INFORMATICĂ · CLASA A XI-A · GRAFURI · LECȚIA 11

Parcurgerea grafurilor: în lățime (BFS) și în adâncime (DFS)

Durată: 60 min Nivel: mate-info Înainte: lecțiile xi-10

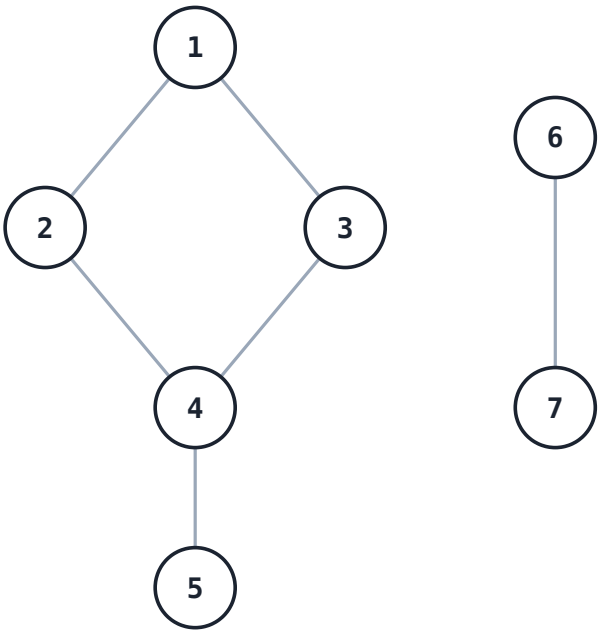
CE VEI ȘTI DUPĂ LECȚIE

- Parcurgi un graf în lățime (BFS) cu o coadă și în adâncime (DFS) recursiv.
- Scrii de mână ordinea de vizitare BFS / DFS pornind dintr-un vârf (grilă de Bac).
- Folosești parcurgerile pentru conexitate, componente conexe și lanțul de lungime minimă.

1. Ideea: cum „vizitezi” un graf

Ai o hartă cu drumuri și vrei să afli **ce orașe poți atinge** din orașul 1. Sunt două strategii naturale:

- **DFS** (în adâncime): mergi cât poți înainte pe un drum; când ajungi într-un capăt mort, te întorci un pas și încerci altă variantă. Ca într-un labirint cu un fir de ață.
- **BFS** (în lățime): vizitezi mai întâi toți vecinii (la distanța 1), apoi vecinii lor (distanța 2), și tot așa. Ca o undă pe apă.



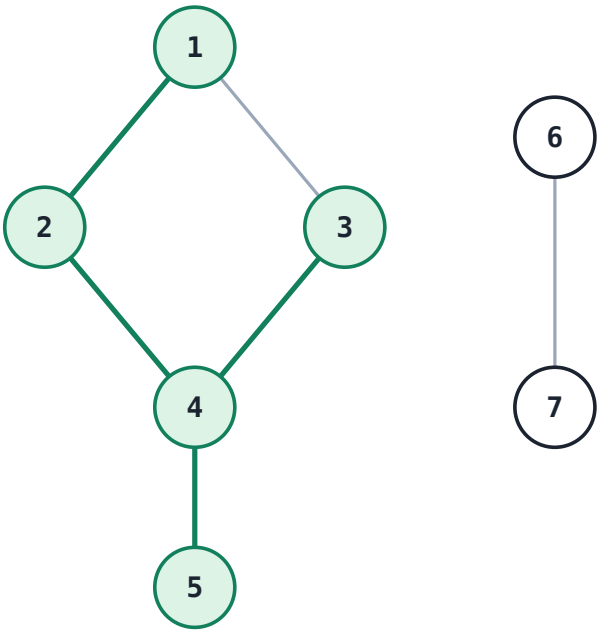
două componente: {1,2,3,4,5} și {6,7}

Graful exemplu

Folosim peste tot graful acesta, dat prin `n = 7`, `m = 6` și muchiile `1 2`, `1 3`, `2 4`, `3 4`, `4 5`, `6 7`. Vecinii se iau în ordinea crescătoare a numerelor.

