OLIMPIADA NAȚIONALĂ DE INFORMATICĂ, ETAPA NAȚIONALĂ
CLASELE 11-12
DESCRIEREA SOLUȚIILOR

COMISIA ȘTIINȚIFICĂ

Problema 1: EXPERIMENTE2

Propusă de: stud. Argherie Ovidiu-Alexandru, Delft University of Technology

Subtask 1: Pentru $n \leq 9$, numărul total de secvențe posibile de operații de combinare[1] este suficient de mic încât se poate utiliza o căutare exhaustivă, pentru a explora toate înlocuirile posibile. Dacă există cel puțin o secvență de operații care conduce la o configurație omogenă, soluția returnează răspunsul afirmativ. La fiecare pas, dacă șirul curent are k elemente, se pot efectua $k-1$ operații de combinare, deci numărul total de secvențe posibile este $(k-1) \times (k-2) \times \cdots \times 1 = (k-1)!$. Cum operațiile intermediare implică și o verificare a egalității tuturor elementelor, costul total al acestei soluții o să fie $\mathcal{O}(n) \times \mathcal{O}((n-1)!)$, deci complexitatea de timp o să fie $\mathcal{O}(Q \times n!)$. Pentru cei interesați, numărul de moduri de a efectua operațiile de combinare este direct legat de numerele lui Schröder-Hiparh (cunoscute și ca numerele supra-Catalan), care conturează numărul de moduri de a paranteza, într-un mod mai general, o secvență de n elemente.

Teorema 1: Fie Γ un șir asupra căruia aplicăm operații de combinare. Se arată că, dacă Γ se poate reduce la $m \geq 2$ elemente egale, atunci Γ poate fi redus la două sau trei elemente egale, în funcție de paritatea lui m.

Demonstrație prin inducție matematică. Fie propoziția $P(m)$ adevărată, dacă și numai dacă Teorema 1 este adevarată, oricare ar fi valoarea comună $x \in \mathbb{N}$ a șirului Γ, redus la m elemente egale.

Cazurile de bază $(m = 2 \vee m = 3)$ sunt trivial adevărate, deoarece am obținut o configurație cu două sau trei elemente egale.

Presupunem că, pentru un număr natural $k > 3$ oarecare, propoziția $P(k)$ este adevărată.

De demonstrat că $P(k) \implies P(k+2)$: Să presupunem că există o secvență de operații care reduce Γ la $(k+2)$ elemente egale. Putem alege oricare două elemente adiacente din Γ și, cum disjuncția exclusiva a acestora va fi 0, vom introduce în mulțime valoarea 0. Apoi, combinând-o cu un alt x adiacent, obținem valoarea x, deoarece $0 \oplus x = x \oplus 0 = x$. Astfel, în două operații succesive, eliminăm două elemente și nu schimbăm valoarea comună a șirului, deci dimensiunea acestuia scade de la $(k+2)$ la k elemente. Din ipoteza de inducție, $P(k)$, știm deja că, dacă există un mod de a obține k elemente egale, atunci putem reduce acel șir la două sau trei elemente egale, ceea ce inseamnă ca $P(k+2)$ este, de asemenea, adevarată. Prin urmare, am arătat că afirmația $P(k) \implies P(k+2)$ este adevărată.

Cum k a fost ales arbitrar și numerele pare și impare partiționează mulțimea numerelor naturale, putem spune că, prin principiul inducției matematice, propoziția $P(m)$ este adevărată, oricare ar fi $m \geq 2$. □

Teorema 2: Fie $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$ un șir asupra căruia aplicăm operații de combinare. Se arată că, pentru oricare $1 \leq i < n$, o operație de combinare a elementelor γ_i și γ_{i+1} nu va schimba disjuncția exclusivă totală $T = \gamma_1 \oplus \gamma_2 \cdots \oplus \gamma_n$ a șirului inițial.

