Concursul Județean de Informatică „Future for Future”
Ediția a II-a, 2025
Clasa a X-a
Descrierea soluțiilor

1. Problema Top Secret

Propusă de: Rareș-Andrei Buzdugan, LTC

Subtask 1 (31 puncte)

Pentru prima cerință este suficient să determinăm numărul de apariții ale caracterului `'/'` într-o variabilă de tip contor, inițializată cu valoarea `1`.

Complexitate timp: $\mathcal{O}(|S|)$

Exemplu de implementare: https://kilonova.ro/pastes/p2nDwKpTlhyO

Subtask 2 (10 puncte)

Putem determina care cifră din tabel are codificarea egală cu șirul S folosind funcția `strcmp` din biblioteca `cstring` sau operatorul `==` pe `std::string`.

Subtask 3 (17 puncte)

Putem folosi un vector de cuvinte pentru codificarea caracterelor (de exemplu, pe primele 26 de poziții poate se memora codificarea literelor și pe următoarele 10 codificarea cifrelor).

Având în vedere faptul că în șirul S codificările sunt separate prin spațiu, putem folosi funcția `strtok` din biblioteca `cstring` sau operatorul `>>` pe `std::stringstream` sau putem verifica manual când caracterul curent este spațiu.

Pentru fiecare codificare găsită, determinăm poziția din vector la care se află aceasta și afișăm caracterul corespunzător poziției.

Subtask 4 (42 puncte) – Soluția finală

Singura diferență față de soluția precedentă este că S poate conține caracterul `'/'`, caz în care se afișează caracterul spațiu la fiecare apariție a lui.

Complexitate timp: $\mathcal{O}(|S|)$

Exemplu de implementare: https://kilonova.ro/pastes/wX0rgpREAJDD

2. Problema Restricționare

Propusă de: Rareș-Andrei Buzdugan, LTC

În cele ce urmează, notăm cu `restrictionare[]` șirul obținut după aplicarea operației de restricționare tuturor numerelor din șirul `A[]`.

Notații/definiții:

- $M = \max_{1 \le i \le N} A_i$,
- $\pi(n)$ = numărul de numere prime mai mici sau egale cu n,
- subsecvența $[l, r]$ = subsecvența formată din numerele corespunzătoare indicilor de la l la r,
- suma subsecvenței $[l, r] = \sigma_{[l,r]} = \sum_{i=l}^{r} \texttt{A[}i\texttt{]}$.

Subtask 1 (8 puncte)

Pentru prima cerință, vom considera toate elementele șirului `A[]` ca fiind pozitive, deoarece evident nu influențează metoda de rezolvare a problemei.

Dacă `B[i] == 0`, atunci `restrictionare[i] = 1`. În caz contrar, `restrictionare[i] = A[i]`. După calcularea șirului `restrictionare[]`, afișăm maximele șirurilor `A[]`, respectiv `restrictionare[]`.

Complexitate timp: $\mathcal{O}(N)$.

Subtask 2 (30 puncte)

Vom calcula `restrictionare[i]` folosind descompunerea în factori primi a lui `A[i]`. Fie f și e factorul prim curent, respectiv exponentul acestuia. Trebuie să calculăm $f^{\min(\texttt{B[i]},e)}$ și să înmulțim toate aceste valori. Calculele se pot face folosind o funcție de ridicare la putere, diferită de funcția `pow()` (deoarece este înceată), sau în timp ce determinăm valoarea lui e.

De asemenea, se poate observa că, pentru un număr natural nenul k, singurul factor prim al lui mai mare decât $\sqrt{k}$ poate fi doar însuși k. Astfel, este suficient să căutăm factorii primi în intervalul $[2, \sqrt{k}]$ și să tratăm separat cazul în care mai rămâne un singur factor prim.

Complexitate timp: $\mathcal{O}(N\sqrt{M})$.

Subtask 3 (10 puncte)

Pentru a descompune mai rapid în factori primi, vom determina toate numerele prime până la M, folosind **ciurul lui Eratostene**. De asemenea, se poate trata separat cazul particular în care `A[i]` este număr prim, crescând eficiența algoritmului.

Complexitate timp: $\mathcal{O}\left(M \log\log M + N \cdot \pi(\sqrt{M})\right)$.

Notă: Având în vedere că $\pi(n) = \mathcal{O}\left(\frac{n}{\log n}\right)$, complexitatea se poate rescrie ca $\mathcal{O}\left(M \log\log M + \frac{N\sqrt{M}}{\log M}\right)$.

