INFORMATICĂ · CLASA A XI-A · GRAFURI (INTENSIV) · LECȚIA 19

Grafuri cu costuri: Dijkstra, Roy-Floyd, arbori parțiali de cost minim (Prim, Kruskal)

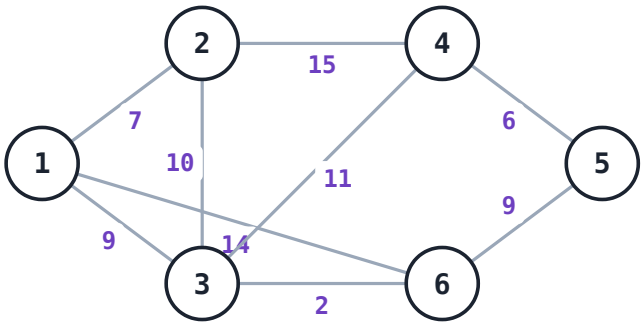
Durată: 60 min Nivel: intensiv informatică / olimpiadă Înainte: lecțiile xi-12, xi-13

CE VEI ȘTI DUPĂ LECȚIE

- Reprezinți un graf cu costuri prin matricea costurilor.
- Calculezi drumurile minime de la un nod la toate celelalte (Dijkstra).
- Calculezi drumurile minime între oricare două noduri (Roy-Floyd).
- Construiești arborele parțial de cost minim (Prim, Kruskal).

1. Graf cu costuri

Până acum o muchie spunea doar „există drum". Acum fiecare muchie are un **cost**: kilometri, minute, lei. Întrebările devin: care este drumul **cel mai ieftin**? cum conectăm toate orașele cu cabluri cât mai ieftin?



costul muchiei = numărul de pe ea (kilometri, lei, minute...)

Graf cu costuri

Se memorează în **matricea costurilor**: `c[i][j]` = costul muchiei (i, j); `0` pe diagonală; `INF` (o valoare foarte mare, de exemplu 10^9) dacă muchia nu există.

```
const int INF = 1000000000;
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
        c[i][j] = (i == j) ? 0 : INF;
// la citirea muchiei (x, y) cu costul z:  c[x][y] = c[y][x] = z;   (orientat: doar c[x][y] = z)
```

DE CE AȘA?

De ce INF și nu 0 pentru „nu există muchie”? Zero ar însemna „muchie gratuită”, iar algoritmii ar folosi muchii inexistente. INF este „atât de scump încât nu merită niciodată”. **De ce 10^9 și nu cel mai mare int?** Când aduni două costuri `INF + INF` depășești `int` și obții un număr negativ (depășire); $10^9 + 10^9 = 2 \cdot 10^9$ încape încă în `int` (limita este ~$2{,}1 \cdot 10^9$), dar adună cu grijă.

2. Dijkstra: drumuri minime dintr-un nod (costuri ≥ 0)

Ideea. Imaginează-te la nodul de start. Distanța până la el este 0. Pornești „un val”: la fiecare pas alegi nodul **nefixat** cu distanța cea mai mică. Distanța lui este **definitivă** (orice alt drum ar trece prin noduri mai depărtate și ar fi mai lung), apoi vezi dacă prin el poți ajunge mai ieftin la vecinii lui.

Pe graful de mai sus, din nodul 1:

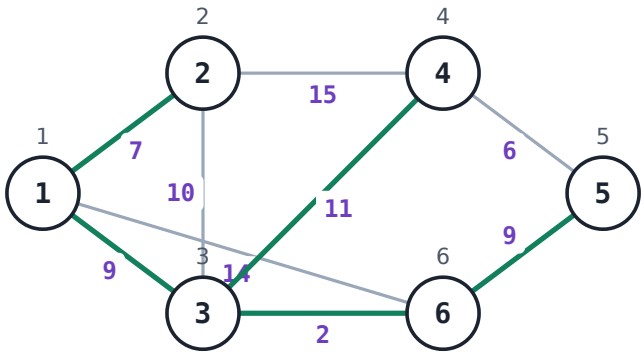
	1	2	3	4	5	6
început	0	∞	∞	∞	∞	∞
fixez 1	**0**	7	9	∞	∞	14
fixez 2	0	**7**	9	22	∞	14
fixez 3	0	7	**9**	20	∞	11
fixez 6	0	7	9	20	20	**11**
fixez 4	0	7	9	**20**	20	11
fixez 5	0	7	9	20	**20**	11

verde = nodul fixat la pasul respectiv (distanța lui nu se mai schimbă)