Demonstrație. Fie șirul $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$ și disjuncția exclusivă totală, inițială $T_{init} = \gamma_1 \oplus \gamma_2 \cdots \oplus \gamma_n$ a acestuia. Pentru oricare $1 \leq i < n$, înlocuim două elemente adiacente

[1] Definim *o operație de combinare* o înlocuire a două elemente adiacente cu disjuncția exclusivă a acestora.

γ_i și γ_{i+1} cu un singur element x, unde $x = \gamma_i \oplus \gamma_{i+1}$. Astfel, cum disjuncția exclusivă este o operație asociativă și comutativă, o să avem că $T_{nou} = T_{init} \oplus \gamma_i \oplus \gamma_{i+1} \oplus x$, adică $T_{nou} = T_{init} \oplus \gamma_i \oplus \gamma_{i+1} \oplus \gamma_i \oplus \gamma_{i+1} = T_{init} \oplus \gamma_i \oplus \gamma_i \oplus \gamma_{i+1} \oplus \gamma_{i+1} = T_{init}$. □

Observația problemei: Având în vedere cele două teoreme demonstrate anterior, problema se reduce la două cazuri fundamentale:

(1) *Cazul 1.* Conform *Teoremei 1*, dacă șirul Γ poate fi redus la m elemente nenule și egale, iar m este par, atunci șirul poate fi redus la două elemente egale. Cum $x \oplus x = 0$, rezultă, conform *Teoremei 2*, că disjuncția exclusivă totală, T, a șirului inițial trebuie să fie 0. Prin urmare, indiferent de tipologia șirului, acest caz constă numai în verificarea directă dacă $T = 0$ în $\Theta(n)$ pași, și **este parte a fiecărui subtask ca o soluție reciproc exclusivă**.

(2) *Cazul 2.* Analog, dacă Γ poate fi redus la m elemente nenule și egale, iar m este impar, atunci Γ se reduce la trei elemente egale. Din moment ce $x \oplus x \oplus x = x$, înseamnă că disjuncția exclusivă totală, T, a șirului inițial trebuie să fie egală, conform *Teoremei 2*, cu valoarea comună a șirului redus la trei elemente egale, adică $T = x$.

De menționat este că, pentru cazul în care șirul Γ se poate transforma astfel încât să aibă m elemente egale cu 0, Γ se poate reduce la oricare dintre cazurile menționate anterior, deoarece, trivial, $0 \oplus 0 = 0$. Pentru a explica următoarele subtask-uri, o să definim funcția $xor : \bigcup_{n=2}^{\infty} \left(\mathbb{N}^n \times \{1, \ldots, n\} \times \{1, \ldots, n\}\right) \to \mathbb{N}$, astfel încât, pentru orice $n \geq 2$, dacă $\Gamma \in \mathbb{N}^n$,

$$xor(\Gamma, i, j) = \begin{cases} \bigoplus_{k=i}^{j} \gamma_k, & \text{dacă } i \leq j, \\ \left(\bigoplus_{k=i}^{n} \gamma_k\right) \oplus \left(\bigoplus_{k=1}^{j} \gamma_k\right), & \text{dacă } i > j. \end{cases}$$

Subtask 2: Vom încerca toate combinațiile posibile de împărțire a șirului în trei subsecvențe consecutive și vom calcula disjuncția exclusivă a fiecărui segment. Pentru a obține un șir de forma $\Gamma = \{\gamma, \gamma, \gamma\}$, oricare ar fi $\gamma \in \mathbb{N}$, vom căuta două poziții i și j $(1 \leq i < j < n)$, astfel încât $xor(\Gamma, 1, i) = xor(\Gamma, i+1, j) = xor(\Gamma, j+1, n)$.
Complexitatea de timp este $\mathcal{O}(Q \times n^3)$.

Subtask 3: Pentru a optimiza soluția anterioară, putem să calculăm, pentru fiecare subsecvență, fiecare disjuncție exclusivă în timp constant, folosind *xor-uri parțiale pe prefix*:

$$prefix[i] = \begin{cases} 0, & i = 0 \\ prefix[i-1] \oplus \gamma_i, & 1 \leq i \leq n \end{cases}$$

Astfel, pentru orice subsecvență definită de pozițiile i și j, putem calcula disjuncția exclusivă a acesteia în $\Theta(1)$, folosind $xor(\Gamma, i, j) = prefix[j] \oplus prefix[i-1]$.
Complexitatea de timp este $\mathcal{O}(Q \times n^2)$.

