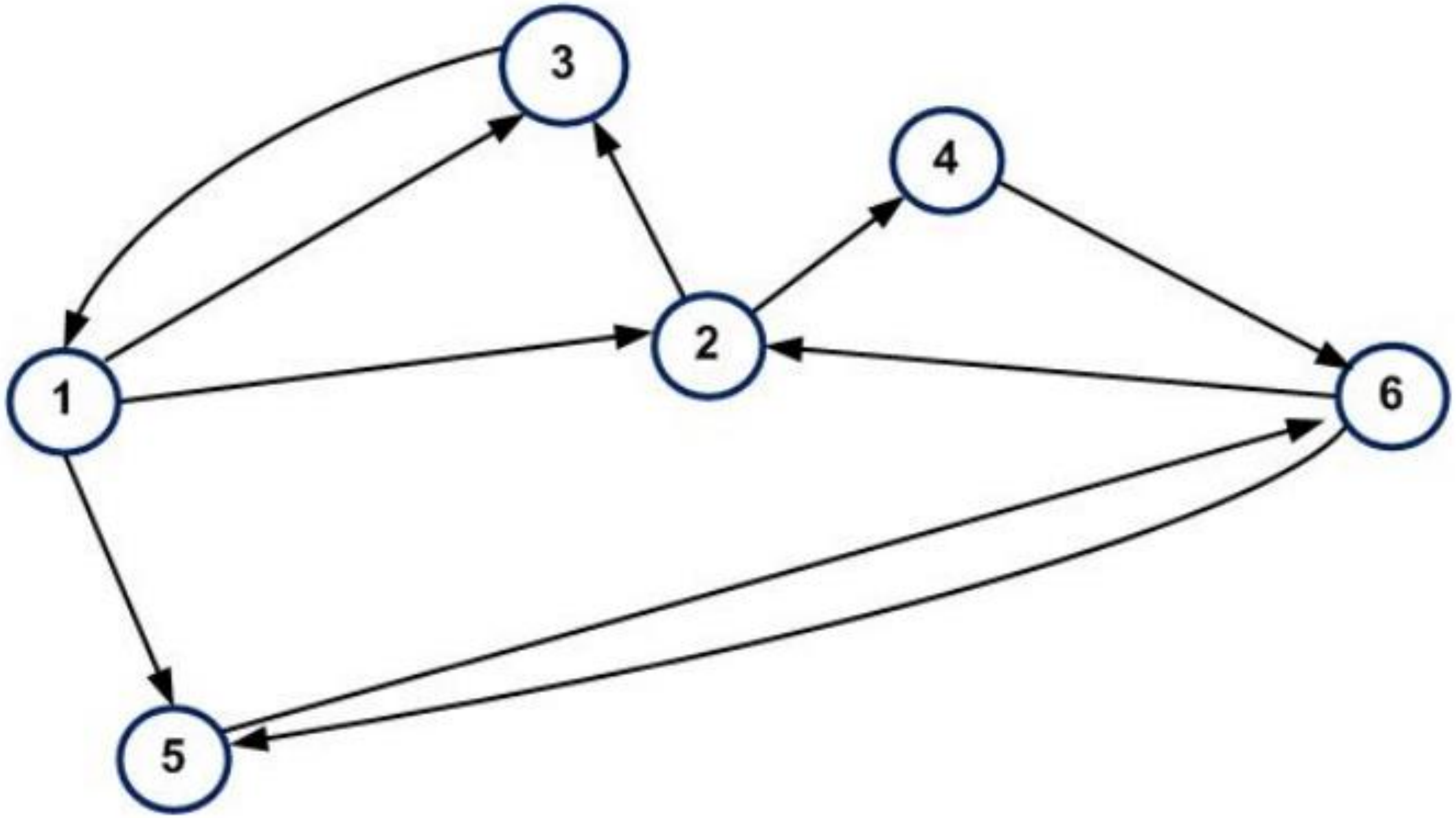
-E = mulțimea muchiilor (formată din perechi ordonate de 2 elemente distincte din V)

-v $\in$ V vârf/nod

-e = (u, v) = uv – arc

-u = e^{-}= vârf inițial

-u = e^{+}= vârf final


3
4
2
1
6
5

$d_G^-(u) = |\{e \in E : u \; extremitate \; finala \; pentru \; e\}|$ - grad interior
$d_G^+(u) = |\{e \in E : u \; extremitate \; initiala \; pentru \; e\}|$ - grad exterior

