OLIMPIADA REPUBLICANĂ LA INFORMATICĂ
EDIȚIA XXXIII

13 MARTIE 2022

Abordări pragmatice în soluționarea problemelor

Cifre Romane

Problema poate fi divizată în mai multe sub probleme. Înainte de a sorta numerele, ele ar trebui să fie transformate într-o reprezentare zecimală, deci numerele ce se citesc de la *stdin* trebuie să fie transformate din reprezentarea romană în zecimală. După aceasta trebuie de sortat tabloul de numere transformat, aici contează foarte mult algoritmul de sortare ales, într-o variantă de problemă se poate folosi algoritmul *quicksort*. În urma unei sortări, elementele ar trebui să fie afișate la ecran sub format roman, deci vom avea nevoie de un algoritm de transformare a reprezentării zecimale în roman.

Totuși, cel mai mult durează sortarea, din acest motiv contează foarte mult complexitatea algoritmului de sortare ales, dar nu putem ignora și algoritmii de transformare a reprezentării numerelor.

Fracții egiptene

Descompunerea fracțiilor ar putea avea loc și în mod iterativ pornind de la numărătorul 2 și tot așa mai departe, dar această metodă va fi mai lentă și nu mereu va fi precisă. În acest caz poate fi algoritmul propus mai jos. Vom nota numărătorul prin *nr*, iar numitorul prin *dr*.

Pas 1 – dacă `nr` sau `dr` este 0, atunci `stop`.

Pas 2 – dacă `nr >= dr`, atunci afișăm partea întreagă, adică *`nr div dr`*, o extragem din fracția noastră și revenim la pasul 1.

Pas 3 - dacă `dr mod nr` va fi egal cu 0, atunci următoarea fracție din șirul nostru va avea numitorul egal cu `dr div nr`, în caz contrar fracția va avea numitorul egal cu *`dr div nr + 1`*. Vom păstra acest rezultat într-o variabilă `aux` și afișăm `1/aux`.

Pas 4 – calculăm noul `nr` și `dr`.

```
nr = nr * aux – dr;
dr = dr * aux;
```

Revenim la pasul 1

Virusul

Problema data ar putea fi gândită într-o manieră puțin diferită. Înainte de a afla care porțiuni ar putea fi infectate, am putea să aflăm care porțiuni NU ar putea fi infectate. Din explicație am observat că doar regiunile înconjurate din toate 4 părți ar putea fi infectate, deci logic, regiunile aflate la marginile hărții nu ar putea fi infectate. Prin urmare ar putea fi pornit un algoritm de propagare a undei de la celulele neinfectate de virus (marcate cu 0) aflate la aceste margini.

Acest algoritm de propagare, va marca celulele ce nu sunt predispuse infectării, pentru aceasta se poate utiliza valoarea 2. După ce recursiv au fost marcate aceste celule cu 2, celulele rămase cu 0 ar fi predispuse la infectare, deoarece unda de marcare a celulelor „sănătoase" nu le-a atins. În urma acestor marcări, zonele cu 0 vor fi transformate în 1, iar zonele cu 2 vor fi transformate în 0. Astfel obținem harta finală.

Exemplu pe etape a algoritmului

0 0 1 1 1 1 0 1 0 1 1 0 1 0 1 1 1 0 1 1	2 2 1 1 1 1 2 1 0 1 1 2 1 0 1 1 1 2 1 1	2 2 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 2 1 1	0 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1

Știri false

Rezolvarea problemei se bazează pe construirea lanțurilor de raționamente pornind de la presupunerea că un autor este onest. Fie că primul autor este onest, deci toate afirmațiile lui sunt juste. Într-un vector sunt notate persoanele care sunt oneste și care mint din perspectiva acestuia. Se va folosi o notație, de exemplu, dacă persoana i minte în vector pe poziția i se va scrie 0, iar dacă este onest se va scrie 1. În condiția că primul autor este onest se analizează afirmațiile celui de-al doilea autor. Se vor căuta contradicțiile:

- Primul autor afirmă că un careva autor i minte, iar al doilea autor spune că autorul i este onest;
- Primul autor afirmă că autorul i este onest, iar al doilea autor afirmă că autorul i minte.