Subtask 4: Pentru a obține o soluție mai optimă decât cea anterioară, o să ne folosim de cazul 2 al observației făcute anterior, care arată că orice secvență de operații de combinare va menține disjuncția exclusivă totală, T, a șirului inițial. Prin urmare, căutăm două poziții i și j $(1 \leq i < j < n)$, astfel încât $xor(\Gamma, 1, i) = xor(\Gamma, i+1, j) = xor(\Gamma, j+1, n) = T$. O soluție posibilă este să calculăm, fără a mai folosi xor-urile parțiale pe prefix, valoarea disjuncției exclusive curente la fiecare pas în parcurgerea șirului Γ. În momentul în care aceasta devine egală cu T, resetăm valoarea și incrementăm numărul de subsecvențe găsite. În final, dacă acest număr este mai mare sau egal cu 3, atunci soluția returnează răspunsul afirmativ.
Complexitatea de timp este $\mathcal{O}(Q \times n)$.

Subtask 5: Similar ca primul subtask, numărul total de secvențe posibile de operații de combinare este suficient de mic încât se pot explora toate înlocuirile posibile. Cum circularitatea șirului implică că se poate lua în considerare, la fiecare pas de combinare, și perechea formată de primul și ultimul element, o să avem k operații posibile de combinare, pentru un șir cu k elemente. Astfel, se obține un cost total de $\mathcal{O}(n \times n!)$, de unde rezultă complexitatea de timp $\mathcal{O}(Q \times (n+1)!)$.

Observație: Pentru a liniariza un șir circular de n elemente, se poate dubla șirul, prin concatenarea cu el însuși, și utiliza o tehnică de tip *sliding window*, în care orice subsecvență circulară de lungime cel mult n corespunde unei subsecvențe liniare de aceeași lungime în șirul dublat. Imaginați-vă că plasăm cele n elemente pe circumferința unui cerc. Dacă există o împărțire a cercului în trei subsecvențe consecutive (arce disjuncte, a căror uniune formează circumferința cercului), fiecare având disjuncția exclusivă egală cu T, atunci putem roti succesiv elementele aflate pe cerc, astfel încât primul element al șirului liniar să coincidă cu poziția unde se află o împărțire optimă. Pentru a demonstra formal această observație, putem crea o bijecție între toate șirurile posibile și fiecare sliding window, astfel:

Demonstrație. Fie $\Gamma = (\gamma_0, \gamma_1, \ldots, \gamma_{n-1})$ un șir circular de n numere naturale. Construim un șir liniar de lungime $2n$, notat cu $\Gamma' = (\gamma_0, \ldots, \gamma_{n-1}, \gamma_0, \ldots, \gamma_{n-1})$ și mulțimea tuturor rotațiilor liniare ale lui Γ, notată cu $S_n = \left\{ \left(\gamma_i, \gamma_{(i+1) \bmod n}, \ldots, \gamma_{(i+n-1) \bmod n} \right) \mid 0 \leq i < n \right\}$. Pentru fiecare $i \in \{0, 1, \ldots, n-1\}$, fie sliding window-ul $w(i) = (\Gamma'[i], \Gamma'[i+1], \ldots, \Gamma'[i+n-1])$. Astfel, definim funcția $s : \{0, 1, \ldots, n-1\} \to S_n : s(i) = w(i)$, unde $s(i) \in S_n$ este șirul liniar din Γ care începe pe poziția i.

Pentru a arăta că s este injectivă, presupunem că, pentru doi indici $i, j \in \{0, \ldots, n-1\}$, avem $s(i) = s(j)$. Din definiția lui Γ', elementele corespund rotațiilor $(\gamma_i, \gamma_{(i+1) \bmod n}, \ldots, \gamma_{(i+n-1) \bmod n})$ și $(\gamma_j, \gamma_{(j+1) \bmod n}, \ldots, \gamma_{(j+n-1) \bmod n})$. Cum șirurile sunt identice, înseamnă că poziția de început a primului șir trebuie să fie identică cu cea a șirului al doilea, deci $i = j$.