Dijkstra

Pas	Nod fixat	Ce se îmbunătățește
1	1 (d = 0)	d[2] = 7, d[3] = 9, d[6] = 14
2	2 (d = 7)	d[4] = 7 + 15 = 22 (3 rămâne 9, fiindcă 7 + 10 = 17 e mai mult)
3	3 (d = 9)	d[4] = 9 + 11 = 20 (mai bine decât 22); d[6] = 9 + 2 = 11 (mai bine decât 14)
4	6 (d = 11)	d[5] = 11 + 9 = 20
5	4 (d = 20)	nimic nou

Pas	Nod fixat	Ce se îmbunătățește
6	5 (d = 20)	nimic nou



verde: muchiile folosite de drumurile minime (formează un arbore)

Arborele drumurilor

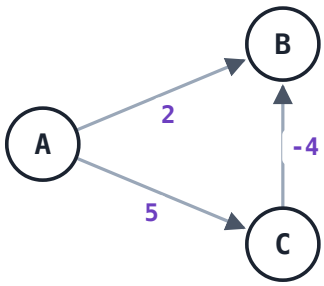
```cpp
#include <iostream>
using namespace std;

const int INF = 1000000000;
int n, m, s, c[101][101], d[101];
bool viz[101];

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            c[i][j] = (i == j) ? 0 : INF;
    for (int i = 1; i <= m; i++)
    {
        int x, y, z;
        cin >> x >> y >> z;
        c[x][y] = c[y][x] = z;
    }
    cin >> s;
    for (int i = 1; i <= n; i++) d[i] = c[s][i];   // distanțe inițiale: doar muchiile
directe
    viz[s] = true;
    for (int pas = 1; pas < n; pas++)
    {
        int mn = INF, x = 0;
        for (int i = 1; i <= n; i++)               // nodul nefixat cu d minim
            if (!viz[i] && d[i] < mn) { mn = d[i]; x = i; }
        if (x == 0) break;                         // restul sunt inaccesibile
        viz[x] = true;
        for (int y = 1; y <= n; y++)               // îmbunătățim vecinii
            if (!viz[y] && c[x][y] < INF && d[x] + c[x][y] < d[y])
                d[y] = d[x] + c[x][y];
    }
    for (int i = 1; i <= n; i++) cout << d[i] << ' ';
    return 0;
}
```

DE CE AȘA?

De ce nodul cu d minim este definitiv? Orice alt drum spre el ar trece prin cel puțin un nod nefixat, iar toate au distanțe ≥ d minim; cum costurile nu sunt negative, drumul alternativ nu poate fi mai scurt. **De ce nu merge cu costuri negative?** Exact acest argument cade: un cost negativ poate „răscumpăra” un ocol.



Dijkstra fixează B cu 2; drumul A → C → B costă 5 − 4 = 1, mai bun

Cost negativ

Complexitate: **O(n²)** (n pași, fiecare cu două parcurgeri de n). Bun pentru n ≤ 2000–3000.

> **DE ȘTIUT**
>
> **Cum obții și drumul, nu doar costul?** Păstrează `tata[y] = x` de fiecare dată când îmbunătățești `d[y]` prin x. Apoi reconstruiești de la destinație spre start (ca la Lee, lecția XI.17).

3. Roy-Floyd: drumuri minime între oricare două noduri

Ideea. Pentru fiecare nod k, întrebăm: „pot ajunge de la i la j mai ieftin trecând prin k?". Dacă da, actualizăm. După ce am încercat toți k, matricea conține distanțele minime între toate perechile.

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

matricea costurilor (înainte)

	1	2	3	4
1	0	3	**5**	**6**
2	**5**	0	2	**3**
3	3	**6**	0	1
4	2	**5**	**7**	0

după Roy-Floyd (verde = valori îmbunătățite)

Roy-Floyd

Exemplu: de la 1 la 3 nu exista arc direct (∞). Prin nodul 2: 3 + 2 = 5. Deci `c[1][3] = 5`.

