Grafuri

Banu Denis
Faculty of Computer Science Iaşi

January 15, 2019

Cuprins

Cuprins

Graf

Definiție

Un graf este o pereche ordonată de mulțimi G = {V, E} unde V este o mulțime de elemente numite noduri(vârfuri) și E este o multime de elemente numite muchii(un element din mulțimea E fiind o pereche(ordonată sau neordonată) de elemente din V.

Graf

Definiție

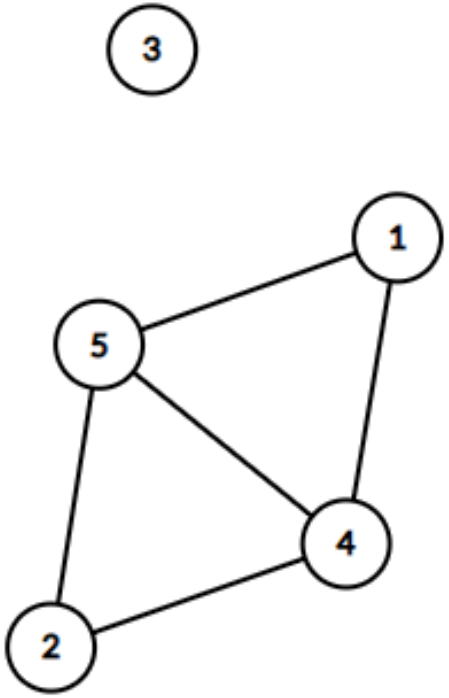
Un graf este o pereche ordonată de mulțimi G = {V, E} unde V este o mulțime de elemente numite noduri(vârfuri) și E este o multime de elemente numite muchii(un element din mulțimea E fiind o pereche(ordonată sau neordonată) de elemente din V.

Informal

Un graf este alcătuit din noduri și muchii. Fiecare muchie unește două noduri.

Matrice de adiacență

Se construiește o matrice A care poate avea două valori (0 sau 1). Daca A[x][y] este 1 atunci există muchie între nodurile x și y.



	1	2	3	4	5
1	0	0	0	1	1
2	0	0	0	1	1
3	0	0	0	0	0
4	1	1	0	0	1
5	1	1	0	1	0

Matrice de adiacență

```
1  int A[NMAX][NMAX], n, m, x, y;
2  cin >> n >> m;
3  for (int i = 1; i <= m; i++){
4       cin >> x >> y;
5       A[x][y] = A[y][x] = 1;
6  }
```

Complexitate memorie: $O(N^2)$

Observații

- Matricea de adiacență se poate folosi atunci când numărul de noduri este mic ($\leqslant$1000).
- Este utilă atunci când trebuie să determinăm rapid dacă există muchie între două noduri.

Liste de adiacență

Pentru fiecare nod se construiește o listă în care sunt introduse toate nodurile cu care acesta are o muchie comună.

Liste de adiacență

```
1  vector<int> v[NMAX];
2  cin >> n >> m;
3  for (int i = 1; i <= m; i++){
4       cin >> x >> y;
5       v[x].push_back(y);
6       v[y].push_back(x);
7  }
```

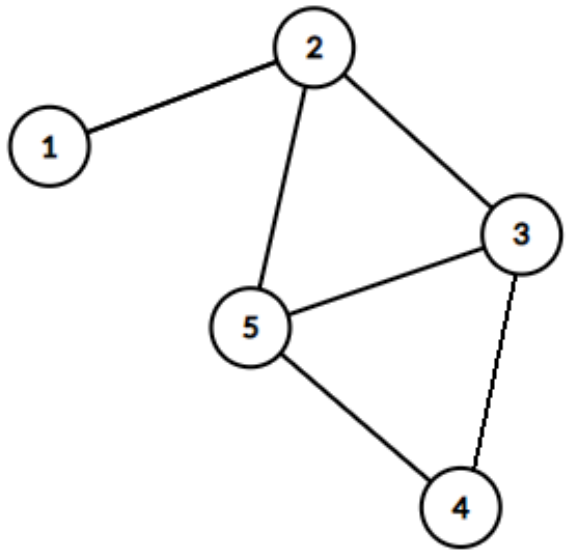
Complexitate memorie: O(m+n)

BFS - parcurgere în lățime

- Nodurile sunt parcurse în ordine în funcție de distanța față de nodul de început.
- Se utilizează o coadă.
- La început se introduce nodul de plecare în coadă.
- În fiecare etapă se scoate câte un nod din coadă, se parcurg toți vecinii acestuia și se introduc în coadă nodurile care nu au mai fost vizitate.
- Algoritmul se termină atunci când nu mai sunt elemente în coadă.

