Curs Lot Național Juniori
Abordarea problemelor ad-hoc

Ștefan Dăscălescu

1 Mai 2023

1 Introducere și motivație

De-a lungul drumului vostru de până acum spre obținerea rezultatelor care v-au adus aici, ați rezolvat foarte multe probleme de diverse tipuri și de diverse nivele de dificultate. Pentru o mare parte dintre ele, le putem clasifica după algoritmul sau tehnica folosită pentru rezolvarea problemei, ceea ce face exersarea acestor probleme și însușirea cunoștințelor necesare mai facilă, prin faptul că putem găsi similitudini cu alte probleme rezolvate anterior, precum și datorită faptului că există numeroase resurse utile pentru învățarea algoritmilor, metodelor de programare și structurilor de date necesare pentru implementarea acelor soluții.

În schimb, există un anumit tip de probleme pe care nu le putem încadra sub umbrela unui algoritm sau a unei tehnici de programare, iar aceste probleme par a fi complet diferite una față de cealaltă. Aceste probleme sunt cele ad-hoc și așa cum sugerează și numele, modul în care găsim soluțiile la aceste probleme depinde în totalitate de circumstanțele care ni se dau de la problemă la problemă. Cu toate acestea, deși problemele ad-hoc sunt în teorie complet originale, putem să ne folosim de câteva principii care ne vor ajuta pentru a facilita găsirea unor soluții pentru aceste probleme.

Scopul pe care acest curs îl are este acela de a prezenta viziunea mea asupra principiilor necesare pentru a găsi idei, abordări și soluții pentru problemele ad-hoc, precum și de a prezenta diverse probleme, aplicând aceste principii. De asemenea, un alt scop este acela de a unifica diversele resurse prezentate anterior pe diverse site-uri cu scopul de a extinde baza de informații în ceea ce privește acest tip de probleme prea puțin discutat și analizat într-o manieră mai sistematică.

2 O strategie de bază pentru abordarea problemelor ad-hoc

2.1 Tratarea cazurilor mici

Un prim pas spre a rezolva cu succes probleme ad-hoc este acela de a găsi soluții pentru exemple mici, lucru ce se poate face fie cu ajutorul hârtiei și a analizei manuale a acelor exemple, fie cu ajutorul unui program ce poate genera toate soluțiile posibile folosind un algoritm de tip brute force. De exemplu, puteți folosi un program brute force pentru a genera toate permutările unui șir de n numere, toate submulțimile mulțimii $1, 2, ..., n$ și asa mai departe.

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{

    int n
    cin >> n;

    vector<int> v;

    for(int i = 1; i <= n; i++)
        v.push_back(i);

```

```
    // generarea permutarilor multimii 1, 2, .., n

    do{
       // procesam permutarea
    }while(next_permutation(v.begin(), v.end()));

    // generarea submultimilor multimii 1, 2, .., n
    // avem 2^n submultimi, fiecare dintre ele putand fi reprezentata folosind
    // o masca pe biti

