AVANSAT · OLIMPIADĂ · LECȚIA 6

Mulțimi disjuncte (DSU / Union-Find) și algoritmul lui Kruskal

Durată: 75 min Nivel: olimpiadă / informatică intensiv Înainte: lecțiile xi-10, xi-19, co-01

CE VEI ȘTI DUPĂ LECȚIE

- Reții grupurile (mulțimile disjuncte) ca o pădure de arbori și implementezi find și union.
- Aplici compresia căilor și unirea după mărime și explici de ce fac operațiile aproape O(1).
- Numeri componente conexe care apar pe măsură ce adaugi muchii.
- Implementezi algoritmul lui Kruskal pentru arborele parțial de cost minim.

1. Problema: „sunt în același grup?"

Ai n persoane. Pe rând, aflii că „a și b sunt prieteni" (și deci în același grup, prin prieteni de prieteni). Între timp ți se pun întrebări: „sunt a și b în același grup?". Cu un BFS la fiecare întrebare ai O(n + m) pe întrebare: prea lent pentru 10^5 întrebări. Structura **DSU** (*Disjoint Set Union*, sau **Union-Find**) rezolvă ambele operații în timp aproape constant.

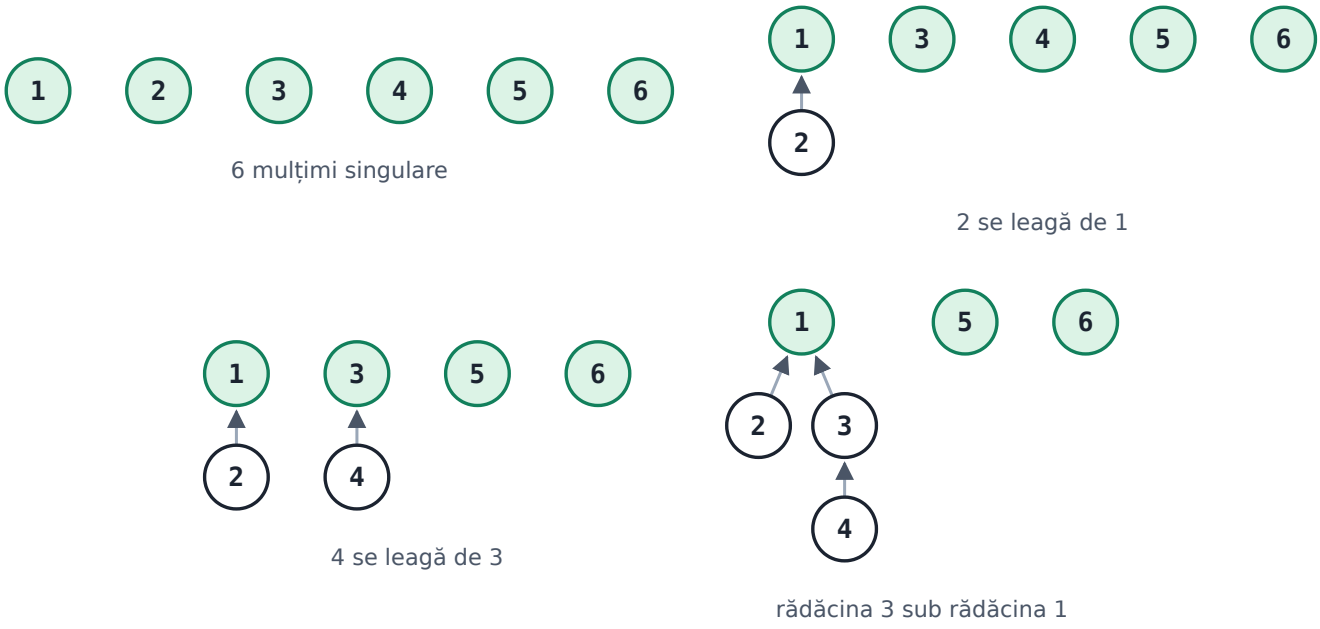
DSU ține **mulțimi disjuncte** (fiecare element este în exact o mulțime) și oferă două operații:

- **find(x)**: „care este reprezentantul (rădăcina) mulțimii lui x?";
- **union(a, b)**: „unește mulțimile lui a și b".