Dacă a fost depistată una din contradicțiile enumerate și având în vedere că am presupus că primul autor este onest, rezultă că al doilea autor minte și afirmațiile lui sunt ignorate. Dacă afirmația celui de-al doilea autor nu contrazice afirmațiile primului autor, atunci afirmația se adaugă la vector (dacă este o afirmație despre un nou autor despre care primul autor nu s-a expus). Se vor analiza afirmațiile tuturor celorlalți autori. La fiecare etapă se vor identifica persoanele care sunt oneste sau care mint. După ce au fost analizate toate afirmațiile sunt numărați autorii onești în condiția că primul autor este onest. Se va ține cont că autorii despre care nu se afirmă că mint sunt și ei onești.

Se vor construi astfel de raționamente pentru toți autorii, după care se va determina numărul maxim posibil de persoane oneste.

Exemplu

Intrare	Matrice intermediară	Vector	Ieșire
4 2 2 4 1 3 1 4 0 0 0 0 1 3	2 1 0 1 2 2 2 1 2 2 2 2 2 2 0 2	1 1 0 1	3

Explicații

Datele din fișierul de intrare sunt înscrise într-o matrice. Pe prima linie a matricei vor fi afirmațiile primului autor. Dacă primul autor afirmă că autorul 2 și 4 sunt onești deci se vor completa cu 1, iar autorul 3 minte deci poziția 3 se va completa cu 0. Dacă nu se cunoaște se va completa cu 2. Deci prima linie a matricei va conține: 2 1 0 1. Presupunem că primul autor este onest. La prima iterație vectorul care conține statutul fiecărui autor va fi: 1 1 0 1. Pe prima poziție este unul pentru că am presupus că primul autor este onest, restul pozițiilor sunt preluate din matrice. Se poate observa că celelalte afirmații ale autorilor 2, 3 și 4 nu contrazic afirmația noastră. Deci după analiza tuturor afirmațiilor determinăm numărul de autori onești posibile în condiția că primul autor este onest. În vector avem trei unități, deci numărul de autori onești este 3.

Comoara

Problema dată ar putea fi rezolvată printr-o căutare în adâncime. Dar nu vom folosi o căutare recursivă simplă. Abordarea simplă ar recalcula de foarte multe ori careva valori, dar noi vom mai folosi un tablou adițional care va salva calculele făcute anterior, ceea ce ne duce la ideea programării dinamice. Spre exemplu pentru matricea dată:

1 4 1 2	*Rezultatul ar fi 3 - calea: 1, 2 ,4*

Vom mai avea încă o matrice bidimensională care va stoca calea maximă ce poate fi parcursă dintr-un oarecare punct. La început inițializăm toată matricea adițională cu 0. Elementele care sunt notate cu 0 reflectă că calea maximă pentru acel element nu a fost calculată.

```
0 0

0 0
```

Putem porni o căutare în adâncime din fiecare celulă a tabloului și să găsim maximul. Nu contează punctul de start, dar vom începe din [0, 0] și recursiv vom căuta calea maximă salvând și rezultatele intermediare. Ne vom deplasa recursiv în direcțiile: jos, sus, dreapta, stânga. La parcurgere trebuie să ducem cont dacă nu

cumva ieșim după limitele tabloului și desigur dacă celula pe care o vizităm ar respecta constrângerea unui șir crescător.

Din [0, 0] în jos nu ne putem deplasa deoarece nu vom avea șir crescător, în sus nu ne putem deplasa deoarece ieșim după limite, în dreapta ne putem deplasa, prin urmare calea maximă va fi egală cu 1 + lungimea maximă a căii din dreapta. Pornim recursiv căutarea din [0, 1] unde aflăm că nu ne putem deplasa în oricare alt punct. Deci, șirul maxim din [0, 1] va avea lungimea 1, respectiv pentru [0, 0] avem 1 + 1, adică 2. Matricea adițională la această etapă arată astfel:

```
2 1

0 0
```

Iterăm prin următoarele celule, după [0, 0] ar urma [0, 1], dar valoarea pentru [0, 1] a fost deja calculată, așa că vom trece la următoarea celulă, adică [1,1]. Din [1,1] pornim căutarea recursivă care va rezulta în tabloul următor.

```
2 1

3 2
```

După ce toate elementele au fost vizitate sau sunt diferite de 0, găsim valoarea maximă din tablou, asta și va fi soluția.