2. DFS în adâncime, pas cu pas

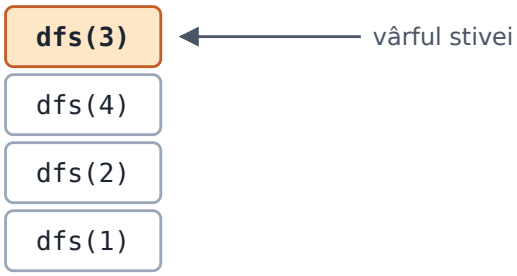
Regula: vizitezi vârful x; pentru fiecare vecin y nevizitat, repeți operația pentru y.



verde: muchiile prin care „coborâm” (arborele DFS); 6 și 7 nu sunt atinse

DFS

Pas	Vârf curent	Vecini	Ce se întâmplă	Ordine
1	1	2, 3	marchez 1; primul vecin nevizitat: 2	1
2	2	1, 4	marchez 2; 1 e vizitat; merg în 4	1 2
3	4	2, 3, 5	marchez 4; 2 e vizitat; merg în 3	1 2 4
4	3	1, 4	marchez 3; ambii vizitați; mă întorc în 4	1 2 4 3
5	4	5	merg în 5	1 2 4 3 5
6	5	4	marchez 5; nimic nou; mă întorc până în 1	1 2 4 3 5



DFS „ține minte” drumul pe stiva recursivității; când vecinii sunt epuizați, se întoarce

Stiva recursivității

```cpp
#include <iostream>
using namespace std;

int a[101][101], viz[101], n, m;

void dfs(int x)
{
    viz[x] = 1;
    cout << x << ' ';                    // „vizităm” vârful
    for (int y = 1; y <= n; y++)
        if (a[x][y] == 1 && !viz[y])     // vecin încă nevizitat
            dfs(y);
}

int main()
{
    cin >> n >> m;
    for (int k = 1; k <= m; k++)
    {
        int x, y;
        cin >> x >> y;
        a[x][y] = a[y][x] = 1;
    }
    int start;
    cin >> start;
    dfs(start);
    return 0;
}
```

DE CE AȘA?

De ce `viz[]` ? Fără marcaj, graful 1-2 ar face `dfs(1)` → `dfs(2)` → `dfs(1)` → ... la infinit. Marcăm vârful **înainte** de a coborî mai adânc. **De ce recursivitate?** Revenirea („mă întorc în 4”) vine gratis: când `dfs(3)` se termină, execuția continuă în `dfs(4)` exact de unde rămăsese (vezi stiva din figură).

La fel, cu liste de adiacență (mai rapid pe grafuri mari):

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

vector<int> L[101];
int viz[101], n, m;

void dfs(int x)
{
    viz[x] = 1;
    cout << x << ' ';
    for (int i = 0; i < (int)L[x].size(); i++)
    {
        int y = L[x][i];
        if (!viz[y]) dfs(y);
    }
}