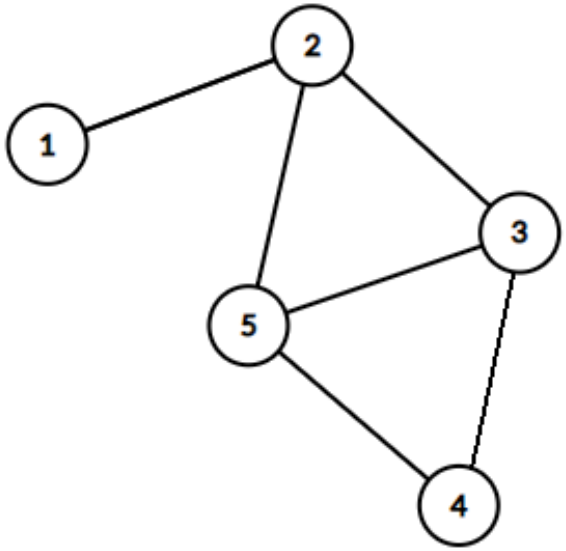
BFS - parcurgere în lățime

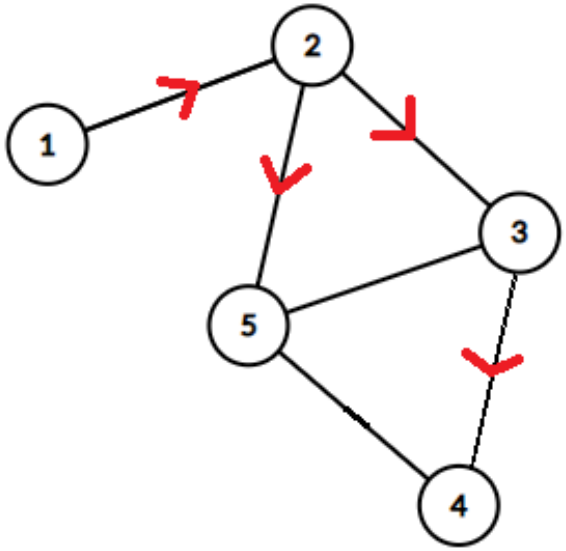
- Nodurile sunt parcurse în ordine în funcție de distanța față de nodul de început.
- Se utilizează o coadă.
- La început se introduce nodul de plecare în coadă.
- În fiecare etapă se scoate câte un nod din coadă, se parcurg toți vecinii acestuia și se introduc în coadă nodurile care nu au mai fost vizitate.
- Algoritmul se termină atunci când nu mai sunt elemente în coadă.



BFS - parcurgere în lățime

- Nodurile sunt parcurse în ordine în funcție de distanța față de nodul de început.
- Se utilizează o coadă.
- La început se introduce nodul de plecare în coadă.
- În fiecare etapă se scoate câte un nod din coadă, se parcurg toți vecinii acestuia și se introduc în coadă nodurile care nu au mai fost vizitate.
- Algoritmul se termină atunci când nu mai sunt elemente în coadă.





1 -> 2 -> 3 -> 5 -> 4

BFS

```
1  bool viz[NMAX];
2  vector<int> v[NMAX];
3  queue<int> Q;
4  Q.push(1);
5  while (!Q.empty()){
6        int nod = Q.front();
7        Q.pop();
8        for (auto it : v[nod]){
9              if (!viz[it]){
10                   viz[it] = 1;
11                   Q.push(it);
12             }
13       }
14 }
```

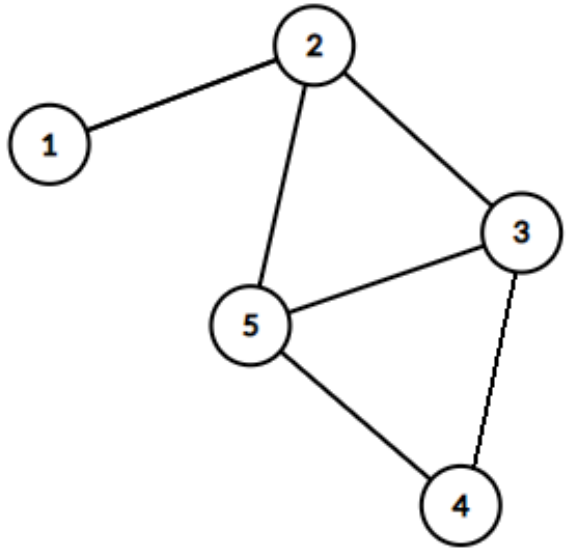
Complexitate timp: O(m)

DFS - parcurgere în adâncime

- În general se implementează recursiv.
- La început se apeleaza funcția DFS pentru nodul 1.
- În interiorul funcției se parcurg toți vecinii nodului curent și se apelează recursiv aceeași funcție pentru cei nevizitați.
- Algoritum se termină când nu mai există noduri nevizitate.