Recursie

Definiție: Șirul s este un anti-palindrom, dacă citit invers obținem șirul $\underline{s}$.

Să presupunem că pentru $i \geq 1$, șirul s_i este un palindrom. Din moment ce $s_i = s_{i-1}\underline{s_{i-1}}$, putem observa că s_{i-1} este un anti-palindrom. Asemănător putem observa că dacă s_i este un anti-palindrom, atunci s_i este un palindrom. Astfel putem observa că dacă s_0 este un palindrom atunci s_i va fi un palindrom pentru orice *i* par, iar dacă s_0 este un anti-palindrom atunci s_i va fi un palindrom pentru orice i impar. Dacă s_0 nu este nici palindrom, nici anti-palindrom, atunci s_i nu va fi un palindrom niciodată.

MEX Suma

Pentru fiecare indice i cu $1 \leq i \leq N$ și pentru fiecare indice j cu $i \leq j \leq N$ avem următoarele proprietăți:

1. $mex(a_i, a_{i+1}, \ldots, a_j) \leq MAX + 1$
2. dacă $j < N$, atunci $mex(a_i, a_{i+1}, \ldots, a_j) \leq mex(a_i, a_{i+1}, \ldots, a_{j+1})$.

Fixând indicele i, pentru fiecare k vom afla cel mai mare indice p_k, astfel încât $mex(a_i, a_{i+1}, \ldots, a_{p_k}) \leq k$. Prin urmare vom avea

$$\sum_{j=i}^{N} mex(a_i, a_{i+1}, \ldots, a_j) = \sum_{k=1}^{MAX+1} (p_k - p_{k-1})k$$

Astfel problema se reduce la aflarea șirului p într-un timp optimal, pornind de la faptul că p_k este primul indice j, astfel încât $\forall t (1 \leq t \leq k)$, unde $t \in \{a_i, a_{i+1}, \ldots, a_j, a_{j+1}\}$ sau

$$p_k = \min_{j, t \in \{a_i, a_{i+1}, \ldots, a_j, a_{j+1}\}, \forall t (1 \leq t \leq k)} j$$

În concluzie, folosind ideea menționată mai sus, pornind de la $i = N$ la $i = 1$, putem calcula p în $O(N \cdot MAX)$ amortizat. Complexitatea finala este $O(N \cdot MAX)$.

Tic Tac Toe

Se menționează următoarele condiții pentru a considera tabelul valid:

1. Fiindcă 'X' este pus de primul jucător, vom avea mereu numărul de 'X' pe tabelă mai mare sau egal cu numărul de 'O',
2. Deoarece ambii jucători merg alternând, vom avea diferența dintre numărul de 'X' și numărul de 'O' egală cu maxim 1,
3. Un singur jucător poate câștiga (partida se termină atunci, când unul din jucători câștiga),
4. Daca nici un jucător nu a câștigat, atunci trebuie să avem tot tabelul plin de 'X' și 'O'.

Pentru a determina dacă primul jucător a câștigat trebuie să verificăm, dacă există trei de 'X' consecutiv orizontal, vertical sau diagonal. Similar putem verifica dacă al doilea jucător a câștigat. În condițiile când tabelul este valid și nici un jucător nu a câștigat, se va afișa "Remiza".