Teoremă: Are loc următoarea relație:

$$\sum_{u \in V} d_G^-(u) = \sum_{u \in V} d_G^+(u) = |E|$$

Reprezentarea grafurilor

I. Matrice de adiacență:

- matrice unde A[i][j] = 1, dacă există muchia ij

II. Liste de adiacență:

- vectori unde în v[i] este un vector cu nodurile către care avem muchie (se poate folosi std::vector din STL)

III. Listă de muchii:

- vector de perechi care reprezintă extremitățile unei muchii din graf (se poate folosi STL pentru implementare)

Lanț. Ciclu

Se numește **lanț** o succesiune de vârfuri L = $[x_1, x_2, \ldots, x_n]$ cu proprietatea că între x_i și x_{i+1} există muchie. Lungimea lanțului este k – 1. Un lanț care conține vârfuri distincte două câte două se numește **lanț elementar**, iar un lanț în care muchiile nu se repetă se numește **lanț simplu**.

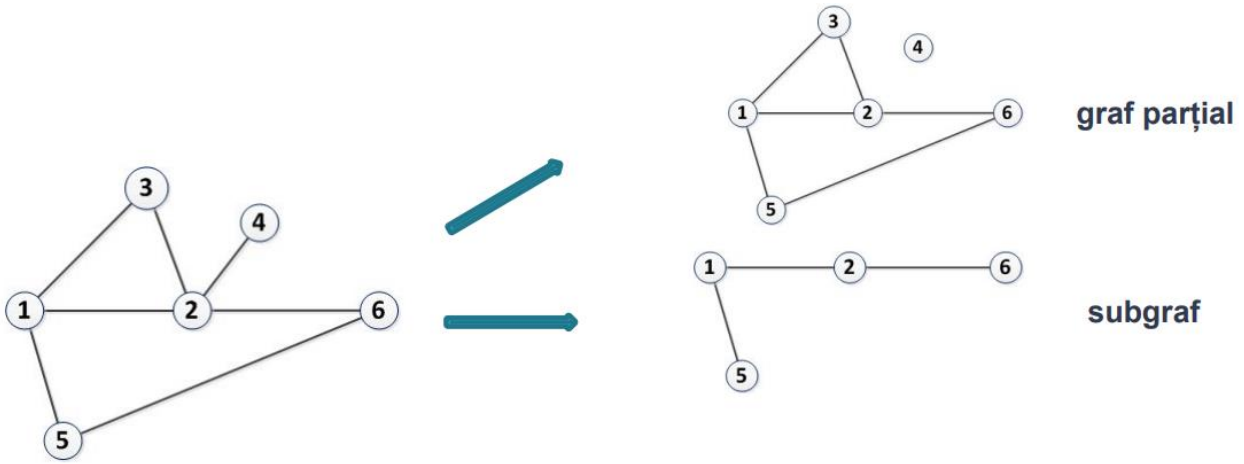
Se numește **ciclu** un lanț în care primul și ultimul vârf sunt aceleași. Lungimea unui ciclu cu k elemente este k – 1. Un **ciclu elementar** este un ciclu cu toate vârfurile distincte două câte două. Un graf care nu conține niciun ciclu se numește **aciclic**.

Drumuri. Circuite

În mod similar se definesc noțiunile de drum, drum simplu, drum elementar, circuit, circuit elementar și circuit simplu în cadrul grafurilor orientate.