Pentru a arăta că s este surjectivă, luăm un șir oarecare din S_n. Prin definiție, există un k cu $0 \leq k < n$, astfel încât acel șir se scrie $(\gamma_k, \gamma_{(k+1) \bmod n}, \ldots, \gamma_{(k+n-1) \bmod n})$, ceea ce corespunde exact subsecvenței $s(k) = (\Gamma'[k], \Gamma'[k+1], \ldots, \Gamma'[k+n-1])$. Așadar, orice element al lui S_n apare ca imagine sub s a unui indice din $\{0, 1, \ldots, n-1\}$, deci s este surjectivă.

Din injectivitate și surjectivitate, rezultă că s este o funcție bijectivă între $\{0, 1, \ldots, n-1\}$ și S_n, adică fiecare rotație a șirului circular Γ corespunde, în mod unic, uneia dintre sliding window-urile de lungime n din șirul dublat Γ'. □

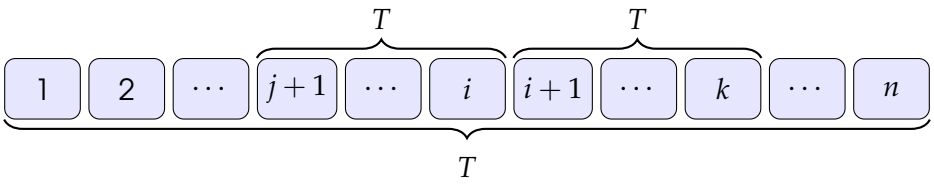
Subtask-urile 6, 7, 8: Putem combina observația anterioară cu ideile din subtask-urile 2, 3 sau 4, fiecare aplicat pe un vector dublat Γ' și verificat pentru fiecare sliding window al acestuia. Costul total al soluțiilor o să crească cu un factor suplimentar de $\mathcal{O}(n)$, deci cea mai optimă implementare o să aibă complexitatea de timp $\mathcal{O}(Q \times n^2)$.

Subtask 9a: Dacă șirul circular se poate împărți în trei subsecvențe consecutive, într-un sliding window k, înseamnă că există doi indici i și j ($st = k \leq i < j < dr = k + n - 1$), astfel încât $xor(\Gamma', st, i) = xor(\Gamma', i+1, j) = xor(\Gamma', j+1, dr) = T$. Astfel, putem să precalculăm xor-uri parțiale pe prefix și, pentru fiecare prefix în parte, îi adăugăm poziția la care a fost găsit într-un dicționar de mulțimi (spre exemplu, `unordered_map<int,vector<int>>`). Prin urmare, cu ajutorul a două căutari binare, putem găsi, în timp logaritmic, cei doi indici, deoarece, pentru fiecare sliding window, știm că $prefix[i] \oplus prefix[st-1] = T$, deci $prefix[i] = prefix[st-1] \oplus T$, și $prefix[j] \oplus prefix[i] = T$, deci, prin înlocuire, $prefix[j] = prefix[st-1]$. Complexitatea de timp este, în medie, $\mathcal{O}(Q \times n \; log \; n)$.

Subtask 9b: *Prima soluție.* Putem optimiza soluția anterioară, ținând în dicționar numai prefixele care se află în sliding window-ul curent. Mai concret, ne interesează să găsim cel mai din stânga indice i și cel mai din dreapta indice j, astfel încât sliding window-ul curent să poată fi împărțit în trei subsecvențe consecutive, a căror disjuncție exclusivă este T. Astfel,

putem folosi un dicționar de cozi cu dublu acces (deque), în care ultima apariție a unui prefix o să fie, dacă există, la finalul cozii, iar prima apariție, dacă există, la începutul cozii.

