Editorial McProiect

Comisia Științifică McProiect

Iunie 2025

1 Mulțumiri

Acest concurs nu a putut avea loc fără următoarele persoane:

- Horia Andrei Boeriu, Traian Mihai Danciu, Darius Hanganu, Rareș Hanganu, Radu Vasile, Ștefan Vîlcescu, autorii problemelor;
- Alex Vasiluță, fondatorul și dezvoltatorul principal al Kilonova;
- Lucian Andrei Badea, Luca Ștefan Camburu, Andrei Bogdan Dumitrache, Tiberiu Mihai Enache și Cătălin Frâncu, testerii concursului, care au dat numeroase sugestii și sfaturi utile pentru buna desfășurare a celor două runde;
- Celor care au participat la concurs.

2 Ziua 1

2.1 McFlurryOreo

Autor: Ștefan Vîlcescu

2.1.1 Soluție completă

Observație 1. *Nu contează în ce ordine sunt numerele.*

Convenție 1. *Vom presupune că* $x_1 \geq x_2 \geq x_3 \geq x_4$.

Cazul 1. *Dacă* $x_1 - x_2 - x_3 - x_4 \geq 2$*, răspunsul este NU.*

Demonstrație 1. *Oricum am pune cele* $x_2 + x_3 + x_4$ *litere, mereu vom avea cel puțin două dintre cele* x_1 *una lângă cealaltă.*

Cazul 2. *Altfel (dacă $x_1 - x_2 - x_3 - x_4 < 2$), răspunsul este DA.*

Demonstrație 2. *Putem pune câte o literă între câte două litere dintre cele x_1, și cum există $x_1 + 1$ locuri în care putem pune cele x_2, x_3 respectiv x_4 litere, mereu se poate.*

2.1.2 Cod sursă

Soluție de 100

2.2 BigTasty

Autori: Radu Vasile, Rareș Hanganu

2.2.1 Soluție completă

Definiție 1. *Fie $dp_{i,j}$ = numărul de moduri în care, folosind primele i tipuri de burgeri, Origu mănâncă j burgeri.*

Recurența este:

$$dp_{i,j} = \begin{cases} dp_{i-1,j}, & \text{dacă } j < p_i \\ dp_{i-1,j} + dp_{i-1,j-p_i} \cdot \binom{m-(j-p_i)}{p_i}, & \text{altfel} \end{cases}$$

Complexitatea memoriei acestei soluții este $O(N \cdot M)$, care nu se încadrează în limitele problemei. Pentru a lua 100, trebuie ținută o singură linie a dp-ului.

Complexitate timp: $O(N \cdot M)$
Complexitate memorie: $O(N + M)$

2.2.2 Cod sursă

Soluție de 100

2.3 McApp

Autor: Ștefan Vîlcescu

2.3.1 Soluție completă

Notație 1. *Notăm cu nrbits(a) câți biți are numărul a.*

Observație 1. *$f(0, a) = f(0, a \ mod \ 2^b) + f(nrbits(a \ mod \ 2^b), \frac{a}{2^b})$. Acest lucru ne dă ideea de a împărți numărul într-un prefix și un sufix.*

Observație 2. *Dacă numărul format din primii k biți este mai mic ca numărul format din primii k biți ai lui x, oricare ar fi următorii biți, numărul va fi mai mic decât x.*

Aceste două observații ne duc la următoarea soluție:

Mergem prin biții lui x de la cel mai mare la cel mai mic.

Notație 2. *Fie p bitul la care suntem acum.*

Notație 3. *Fie a numărul format din primii p biți.*

Cazul 1. *Dacă bitul p este* 0*, nu facem nimic.*

Cazul 2. *Dacă bitul p este* 1*, trebuie să aflăm într-un mod eficient* $max(f(nrbits(nr), a) + f(0, nr))$*, pentru* $0 \leq nr < 2^{k-p-1}$.

Răspunsul va fi maximul dintre aceste maxime și $f(0, x)$.

Pentru a calcula eficient $\max(f(\text{nrbits}(nr), a) + f(0, nr))$, vom calcula $f(\text{nrbits}(nr), a)$ si $f(0, nr)$ în paralel.