Doi elemente sunt în același grup exact când `find(a) == find(b)`.

2. Pădure de arbori

Fiecare mulțime este un **arbore** în care fiecare nod își reține **tatăl** (`tata[x]`); rădăcina este propriul ei tată (`tata[r] = r`). Reprezentantul mulțimii este rădăcina.



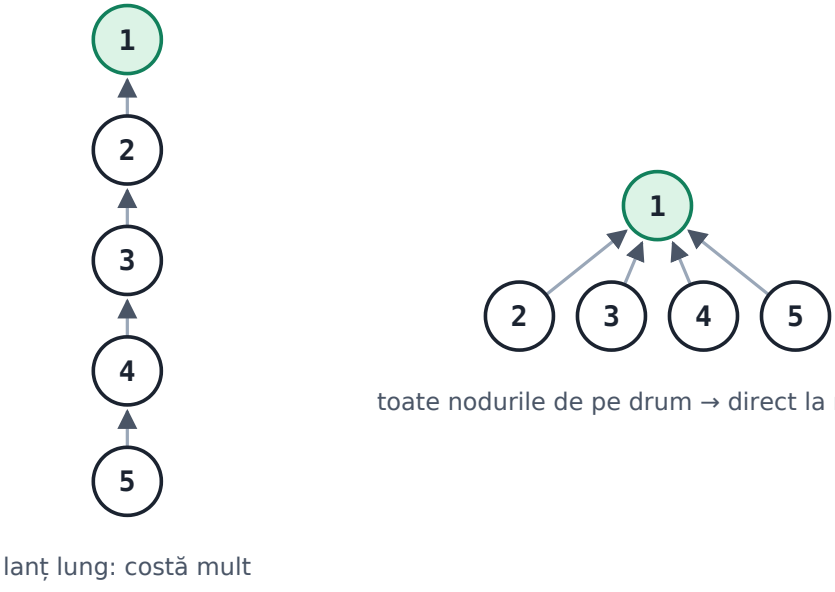
Pădurea DSU

Operația	Ce face
`find(x)`	urcă din tată în tată până găsește rădăcina
`union(a, b)`	află rădăcinile lui a și b; dacă sunt diferite, leagă o rădăcină sub cealaltă

Doar cu acestea, arborele poate ajunge un **lanț** (1 ← 2 ← 3 ← ... ← n), iar `find` costă O(n). Două îmbunătățiri simple repară asta.

Îmbunătățirea 1: compresia căilor

Când `find(5)` urcă prin 4, 3, 2 până la rădăcina 1, legăm **toate** nodurile traversate **direct** de rădăcină. Următorul `find` pe oricare dintre ele are un singur pas.



Compresia căilor

Îmbunătățirea 2: unirea după mărime

La `union`, atașăm arborele **mai mic** sub cel **mai mare** (reținem `sz[r]` = numărul de noduri). Astfel, înălțimea arborilor crește foarte încet.

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

int tata[100005], sz[100005];

int gaseste(int x)                          // rădăcina mulțimii lui x
{
    if (tata[x] == x) return x;
    return tata[x] = gaseste(tata[x]);      // compresia căilor: x se leagă direct de
rădăcină
}

void uneste(int a, int b)
{
    a = gaseste(a);
    b = gaseste(b);
    if (a == b) return;                     // sunt deja în aceeași mulțime
    if (sz[a] < sz[b]) swap(a, b);          // atașăm arborele mai mic sub cel mai mare
    tata[b] = a;
    sz[a] += sz[b];
}