    for(int i = 0; i < (1<<n); i++)
    {
        vector<int> subset;
        for(int j = 0; j < n; j++)
        {
            if(i & (1<<j))
            {
                subset.push_back(j);
                // procesam acel numar ca parte a submultimii gasite
            }
        }
    }
}
```

2.2 Găsirea unor patternuri sau a unor relații între diverse valori din input

De multe ori, după ce generăm răspunsuri pentru valori mici folosind o metodă sau alta, putem să observăm diferite relații sau patternuri în funcție de răspunsurile pe care le obținem. De asemenea, ne putem folosi de anumite particularități pe care anumite valori din restricții le au, lucruri ce se întâlnesc des în problemele ad-hoc.

Un exemplu concret este problema asort de la barajul de juniori din 2016.

De exemplu, pentru N=8, aplicând de N ori regula "$\mathscr{A}$" se obțin șirurile:

Șir inițial	1	2	3	4	5	6	7	8
Prima aplicare a regulii "$\mathscr{A}$"	2	1	4	3	6	5	8	7
A doua aplicare a regulii "$\mathscr{A}$"	7	4	1	6	3	8	5	2
A treia aplicare a regulii "$\mathscr{A}$"	4	7	6	1	8	3	2	5
A patra aplicare a regulii "$\mathscr{A}$"	5	6	7	8	1	2	3	4
A cincea aplicare a regulii "$\mathscr{A}$"	6	5	8	7	2	1	4	3
A șasea aplicare a regulii "$\mathscr{A}$"	3	8	5	2	7	4	1	6
A șaptea aplicare a regulii "$\mathscr{A}$"	8	3	2	5	4	7	6	1
A opta aplicare a regulii "$\mathscr{A}$"	1	2	3	4	5	6	7	8

Se poate observa că prin aplicarea de un număr impar de ori a regulii, șirul rezultat are pe pozițiile impare numerele pare. Numere pare au apariție periodică în șirurile rezultate în ordinea $2, 4, 6, 8, 2, 4, 6, \ldots$. Analog, numerele impare apar pe poziții impare în ordinea $1, 3, 5, 7, 1, 3, 5, \ldots$.

Dacă se aplică regula de un număr par de ori, pe pozițiile pare apar numerele pare. Pe baza acestei observații, putem găsi modalitatea în care valorile din șirul inițial se vor deplasa de-a lungul algoritmului.

2.3 Folosiți-vă de diferite restricții speciale din enunț (dacă apar asemenea restricții)

.

De multe ori, pot apărea anumite restricții care sunt foarte particulare raportat la problema în sine. De obicei, o parte importantă a soluției în acest caz constă în observarea rolului din spatele acestor restricții.

Deși acesta nu este un sfat util doar pentru problemele ad-hoc, de multe ori analizarea atentă a acestor părți din problemă poate ușura rezolvarea problemei sau cel puțin a unor subprobleme care duc la soluția finală a problemelor.

2.4 Presupunerea unor soluții rezonabile

Acest pas este unul ce depinde de la problemă la problemă, dar de obicei e important să încercăm să reducem cât se poate aspectele variabile din problemă (reducem intervalul posibil al soluției, ignorăm informații neimportante, tratăm cazurile evidente etc.).

2.5 Convinge-te singur că soluția găsită anterior este corectă (sau incorectă)

Acest ultim pas este de departe cel mai dificil deoarece de multe ori se poate întâmpla una din cele două situații:

1. soluția să pară corectă dar să apară greșeli de idee / submisii care obtin 0 sau un scor foarte mic.

2. soluția să fie corectă dar să fie foarte greu de găsit cel putin o intuiție dacă nu o demonstrație completă a acesteia.

Indiferent de caz, este important să ne asigurăm că soluția obținută nu obține un răspuns greșit pe un caz evident, dar și să încercăm pe cât posibil să trimitem soluția când algoritmul și implementarea au sens.

Atenție!: Deși acesta este un sfat general, este foarte important să evitați să trimiteți soluția imediat ce găsiți un bug pe care l-ați rezolvat (cu excepția cazului când concursul se apropie de final și orice secundă contează) - pe lângă faptul că numărul de submisii este limitat, este de asemenea important să reușiți să rezolvați problema din mai puține submisii, deoarece din experiența personală, ajută în ceea ce privește încrederea pe care o căpătați pentru celelalte probleme în timpul concursului.

3 Un prim exemplu de problemă ad-hoc