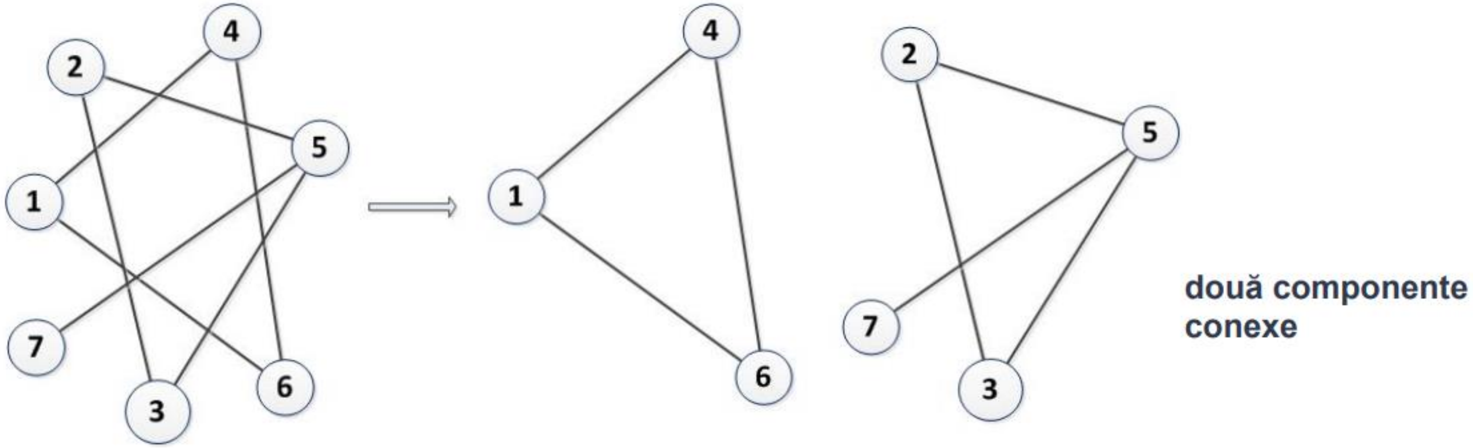
Subgrafuri. Grafuri partiale

Se numește **graf partial** al grafului neorientat G = (X, U) un graf G'(X, U_1), unde $U_1 \subseteq U$.
Se numește **subgraf** al grafului neorientat G = (X, U) un graf G''(X'', U''), unde $U'' \subseteq U \ si\ X'' \subseteq X$.



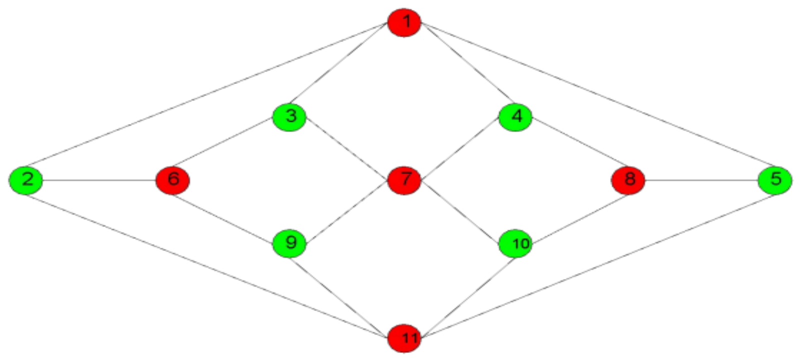
Grafuri conexe

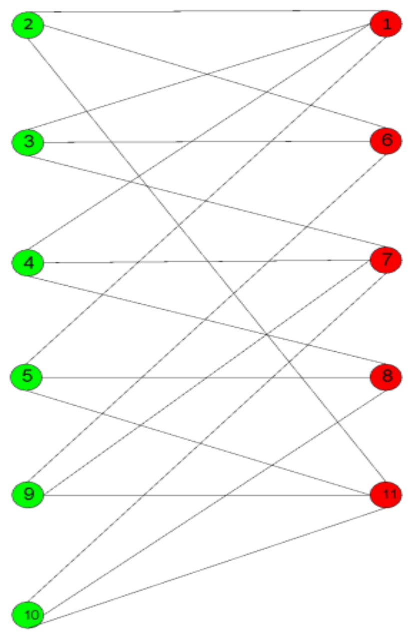
Un **graf conex** este un graf unde există drum de la un nod la oricare altul. O **componentă conexă** este un subgraf obținut din graful inițial și care este conex.



Graf bipartit

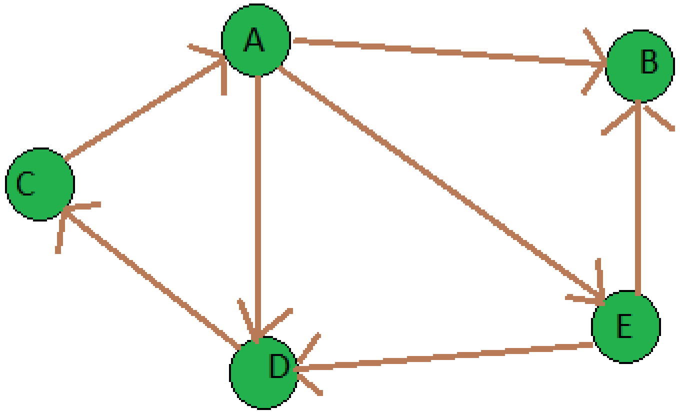
Un **graf bipartit** este un graf în care mulțimea nodurilor poate fi împărțită în două mulțimi nevide disjuncte, astfel incât orice muchie are câte o extremitate în una dintre cele două mulțimi.