Subtask 4 (3 puncte)

Având în vedere că $M < 2^{20}$, nu se poate restricționa niciun număr din șirul `A[]`. Deoarece este garantat că există soluție, răspunsul este 0.

Subtask 5 (22 puncte)

Vrem să restricționăm numerele din subsecvențe cu sume cât mai mari[1]. Numim subsecvență *interesantă*[2] a șirului `A[]` o subsecvență pentru care toate prefixele acesteia, excluzând însăși subsecvența, au suma cel mult S. Încercăm să extindem subsecvența curentă, păstrându-i proprietatea de subsecvență *interesantă*, până ajungem la una cu suma mai mare decât S. Următoarele observații ne vor scuti de analizarea unui număr prea mare de subsecvențe.

Presupunem că avem în vedere o subsecvență *interesantă* $[l, r]$. În primul rând, dacă suma ei este negativă, nu este convenabil să o extindem cu încă un element, deoarece $\sigma_{[l,r]} + \texttt{A[}r+1\texttt{]} < \texttt{A[}r+1\texttt{]}$. Așadar, putem schimba subsecvența candidată în $[r+1, r+1]$, pe care o extindem mai departe.

Acum, presupunând că subsecvența candidată $[l, r]$ are suma pozitivă, putem afirma că toate prefixele ei au suma pozitivă. Într-adevăr, dacă ar exista $l \leq r' < r$ cu $\sigma_{[l,r']} < 0$, atunci, potrivit observației de mai sus, subsecvența candidată ar trebui să înceapă cu termenul de indice $r' + 1 \neq l$, contradicție.

Mai mult, dacă subsecvența candidată are suma pozitivă și cel mult S, atunci orice subsecvență $[l', r']$ inclusă în ea are suma cel mult S. Știm că (i) orice prefix $[l, r']$ al subsecvenței $[l, r]$ are suma cel mult S (datorită proprietății de subsecvență *interesantă*) și că (ii) prefixul $[l, l'-1]$ al subsecvenței $[l, r']$ are suma pozitivă, deci:

$$\sigma_{[l',r']} = \sigma_{[l,r']} - \sigma_{[l,l'-1]} \overset{\text{(i)}}{\leq} S - \sigma_{[l,l'-1]} \overset{\text{(ii)}}{\leq} S.$$

În fine, dacă subsecvența candidată are suma mai mare decât S, putem folosi o strategie de tip *greedy* pentru scăderea sumei: aplicăm **acum** operația de restricționare numerelor din subsecvență în ordinea

[1] Teoretic, mai întâi trebuie să ne asigurăm că aplicăm mereu operația de restricționare în subsecvențe astfel încât să le scădem suma. Este limpede că restricționând un număr negativ obținem un număr mai mare. Atunci, restricționând toate numerele pozitive dintr-o subsecvență, obținem minimul la care poate ajunge suma ei. Așadar, având în vedere garanția existenței unei soluții, putem aplica operația doar pe numerele pozitive, scăzând mereu suma subsecvenței în care se află.

[2] Formal, o subsecvență $[l, r]$ este *interesantă* dacă $l < r$ și $\sigma_{[l,r']} \leq S$ pentru orice $l \leq r' < r$ sau $l = r$.

descrescătoare a diferenței `A[i] - restrictionare[i]`. Dacă am amâna operația de restricționare pentru o secvență $[l, r']$ cu $r' > r$ și $\sigma_{[l,r']} > S$, ar putea exista numere în subsecvența $[r+1, r']$ cu diferența `A[i] - restrictionare[i]` mai mare decât diferențele numerelor din $[l, r]$, astfel încât ele să fie suficiente pentru a scădea $\sigma_{[l,r']}$ sub S. Însă, $\sigma_{[l,r]}$ ar rămâne în continuare peste S.

Pentru a aplica această strategie, ținem minte șirul `diferente[]` conținând diferențele de tipul `A[i] - restrictionare[i]` pentru numerele încă nerestricționate din $[l, r]$. De fiecare dată când vrem să reducem $\sigma_{[l,r]}$, determinăm valoarea maximă din `diferente[]`, o scădem din $\sigma_{[l,r]}$ și o eliminăm din șir, repetând acest proces până când $\sigma_{[l,r]} < S$. De asemenea, dacă $\sigma_{[l,r]} < 0$, golim șirul `diferente[]` și adăugăm diferența `A[r + 1] - restrictionare[r + 1]`.