CF 1541A

Se dă n și vrem să construim o permutare a mulțimii $1, 2, ..., n$ astfel încât $p[i]! = i$ și suma $abs(p[i] - i)$ să fie minim posibilă.

3.1 Soluție

Putem începe prin a găsi diverse răspunsuri pentru valori mici ale lui n.

$n = 2$ - $[2, 1]$

$n = 3$ - $[3, 1, 2]$

$n = 4$ - $[2, 1, 4, 3]$

$n = 5$ - $[3, 1, 2, 5, 4]$

Se poate observa că în cazul acestei probleme, este îndeajuns să găsim câteva soluții pentru valori mici ale lui n pentru a observa un pattern pentru valorile pare și impare ale lui n. Astfel, pentru $n > 3$, putem pleca de la soluția pentru $n-2$ la care adăugăm perechea $[n, n-1]$. Această soluție este una optimă deoarece fiecare element va fi situat la o distanță de 1 de poziția lui, cu excepția lui 3 când n e impar, care va fi situat la distanța 2.

Deși această problemă este una relativ facilă, procesul folosit pentru rezolvarea ei va fi regăsit în multe alte probleme de acest gen în viitor.

4 Problema Yinyang (OJI 2019, clasa a X-a)

Se dă o matrice A cu N linii şi M coloane, cu valori cuprinse între 1 şi $N * M$ inclusiv, nu neapărat distincte. O operaţie constă în selectarea a două linii sau două coloane consecutive şi interschimbarea acestora (swap).

O matrice yin-yang este o matrice în care $A[i][j] \geq A[i][j–1]$, pentru orice pereche (i, j) cu $1 \leq i \leq N$ şi $2 \leq j \leq M$ şi $A[i][j] \geq A[i–1][j]$, pentru orice pereche (i, j) cu $2 \leq i \leq N$ şi $1 \leq j \leq M$. Să se determine numărul minim de operaţii necesare pentru a transforma matricea dată într-o matrice yin-yang.

4.1 Observații inițiale

La o primă citire a enunțului, putem observa faptul că noi trebuie să sortăm matricea folosind un algoritm de interschimbare a două linii sau coloane consecutive. De aici, putem observa faptul că putem să verificăm rezultatele pe care le-am obține folosind un asemenea algoritm.

Cei doi algoritmi de sortare care interschimbă valori adiacente la fiecare pas sunt bubble sort și insertion sort, deci putem verifica cei doi algoritmi și vedem care ar oferi rezultate mai optime. Dar oare este necesar?

4.2 Generalizarea răspunsului pe o matrice

Știm deja de la lucrul cu vectori că numărul de inversiuni al unui vector este echivalent cu numărul de operații de interschimbare pe care îl face algoritmul bubble sort, ceea ce ne duce cu gândul la folosirea bubble sort pentru a calcula numărul minim de interschimbări.

De asemenea, deoarece atunci când interschimbăm două linii (sau coloane), valorile nu își schimbă ordinea pe coloană (sau pe linie), putem trata în mod independent sortarea liniilor și a coloanelor pentru a obține răspunsul.

După găsirea acestor observații, putem să sortăm liniile, apoi coloanele folosind bubble sort, iar tot ceea ce avem de făcut apoi este să verificăm condițiile din enunț.

Soluție de 100 de puncte

5 Problema Preparing Boxes (ABC 134)

Se dă un vector a de n valori care pot fi doar 1 si 0, și se cere să construim un vector b de n valori care pot fi doar 1 și 0 astfel încât pentru fiecare poziție i de la 1 la n, suma valorilor de pe pozițiile multiplu de i să aibă restul împărțirii la 2 egal cu $a[i]$.

5.1 Observații inițiale

Mai întâi, trebuie să ne gândim la informațiile pe care le putem afla mai ușor cu privire la valorile din vectorul b. De exemplu, $b[1] \mod 2 = (a[1] + a[2] + ... + a[n]) \mod 2$. Totuși, $b[n] \mod 2 = a[n] \mod 2$.

De asemenea, numărul total de valori pe care trebuie să le verificăm este $O(nlogn)$, o observație ce este destul de clară pentru oricine a implementat cel puțin o dată ciurul lui Eratostene.

Din aceste observații, putem deduce faptul că este optim să fixăm valorile din b de la dreapta la stânga, iar parcurgerea lor folosind brute force este îndeajuns de rapidă.

5.2 Soluția finală

Din cele două observații de mai sus, putem deduce un algoritm pentru implementarea soluției, care trece prin valori în ordine descrescătoare a poziției, iar pentru fiecare valoare, putem să determinăm dacă va fi 0 sau 1 în functie de restul împărțirii la 2 al sumei valorilor relevante.