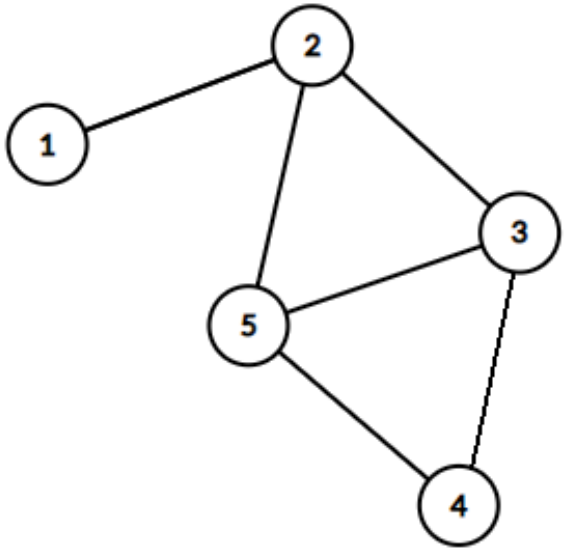
DFS - parcurgere în adâncime

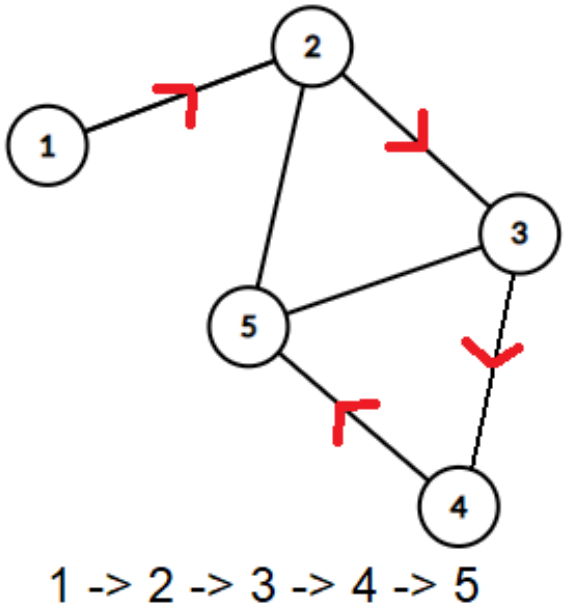
- În general se implementează recursiv.
- La început se apeleaza funcția DFS pentru nodul 1.
- În interiorul funcției se parcurg toți vecinii nodului curent și se apelează recursiv aceeași funcție pentru cei nevizitați.
- Algoritum se termină când nu mai există noduri nevizitate.



DFS - parcurgere în adâncime

- În general se implementează recursiv.
- La început se apeleaza funcția DFS pentru nodul 1.
- În interiorul funcției se parcurg toți vecinii nodului curent și se apelează recursiv aceeași funcție pentru cei nevizitați.
- Algoritum se termină când nu mai există noduri nevizitate.





1 -> 2 -> 3 -> 4 -> 5

DFS

```
1  bool viz[NMAX];
2  vector<int> v[NMAX];
3  void DFS(int nod){
4      for (auto it : v[nod])
5          if (!viz[it]){
6              viz[it] = 1;
7              DFS(it);
8          }
9  }
```

Complexitate timp: O(m)

Friend of Friend

Enunț

Intr-o retea de socializare sunt N utilizatori. Doi utilizatori sunt prieteni daca intre ei exista o relatie de prietenie in reteaua de socializare.

Cerință

Pentru fiecare dintre cei N utilizatori afisati toate persoanele cu care are cel putin un prieten comun, dar cu care nu este prieten.

Restricții

- $1 \leqslant N \leqslant 1000$
- $1 \leqslant M \leqslant 5000$ (numarul de muchii)

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

Soluție

- Fie u un nod si v un nod care nu are muchie cu u dar are un vecin comun.