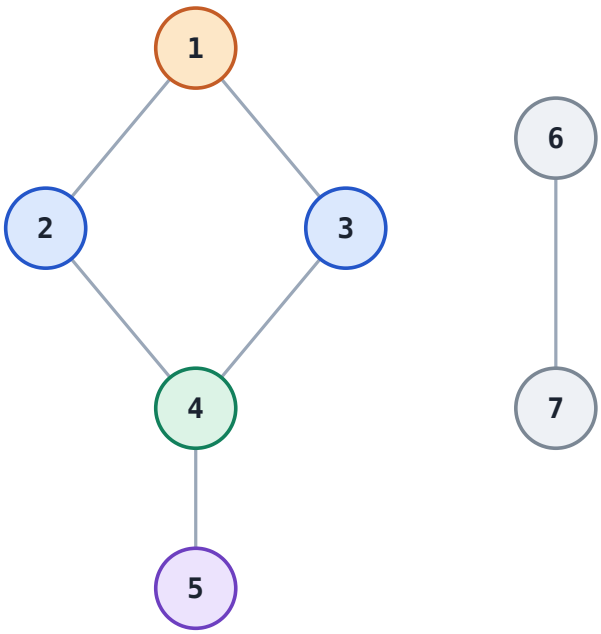
int main()
{
    cin >> n >> m;
    for (int k = 1; k <= m; k++)
    {
        int x, y;
        cin >> x >> y;
        L[x].push_back(y);
        L[y].push_back(x);
    }
    for (int i = 1; i <= n; i++) sort(L[i].begin(), L[i].end());    // vecini în ordine crescătoare
    int start;
    cin >> start;
    dfs(start);
    return 0;
}
```

DE ȘTIUT

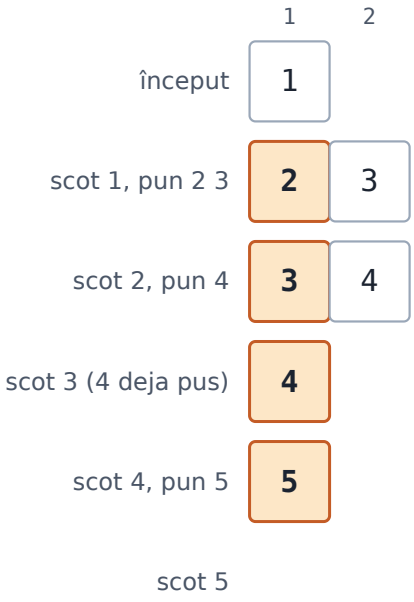
Complexitate: cu matrice **O(n²)** (pentru fiecare vârf scanezi o linie întreagă), cu liste **O(n + m)** (fiecare vârf și fiecare muchie sunt atinse de un număr constant de ori). Pentru n = 100 000 doar varianta cu liste e practică.

3. BFS în lățime și distanțele minime

Regula: punem vârful de start într-o **coadă**. Cât timp coada nu e goală: scoatem primul vârf x; pentru fiecare vecin y nevizitat, îl marcăm, îi punem distanța `d[x] + 1` și îl băgăm în coadă.



portocaliu: 0 · albastru: 1 · verde: 2 · mov: 3 · gri: nu se poate ajunge



un vârf intră în coadă o singură dată: când este descoperit

De ce BFS dă distanțele minime? Coada păstrează vârfurile **în ordinea distanței**: mai întâi toate cele cu distanța 1, apoi 2, etc. Prima dată când un vârf e descoperit, este descoperit pe drumul cel mai scurt.

Trace pentru `d[]` (-1 = nevizitat):

Pas	scot	coada după	d[1..5]
0		1	0, -1, -1, -1, -1
1	1	2 3	0, 1, 1, -1, -1
2	2	3 4	0, 1, 1, 2, -1

Pas	scot	coada după	d[1..5]
3	3	4	0, 1, 1, 2, -1 (4 e deja marcat)
4	4	5	0, 1, 1, 2, 3
5	5	(gol)	0, 1, 1, 2, 3

```cpp
#include <iostream>
using namespace std;

int a[101][101], d[101], coada[101], n, m;

int main()
{
    cin >> n >> m;
    for (int k = 1; k <= m; k++)
    {
        int x, y;
        cin >> x >> y;
        a[x][y] = a[y][x] = 1;
    }
    int start;
    cin >> start;