A doua soluție. Fără a mai folosi tehnica *sliding window*, putem să folosim o abordare asemănătoare cu aceea a problemei clasice *2-sum*. Mai exact, pe măsură ce parcurgem șirul de prefix de la stânga la dreapta, pentru fiecare poziție i ($1 \leq i < n$), calculăm complementul $prefix[i] \oplus T$. Dacă, pentru un anumit i, există —într-un dicționar care ține poziția ultimei apariții a unui prefix până la i— un indice j mai mic decât i, pentru care $prefix[j] = prefix[i] \oplus T$, atunci subsecvența de la poziția $j+1$ la i are disjuncția exclusivă egală cu T. Odată fixat acest segment, încercăm să găsim, în dreapta lui i, un alt indice k, astfel încât $xor(\Gamma, i+1, k) = T$. Dacă două subsecvențe consecutive au disjuncția exclusivă egală cu T, atunci, datorită proprietăților asociative și comutative ale operației **xor**, și segmentul rămas, considerat în mod circular, va avea disjuncția exclusivă egală cu cea inițială, adică $xor(\Gamma, k+1, j) = T$. Complexitatea de timp este, în medie, $\mathcal{O}(Q \times n)$.



Problema 2: FESTIVAL

Propusă de: stud. Moca Andrei-Cătălin, Babes-Bolyai University

Recapitularea problemei. Avem un set de N festivaluri, fiecare descris prin tripletul (t_i, x_i, s_i), unde:

- t_i este timpul la care are loc festivalul,
- x_i este locația festivalul,
- s_i reprezintă satisfacția obținută dacă festivalul este selectat.

Andrei dorește să participe la o parte dintre ele astfel încât:

(1) Timpul să fie crescător, adică dacă festivalurile selectate sunt $i_1, i_2, \ldots, i_k$, atunci $t_{i_1} \leq t_{i_2} \leq \cdots \leq t_{i_k}$.
(2) Tranziția de la festivalul i la festivalul j este permisă dacă

$$|x_j - x_i| \leq D \quad \text{și} \quad |x_j - x_i| \leq t_j - t_i.$$

Subtask 1: Pentru un N mic ($N \leq 20$) putem explora toate submulțimile festivalurilor (numărul total de submulțimi este 2^N). Deoarece evenimentele trebuie să fie alese în ordine strict crescătoare după timp, vom sorta evenimentele în prealabil după t_i. Pentru fiecare submulțime (reprezentată de un bitmask) verificăm dacă secvența selectată (în ordinea naturală a indexării după sortare) respectă condițiile de tranziție:

$$|x_j - x_i| \leq D \quad \text{și} \quad |x_j - x_i| \leq t_j - t_i,$$

Pentru un bitmask valid calculăm suma de satisfacție a evenimentelor selectate și actualizăm soluția maximă.

Complexitatea soluției este de $\mathcal{O}(2^N \cdot N)$, care este acceptabil pentru $N \leq 20$.

Subtask 2: Pentru $N \leq 2000$, putem folosi o abordare clasică de programare dinamică:

- Sortăm evenimentele după t_i.
- Inițializăm $dp[j] = s_j$ pentru fiecare $1 \leq j \leq N$.
- Pentru fiecare eveniment j, pentru fiecare eveniment i cu $i < j$, verificăm dacă tranziția $i \to j$ este permisă, adică:

$$|x_j - x_i| \leq D \quad \text{și} \quad |x_j - x_i| \leq t_j - t_i.$$

- Dacă condiția este verificată, actualizăm:

$$dp[j] = \max(dp[j], dp[i] + s_j).$$

Numărul de operații este de ordinul $O(N^2)$.

Subtask 3: Când $D = 0$, condiția

$$|x_j - x_i| \leq D$$

impune că $|x_j - x_i| = 0$, deci $x_j = x_i$. În această situație, tranziția este permisă doar între evenimente care au aceeași locație.

- Pentru fiecare locație x menținem valoarea maximă $sum[x]$, care reprezintă suma maximă de satisfacție obținută până în acel moment la locația respectivă.
- Pentru fiecare eveniment $e = (t, x, s)$, soluția optimă care se termină la e este:

$$dp = s + sum[x].$$

- Actualizăm apoi $sum[x] \leftarrow \max(sum[x], dp)$.

