

# Curs Lot Național Juniori

## Paduri de mulțimi disjuncte

Ștefan Dăscălescu

10 Mai 2023

## 1 Introducere

Structurile de date sunt de multe ori foarte utile în multe contexte în programare, acestea dovedindu-se a fi în special foarte puternice și esențiale în lucrul problemelor date la diverse olimpiade și concursuri de informatică. Deși se pot vorbi foarte multe lucruri despre acest capitol, astăzi voi discuta despre o structură de date care nu e la prima vedere foarte complicată față de alte structuri de date mai consacrate, dar care se dovedește a fi foarte puternică în rezolvarea multor probleme de toate felurile.

Așa cum sugerează și titlul, voi prezenta în acest curs pădurile de mulțimi disjuncte, sau union-find, denumire dată după cele două operații principale pe care această structură de date le oferă. Union-find poate fi folosit cu mare ușurință pentru probleme de tipul acelorora în care ni se cere să aflăm pe parcurs ce valori sunt legate între ele printr-o relație, presupunând că relațiile dintre valori se adaugă treptat. Pe parcurs se vor remarca diverse optimizări, precum și diferitele clase de probleme în care se poate folosi o asemenea structură de date.

Pentru ușurarea explicațiilor, vom presupune că avem o situație ipotetică în care avem  $n$  prieteni și ni se dau operații în care fie două persoane devin prietene, fie trebuie să decidem dacă două persoane aparțin aceluiași grup de prieteni.

## 2 Definirea operațiilor și funcționalității structurii de date

### 2.1 Fundamente

Pentru a reprezenta datele, vom ține într-un vector dimensiunea fiecărei mulțimi, iar într-un alt vector vom ține pentru fiecare poziție, nodul reprezentativ corespunzător grupului de prieteni din care face parte, la început fiecare nod fiind reprezentantul lui însuși.

```
1 // vectorii rad si card sunt de dimensiunea n+1
2 for(int i = 1; i <= n; ++i)
3     rad[i] = i, card[i] = 1;
```

### 2.2 Operația Union

La acest pas, ni se dau două persoane și trebuie să stabilim relația de prietenie dintre ei. Deși această operație se face în timp constant, contează foarte mult modul în care facem relația de atribuire, aceasta putând schimba radical complexitatea algoritmului. Astfel, voi introduce prima optimizare, și anume optimizarea de unire după cardinalul mulțimii, astfel încât vom uni mereu mulțimea cu cardinal mai mic la mulțimea cu cardinal mai mare.

Motivul pentru care această optimizare duce la o complexitate mai mică va fi dat de numărul mai mic de operații pe care funcția Find le va face la fiecare pas. De asemenea, această optimizare de a uni mulțimile mai mici la cele mai mari se regăsește în mod frecvent și în alte contexte în diverse structuri de date și nu numai.

```

1 void Union(int a, int b)
2 {
3     if(card[a] < card[b]) // vom vrea sa atasam b la a
4         swap(a, b);
5     rad[b] = a; // radacina lui b devine a
6     card[a] += card[b]; // crestem cardinalul lui a cu cardinalul lui b
7 }

```

## 2.3 Operația Find

La această operație, vrem să găsim pentru un nod, poziția nodului reprezentativ în structura noastră de date. În mod normal, această operație poate face cel mult  $O(n)$  pași, în cazul în care arborele rezultat ar fi un lanț. Totuși, putem să ne folosim de parcurgerile pe care le facem pentru a reține rezultatele pentru toate nodurile de pe parcursul aceluși drum, astfel încât la o parcurgere ulterioară, numărul de pași să se reducă spre un număr constant, structura arborelui ajungând similară cu cea a unui arbore stea.

```

1 int Find(int x)
2 {
3     if(rad[x] == x) // daca nodul e radacina
4         return x;
5     rad[x] = Find(rad[x]); // radacina nodului nostru e radacina radacinii curente
6     return rad[x];
7 }

```

## 2.4 Prime concluzii

Operația de union are complexitatea  $O(1)$ , iar operația de find are complexitatea  $O(n)$ . Totuși, datorită optimizărilor menționate mai sus (compresia drumurilor și unirea după dimensiunea mulțimilor), numărul total de operații făcute este  $O(n \log * n)$ , unde  $\log * x$  reprezintă inversul funcției Ackermann, valoare care se poate aproxima ca fiind o constantă.

De asemenea, nefolosirea optimizării de compresie a drumurilor ar duce la complexitatea  $O(n \log n)$ , rezultat foarte important în contextul altor optimizări, cum ar fi [tehnica small-to-large](#) sau în general în demonstrarea diverselor rezultate ce tin de sume armonice.

## 3 Problemă exemplu

[disjoint \(infoarena\)](#)

Pentru fiecare operație citită de la intrare, vom implementa funcțiile necesare pentru a obține rezultatul problemei. Unirea a două mulțimi implică mai întâi folosirea funcției Find pentru a găsi rădăcinile, iar mai apoi folosim funcția Union pentru a face unirea propriu-zisă. Folosirea ambelor optimizări pentru îmbunătățirea complexității duce la soluția optimă, ce rulează într-un timp aproximativ liniar raportat la numărul de valori citite.