Complexitate timp: $\mathcal{O}(N^2)$ plus complexitatea cerinței 1 (aferentă restricționării fiecărui număr din șir).

Subtask 6 (27 puncte) – Soluția finală

Putem memora diferențele într-o structură de date (de ex. `priority_queue` din STL) care suportă operațiile de: adăugare în complexitate logaritmică, eliminarea valorii maxime în complexitate logaritmică și accesarea valorii maxime în complexitate constantă.

Complexitate timp: $\mathcal{O}(N \log N)$ plus complexitatea cerinței 1 (aferentă restricționării fiecărui număr din șir).

Exemplu de implementare: https://kilonova.ro/pastes/AefAVbm91aZa

Observații finale

Există soluții care folosesc anumite optimizări de finețe pentru a obține un punctaj foarte mare, de minimum 90 de puncte, fără a folosi ciurul lui Eratostene. De asemenea, există și soluții mai rapide decât cea prezentată aici, dar acestea depășesc programa concursului.

3. Problema Constangeles

Propusă de: Rareș-Ștefan Stanciu, ICHC și Rareș-Andrei Buzudgan, LTC

Subtask 1 (21 puncte)

Vom adăuga fiecare stâlp, în ordinea dată, până când există un drum de la $(1, 1)$ la (N, M) format doar din bucăți de drum iluminate, iar apoi vom afișa stâlpul la care ne-am oprit. Pentru a adăuga un stâlp, vom parcurge fiecare element din matrice și vom verifica, folosind condiția din enunț, dacă această bucată de drum este iluminată de stâlpul pe care îl adăugăm.

Pentru a verifica dacă există un drum de la $(1, 1)$ la (N, M) vom folosi un algoritm de tip **fill/Lee**, în care vom parcurge doar poziții iluminate de stâlpi și vom verifica dacă am ajuns la poziția (N, M), plecând din poziția $(1, 1)$.

Complexitate timp: $\mathcal{O}(N \cdot M \cdot K)$.

Exemplu de implementare: https://kilonova.ro/pastes/axqegrNXEQW1

Subtask 2 (17 puncte)

Se poate observa că forma geometrică a bucăților de drum iluminate de un stâlp de iluminat este un pătrat (plasat oblic) cu aria aproximativ $2R^2$. Astfel, putem aprinde doar bucățile de drum relevante pentru stâlpul de iluminat curent, folosindu-ne de formula dată în enunț.

În plus, dacă aprindem toți stâlpii până la stâlpul i și există un drum valid, atunci va exista un drum valid și pentru $j > i$. De aceea, putem căuta binar indicele cel mai mic al unui stâlp pentru care există un astfel de drum.

Complexitate timp: $\mathcal{O}((R^2 + N \cdot M) \log K)$

Exemplu de implementare: https://kilonova.ro/pastes/b3j9Z6bE9Sib

Subtask 3 (62 puncte)

Vrem să eficientizăm aprinderea tuturor stâlpilor până la un anumit stâlp i.

Un stâlp de rază R poate „pasa” iluminarea bucăților de drum altor patru stâlpi imaginari aflați pe poziții adiacente lui, cu raza $R - 1$.

Ar fi de preferat să adăugăm toți stâlpii într-o coadă pentru algoritmul lui Lee, pentru a putea opera o singură dată pe matrice, tratându-i în același timp pe toți. Totuși, există posibilitatea ca doi stâlpi, x și y, să se întâlnească pe o bucată de drum pe care unul dintre ei, de exemplu x, are raza mai mică decât cea a lui y. Atunci, formăm $R_{\max}$ cozi, astfel încât în coada i vom adăuga doar stâlpi de rază i.

Întrucât vrem să maximizăm raza stâlpului pe fiecare bucată de drum, vom parcurge mulțimea de cozi în ordine descrescătoare (de la $R_{\max}$ la 1). În momentul în care scoatem un stâlp de iluminat din coada i, trebuie să verificăm dacă poziția pe care se află nu este deja iluminată de un alt stâlp, caz în care acest stâlp trebuie ignorat. În caz contrar, raza acestui stâlp reprezintă raza maximă a unui stâlp de iluminat pe această poziție. Astfel, vom adăuga vecinii acestuia în coada $i - 1$. Repetăm acest proces pentru fiecare coadă, iar la final vom ilumina și pozițiile stâlpilor cu raza 0.

Complexitate timp: $\mathcal{O}(N \cdot M \cdot \log K)$.

Exemplu de implementare: https://kilonova.ro/pastes/ISuQOTiaKdue