Vulcan

Soluția pentru problema Vulcan se poate baza pe algoritmul de propagare a unei (a vedea algoritmul Lee). O detaliere se poate regăsi la problema **Muzeu.**

Șir frumos

Putem observa că dacă șirul (a_i) este frumos, atunci pentru orice $i>1$, $a_i - a_{i-1}=b$, o valoare constantă. Astfel, dacă transformăm șirul inițial (a_i) în șirul de diferențe ale elementelor adiacente (d_i), unde $d_i = a_i - a_{i-1}$, problema se reduce la a afla numărul minim de operații pentru a obține un șir constant. În șirul de diferențe, o operație de tip (1) cu indicele p este echivalentă cu operația $d_{p+1} \leftarrow d_{p+1} - 1$, iar o operație de tip (2) cu indicele p este echivalentă cu operația $d_{p+1} \leftarrow d_{p+1} + 1$. Astfel, dacă valoarea finală în șir va fi b, numărul de operații necesare va fi:

$$|d_2 - b| + |d_3 - b| + \cdots + |d_n - b|$$

Putem încerca orice valoare posibilă pentru b, soluție $O(n|a_i|)$ ce ia 30 puncte. Dacă observăm că e necesar să testăm doar valorile $b = d_i$ pentru fiecare i, obținem o soluție $O(n^2)$, care ia 50 puncte. Observația finală e că valoarea b optimă va fi mediana șirului, care o putem calcula în $O(n)$, obținând soluția finală.

Bilete

Categoria de teste 1

Aplicăm metoda forței brute. În acest scop folosim 6 cicluri imbricate și analizăm cele b^6 numere posibile de bilete. Evident, complexitatea unui astfel de algoritm va fi $O(b^6)$.

Categoriile de teste 1 și *2*

Se observă că dacă sunt a_s moduri de a alege 3 numere cu suma s, atunci sunt a_s^2 bilete norocoase cu această sumă, deoarece cele doua jumătăți ale numerelor de pe bilete sunt independente una de alta. Astfel, este suficient să numărăm doar numărul de moduri pentru fiecare sumă posibilă (între 0 și $3b$-3), iar apoi să adunăm pătratele rezultatelor. Putem ușor găsi a_s iterând prin toate cele b^3 jumătăți de numere posibile de bilete. Complexitate unui astfel de algoritm este $O(b^3)$.

Categoriile de teste 1, 2 și *3*

Optimizăm modul în care se calculează a_s folosind programarea dinamică.

Fie $dp_{i,j}$ numărul de moduri de a forma suma j folosind i numere. Astfel $a_s = dp_{3,s}$. Când alegem al i-lea număr, putem alege orice număr între 0 și b-1. Astfel obținem relația

$$dp_{i,j} = dp_{i-1,j} + dp_{i-1,j-1} + dp_{i-1,j-2} + \ldots + dp_{i-1,j-b+1}.$$

Împreună cu relațiile $dp_{0,0} = 1$, $dp_{0,j} = 0$ pentru orice $j \neq 0$, $dp_{i,j} = 0$ pentru orice $j < 0$, putem ușor găsi toate valorile căutate.

Complexitate unui astfel de algoritm este $O(b^2)$.

Categoriile de teste 1, 2, 3* și *4

Observăm că în relația de recurență, avem o sumă consecutivă de b valori, pe care noi o calculăm la fiecare parcurgere. Putem exclude acest calcul prin implementarea tehnicii "suma prefixelor".

Fie $s_{i,j} = dp_{i,0} + dp_{i,1} + \ldots + dp_{i,j}$. Atunci $dp_{i,j} = s_{i-1,j} - s_{i-1,j-b}$, iar $s_{i,j} = s_{i,j-1} + dp_{i,j}$.

Astfel, putem calcula eficient toate aceste valori. Deoarece acum în formule apare scăderea, pentru a nu obține valori negative, trebuie să fim atenți atunci când lucrăm cu operatorul **mod**.

Complexitate acestui algoritm este $O(b)$.

Arbore

Pentru prima restricție de `N, A, B ≤ 100`, problema se rezolvă printr-o simplă căutare liniară a înălțimii minime și construirea unui arbore binar cu număr maxim de noduri cu înălțimea fixată. Complexitatea acestei soluții este `O(N * H)`, unde `H` este înălțimea minimă a arborelui.