int main()
{
    int n, q;
    cin >> n >> q;
    for (int i = 1; i <= n; i++) { tata[i] = i; sz[i] = 1; }
    while (q--)
    {
        int tip, a, b;
        cin >> tip >> a >> b;
        if (tip == 1) uneste(a, b);
        else cout << (gaseste(a) == gaseste(b) ? "DA" : "NU") << '\n';
    }
    return 0;
}
```

Operația	Efect	Răspuns
`1 1 2`	uniunea grupurilor lui 1 și 2	–
`2 1 2`	find(1) = find(2) = 1	DA
`2 1 3`	find(1) = 1, find(3) = 3	NU
`1 3 4`	4 sub 3	–
`1 2 4`	rădăcinile 1 și 3 au aceeași mărime (2): 3 sub 1	–
`2 1 3`	find(3) = 1	DA
`2 5 6`	5 și 6 nu au fost unite	NU

DE CE AȘA?

De ce se unesc rădăcinile și nu nodurile `a` și `b` direct? Apartenența la o mulțime este dată de rădăcină. Dacă legi `tata[a] = b` fără să cauți rădăcinile, rupi mulțimea lui `a` de restul nodurilor ei. **Cât costă acum operațiile?** Cu ambele îmbunătățiri, un șir de m operații costă O(m · α(n)), unde α este inversa funcției lui Ackermann: sub 5 pentru orice n imaginabil. În practică spui „O(1) amortizat". Chiar și **numai** cu compresia, costul este O(log n) amortizat: pentru concurs, compresia singură ajunge de obicei, dar ambele sunt ușor de scris. **De ce `a == b → return` ?** Dacă sunt deja în aceeași mulțime, a lega rădăcina de ea însăși ar crea un ciclu de lungime 1 (și ar strica `sz`).

ATENȚIE — GREȘELI FRECVENTE

Nu uita inițializarea: `tata[i] = i` și `sz[i] = 1` pentru fiecare i. Dacă `tata` rămâne 0 (global), `gaseste` urcă la 0 și toate elementele par a fi în același grup.

3. Câte componente după fiecare muchie?

Citim muchiile unui graf pe rând; după fiecare muchie afișăm câte componente conexe are graful și mărimea celei mai mari. Fiecare muchie care unește **două mulțimi diferite** scade numărul de componente cu 1; o muchie în interiorul unei componente nu schimbă nimic.

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

int tata[100005], sz[100005];

int gaseste(int x)
{
    if (tata[x] == x) return x;
    return tata[x] = gaseste(tata[x]);
}

int main()
{
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) { tata[i] = i; sz[i] = 1; }
    int componente = n;                    // la început fiecare nod e singur
    int maxim = 1;                         // dimensiunea celei mai mari componente
    while (m--)
    {
        int a, b;
        cin >> a >> b;
        a = gaseste(a);
        b = gaseste(b);
        if (a != b)
        {
            if (sz[a] < sz[b]) swap(a, b);
            tata[b] = a;
            sz[a] += sz[b];
            componente--;                  // două componente au devenit una
            maxim = max(maxim, sz[a]);
        }
        cout << componente << ' ' << maxim << '\n';
    }
    return 0;
}
```

Muchia	Efect	Componente	Cea mai mare
1–2	unește {1} și {2}	5	2
3–4	unește {3} și {4}	4	2
2–4	unește {1,2} și {3,4}	3	4
1–3	1 și 3 sunt deja împreună: nu schimbă nimic	3	4
5–6	unește {5} și {6}	2	4

DE ȘTIUT

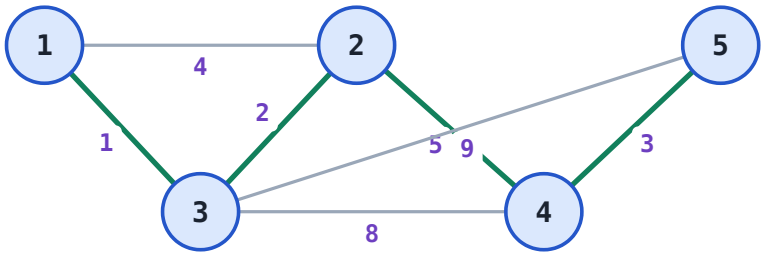
Observația „o muchie care leagă două noduri **deja** în aceeași componentă închide un **ciclu**" este tot ce trebuie ca să detectezi cicluri într-un graf neorientat cu DSU: dacă `gaseste(a) == gaseste(b)` înainte de uniune, muchia (a, b) închide un ciclu.