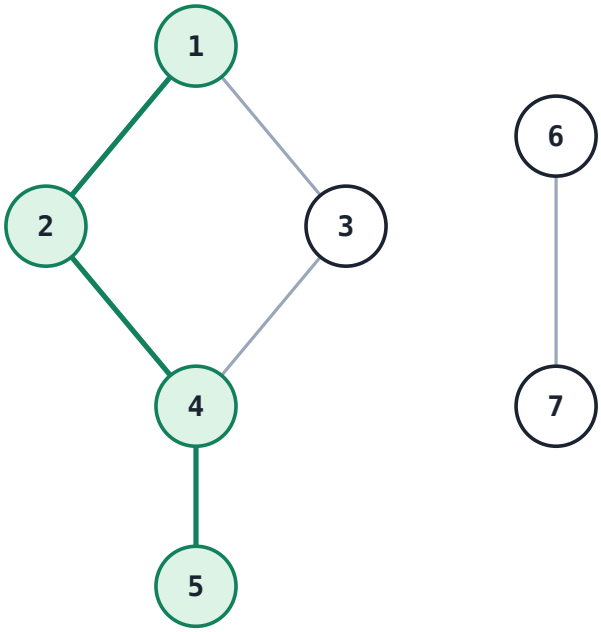
    for (int i = 1; i <= n; i++) d[i] = -1;      // -1 = nevizitat
    int prim = 0, ultim = 0;
    coada[ultim++] = start;
    d[start] = 0;
    while (prim < ultim)
    {
        int x = coada[prim++];                    // scot primul
        for (int y = 1; y <= n; y++)
            if (a[x][y] && d[y] == -1)
            {
                d[y] = d[x] + 1;
                coada[ultim++] = y;               // adaug la sfârșit
            }
    }
    for (int i = 1; i <= n; i++) cout << d[i] << ' ';
    return 0;
}
```

DE CE AȘA?

De ce `d[y] == -1` ca „nevizitat”? Distanța 0 e validă (pentru start), deci nu putem folosi 0 ca „necunoscut”. **De ce coada este un vector cu doi indici?** Fiecare vârf intră o singură dată, deci nu depășim n elemente: `prim` și `ultim` ajung. **De ce rămân -1 la vârfurile 6 și 7?** Nu sunt în componenta lui 1: nu se poate ajunge la ele.

Drumul minim efectiv: vectorul de părinți

Pe lângă `d[y]`, reținem `tata[y] = x` (de unde l-am descoperit). Apoi reconstruim drumul de la destinație înapoi.



părinții: p[2] = 1, p[4] = 2, p[5] = 4; se reconstruiește de la 5 înapoi

Drumul minim

```cpp
#include <iostream>
using namespace std;

int a[101][101], d[101], tata[101], coada[101], n, m;

void afisDrum(int x)
{
    if (x == 0) return;
    afisDrum(tata[x]);                  // mai întâi drumul până la părinte
    cout << x << ' ';
}

int main()
{
    cin >> n >> m;
    for (int k = 1; k <= m; k++)
    {
        int x, y;
        cin >> x >> y;
        a[x][y] = a[y][x] = 1;
    }
    int start, final;
    cin >> start >> final;

    for (int i = 1; i <= n; i++) d[i] = -1;
    int prim = 0, ultim = 0;
    coada[ultim++] = start;
    d[start] = 0;                        // tata[start] = 0 (global)
    while (prim < ultim)
    {
        int x = coada[prim++];
        for (int y = 1; y <= n; y++)
            if (a[x][y] && d[y] == -1)
            {
                d[y] = d[x] + 1;
                tata[y] = x;
                coada[ultim++] = y;
            }
    }
    if (d[final] == -1) cout << "nu exista";
    else afisDrum(final);
    return 0;
}
```

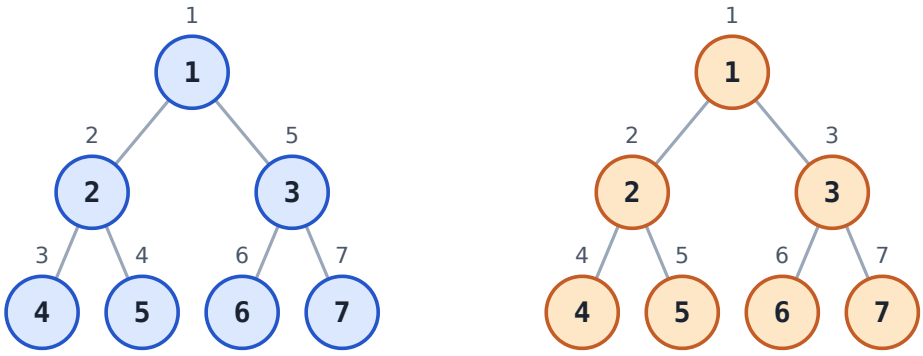
DE CE AȘA?

De ce reconstruim „înapoi" și afișăm „înainte"? Din `final` știm doar părintele (`tata[final]`), nu și urmașul. Mergem înapoi până la start (unde `tata = 0`), iar afișarea se face **după** apelul recursiv ca ordinea să fie de la start la final. Aceeași idee ca la afișarea unui număr în baza 2 (lecția XI.3).

Intrare / ieșire

Program	Intrare	Ieșire
DFS	graful exemplu + start 1	`1 2 4 3 5`
BFS, distanțe	graful exemplu + start 1	`0 1 1 2 3 -1 -1`
drum minim 1 → 5	graful exemplu + `1 5`	`1 2 4 5`
drum minim 1 → 7	graful exemplu + `1 7`	`nu exista`

4. DFS sau BFS?



DFS: în adâncime (1 2 4 5 3 6 7)

BFS: pe niveluri (1 2 3 4 5 6 7)

DFS și BFS

	DFS	BFS
structură	stiva recursivității	coadă
cum explorează	în adâncime	pe niveluri
distanțe minime	**nu**	**da** (graf fără costuri)
componente conexe	da	da
ciclu / bipartit / conexitate tare	foarte potrivit	posibil
memorie	adâncimea (poate fi n)	cel mult un nivel

> **SFAT**
>
> „Drum **cel mai scurt**" într-un graf fără costuri → BFS. „Se poate ajunge?", „câte componente?", „are ciclu?" → DFS (mai scurt de scris).

5. Componentele conexe: număr și dimensiuni

Același `dfs` din `main`, numărând vârfurile din fiecare insulă: afișăm dimensiunea celei mai mari.