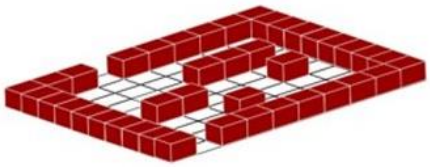
Pentru restricția `N ≤ 100 000`, algoritmul poate fi îmbunătățit prin printr-o căutare binară a înălțimii minime și construirea arborelui ca în soluția precedentă. Complexitatea este `O(N * log H)`.

Pentru restricția `N ≤ 1 000 000`, problema se rezolvă prin programarea dinamică `D[i]` = înălțimea minimă a unui arbore binar care are `i` noduri. Recurența este `D[i] = min(max(A + D[j], B + D[i − j − 1]), 0 ≤ j < i)`. Complexitatea este `O(N)`, iar pentru `A, B ≤ 10 000`, `D[i]` = numărul maxim de noduri pe care îl poate conține un arbore cu înălțime `i`. Recurența va fi `D[i] = 1 + D[i − A] + D[i − B]`. Această soluție are complexitate `O(H)`.

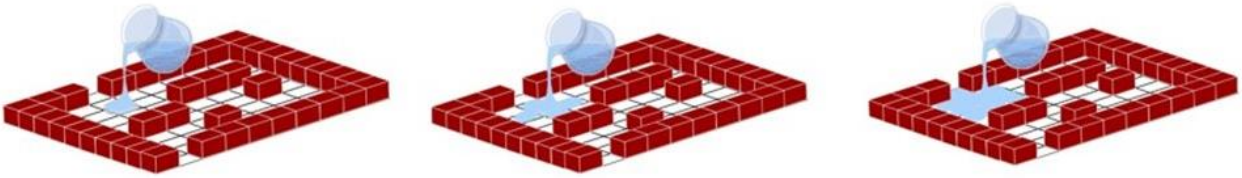
Soluția optima presupune o aboradare diferită: În primul rând, se aplică căutarea binară la căutare înălțimii arborelui, apoi se numără toate nodurile care au pe drumul până la rădăcină `x` muchii de lungime `A` și `y` muchii de lungime `B` (adâncimea acestor noduri fiind `x * A + y * B`). Răspunsul la această întrebare este `C(x + y, x) = C(x + y, y)`. Astfel, numărul maxim de noduri pe care îl poate conține un arbore cu înălțime `H` este `sum(C(x + y, x), x * A + y * B ≤ H)`. Putem presupune fără a pierde din generalitatea soluției că `A ≤ B`. Vom fixa `y` numărul de muchii de lungime `B`, iar numărul maxim de muchii de lungime `A` va fi `x = (H − y * B) / A`. Astfel, numărul nodurilor situate la adâncime cel mult `H` care au exact `y` muchii de lungime `B` pe drumul până la rădăcină va fi `sum(C(i + y, i), 0 ≤ i ≤ x) = C(x + y + 1, y + 1)`. Este suficient să observăm că pentru orice `n > 100` și `k > 5`, avem `C(n, k) > 1 000 000 000`, deci sumele descrise mai sus vor conține un număr foarte mic de termeni mai mici decât `N`.

Muzeu

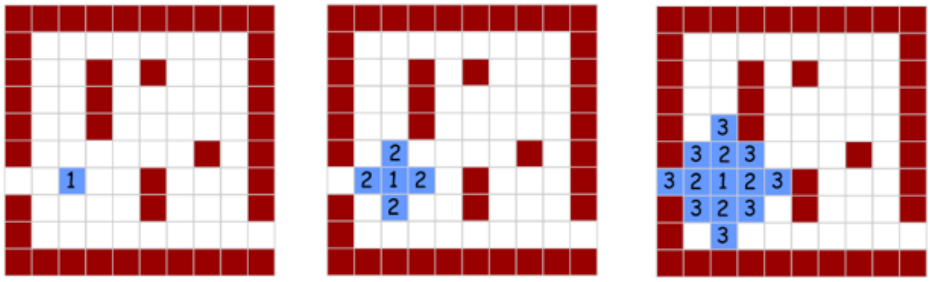
Să ne închipuim o suprafață plană, divizată în pătrate cu latura de o unitate. Pe unele pătrate sunt puse cuburi, și ele cu latura unitară:



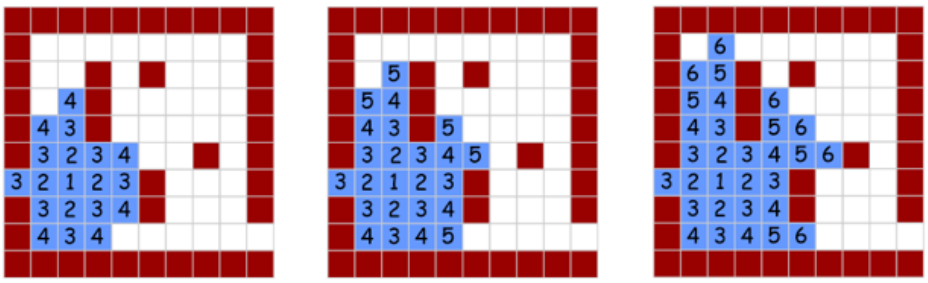
Pe unul din pătrate se toarnă vopsea. Pata de vopsea are proprietatea de a se extinde în direcțiile posibile (Nord, Sud, Est, West) cu viteza de un pătrat unitar pe secundă. Mai mult, datorită proprietăților sale speciale, vopseaua nu trece din pătrat în pătrat decât pe laturi vecine!



Pentru a modela mai simplu ce se întâmplă, vom trece la un model bidimensional:



Observăm, că după fiecare secundă "pata" se extinde, ocupând pătrățelele vecine față de cele deja ocupate, exact ca în situațiile reale (cu excepția formei petei). Dacă înscriem în fiecare pătrat valoarea timpului trecut până la colorarea lui, observăm următoarea legitate:

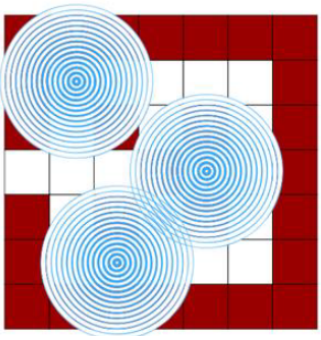


- ⇨ Pătratul din care începe extinderea petei de culoare are valoarea 1 – el e colorat primul.
- ⇨ Pătratele vecine cu el după o latură se colorează în secunda următoare (a doua) și capătă indicatorul 2
- ⇨ Pătratele necolorate care au o latură vecină cu un pătrat colorat în secunda `i`, vor fi colorate în secunda `i+1`!
- ⇨ Și da, obstacolele rămân necolorate – ele sunt mai înalte decât nivelul vopselei!

Continuând procesul de "colorare" vom ajunge într-un moment dat la situația când toată suprafața inițială va fi "colorată". Valoarea înscrisă în fiecare celulă va corespunde timpului **minimal** necesar pentru ca unda să acopere acea celulă. Respectiv, în cazul paznicilor care se mișcă conform condițiilor indicate în enunț valoarea înscrisă în fiecare celulă va corespunde timpului **minimal** necesar pentru ca paznicul să ajungă în sala respectivă - 1, (de ce?).

Valoarea maximală care va apare pe hartă corespunde sălii de expoziție, accesul paznicilor până la care durează cel mai mult! – prin urmare este tocmai sala de care are nevoie Micky.

Mai mulți paznici? – mai multe unde lansate concomitent!



Prin urmare

1. Identificăm sălile în care sunt paznici
2. Lansăm algoritmul de propagare a undei concomitent din toate sălile cu paznici
3. După finalizarea propagării undei, verificăm care este valoarea maximală în tabloul de propagare... și afișăm cu 1 mai puțin.