[Sursă de 100 de puncte](#)

### 3.1 Oare putem implementa mai eficient?

Inițial, noi am implementat această structură folosind doi vectori, anume cel în care ținem cardinalul fiecărei mulțimi, precum și cel în care ținem rădăcina fiecărei mulțimi. Totuși, se poate observa faptul că noi folosim o grămadă de informație inutilă din cauza faptului că pentru fiecare număr, practic ne interesează doar dacă e o rădăcină a unei mulțimi de valori sau nu. Astfel, vom recurge la a reprezenta pozițiile corespunzătoare rădăcinilor cu numere negative, reprezentând  $-x$ , unde  $x$  e cardinalul mulțimii reprezentat de acea valoare, respectiv reprezentarea nodurilor adiacente cu numere pozitive, reprezentând rădăcina mulțimii din care acea valoare face parte.

[Sursă de 100 de puncte cu optimizarea de memorie](#)

## 4 Problema bile

[bile de pe infoarena](#)

”Pe o tabla patratica impartita in  $N*N$  patratele ( $N$  linii si  $N$  coloane), se afla asezate  $N*N$  bile (cate una in fiecare patrat a tablei). Lui Gigel ii plac bilele foarte mult, astfel ca el ia, pe rand, cate o bila de pe tabla, pana cand nu mai ramane pe tabla nici o bila. Gigel este, de asemenea, un baiat foarte curios. El a constatat ca bilele pot fi impartite in componente conexe, astfel:

fiecare bila face parte din exact o componenta conexa, iar daca 2 bile sunt invecinate pe orizontala sau verticala, atunci ele fac parte din aceeasi componenta conexa (adica daca una se afla imediat deasupra, dedesubtul, la dreapta sau la stanga celeilalte)

Dimensiunea unei componente conexe este egala cu numarul de bile care fac parte din componenta conexa respectiva. Dupa fiecare bila luata, Gigel vrea sa stie care este valoarea maxima dintre dimensiunile componentelor conexe din care fac parte bilele ramase.”

### 4.1 Soluție

Mai întâi, trebuie observat faptul că problema determinării conectivității dinamice este una foarte dificil de rezolvat [articol de pe wikipedia](#), deci nu are sens să ne chinuim cu asemenea implementări care nu fac obiectul cursului nostru sau în general a programelor olimpiadelor de informatică.

Asta ne duce cu gândul să încercăm să privim problema dintr-o perspectivă diferită, în special și datorită faptului că nu suntem forțați să răspundem la actualizări online. Din acest motiv, vom introduce o abordare care se folosește la multe soluții ce se bazează pe folosirea pădurilor de mulțimi disjuncte.

Practic, în loc să privim problema de la început la final, vom rezolva problema inversă, în care putem adăuga bile, ceea ce ne ajută să reducem problema la o aplicație standardă a pădurilor de mulțimi disjuncte, răspunsurile ajungând în cele din urmă să fie afișate în ordinea inversă în care le-am aflat.

[Soluție de 100 de puncte](#)

## 5 Problema Secvmax

[secvmax de pe infoarena](#)

”Fiona are o secvență de  $N$  numere naturale. Ea se întreaba din când în când pentru un anumit număr  $Q$  care este cea mai lungă subsecvență care are toate numerele mai mici sau egale cu  $Q$ . Ajutați-o pe Fiona să își rapunda la toate întrebările.”

### 5.1 Soluție

Aici putem folosi din nou prelucrarea numerelor în ordine crescătoare a numerelor din vector, iar atunci când adăugăm valorile în considerare, vom verifica fiecare vecin să vedem dacă putem uni valorile din cele două mulțimi, iar la fiecare pas răspunsul e cardinalul maxim al unei mulțimi, care e crescător pe măsură ce creștem valorile adăugate.

[Soluție de 100 de puncte](#)

## 6 Probleme suplimentare

### 6.1 Probleme de la olimpiade

1. [Chemical table - EJOI 2018](#)
2. [MexC ONI 2008](#)
3. [COCI 13-ladice](#)
4. [USACO MooTube](#)
5. [USACO Wormhole Sort](#)

### 6.2 Probleme de pe Codeforces/AtCoder

1. [Galleries - AGM 2020](#)
2. [DSU Step 1 - Codeforces EDU](#)
3. [DSU Step 2 - Codeforces EDU](#)

## 7 Bibliografie și lectură suplimentară

Am ordonat resursele suplimentare în ordinea dificultății înțelegerii și într-o ordine logică pentru a ușura obținerea de cunoștințe despre tehnicile, abordările și problemele discutate în acest curs.

1. [Păduri de mulțimi disjuncte - CS Academy](#)
2. [Algoritmul Union-Find - Algopedia](#)
3. [Link ce trebuie accesat pentru înscrierea la cursul despre DSU facut de ITMO Academy](#)
4. [Articol USACO Guide - DSU](#)
5. [Curs despre DSU - Codeforces](#) (este necesar un cont pentru a putea accesa acest curs, plus accesarea linkului de mai sus)
6. [Sack \(dsu on tree\) - Avansat](#)
7. [Smenul de manevrare a query-urilor offline cu DSU](#)

## 8 Cuvânt de încheiere

Dacă aveți sugestii, păreri, întrebări sau altceva ce ați vrea să împărtășiți cu privire la articol sau la alte aspecte legate de conținuturile prezentate, mă puteți contacta pe Discord ([stefdasca#9249](#)).

Mult succes mai departe.