```cpp
#include <iostream>
using namespace std;

int a[101][101], viz[101], n, m, marime;

void dfs(int x)
{
    viz[x] = 1;
    marime++;                              // încă un vârf în insula curentă
    for (int y = 1; y <= n; y++)
        if (a[x][y] && !viz[y]) dfs(y);
}

int main()
{
    cin >> n >> m;
    for (int k = 1; k <= m; k++)
    {
        int x, y;
        cin >> x >> y;
        a[x][y] = a[y][x] = 1;
    }
    int comp = 0, maxim = 0;
    for (int i = 1; i <= n; i++)
        if (!viz[i])
        {
            comp++;
            marime = 0;
            dfs(i);
            if (marime > maxim) maxim = marime;
        }
    cout << comp << ' ' << maxim;
    return 0;
}
```

DE CE AȘA?

De ce `marime = 0` înainte de fiecare `dfs`? Variabila e globală și se adună în toate apelurile; pentru fiecare insulă numărăm de la zero.

6. Greșeli frecvente

Greșeala	Efect	Corect
nu marchezi `viz[x]`	recursivitate infinită	`viz[x] = 1` la intrare
marchezi la scoaterea din coadă, nu la introducere	un vârf intră de mai multe ori	marchezi când îl pui în coadă
`d[]` inițializat cu 0	nu mai deosebești „nevizitat”	inițializezi cu -1
coadă prea mică	depășire	`coada[n + 1]` ajunge (fiecare vârf o dată)
citești start/final după BFS	ordinea datelor greșită	după `m` muchii

Greșeala	Efect	Corect
BFS pentru drum cu costuri	rezultat greșit	Dijkstra (lecția XI.19)

7. La BAC

> **LA BAC**
>
> BFS/DFS nu apar ca întrebări „scrie algoritmul", dar sunt în spatele multor itemi: „care este **lungimea celui mai scurt lanț** între x și y", „câte componente conexe", „care vârfuri sunt la distanța k de x", „în ce ordine sunt vizitate vârfurile". Pe grafuri mici le rezolvi cu o coadă pe hârtie, ca în tabelul de la BFS. La subiectul cu programare (II/III) se cer des componente conexe și drumuri minime, ca aici.

8. Exersează pe hârtie

1. Pe graful cu n = 6 și muchiile [1,2], [1,3], [2,4], [3,4], [4,5], [5,6], scrie ordinea DFS din 1 (vecinii în ordine crescătoare).

2. Pe același graf scrie ordinea BFS din 1 și vectorul `d[1..6]`.

3. Care este lungimea celui mai scurt lanț între 1 și 6 în graful de la exercițiul 1? Scrie un asemenea lanț.

4. Un graf are 8 vârfuri și muchiile [1,2], [2,3], [4,5], [6,7], [7,8], [6,8]. Câte componente conexe are? Dar câte muchii minime trebuie adăugate ca să fie conex?

5. Desenează un graf cu 5 vârfuri în care DFS din 1 și BFS din 1 dau aceeași ordine. Poate fi un lanț? Dar un graf complet?

6. (BAC) Într-un arbore cu rădăcina 1 și muchiile [1,2], [1,3], [2,4], [2,5], [3,6], scrie ordinea BFS și ordinea DFS.

7. Pe graful exemplu din lecție, ce se afișează dacă `dfs` afișează vârful **după** apelurile recursive (la ieșire, nu la intrare)?

8. Ce număr maxim de elemente poate avea coada la BFS pe un graf cu n vârfuri? Dă un exemplu în care se atinge.

9. Modifică programul de BFS pentru a afișa și numărul de vârfuri care pot fi atinse din `start` (inclusiv el).