Structuri de date

Un tablou bidimensional pentru stocarea hărții: **`a[n][m]`**

Un tablou bidimensional pentru modelarea cozii de parcurgere **`b[3][ n*m]`**
(sau, mai comod, un tablou unidimensional de structuri care descriu zonele unitare acoperite de lavă)

```
struct cell{int x; int y; int d;} b[n*m];
```

(sau o listă alocată dinamic , un vector, etc)

Complexitate

Numărul de operații efectuate este proporțional:

1. **La citire cu dimensiunile hărții. Complexitate asimptotică O(n * m)**
2. **La etapa de propagare cu dimensiunile hărții. Complexitate asimptotică O(n * m)**
3. **Determinarea timpului constant. Complexitate asimptotică O(C)**

Prin urmare, complexitatea algoritmului O(n * m) m), ceea ce permite rularea soluției în condițiile impuse în enunț.

Librărie

Categoria de teste 1

Pentru `n=1`, primul jucător va alege cartea cu scorul mai mare la prima mutare, iar jucătorului al doilea îi va reveni cartea cu scorul minim.

Pentru `n=2`, fie cele 4 cărți `[a, b, c, d]`. Primul jucător poate alege fie `[a, b]`, fie `[c, d]`, după care al doilea jucător va alege cartea cu scorul maxim din cele două cărți rămase, iar în final primului jucător îi va mai reveni cartea cu scorul minim. Putem analiza ambele alegeri inițiale posibile ale primului jucător și afla care este cea ce îi oferă un scor mai mare.

Categoriile de teste 1, 2 și 3

Fie `dp[l][r]` perechea de scoruri obținute de cei doi jucători, daca ei vor juca cu cărțile de pe pozițiile de la `l` (inclusiv) până la `r` (exclusiv). Dacă definim `m=(l+r)/2`, adică `m` este mijlocul segmentului de la `l` la `r`, atunci putem observa că problema se reduce la cazul `n=2`, cu șirul `[dp[l][m].first, dp[l][m].second, dp[m][r].first, dp[m][r].second]`.

Astfel putem calcula toate valorile `dp[l][r]` după formula din cazul precedent, obținând o soluție `O(N2)`, care trece subtask-ul 2.

Totuși într-un joc adevărat nu e posibil să ajungem în orice interval `(l,r)`, în realitate putem ajunge la doar `O(N log N)` din ele: `(0,N)`, `(0,N/2)`, `(N/2,N)`, `(0,N/4)`, `(N/4, N/2)`, … și așa mai departe. Astfel calculând doar valorile `dp` pentru intervalele necesare obținem o soluție `O(N log N)`, care ia scor maxim.

Prefix

Știind că suma tuturor elementelor din șir este maxim `MAX = 2 000 000`, putem folosi algoritmul *Ciurul lui Erathostenes* pentru a precalcula toate numerele prime mai mici sau egale cu `MAX`. Această parte va avea `O(MAX log log MAX)`. După ce folosim *Ciurul lui Erathostenes*, putem verifica dacă un număr este prim în `O(1)`.

Pentru a afla șirul optimal în care suma fiecărui *prefix* este număr prim, se va folosi programarea dinamică pe mulțimi. Se menționează că se va afla șirul optimal pentru fiecare submulțime a șirului `a`.

Notând cu dp_S șirul optimal care conține indicii `S`. Se va inițializa $dp_{\{k\}} = a_k$, dacă a_k este prim, în caz contrar nu există soluție. Pentru orice mulțime `S` cu `|S| > 1`, dacă $\sum_{i \in S} a_i$ nu este prim, atunci nu avem soluție, în caz contrar, vom folosi următoarea formulă recursivă

$$dp_S = \min_{\text{i;dp}_{S\setminus\{i\}} \text{ are solutie}} (\text{dp}_{S\setminus\{i\}}(0), \text{dp}_{S\setminus\{i\}}(1), \dots, \text{dp}_{S\setminus\{i\}}(|S|-1), 1)$$

Programarea dinamica și operațiile pe mulțimi se pot efectua rapid și comod cu ajutorul operațiilor pe biți. Complexitatea finală este $O(MAX \log \log MAX + N \cdot 2^N)$.