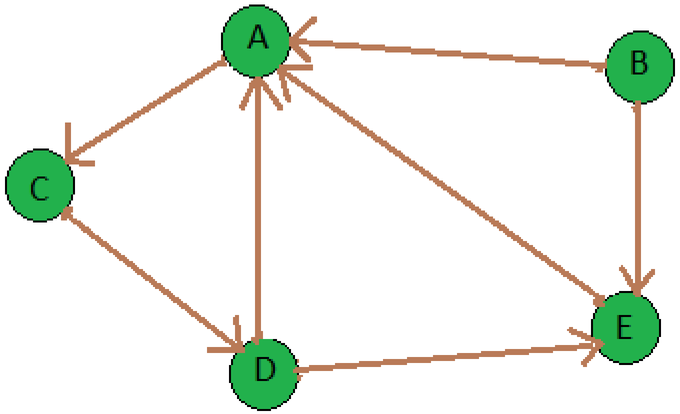


Graf transpus

Graful transpus al unui graf orientat G(X, U) este graful obținut prin schimbarea sensului fiecărui arc din G.



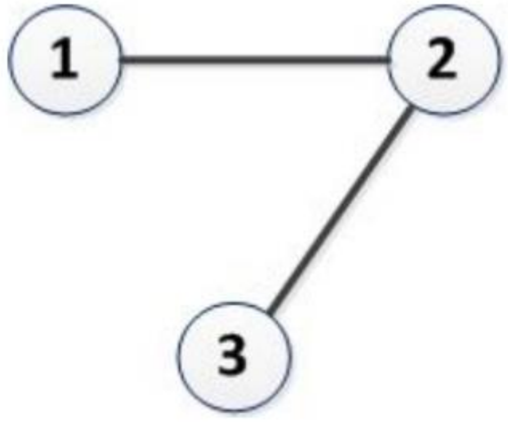
(i)



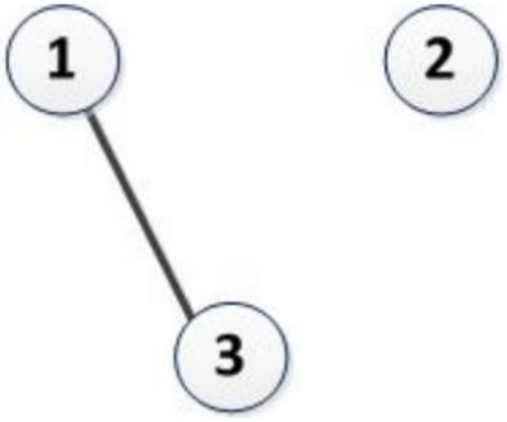
(ii)

Graful complementar

Graful complementar al unui graf neorientat G(X, U) este obținut prin eliminarea muchiilor actuale și adăugarea celor care lipseau în graful inițial.