10. Dacă graful este un lanț 1–2–3–…–n, câți pași (apeluri) face DFS din 1? Ce adâncime are stiva?

9. Mai departe (pentru cine vrea mai mult)

DFS fără recursivitate. Folosești o stivă explicită (`stack<int>`): pui startul; cât timp stiva nu e goală, iei vârful x; dacă nu e vizitat, îl marchezi și pui toți vecinii nevizitați. Util când adâncimea ar da depășire de stivă ($n \approx 10^6$).

BFS pe grilă (algoritmul lui Lee). O matrice cu zid / liber este un graf în care fiecare celulă are până la 4 vecini; BFS dă drumul minim prin labirint (lecția XI.17).

Testul de ciclu și arborele DFS. Dacă DFS întâlnește un vecin deja vizitat care nu este părintele, graful (neorientat) are un ciclu.

Parcurgere multi-sursă. Pui în coadă de la început mai multe vârfuri-sursă cu `d = 0`; BFS-ul dă pentru fiecare vârf distanța până la cea mai apropiată sursă (de exemplu „care e cea mai apropiată farmacie?").

Rezumat

- DFS: vizitezi, apoi pentru fiecare vecin nevizitat apelezi recursiv; marchezi `viz[x]` la intrare. Complexitate $O(n^2)$ cu matrice, $O(n + m)$ cu liste.
- BFS: coadă + vector de distanțe `d` (-1 = nevizitat); găsește drumul minim fără costuri; `tata[]` reconstruiește drumul.
- Componentele conexe: apelezi o parcurgere din fiecare vârf nevizitat.
- Marchezi vârful **când îl pui în coadă / când intri în el**, nu mai târziu.

Probleme de lucru (pbinfo)

Începe cu acestea, în ordine: sunt alese să urmeze pașii din lecție.

- [ ] `#19` **bfs** — ușoară
  https://www.pbinfo.ro/probleme/19/bfs
- [ ] `#539` **dfs** — ușoară
  https://www.pbinfo.ro/probleme/539/dfs
- [ ] `#4276` **nrcompconexe** — ușoară
  https://www.pbinfo.ro/probleme/4276/nrcompconexe

Temă (pbinfo)

Trimite soluțiile pe pbinfo și notează punctajul obținut. Bifează ce ai terminat.

- [ ] `#484` **lantminim** — medie
  BFS + vectorul de tați pentru reconstituirea lanțului.
  https://www.pbinfo.ro/probleme/484/lantminim
- [ ] `#538` **lungimeminima** — medie
  https://www.pbinfo.ro/probleme/538/lungimeminima
- [ ] `#126` **dmax** — medie
  https://www.pbinfo.ro/probleme/126/dmax
- [ ] `#4204` **estearbore** — ușoară
  https://www.pbinfo.ro/probleme/4204/estearbore
- [ ] `#963` **bazine** — ușoară
  https://www.pbinfo.ro/probleme/963/bazine

- [ ] #549 **epidemie** — medie
  https://www.pbinfo.ro/probleme/549/epidemie

Pentru cei care vor mai mult

Probleme mai grele sau de concurs, pentru cine vrea să meargă dincolo de BAC.

- [ ] #4067 **ccmax** — medie
  https://www.pbinfo.ro/probleme/4067/ccmax

Grile din subiectele de BAC (pbinfo)

Exerciții de tip grilă din variantele de Bacalaureat, rezolvate online pe pbinfo, cu verificare automată a răspunsului.

- [ ] #76 **Exercițiul #76**
  graf dat prin muchii: lanțuri și cicluri de citit de pe desen
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=76
- [ ] #105 **Exercițiul #105**
  graf cu 8 noduri dat prin muchii: componente / grade
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=105