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

const int INF = 1000000000;
int n, m, c[101][101];

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            c[i][j] = (i == j) ? 0 : INF;
    for (int i = 1; i <= m; i++)
    {
        int x, y, z;
        cin >> x >> y >> z;
        c[x][y] = z;                                // graf orientat
    }
    for (int k = 1; k <= n; k++)                    // nodul intermediar: cel mai exterior!
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                if (c[i][k] < INF && c[k][j] < INF && c[i][k] + c[k][j] < c[i][j])
                    c[i][j] = c[i][k] + c[k][j];
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++) cout << c[i][j] << ' ';
        cout << '\n';
    }
    return 0;
}
```

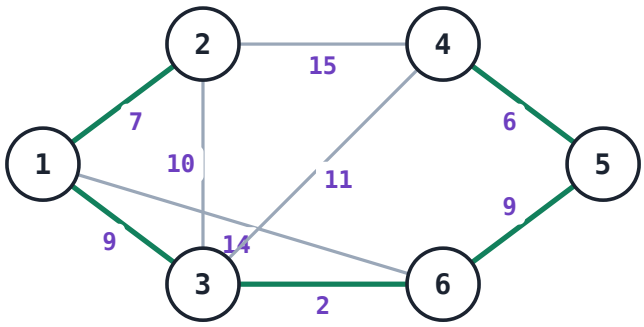
DE CE AȘA?

De ce `k` este bucla exterioară? După ce ai terminat pasul k, `c[i][j]` este cel mai bun drum care folosește ca intermediare doar nodurile 1...k. Dacă inversezi ordinea buclelor, această proprietate se pierde și răspunsul poate fi greșit (o greșeală foarte frecventă). **De ce testul `< INF`?** Fără el aduni `INF + ceva` și poți „îmbunătăți" o valoare cu un drum care nu există.

Complexitate **O(n³)**: n ≤ 200–400. Dijkstra de la fiecare nod ar costa tot $n \cdot n^2 = n^3$, dar Floyd are cod mult mai scurt.

4. Arbore parțial de cost minim (APM)

Un orășel are n localități și vrea să le lege cu cabluri; fiecare legătură posibilă are un preț. Vrem **conectarea tuturor** cu **costul total minim**. Soluția are n − 1 muchii (un arbore: mai puține nu conectează, mai multe închid cicluri inutile și scumpe).

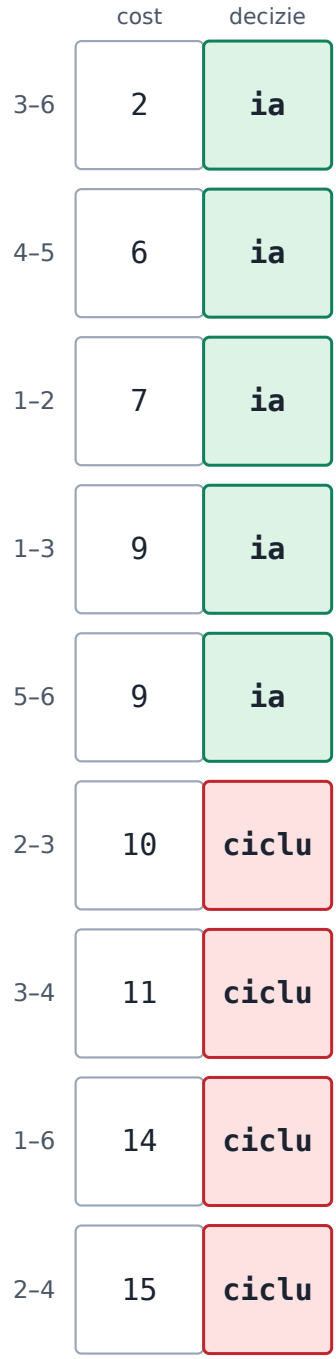


5 muchii = n − 1, fără cicluri, suma costurilor minimă

APM

Kruskal: muchiile cele mai ieftine primele

Sortezi muchiile după cost și le iei pe rând, dar **doar dacă nu închid un ciclu** (adică leagă două componente diferite).



o muchie se ia doar dacă leagă două componente diferite (altfel ar închide un ciclu)

Kruskal

Pas	Muchie (cost)	Componente înainte	Decizie
1	3–6 (2)	{3}, {6}	ia
2	4–5 (6)	{4}, {5}	ia
3	1–2 (7)	{1}, {2}	ia
4	1–3 (9)	{1,2}, {3,6}	ia
5	5–6 (9)	{4,5}, {1,2,3,6}	ia (5 muchii: gata)
6	2–3 (10)	aceeași componentă	respinsă (ciclu)

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

struct muchie { int x, y, cost; };
muchie e[10001];
int n, m, comp[101];

bool cmp(muchie a, muchie b) { return a.cost < b.cost; }

int main()
{
    cin >> n >> m;
    for (int i = 0; i < m; i++) cin >> e[i].x >> e[i].y >> e[i].cost;
    sort(e, e + m, cmp);                              // crescător după cost
    for (int i = 1; i <= n; i++) comp[i] = i;         // fiecare nod: componenta lui
    int total = 0, luate = 0;
    for (int k = 0; k < m && luate < n - 1; k++)
        if (comp[e[k].x] != comp[e[k].y])             // componente diferite → nu face ciclu
        {
            total += e[k].cost;
            luate++;
            int vechi = comp[e[k].y], nou = comp[e[k].x];
            for (int i = 1; i <= n; i++)              // unim componentele
                if (comp[i] == vechi) comp[i] = nou;
        }
    cout << total;
    return 0;
}
```