4. Kruskal: arborele parțial de cost minim

Problemă. Un graf conex cu costuri pe muchii. Alege n − 1 muchii care leagă toate nodurile, cu suma costurilor minimă (**arbore parțial de cost minim**, APM).

Algoritmul lui Kruskal (greedy): 1. sortează muchiile crescător după cost;

2. parcurge-le în ordine; **alege** muchia dacă leagă două componente diferite (altfel ar închide un ciclu), apoi unește componentele;
3. oprește-te când ai n − 1 muchii alese.



verde = muchii alese: 1 + 2 + 3 + 5 = 11 · (1,2), (3,4), (3,5) ar închide cicluri

Kruskal

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

struct Muchie { int a, b, cost; };

int tata[100005], sz[100005];

int gaseste(int x)
{
    if (tata[x] == x) return x;
    return tata[x] = gaseste(tata[x]);
}

bool uneste(int a, int b)                  // întoarce true dacă a unit două mulțimi diferite
{
    a = gaseste(a);
    b = gaseste(b);
    if (a == b) return false;
    if (sz[a] < sz[b]) swap(a, b);
    tata[b] = a;
    sz[a] += sz[b];
    return true;
}

int main()
{
    int n, m;
    cin >> n >> m;
    vector<Muchie> e(m);
    for (auto &x : e) cin >> x.a >> x.b >> x.cost;
    sort(e.begin(), e.end(), [](const Muchie &p, const Muchie &q) { return p.cost < q.cost; });
    for (int i = 1; i <= n; i++) { tata[i] = i; sz[i] = 1; }
    long long total = 0;
    int alese = 0;
    for (auto &x : e)
        if (uneste(x.a, x.b))              // muchia nu închide ciclu: o alegem
        {
            total += x.cost;
            alese++;
        }
    if (alese == n - 1) cout << total;
    else cout << "graful nu este conex";
    return 0;
}
```

Muchia (cost)	Decizie	Componente
(1,3) 1	alege	{1,3} {2} {4} {5}
(2,3) 2	alege	{1,2,3} {4} {5}
(4,5) 3	alege	{1,2,3} {4,5}

Muchia (cost)	Decizie	Componente
(1,2) 4	1 și 2 deja împreună: **sare**	–
(2,4) 5	alege	{1,2,3,4,5}

Au fost alese 4 = n − 1 muchii: 1 + 2 + 3 + 5 = **11**.

> **DE CE AȘA?**
>
> **De ce este corect să alegem mereu cea mai ieftină muchie care nu închide ciclu?** Dacă o muchie `e` de cost minim leagă două componente `X` și `Y`, există un APM care o conține: într-un APM care nu o are, adăugând `e` apare un ciclu; scoți din ciclu o altă muchie care trece între `X` și `Y`, care costă cel puțin cât `e`: suma nu crește. Alegerea lacomă nu strică niciodată optimul. **Cât costă?** Sortarea domină: O(m log m); apoi m operații DSU, aproape O(1) fiecare. **De ce verificăm `alese == n - 1`?** Dacă graful nu este conex, nu se pot alege n − 1 muchii: ai obținut o **pădure** de arbori (un APM pe fiecare componentă).

5. Ce alte probleme se rezolvă cu DSU?

Problema	Idee
grupuri de prieteni / echivalențe	`union` pentru fiecare relație; numărul de grupuri = numărul de rădăcini
detectarea unui ciclu într-un graf neorientat	muchie cu capetele deja în aceeași mulțime
componente conexe într-o matrice (zone de același fel)	unește celulele vecine cu aceeași valoare
conexiuni care apar în timp („după fiecare muchie ...")	DSU la fiecare adăugare
Kruskal (APM)	sortare + DSU
ștergeri de muchii (offline)	procesezi invers, ca adăugări

> **DE ȘTIUT**
>