Soluție care ia accepted

6 Problema Karte (COCI 2018, runda 5)

Pentru multe probleme de acest gen, o sortare a datelor este foarte utilă deoarece ne ajută să vedem lucrurile într-o perspectivă diferită. În cazul acestei probleme, e mai simplu să presupunem că valorile mai mici ne vor duce mai ușor la răspunsul cerut, așa că vom sorta datele mai întâi.

După ce grupăm datele, vom crea o nouă listă în care vom pune mai întâi cele mai mici $n-k$ valori în ordine descrescătoare, iar mai apoi punem celelalte valori, tot în ordine descrescătoare.

În cele din urmă, tot ce trebuie să mai facem este să verificăm dacă numărul de false ipoteze este exact k, conform enunțului.

Soluție care ia punctaj maxim

7 Problema Game (EJOI 2017)

Se dă un vector de n elemente și avem un joc între Alice și Bob. La început, luăm primele p valori. De-a lungul jocului, cei doi jucători vor alterna mutările, fiecare luând o valoare din vector, iar după acea mutare, câte o valoare nouă din vector este adăugată, atâta timp cât nu am adăugat încă toate valorile. Scopul este de a afla pentru k jocuri care se bazează pe vectorul inițial scorul final obținut (p se poate schimba de la joc la joc).

7.1 O soluție simplă

În primul rând este clar că fiecare jucător va vrea să ia elementul cu valoarea maximă la fiecare pas. Acest lucru ne duce la o soluție relativ ușor de găsit care implică un set (la fiecare pas luăm valoarea maximă și eventual adăugăm valoarea nouă în set). Complexitatea este $O(n * k * \log n)$, ceea ce obține 50 de puncte. Cum putem îmbunătăți această soluție?

7.2 Observații suplimentare

Deocamdată nu am folosit încă faptul că valorile din input nu sunt foarte mari și putem să folosim un vector de frecvență pentru a simula procesul. Totuși, acest lucru nu este îndeajuns pentru a obține punctajul maxim, deoarece nu am stabilit încă ce facem cu elementele pe care le adăugăm pe parcurs.

Să analizăm ce se întâmplă la un pas al jocului. După ce știm valoarea maximă inițială, avem două cazuri în ceea ce privește noua valoare adăugată.

1. Fie e mai mare decât maximul curent, iar atunci jucătorul aflat la mutare va lua acea valoare.
2. Altfel, acea valoare va fi adăugată în vectorul de frecvență, iar calculul va continua ca de obicei.

Astfel, complexitatea soluției va deveni $O(n * k)$.

Soluție care ia 100 de puncte

8 Alte sfaturi practice

1. Folosiți tot felul de idei și analizați de ce sunt incorecte
2. Aveți grijă la cazurile limită și cazurile mici.
3. Verificați de două ori soluțiile la cazurile pe care le rezolvați manual sau cu programul brute force
4. Nu vă gândiți prea mult la diferite abordări foarte sofisticate, deoarece de multe ori aceste probleme au soluții mai simple decât par.

9 Probleme suplimentare

9.1 Probleme de la olimpiade

1. yinyang (OJI 2019)
2. traseu4 (OJI 2019)
3. asort (baraj juniori 2016)
4. aranjare (infoarena)
5. palind (infoarena)
6. nane (infoarena)
7. secvmin (infoarena)
8. Quality (IOI 2010)
9. COCI 13-Organizator
10. COCI 18-karte
11. EJOI 17-game

9.2 Probleme de pe Codeforces/AtCoder

1. CF 1332B
2. ABC134 D
3. CF 1804B
4. CF 1815A
5. CF 1797C
6. CF 1815B
7. ARC158 A
8. ARC153 B
9. ARC145 B
10. CF 1407C
11. CF 1776F
12. ARC118 C
13. CF 1776G
14. ARC159 C
15. CF 1736D

10 Bibliografie și lectură suplimentară

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

1. Probleme constructive si interactive - HKOI
2. Tehnici pentru rezolvarea problemelor ad-hoc - Codeforces
3. Articolul SEPI din 2021 despre problemele ad-hoc
4. Alte sfaturi pentru problemele constructive - Codeforces
5. Probleme bune ad-hoc - Codeforces
6. Demonstrarea solutiilor la problemele de programare competitiva - Codeforces

11 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 la loturi și abia aștept să vă văd pe toți peste câteva zile.