Soluție

- Fie u un nod si v un nod care nu are muchie cu u dar are un vecin comun.
- Observam ca intre oricare doua noduri u si v descrise mai sus, exista un drum de lungime 2 care trece prin vecinul comun. In plus nu exista muchie intre u si v deci acel drum de lungime 2 este cel mai scurt.

Soluție

- Fie u un nod si v un nod care nu are muchie cu u dar are un vecin comun.
- Observam ca intre oricare doua noduri u si v descrise mai sus, exista un drum de lungime 2 care trece prin vecinul comun. In plus nu exista muchie intre u si v deci acel drum de lungime 2 este cel mai scurt.
- Pentru **fiecare** nod u, facem o pargurgere BFS in care retinem si lungimea drumului parcurs.

Soluție

- Fie u un nod si v un nod care nu are muchie cu u dar are un vecin comun.
- Observam ca intre oricare doua noduri u si v descrise mai sus, exista un drum de lungime 2 care trece prin vecinul comun. In plus nu exista muchie intre u si v deci acel drum de lungime 2 este cel mai scurt.
- Pentru **fiecare** nod u, facem o pargurgere BFS in care retinem si lungimea drumului parcurs.
- Ne oprim atunci cand am vizitat toate nodurile care au distanta 2 de nodul de start, iar acestea sunt nodurile cautate.

Soluție

- Fie u un nod si v un nod care nu are muchie cu u dar are un vecin comun.
- Observam ca intre oricare doua noduri u si v descrise mai sus, exista un drum de lungime 2 care trece prin vecinul comun. In plus nu exista muchie intre u si v deci acel drum de lungime 2 este cel mai scurt.
- Pentru **fiecare** nod u, facem o pargurgere BFS in care retinem si lungimea drumului parcurs.
- Ne oprim atunci cand am vizitat toate nodurile care au distanta 2 de nodul de start, iar acestea sunt nodurile cautate.
- Complexitate: O(N * M)

Conexitate

Un graf este **conex** dacă se poate ajunge de la orice nod al său în oricare alt nod.

Conexitate

Un graf este **conex** dacă se poate ajunge de la orice nod al său în oricare alt nod.

Ciclu

Un **ciclu** este un drum care trece prin muchii diferite și care începe și se termină în același nod.

Arbori

Definiție

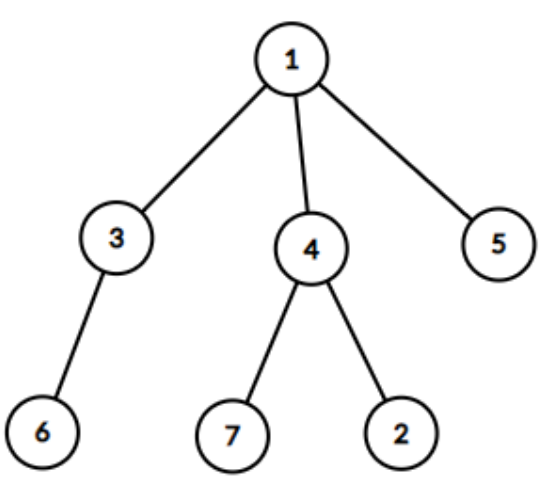
Un **arbore** este un graf conex aciclic.

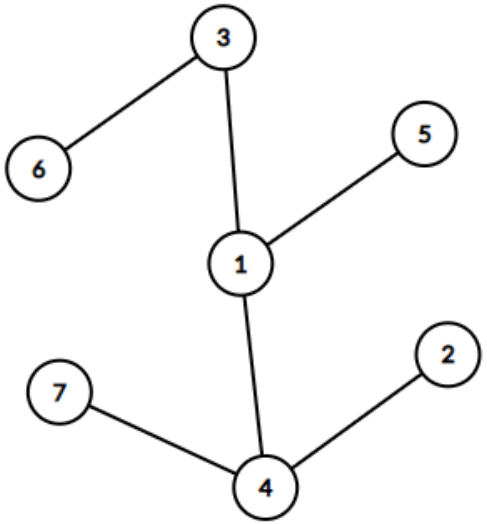
Arbori

Definiție

Un **arbore** este un graf conex aciclic.