Subtask 4: Inițial, condițiile pentru a putea trece de la festivalul i la festivalul j (cu $t_j > t_i$) sunt:

$$|x_j - x_i| \leq D \quad \text{și} \quad |x_j - x_i| \leq t_j - t_i.$$

Având $D = 10^9$, restricția $|x_j - x_i| \leq D$ dispare (deoarece diferențele $|x_j - x_i|$ vor fi mai mici sau egale cu 10^9). Astfel, ne concentrăm asupra condiției

$$|x_j - x_i| \leq t_j - t_i.$$

Observăm că inegalitatea

$$|x_j - x_i| \leq t_j - t_i$$

este echivalentă cu:

$$x_j - x_i \leq t_j - t_i \quad \text{și} \quad -(x_j - x_i) \leq t_j - t_i.$$

Rescriem aceste inegalități astfel:

(1) Prima inegalitate:

$$x_j - x_i \leq t_j - t_i \quad \Longrightarrow \quad t_i - x_i \leq t_j - x_j.$$

Definim:

$$L_i = t_i - x_i.$$

Astfel, condiția devine:

$$L_i \leq L_j.$$

(2) A doua inegalitate:

$$-(x_j - x_i) \leq t_j - t_i \quad \Longrightarrow \quad x_i - x_j \leq t_j - t_i.$$

Aceasta se poate rescrie ca:

$$t_i + x_i \leq t_j + x_j.$$

Definim:

$$R_i = t_i + x_i.$$

Astfel, se obține:

$$R_i \leq R_j.$$

Prin urmare, condiția esențială pentru a putea merge de la festivalul i la festivalul j se reduce la:

$$L_i \leq L_j \quad \text{și} \quad R_i \leq R_j.$$

Observația esențială este următoarea: dacă avem

$$L_i \leq L_j \quad \text{și} \quad R_i \leq R_j,$$

atunci, adunând cele două inegalități, obținem:

$$(t_i - x_i) + (t_i + x_i) \leq (t_j - x_j) + (t_j + x_j),$$

ceea ce se simplifică la:

$$2t_i \leq 2t_j \quad \Longrightarrow \quad t_i \leq t_j.$$

Astfel, condiția de timp $t_i \leq t_j$ este implicit asigurată de condițiile $L_i \leq L_j$ și $R_i \leq R_j$. În concluzie, dacă festivalurile sunt aranjate astfel încât vectorii (L_i) și (R_i) să fie monoton crescători, condiția inițială $|x_j - x_i| \leq t_j - t_i$ este garantată.

Având în vedere că restricția pe distanță este eliminată (practic), soluția se bazează pe următoarele idei:

(1) **Sortarea evenimentelor:** Sortăm festivalurile în ordine crescătoare după valoarea lui $L = t - x$. Această sortare asigură că, pentru orice tranziție de la evenimentul i la evenimentul j cu $i < j$, avem automat $L_i \leq L_j$.

(2) **Gestionarea condiției pe R:** Pentru a impune și condiția $R_i \leq R_j$, folosim o structură de date eficientă, cum ar fi un *arbore indexat binar* (AIB). În AIB, evenimentele sunt indexate în funcție de valoarea lui R, iar pentru fiecare festival j, putem interoga rapid maximul valorii $dp[i]$ printre evenimentele anterioare (cu $i < j$) care au $R_i \leq R_j$.

(3) **Recurența DP:** Definim $dp[j]$ ca fiind satisfacția cumulată maximă pentru un lanț ce se termină la festivalul j. Astfel, recurenta este:

$$dp[j] = s_j + \max_{\substack{i<j \\ R_i \leq R_j}} \{dp[i]\}.$$

(4) **Actualizarea AIB-ului:** După ce calculăm $dp[j]$, actualizăm AIB-ul la poziția corespunzătoare valorii R_j cu noua valoare $dp[j]$, astfel încât informația să poată fi folosită pentru evenimentele viitoare.