- [ ] #154 **Exercițiul #154**
  graf cu 7 noduri: numărul de componente conexe
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=154
- [ ] #162 **Exercițiul #162**
  graf dat prin liste de adiacență: ce se poate afirma
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=162
- [ ] #174 **Exercițiul #174**
  graf cu 7 noduri dat prin muchii: noduri la distanță dată
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=174
- [ ] #181 **Exercițiul #181**
  graf cu 7 noduri: componente și lanțuri
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=181
- [ ] #291 **Exercițiul #291**
  graf cu 6 noduri dat prin muchii: lanțuri elementare
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=291
- [ ] #1004 **Exercițiul #1004**
  indicați un lanț elementar în graful dat
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1004

- [ ] #1135 **Exercițiul #1135**
  graf cu 8 noduri, muchii date: proprietăți ale lanțurilor
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1135

- [ ] #1177 **Exercițiul #1177**
  graf cu 10 noduri: componente conexe
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1177

Citește și exersează în plus

- Parcurgerea grafurilor neorientate https://www.pbinfo.ro/articole/8/parcurgerea-grafurilor-neorientate
- VisuAlgo — DFS / BFS — animații pas cu pas pentru ambele parcurgeri, pe graful tău (engleză)
  https://visualgo.net/en/dfsbfs
- USACO Guide — BFS — teorie + probleme de concurs cu distanțe minime (engleză)
  https://usaco.guide/silver/bfs
- CSES — Counting Rooms / Labyrinth — componente conexe și drum minim pe grilă (engleză)
  https://cses.fi/problemset/task/1192

Răspunsuri

Exersează pe hârtie

1. `1 2 4 3 5 6` (din 1 la 2, din 2 la 4, din 4 la 3 (vecinul nevizitat cu cel mai mic număr), apoi la 5 și 6).

2. BFS: `1 2 3 4 5 6`; `d = 0 1 1 2 3 4`.

3. Lungime **4**, de exemplu `1 → 2 → 4 → 5 → 6` (4 muchii).

4. Componente: {1,2,3}, {4,5}, {6,7,8}: **3**. Pentru conexitate sunt necesare cel puțin 3 − 1 = **2** muchii.

5. Da, pentru amândouă. Lanțul 1–2–3–4–5: DFS și BFS dau `1 2 3 4 5`. Graful complet K_5: BFS dă `1 2 3 4 5` (toți sunt vecini ai lui 1); DFS merge 1 → 2 → 3 → 4 → 5 (la fiecare pas cel mai mic vecin nevizitat), deci tot `1 2 3 4 5`. Ca să vezi o diferență între cele două parcurgeri, ia arborele cu 7 vârfuri din figura secțiunii 4.

6. BFS: `1 2 3 4 5 6`; DFS: `1 2 4 5 3 6`.

7. DFS din 1 vizitează 1 → 2 → 4 → 3 (3 se termină primul), apoi 5; apelurile se termină în ordinea 3, 5, 4, 2, 1. Se afișează **`3 5 4 2 1`**.

8. Cel mult **n − 1** elemente (de exemplu un graf stea: centrul 1 legat de celelalte n − 1 vârfuri; după scoaterea lui 1 toate cele n − 1 sunt în coadă).

9. Un contor: `int atinse = 0;` și `atinse++` la fiecare vârf adăugat în coadă (inclusiv start); sau numeri câte `d[i] != -1` la final.

10. DFS face **n** apeluri și stiva ajunge la adâncimea **n** (de aceea la n mare se folosește varianta iterativă).