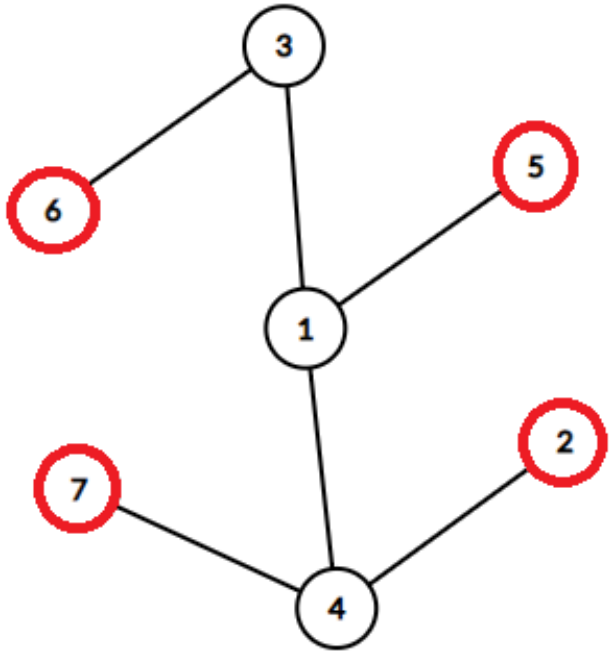
Acesta poate să aibă sau să nu aibă o radacină.





Frunze

Nodurile care au un singur vecin se numesc **frunze**.



Diametrul unui arbore

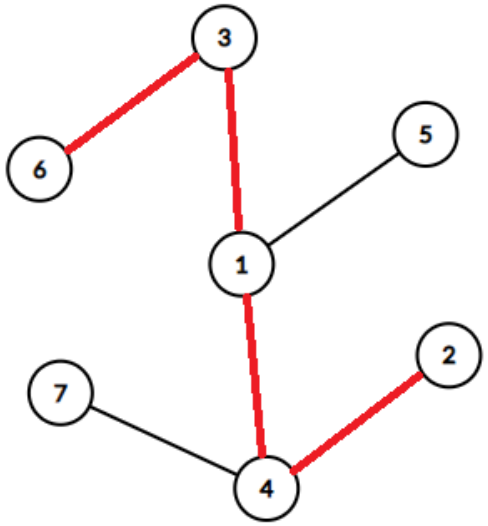
Diametrul unui arbore este cel mai lung drum care unește două frunze.

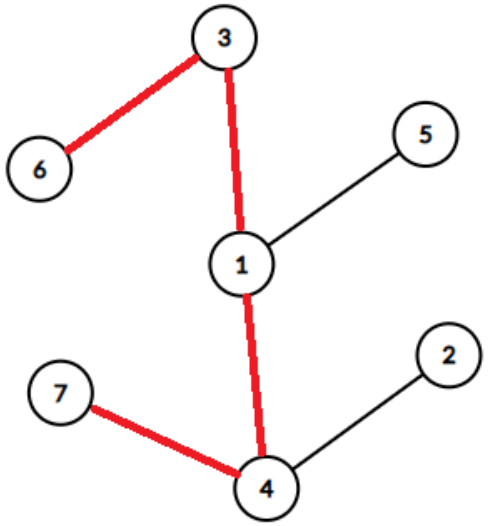
Diametrul unui arbore

Diametrul unui arbore este cel mai lung drum care unește două frunze.

- Pentru a găsi diametrul arborelui putem pornim un DFS din orice nod și să căutam cel mai depărtat nod de acesta. Nodul găsit este o frunză ce reprezintă unul din capetele diametrului.
- Apelam aceeași funcție DFS pentru nodul găsit pentru a determina celălalt capăt al diametrului.
- Complexitate: O(n)

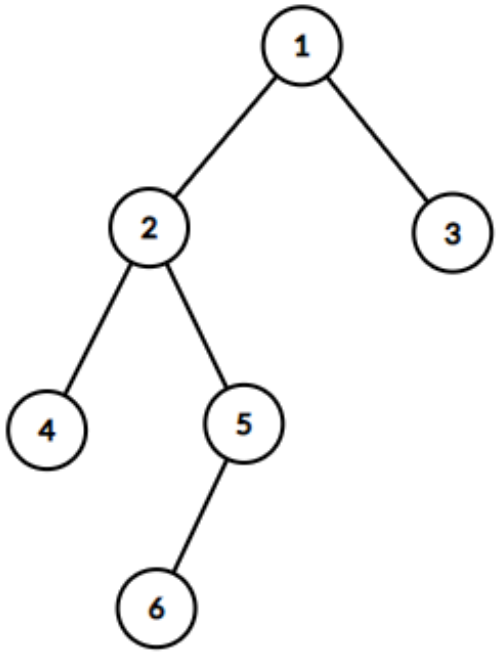
Diametrul unui arbore nu este unic.