> DSU știe doar să **uneze** mulțimi, nu să le despartă. Dacă o problemă cere ștergeri, caută varianta „offline, în ordine inversă": muchiile șterse devin muchii adăugate. DSU este potrivit și pentru grafuri **neorientate**; pentru drumuri în grafuri orientate nu ajută.

6. Greșeli frecvente

Greșeala	Ce se întâmplă	Corect
`tata[i] = i` nefăcut	toate elementele par unite (tata 0)	inițializare în `main`
`tata[a] = b` fără `gaseste`	rupi mulțimea lui a	unește rădăcinile

Greșeala	Ce se întâmplă	Corect
`gaseste` fără compresie, fără mărime	O(n) pe operație în cazul rău	adaugă `tata[x] = gaseste(tata[x])`
recursivitate adâncă pe lanț foarte lung	depășirea stivei (rar, cu unirea după mărime)	folosește și unirea după mărime sau varianta iterativă
Kruskal: nu sortezi muchiile	arbore corect dar fără cost minim	sort crescător după cost
Kruskal: costul în `int`	depășire	`long long`
uiți să verifici conexitatea	afișezi un cost parțial	`alese == n - 1`

7. La olimpiadă

DSU apare ca **instrument** în multe probleme (nu doar ca „problemă DSU"): „după fiecare muchie, câte componente?", „primul moment în care grafurile devin conexe", „numărul minim de muchii de adăugat ca să devină conex" (răspunsul este numărul de componente − 1). Kruskal este algoritmul-standard pentru APM la n, m mari: Prim cu heap este alternativa.

8. Exersează pe hârtie

1. Cu n = 5, execută: uneste(1,2), uneste(3,4), uneste(2,3). Răspunde: același grup (1,4)? (1,5)?

2. După operațiile de la exercițiul 1, ce mărimi au grupurile?

3. De ce, după `gaseste(5)` cu compresia căilor, un al doilea `gaseste(5)` costă un singur pas?

4. Rulează Kruskal pe graful cu 4 noduri: (1,2) cost 3, (2,3) cost 1, (3,4) cost 4, (1,4) cost 2, (1,3) cost 5. Care e costul APM și ce muchii alege?

5. Un graf cu 8 noduri are muchiile (1,2), (2,3), (4,5), (6,7), (7,8), (8,6), (3,1). Câte componente conexe are? Care muchii închid cicluri?

6. Câte muchii trebuie adăugate la un graf cu c componente ca să devină conex?

7. Cum numeri grupurile de prieteni după ce ai citit toate perechile (n persoane, m perechi)?

8. Ce schimbi în Kruskal ca să obții un arbore parțial de cost **maxim**?

9. Dă un exemplu de mică problemă în care DSU **nu** ajută (indiciu: ștergeri sau graf orientat).

10. Estimează numărul de operații pentru Kruskal cu n = 10^5, m = $2 \cdot 10^5$.

9. Mini-grile

1. După `uneste(a, b)`, `gaseste(a) == gaseste(b)` este: a) mereu adevărat b) mereu fals c) adevărat doar dacă `a == b` d) imposibil de știut

2. Compresia căilor face ca: a) arborele să fie mai mic b) nodurile traversate să se lege direct de rădăcină c) rădăcina să se schimbe d) mulțimile să se separe

3. În Kruskal, o muchie se sare dacă: a) este cea mai scumpă b) capetele ei sunt deja în aceeași componentă c) are cost negativ d) este prima

4. Un graf conex cu n noduri are APM cu: a) n muchii b) n − 1 muchii c) n + 1 muchii d) m muchii

10. Mai departe (pentru cine vrea mai mult)

- **DSU cu rang** (în loc de mărime) și **DSU cu informații în rădăcină** (minim, sumă, număr de elemente).
- **DSU cu potențial** (diferența față de rădăcină), pentru probleme cu relații „a este cu x mai mare decât b".
- **Prim** cu `priority_queue`: alt algoritm pentru APM (lecția XI.19).
- **Arbore de intervale / LCA** peste un APM: probleme cu drumuri în arbore.