DE CE AȘA?

De ce greedy (alegerea celei mai ieftine muchii) dă optimul aici, dar nu la monede (lecția XI.15)? Pentru APM se demonstrează că cea mai ieftină muchie care nu închide ciclu poate fi inclusă întotdeauna într-o soluție optimă („proprietatea de tăiere"). La monede nu există o astfel de garanție. **De ce `comp[x] != comp[y]` înseamnă „fără ciclu"?** O muchie între două noduri din aceeași componentă ar crea un al doilea drum între ele, deci un ciclu.

Complexitate: sortarea O(m log m) + unirile O(n · n). Cu **păduri de mulțimi disjuncte (DSU)** unirile devin aproape O(1) (vezi „Mai departe").

Prim: arborele crește dintr-un nod

Pornești cu nodul 1. La fiecare pas adaugi cea mai ieftină muchie care leagă arborele de un nod **din afara** lui. Pe graful nostru: 1–2 (7), 1–3 (9), 3–6 (2), 6–5 (9), 5–4 (6): total 33, la fel ca Kruskal (APM-ul poate diferi la costuri egale, dar costul total este același).

Algoritmul seamănă cu Dijkstra: `d[i]` = cel mai ieftin cost al unei muchii care leagă nodul i de arbore.

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

const int INF = 1000000000;
int n, m, c[101][101], d[101];
bool inArbore[101];