Arbore binar

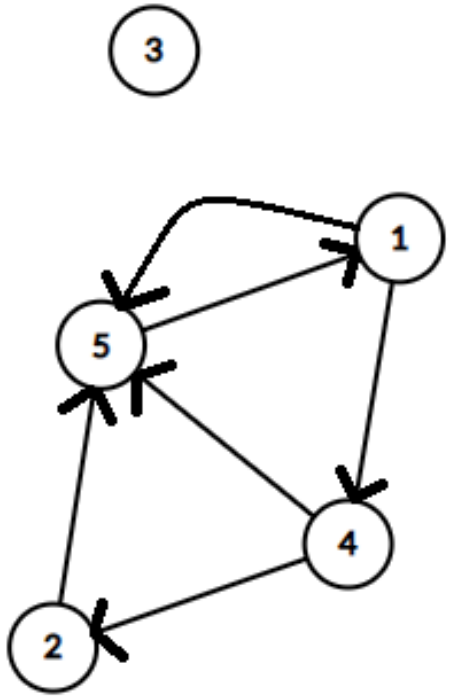
Un arbore binar este un arbore în care orice nod are maxim doi fii.



Grafuri orientate

Definiție

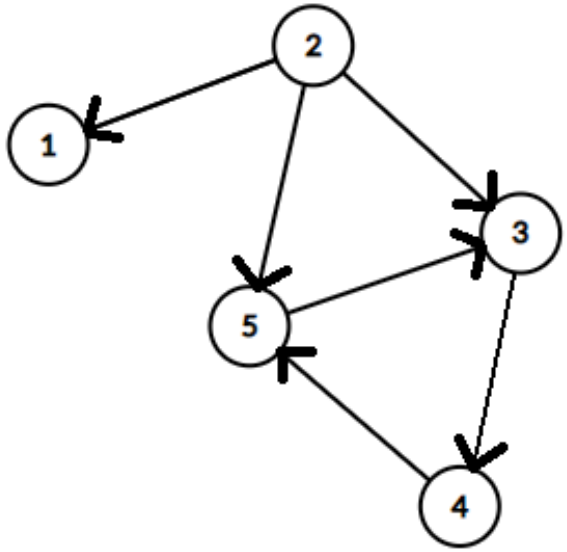
Grafurile orientate sunt asemănătoare grafurilor neorientate cu excepția că muchiile dintre două noduri nu pot fi parcurse decât într-un sens.



Componente tare conexe

Definiție

O componentă **tare conexă** este o submulțime de noduri și muchii ale unui graf orientat(subgraf) în care se poate ajunge de la orice nod la orice nod.



CTC

```
1  int main(){
2      for (int i=1;i<=m;i++){
3          cin >> a >> b;
4          v[a].push_back(b);
5          v2[b].push_back(a);
6      }
7      for (int i=1;i<=n;i++) if (!uz[i]) dfs(i);
8      memset(uz,0,sizeof(uz));
9      for (int i = H.size() - 1; i >= 0; i--)
10             if (!uz[H[i]]){
11                 ans.push_back(vector<int>());
12                 dfs2(H[i]);
13             }
14     }
15 }
```

CTC

```
1  void dfs(int nod){
2       uz[nod] = 1;
3       for (auto it : v[nod]){
4            if (uz[it]) continue;
5            dfs(it);
6       }
7       H.push_back(nod);
8  }
9
10 void dfs2(int nod){
11      uz[nod] = 1;
12      for (auto it : v2[nod]){
13           if (uz[it]) continue;
14           dfs2(it);
15      }
16      ans.back().push_back(nod);
17 }
```

Sortare topologică

Descriere

O **sortare topologică** a vârfurilor unui graf orientat aciclic este o operație de ordonare liniară a vârfurilor, astfel încat, dacă există un arc (i, j), atunci i apare înaintea lui j în această ordonare.

Soluție

- Folosim o listă în care introducem pe rând nodurle grafului.
- La început introducem în listă toate nodurile care au gradul interior zero.
- Atunci când este introdus un nod în listă, scoatem nodul respectiv din graf.
- După fiecare introducere a unui nod in listă, verificăm dacă prin ștergerea sa apar noduri noi cu gradul interior zero și le introducem în continuare în listă.
- Ordinea nodurilor din listă este ordinea din sortarea topologică a grafului.

Probleme

- BFS
- DFS
- Friend of Friend
- Diametrul unui arbore
- Componente tare conexe
- Rețele
- Graf
- PolandBall and Forest
- Kefa and Park
- Andryusha and Colored Balloons
- Leha and...