- **Oglindirea problemei:** „muchia de cost maxim pe drumul dintre a și b din APM" (minimax).

Rezumat

- DSU = pădure de arbori; `find` urcă la rădăcină, `union` leagă două rădăcini.
- Compresia căilor + unirea după mărime → operații aproape O(1) amortizat.
- Muchie între două noduri din aceeași mulțime = ciclu.
- **Kruskal:** sortează muchiile, ia-o dacă leagă două componente diferite; O(m log m).
- Se oprește la n − 1 muchii alese; dacă nu s-au găsit, graful nu este conex.

Probleme de lucru (concurs)

Probleme de pe infoarena, CSES, Kilonova sau pbinfo, în ordinea dificultății. Trimite soluțiile pe platformă și bifează ce ai terminat.

- [ ] `infoarena` **disjoint** — ușoară
  problema clasică DSU: exact programul din lecție
  https://www.infoarena.ro/problema/disjoint
- [ ] `CSES` **Building Roads** — ușoară
  câte drumuri trebuie adăugate ca graful să devină conex
  https://cses.fi/problemset/task/1666
- [ ] `CSES` **Road Reparation** — medie
  arbore parțial de cost minim cu Kruskal
  https://cses.fi/problemset/task/1675
- [ ] `infoarena` **apm** — medie
  arbore parțial de cost minim, cu afișarea muchiilor
  https://www.infoarena.ro/problema/apm

Temă (concurs)

Mai grele: încearcă-le după ce le-ai terminat pe cele de mai sus.

- [ ] CSES **Road Construction** — medie
  componentele și dimensiunea maximă după fiecare muchie (exemplul c din lecție)
  https://cses.fi/problemset/task/1676
- [ ] CSES **Flight Routes Check** — dificilă
  graf orientat: DSU nu mai ajunge; ce algoritm folosești? (tare conexitate)
  https://cses.fi/problemset/task/1682

Citește și exersează în plus

- CP-Algorithms — Disjoint Set Union — articolul de referință: compresie, uniune după rang, aplicații (engleză) https://cp-algorithms.com/data_structures/disjoint_set_union.html
- USACO Guide — Disjoint Set Union — probleme graduate de concurs (engleză) https://usaco.guide/gold/dsu
- VisuAlgo — Union-Find — animație pentru find și union (engleză) https://visualgo.net/en/ufds

Răspunsuri

Exersează pe hârtie

1. (1,4): DA (1–2 și 3–4 s-au unit prin 2–3); (1,5): NU.

2. Grupul {1,2,3,4} are mărimea 4; grupul {5} are mărimea 1.

3. Primul `gaseste(5)` leagă 5 (și nodurile de pe drum) direct de rădăcină; al doilea găsește rădăcina în 1 pas.

4. Muchiile sortate: (2,3) 1, (1,4) 2, (1,2) 3, (3,4) 4, (1,3) 5. Alege (2,3) și (1,4); (1,2) unește {2,3} cu {1,4}: se alege. Sunt 3 = n − 1 muchii: costul **6**, muchiile (2,3), (1,4), (1,2).

5. Componentele: {1,2,3}, {4,5}, {6,7,8}: **3 componente**. Închid cicluri: (8,6) și (3,1).

6. c − 1 muchii (câte una între două componente, pe rând).

7. Pentru fiecare pereche `union`, apoi numeri nodurile `i` cu `gaseste(i) == i` (rădăcinile), sau scazi 1 din n la fiecare uniune reușită.

8. Sortezi muchiile **descrescător** după cost (restul rămâne la fel).

9. Ștergerea unei muchii dintr-un graf neorientat („mai sunt a și b conectați?", online) sau tare conexitatea într-un graf orientat: DSU nu poate separa mulțimi și nu ține cont de direcția arcelor.

10. Sortare: $2 \cdot 10^5 \cdot \log_2(2 \cdot 10^5) \approx 3{,}5 \cdot 10^6$; DSU: $2 \cdot 10^5$ operații aproape O(1). În total sub $5 \cdot 10^6$ (zeci de milisecunde).

Mini-grile

1. a) mereu adevărat. **2.** b). **3.** b). **4.** b) n − 1 muchii.