- Sortarea evenimentelor după L se face în timp $\mathcal{O}(N \log N)$.
- Procesul de *normalizare* (compresie de coordonate) asupra valorilor lui R are, de asemenea, complexitate $\mathcal{O}(N \log N)$.
- Fiecare interogare și actualizare în AIB se execută în timp $\mathcal{O}(\log N)$, iar acestea sunt efectuate pentru fiecare eveniment, ceea ce duce la complexitatea totală de $\mathcal{O}(N \log N)$.

Astfel, soluția finală se execută în timp $\mathcal{O}(N \log N)$.

Subtask 5: Pe lângă condițiile de ordin pentru L și R, trebuie impusă și restricția pe locație:

$$|x_j - x_i| \leq D.$$

Această condiție se implementează prin filtrarea evenimentelor candidate pentru tranziție, astfel încât, pentru un eveniment j, se consideră numai evenimentele i pentru care x_i se află în intervalul $[x_j - D,\ x_j + D]$.

Valorile lui x, L și R pot fi foarte mari (până la 10^9), ceea ce face dificilă utilizarea lor directă într-o structură de date eficientă (precum un AIB sau un arbore de intervale). Pentru a depăși acest obstacol, se efectuează o compresie de coordonate (normalizare). Această etapă se realizează în timp $\mathcal{O}(N \log N)$ și este esențială pentru ca ulterior să se poată construi și opera structurile de date în timp eficient.

Pentru a răspunde rapid la interogări asupra lanțurilor posibile, se folosește o structură de date bidimensională (de exemplu, un AIB 2D sau un arbore de intervale în care la fiecare nod se construiește o altă structură de date). Această structură de date permite:

- Actualizarea rapidă a valorilor dp la poziția corespunzătoare valorilor compresate ale lui L și x.
- Interogarea maximului valorii $dp[i]$ pentru evenimentele anterioare care îndeplinesc condițiile impuse (în special, filtrul pe locație și condiția $R_i \leq R_j$).

Definim $dp[j]$ ca fiind suma maximă de satisfacție a unui lanț valid care se termină la evenimentul j. Recurența poate fi scrisă astfel:

$$dp[j] = s_j + \max\{dp[i] \mid i < j,\ x_i \in I_j \text{ și condițiile } L_i \leq L_j,\ R_i \leq R_j \text{ sunt îndeplinite}\},$$

unde I_j este intervalul de locații relevant ($[x_j - D, x_j + D]$). Prin interogarea structurii de date 2D pe acest interval, se obține valoarea maximă între toate $dp[i]$ relevante pentru evenimentul j, iar actualizarea acesteia se realizează ulterior.

Complexitatea totală se ridică astfel la $\mathcal{O}(N \log^2 N)$.

Problema 3: PRISON BREAK

Propusă de: Posdărăscu Eugenie Daniel, Youni

O soluție naivă este să fixăm cele 2 culori A, B selectate și să aplicăm algoritmul Dijkstra parcurgând doar noduri ce au culorile A și B. Complexitate $O(N^2 \cdot M \log M)$.

Construim o dinamică `d[node][another_colour]` semnificând costul minim să ajungem în nodul `node` pornind din sursa 1, iar una din culori este `another_colour`. Observația este că având selectat un nod, automat știm o culoare (să o notăm A) pe care o folosim în parcurgere (culoarea `color[node]` din acel nod). Prin urmare, `another_colour` putem forța să fie cealaltă culoare B din parcurgere. Menținând aceste proprietăți ne putem asigura că la fiecare pas știm care sunt cele 2 culori pe care le folosim. Această dinamică o rezolvăm asemănător algoritmului Dijkstra considerând că nodurile grafului sunt stările (`node`, `culoare`). În total $O(N^2)$ stări iar complexitatea este $O(N \cdot M \log)$.