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            c[i][j] = INF;
    for (int i = 1; i <= m; i++)
    {
        int x, y, z;
        cin >> x >> y >> z;
        c[x][y] = c[y][x] = z;
    }
    for (int i = 1; i <= n; i++) d[i] = c[1][i];
    inArbore[1] = true;
    int total = 0;
    for (int pas = 1; pas < n; pas++)
    {
        int mn = INF, x = 0;
        for (int i = 1; i <= n; i++)                // nodul din afară cel mai aproape de
arbore
            if (!inArbore[i] && d[i] < mn) { mn = d[i]; x = i; }
        inArbore[x] = true;
        total += mn;
        for (int y = 1; y <= n; y++)
            if (!inArbore[y] && c[x][y] < d[y]) d[y] = c[x][y];
    }
    cout << total;
    return 0;
}
```

DE CE AȘA?

Diferența față de Dijkstra. Dijkstra compară `d[x] + c[x][y]` (costul **întregului drum** de la sursă); Prim compară doar `c[x][y]` (costul **muchiei** care leagă y de arbore). Aceeași structură, alt criteriu.

	Dijkstra	Roy-Floyd	Prim	Kruskal
Răspunde la	drum minim dintr-un nod	drum minim între toate perechile	APM	APM
Complexitate	$O(n^2)$	$O(n^3)$	$O(n^2)$	O(m log m + ...)
Costuri negative	nu	da (fără cicluri negative)	da	da
Graf	orientat sau nu	orientat sau nu	neorientat conex	neorientat

	Dijkstra	Roy-Floyd	Prim	Kruskal
Bun pentru	graf mare, o sursă	n mic, multe întrebări	graf dens	graf rar

5. Greșeli frecvente

Greșeala	Efect	Corect
`c[i][j] = 0` pentru „fără muchie”	algoritmii folosesc muchii inexistente	INF
Floyd cu `k` în buclă interioară	rezultate greșite	`k` este cel mai exterior
uitat `< INF` la adunare	depășire / drumuri false	testează `c[i][k] < INF && c[k][j] < INF`
Dijkstra pe costuri negative	distanțe greșite	Bellman-Ford sau Floyd
Kruskal fără sortare	nu este minim	sortezi după cost
`int` pentru totaluri mari	depășire	`long long`

6. La BAC

> **LA BAC**
>
> Subiectul cu grafuri cu costuri apare rar la BAC (în special arbori parțiali: „câte muchii trebuie eliminate ca să rămână un arbore” → **m – (n – 1)**, dacă graful este conex). Dijkstra, Floyd, Prim, Kruskal sunt pentru **olimpiadă** și pentru întrebările de ordine din probleme de algoritmică.

7. Exersează pe hârtie

1. Aplică Dijkstra din nodul 4 pe graful din lecție. Scrie vectorul d după fiecare pas.

2. Ce distanță minimă este între 1 și 5 pe graful din lecție? Care este drumul?

3. Un graf conex are 8 noduri și 15 muchii. Câte muchii trebuie eliminate ca să obții un arbore parțial?

4. Aplică Kruskal pe un graf cu muchiile: A–B 4, A–C 1, B–C 2, B–D 5, C–D 8, C–E 10, D–E 2. Care este costul APM?

5. Costurile sunt negative: A→B 2, A→C 5, C→B −4. Ce distanță A→B calculează Dijkstra? Care este cea corectă?

6. Pe graful orientat 1→2 (4), 2→3 (1), 1→3 (7), 3→1 (2): scrie matricea costurilor, apoi matricea Roy-Floyd.

7. De ce Prim și Kruskal pot da arbori diferiți, dar cu același cost?

8. Câte muchii are un APM al unui graf cu n noduri și p componente conexe (pădure parțială)?

9. Pentru un graf cu 1000 de noduri și 3000 de muchii, ce algoritm alegi pentru drumuri minime din nodul 1: Dijkstra $O(n^2)$ sau Floyd $O(n^3)$? Ce număr aproximativ de operații fac?

10. (olimpiadă) Cum modifici Dijkstra ca să numeri **câte** drumuri minime există din 1 la fiecare nod?

8. Mai departe (pentru cine vrea mai mult)

Dijkstra cu `priority_queue`, O(m log n). Pui în coadă perechi (distanță, nod) și scoți mereu cea mai mică; folosit pe grafuri cu zeci de mii de noduri.