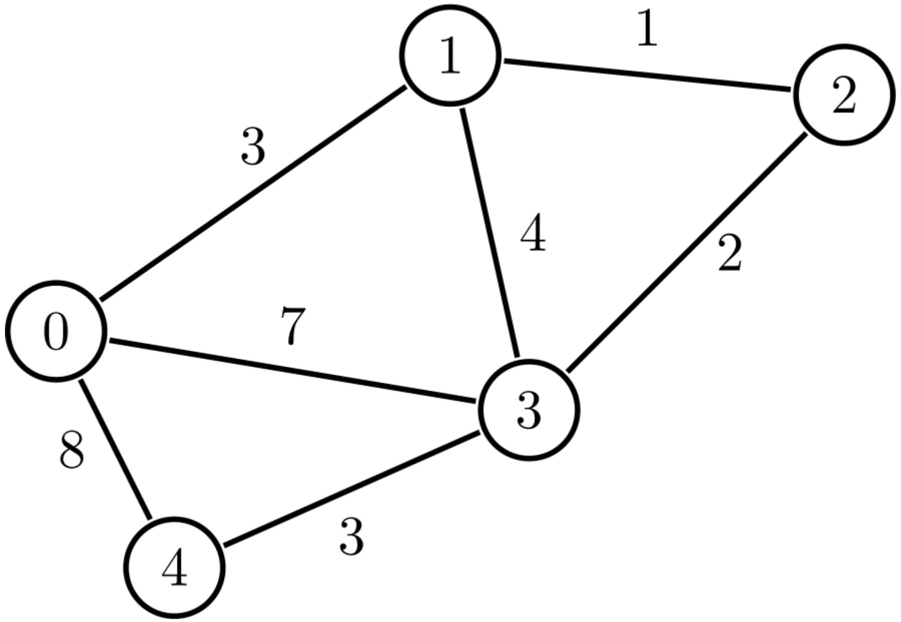
G



$\overline{G}$

Grafuri ponderate

Putem extinde noțiunile învățate deja prin adăugarea unui cost fiecărei muchii, creând astfel o nouă noțiune, aceea de **graf ponderat**.



Parcurgerea în adâncime (DFS)

Vrem să parcurgem graful nostru. Pornind dintr-un nod, vrem să trecem în noduri adiacente și să vedem în ce noduri putem ajunge folosind acest procedeu.

Parcurgerea în adâncime (DFS)

Vrem să parcurgem graful nostru. Pornind dintr-un nod, vrem să trecem în noduri adiacente și să vedem în ce noduri putem ajunge folosind acest procedeu.

O idee este să pornim dintr-un nod și să mergem cât de adânc (depth first) se poate in graf, iar, cand nu se mai poate, să revenim într-un nod din lanțul format, din care să putem continua parcurgerea.

Parcurgerea în adâncime (DFS)

Aplicații ale parcurgerii in adâncime:

- aflarea componentelor conexe;
- să verificăm dacă un graf este bipartit;
- să facem o sortare topologică a grafului;

Parcurgerea în adâncime (DFS)

Algoritm iterativ:

1. Adăugăm nodul de început într-o stivă.
2. Marcăm nodul ca vizitat.
3. Parcurgem nodurile adiacente cu nodul din vârful stivei, iar pe cele care nu au fost vizitate le adăugăm in stivă.
4. Repetăm pașii 2 și 3 până când stiva devine goală.

Complexitate: O(n + m), unde n = numarul de noduri, m = numarul de muchii

Parcurgerea în adâncime (DFS)

Problemă (implementare DFS):
https://infoarena.ro/problema/DFS
Soluție:
https://infoarena.ro/job_detail/3152566?action=view-source

Parcurgerea în adâncime (DFS)

Să rezolvăm următoarea problemă: https://infoarena.ro/problema/cerere .

Parcurgerea în adâncime (DFS)

Să rezolvăm următoarea problemă: https://infoarena.ro/problema/cerere .

Hint: Pentru un nod i putem calcula răspunsul dacă știm care este răspunul pentru cel de-al $k_i - lea$ strămoș. Cum putem afla cel de-al $k_i - lea$ strămoș?

Parcurgerea în adâncime (DFS)

Să rezolvăm următoarea problemă: https://infoarena.ro/problema/cerere .

Hint: Pentru un nod i putem calcula răspunsul dacă știm care este răspunul pentru cel de-al $k_i - lea$ strămoș. Cum putem afla cel de-al $k_i - lea$ strămoș?

Soluție: Folosind parcurgerea in adâncime, dacă implementăm de mână stiva, vom avea acces la toți strămoșii nodului curent. Deci, implementăm astfel și calculăm pentru fiecare nod răspunsul, folosindu-ne de strămoșii săi aflați în stivă.