Un subtask relevant de rezolvat pentru a ajunge la soluția finală este acela în care nu avem muchie între 2 noduri de aceeași culoare. În acest caz, observăm că șirul nostru este un șir de culori alternant (dacă avem 2 culori A și B, șirul o să fie A B A B ...). Dacă am construi o dinamică / Dijkstra unde stările sunt muchiile grafului în loc de noduri, o muchie automat ne oferă ca și informație care sunt cele 2 culori folosite (întrucât nu există o muchie de la o culoare A la altă culoare A). Indiferent dacă aplicăm algoritmul Dijkstra pe muchii în loc de noduri; sau pe stări (`nod`, C) cu proprietatea că C este o culoare vecină cu nodul `nod`, numărul total de stări se amortizează la $O(M)$, complexitatea finală pe acest subtask particular fiind $O(M \log M)$. Trebuie avut grijă în schimb la implementare că atunci când suntem la o muchie de la culoare A la culoarea B, să nu iterăm prin toți vecinii nodului care are culoarea B, ci doar prin aceia care au și ei în continuare culoarea A.

Dorim să construim o extindere a soluției de mai sus, dar problema acum este că avem și muchii între noduri cu aceeași culoare, acest lucru însemnând că o muchie nu îți mai garantează ca și informație ambele culori. Aici intervine momentul în care o să folosim observația că există componente conexe de dimensiune maximă 50 în graf care au aceeași culoare. Ce dorim să facem este ca în continuare să facem Dijkstra-ul pe stări (`nod`, C) unde C reprezintă evident cealaltă culoare decât cea oferită de nodul `nod`, dar în același timp fie să reușim să sărim peste lanțurile din graf de aceeași culoare, fie să ne asigurăm că nu depășim numărul de stări.

Dacă dorim să sărim lanțuri de aceeași culoare pentru a opera doar în stări cu culori diferite, o observație intuitivă este că putem să selectăm fiecare componentă conexă de noduri cu aceeași culoare și să aplicăm algoritmul Roy-Floyd pentru a determina distanța minimă între oricare 2 noduri din acea componentă. După pentru fiecare stare (`nod`, C), în loc să ne extindem doar în nodurile vecine de culoare C, ne putem extinde în acestea plus toate nodurile din componentele conexe acestora, multiplicând astfel numărul de stări cu maxim 50. Preprocesarea algoritmului Roy-Floyd oferă complexitate maximă $O(N/50 \cdot 50^3) = O(N \cdot 50^2)$, iar calcularea noului Dijkstra devine $O(M \log M \cdot 50)$.

O soluție alternativă mai elegantă de implementat este să propagăm culorile din stările de tipul (`nod`, C) - unde culoarea C este vecină nodului `nod` - în stări de tipul (`nod2`, C) - unde culoarea C nu mai este neapărat vecină nodului `nod2` - iar numărul total de stări și în acest caz se multiplică cu maxim 50 de noduri. Complexitate finală $O(M \log M \cdot 50)$.

Echipa

Problemele pentru această etapă au fost pregătite de:

- Argherie Ovidiu-Alexandru, Delft University of Technology

- Moca Andrei Cătălin, Universitatea ”Babeș-Bolyai”, Cluj-Napoca
- Posdărăscu Eugenie Daniel, Youni, București
- Szabó Zoltan, Inspectoratul Școlar Județean, Târgu-Mureș
- Bogdan Vlad-Mihai, Universitatea București
- Ciortea Liviu, Pexabit, București
- Constantinescu Andrei-Costin, ETH Zurich
- Feodorov Andrei, ETH Zurich
- Floare Doru, Universitatea ”Babeș-Bolyai”, Cluj Napoca
- Grigorean Andrei, CS Academy, București
- Ivan Andrei-Cristian, Universitatea Politehnica, București
- Oncescu Costin-Andrei, Harvard University
- Popescu Adrian Andrei, Universitatea Politehnica, București
- Silion Liviu-Mihai, Universitatea ”Babeș-Bolyai”, Cluj-Napoca
- Stănescu Matei-Octavian, Universitatea Politehnica, București
- Todoran Alexandru-Raul, Harvard University
- Tinca Matei, VU Amsterdam