```
priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> pq;
pq.push({0, s}); d[s] = 0;
while (!pq.empty())
{
    auto [dist, x] = pq.top(); pq.pop();
    if (dist > d[x]) continue;                    // intrare veche
    for (auto [y, cost] : g[x])
        if (d[x] + cost < d[y]) { d[y] = d[x] + cost; pq.push({d[y], y}); }
}
```

Bellman-Ford. Acceptă costuri negative, O(n · m): relaxează toate muchiile de n − 1 ori; la a n-a rundă detectează cicluri negative.

DSU (union-find). Structură pentru „sunt x și y în aceeași componentă?", aproape O(1) cu compresia căilor: face Kruskal O(m log m).

Probleme de olimpiadă. Drum minim cu muchii „gratuite" (CSES Flight Discount), APM cu muchii obligatorii, al doilea cel mai bun arbore.

Rezumat

- Matricea costurilor: `INF` = fără muchie, `0` pe diagonală.
- **Dijkstra**: fixezi nodul nefixat cu d minim, îmbunătățești vecinii; fără costuri negative; $O(n^2)$.
- **Roy-Floyd**: `k` exterior, toate perechile, $O(n^3)$.
- **APM**: n − 1 muchii; **Kruskal** (sortare + fără cicluri) sau **Prim** (arbore care crește).
- Eliminarea muchiilor pentru a obține arbore: **m – (n – 1)**.

Probleme de lucru (pbinfo)

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

- [ ] `#589` **roy-floyd** — ușoară
  https://www.pbinfo.ro/probleme/589/roy-floyd
- [ ] `#588` **dijkstra** — ușoară
  https://www.pbinfo.ro/probleme/588/dijkstra

Temă (pbinfo)

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

- [ ] `#590` **prim** — ușoară
  https://www.pbinfo.ro/probleme/590/prim

- [ ] #592 **kruskal** — ușoară
  https://www.pbinfo.ro/probleme/592/kruskal
- [ ] #1651 **graf** — ușoară
  https://www.pbinfo.ro/probleme/1651/graf
- [ ] #1652 **rf** — ușoară
  https://www.pbinfo.ro/probleme/1652/rf
- [ ] #591 **firma** — medie
  https://www.pbinfo.ro/probleme/591/firma

Grile din subiectele de BAC (pbinfo)

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

- [ ] #793 **Exercițiul #793**
  câte muchii se elimină ca să rămână un arbore (n − 1 muchii)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=793
- [ ] #1683 **Exercițiul #1683**
  același raționament: graf conex → arbore parțial
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1683
- [ ] #1149 **Exercițiul #1149**
  graf complet: câte muchii trebuie eliminate pentru a obține un arbore
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1149
- [ ] #505 **Exercițiul #505**
  graf parțial cu componente conexe: număr de muchii
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=505

Citește și exersează în plus

- Algoritmul Roy-Floyd https://www.pbinfo.ro/articole/5891/drumuri-de-cost-minim-intr-un-graf-orientat
- Algoritmul lui Dijkstra https://www.pbinfo.ro/articole/6135/algoritmul-lui-dijkstra
- Algoritmul lui Prim https://www.pbinfo.ro/articole/2443/algoritmul-lui-prim
- Algoritmul lui Kruskal https://www.pbinfo.ro/articole/19048/algoritmul-lui-kruskal
- VisuAlgo — Single-Source Shortest Paths — Dijkstra animat pas cu pas, pe grafuri desenate de tine (engleză) https://visualgo.net/en/sssp
- VisuAlgo — Minimum Spanning Tree — Prim și Kruskal animate pe același graf (engleză) https://visualgo.net/en/mst
- CSES Problem Set — Graph Algorithms — Shortest Routes, Flight Discount, Road Reparation: probleme cu verificare automată (engleză) https://cses.fi/problemset/
- Wikipedia — Algoritmul lui Dijkstra — descriere și istoric
  https://ro.wikipedia.org/wiki/Algoritmul_lui_Dijkstra

Răspunsuri

Exersează pe hârtie

1. Din 4: d = (∞, ∞, ∞, 0, ∞, ∞). Fixez 4: d[2] = 15, d[3] = 11, d[5] = 6. Fixez 5 (6): d[6] = 6 + 9 = 15. Fixez 3 (11): d[1] = 20, d[2] = min(15, 21) = 15, d[6] = min(15, 13) = 13. Fixez 6 (13): d[1] = min(20, 27) = 20. Fixez 2 (15), apoi 1 (20). Rezultat final: **d = (20, 15, 11, 0, 6, 13)**.

2. 20. Drumul 1 → 3 → 6 → 5 (9 + 2 + 9 = 20). Alt drum, 1 → 3 → 4 → 5, costă 9 + 11 + 6 = 26.

3. 15 − (8 − 1) = **8** muchii.

4. Sortate: A–C 1, B–C 2, D–E 2, A–B 4 (ciclu, respinsă), B–D 5 (leagă {A,B,C} cu {D,E}: ia), … Muchiile alese: A–C 1, B–C 2, D–E 2, B–D 5 → costul **10**.

5. Dijkstra fixează B cu 2 (înainte de C), deci spune **2**. Corect: A → C → B = 5 − 4 = **1**.

6. Matricea costurilor: (0, 4, 7 / ∞, 0, 1 / 2, ∞, 0). După Floyd: (0, 4, **5** / **3**, 0, 1 / 2, **6**, 0). De exemplu 1→3: prin 2: 4 + 1 = 5 < 7; 2→1: prin 3: 1 + 2 = 3; 3→2: prin 1: 2 + 4 = 6.

7. Când mai multe muchii au același cost, există mai multe alegeri la fel de bune; arborii diferă, dar costul minim este unic.

8. Pe fiecare componentă are $n_i - 1$ muchii, în total **n – p** muchii.

9. Dijkstra: ~ $n^2 = 10^6$ operații; Floyd: $n^3 = 10^9$ (prea mult pentru o secundă). Alegi **Dijkstra** (sau varianta cu `priority_queue`, ~ 3000 · log 1000 ≈ 30 000 operații).

10. Păstrezi `nr[x]` = numărul de drumuri minime până la x. La îmbunătățire strictă (`d[x] + c < d[y]`): `nr[y] = nr[x]`; la egalitate (`d[x] + c == d[y]`): `nr[y] += nr[x]`.