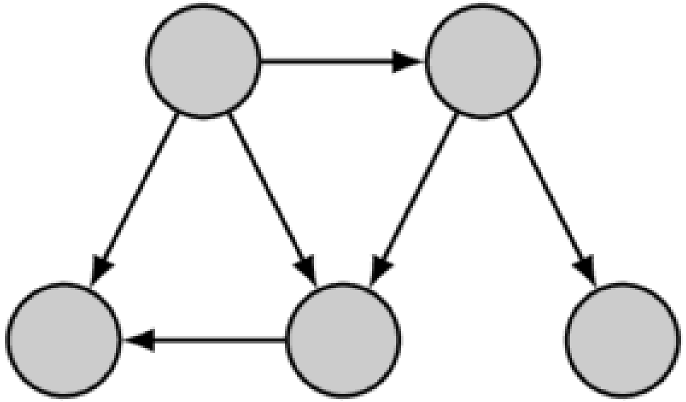
Link solutie: https://www.infoarena.ro/job_detail/3128037?action=view-source

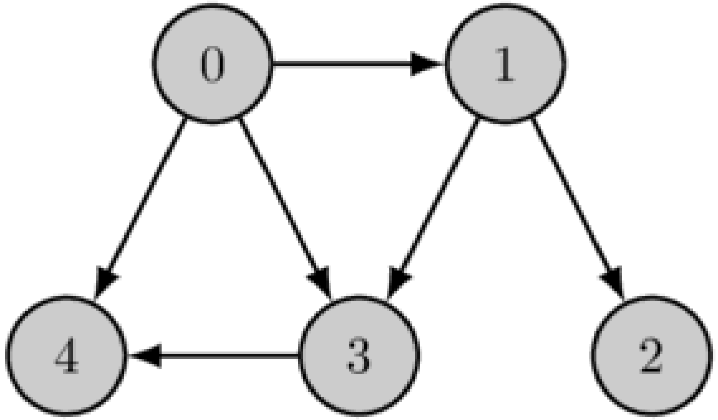
Sortare topologică

Avem un graf orientat și vrem să găsim o ordine a nodurilor, astfel incât dacă i este înaintea lui j în această ordonare, nu există muchie de la j la i.

Sortare topologică

Avem un graf orientat și vrem să găsim o ordine a nodurilor, astfel incât dacă i este înaintea lui j în această ordonare, nu există muchie de la j la i.





Sortare topologică

Dacă există muchie de la u la v, atunci u trebuie să fie înaintea lui v. Atunci, parcurgând în adâncime graful, obținem un drum elementar, inversând acest drum, obținem o posibilă sortare topologică pentru subgraful acesta. Făcând acest procedeu pentru toate vârfurile nevizitate, vom obține o sortare topologica validă.

Sortare topologică

Algoritm:

1. Parcurgem nodurile grafului.
2. Din fiecare nod care nu este deja vizitat, pornim o parcurgere in adâncime.
3. În parcurgere, adăugăm vârfurile într-un vector, în ordinea inversă față de cea în care le parcurgem.
4. Inversăm vectorul.

Complexitate : O(n + m), unde n = numarul nodurilor, m = numarul muchiilor

Sortare topologică

Problemă(implementare a algoritmului):

https://www.infoarena.ro/problema/sortaret

Soluție: https://www.infoarena.ro/job_detail/3159001?action=view-source

Parcurgerea în lățime (BFS)

Parcurgerea în lățime explorează sistematic muchiile unui graf, pentru a afla care sunt toate nodurile accesibile din nodul de plecare.

Parcurgerea în lățime (BFS)

Aplicații ale parcurgerii în lățime:

- aflarea lungimii lanțului minim de la un nod sursă la toate celelalte;
- aflarea componentelor conexe;
- să facem o sortare topologică a grafului;

Parcurgerea în lățime (BFS)

Algoritm:

1. Adăugăm nodul de început într-o coadă.
2. Marcăm nodul ca vizitat.
3. Parcurgem nodurile adiacente cu nodul din fața(vârful) cozii, iar pe cele care nu au fost vizitate le adăugăm in coadă.
4. Repetăm pașii 2 și 3 până când coada devine goală.

Complexitate: O(n + m), unde n = numarul de noduri, m = numarul de muchii

Parcurgerea în lățime (BFS)

Problemă (implementare BFS):
https://infoarena.ro/problema/bfs
Soluție:
https://infoarena.ro/job_detail/2542435?action=view-source

Parcurgerea în lățime (BFS)

Să rezolvăm următoarea problemă: https://infoarena.ro/problema/cifre4.

Parcurgerea în lățime (BFS)

Să rezolvăm următoarea problemă: https://infoarena.ro/problema/cifre4.

Hint: Încercăm să reducem problema la o parcurgere în lățime.

Parcurgerea în lățime (BFS)

Să rezolvăm următoarea problemă: https://infoarena.ro/problema/cifre4.

Soluție: Pornim de la stările inițiale 2, 3, 5 și 7 și ne folosim de un vector de tați, în care ținem minte proveniența stării curente. Pentru stările inițiale, în vectorul de tați avem -1. Rezolvăm problema similar cu o parcurgere BFS, unde stările sunt reprezentate de resturile la împărțirea cu P. Fie r starea noastră curentă, atunci, newR=(r*10+digit) mod P, unde variabila digit ia, pe rând, valorile 2,3,5 și 7. Dacă nu am mai fost în starea newR, atunci o adăugăm în coadă și actualizăm vectorul de tați.

Parcurgerea în lățime (BFS)

Să rezolvăm următoarea problemă: https://infoarena.ro/problema/cifre4.

Soluție: Când se finalizează algoritmul BFS, dacă avem că tata[n]=0, atunci nu există soluție și afișăm -1, altfel facem reconstrucția soluției folosindu-ne de o stivă. Cât timp părintele nodului curent nu este -1, verificăm ce cifră am adăugat părintelui pentru a crea o starea curentă și, în caz că există mai multe posibilități, luăm valoarea minimă. Astfel, la final, de la capul până la baza stivei avem cifrele în ordinea corectă (de la cea mai importantă, la cea mai puțin importantă). Așa că afișăm cifrele de la capul la baza stivei și obținem soluția corectă.
Implementare: https://infoarena.ro/job_detail/2950074?action=view-source

Problemă

Să rezolvăm următoarea problemă:https://infoarena.ro/problema/berarii2.

Problemă

Să rezolvăm următoarea problemă:https://infoarena.ro/problema/berarii2.

Hint: Construim graful transpus.

Problemă

Să rezolvăm următoarea problemă:https://infoarena.ro/problema/berarii2.

Soluție: Construim graful transpus, pentru a putea porni din berării și să aflăm din care intersecții se poate ajunge la ele. Nodurile care rămân nevizitate reprezintă intersecțiile din care nu se poate ajunge la nicio berărie.

Cum implementăm?

Problemă – BFS multisource

Să rezolvăm următoarea problemă:https://infoarena.ro/problema/berarii2.

Soluție: Construim graful transpus, pentru a putea porni din berării și să aflăm din care intersecții se poate ajunge la ele. Nodurile care rămân nevizitate reprezintă intersecțiile din care nu se poate ajunge la nicio berărie.

Pentru o implementare eficientă, actualizăm puțin algoritmul elementar de BFS, astfel încât să plecăm din mai multe noduri deodată. Acest procedeu se numește BFS multisource.

Implementare:https://infoarena.ro/job_detail/3158876?action=view-source

Problemă

Să rezolvăm următoarea problemă:https://infoarena.ro/problema/camionas.

Problemă

Să rezolvăm următoarea problemă:https://infoarena.ro/problema/camionas.

Hint: Extindem algoritmul BFS.

Problemă – BFS 0-1

Să rezolvăm următoarea problemă:https://infoarena.ro/problema/camionas.

Soluție: Se realizează o extindere a algoritmului BFS. Putem reține în loc de greutatea fiecărei muchii, dacă aceasta este sau nu mai grea decât greutatea camionașului. Dacă aceasta este mai ușoară, reținem costul 1 pe muchia respectivă, altfel reținem costul 0. Acum, în loc de o coadă obișnuită, putem folosi un deque pentru implementarea algoritmului BFS. Nodurile în care intrăm folosind o muchie de cost 1 le adăugăm la capătul din dreapta, iar celelalte la capătul din stânga. Astfel, se păstrează proprietatea că algoritmul BFS ne oferă lanțul de lungime minimă de la un nod sursă la oricare altul.
Implementare: https://infoarena.ro/job_detail/3158978?action=view-source

Temă

- https://infoarena.ro/problema/graf
- https://infoarena.ro/problema/sate
- https://www.pbinfo.ro/probleme/3110/genius
- https://codeforces.com/problemset/problem/510/C
- https://codeforces.com/problemset/problem/580/C
- https://codeforces.com/contest/1063/problem/B

Probleme suplimentare

- https://infoarena.ro/problema/easygraph
- https://codeforces.com/contest/1144/problem/F
- https://codeforces.com/contest/1881/problem/F
- https://codeforces.com/problemset/problem/920/E