Observație 3. *Nu contează exact cât este nr pentru a calcula* $f(nrbits(nr), a)$*, ci numărul de biți, deci trebuie să aflăm* $max(f(0, nr))$ *eficient.*

Definiție 1. *Vom calcula* $dp_{msb,\ nrb}$ *= maximul dacă s-au folosit nrb biți până acum, iar cel mai semnificativ bit este msb.*

Recurența va fi:

$$\text{dp}_{\text{msb, nrb}} = \max(\text{msb} \cdot \text{mat}_{\text{msb, nrb}} + \text{dp}_{\text{msb}_2,\ \text{nrb}-1}),\ \text{pentru } 0 \leq \text{msb}_2 < \text{msb}$$

Cum $\text{msb} \cdot \text{mat}_{\text{msb, nrb}}$ este constant, trebuie să aflăm $\max(\text{dp}_{\text{msb}_2,\ \text{nrb}-1})$ eficient.

Definiție 2. *Redefinim astfel:* $dp_{msb,\ nrb}$ *= maximul dacă s-au folosit nrb biți până acum, iar cel mai semnificativ este cel mult msb.*

Acum putem afla ușor $\max(\text{dp}_{\text{msb}_2,\ \text{nrb}-1})$ apelând $\text{dp}_{\text{msb}-1,\ \text{nrb}-1}$. Noua recurență va fi:

$$\text{dp}_{\text{msb, nrb}} = \max(\text{dp}_{\text{msb}-1,\ \text{nrb}},\ \text{msb} \cdot \text{mat}_{\text{msb, nrb}} + \text{dp}_{\text{msb}-1,\ \text{nrb}-1})$$

Acum, dacă ne fixăm $\text{nrbits}(nr)$, putem afla $\max(f(0, nr))$ în timp optim.

Ca să calculăm $f(\text{nrbits}(nr), a)$, vom folosi cum se calculează $f(\text{bit}, \text{number})$ ajutându-ne de $f(\text{bit} + 1, \text{number} - 2^{\text{lsb(number)}})$.

Notație 4. *Vom menține un* $sum_{bit} = f(bit, a)$.

Când dăm de un bit 1 în x pe poziția poz, sum_{bit} va deveni $\text{sum}_{\text{bit}+1} + \text{bit} \cdot \text{mat}_{\text{bit},\ k-p-1}$, acestea putând fi calculate în timp optim.

Având aceste două tablouri, pentru fiecare bit de 1 putem calcula maximele astfel:

Ne vom fixa nrbits(nr). Fie ans = maximul maximelor calculate pentru fiecare bit până acum. Acum, putem afla în $O(1)$ $\max(f(\text{nrbits}(nr), a) + f(0, nr))$, care este $\text{sum}_{\text{nrbits}(nr)} + \text{dp}_{\text{nrbits}(nr),\ k-p-1}$. Actualizăm ans cu acest maxim.

La final se va afișa răspunsul, ca mai sus: $\max(\text{ans}, f(0, x))$

2.3.2 Cod sursă

Soluție de 100

2.4 Cartofi Homestyle Dippers

Autor: Rareș Hanganu

2.4.1 Subtask 1: $1 \leq n \leq 200$

Pentru fiecare subsecvență $[l, r]$ fixată, vom verifica brut dacă toate elementele sunt divizibile cu minimul.

Complexitate timp: $O(N^3)$
Complexitate memorie: $O(N)$

2.4.2 Subtask 2: $1 \leq n \leq 3000$

Observație 1. *Din moment ce toate elementele din subsecvență trebuie să fie divizibile cu minimul, înseamnă că și gcd-ul (cel mai mare divizor comun) acestor elemente va fi divizibil cu minimul. Dar minimul apare în subsecvență, ceea ce înseamnă că ne interesează subsecvențele pentru care gcd-ul numerelor este egal cu minimul.*

Cu această observație, putem fixa capătul stânga l al subsecvenței, iar după aceea pentru fiecare capăt dreapta r să menținem gcd-ul și minimul numerelor.

Complexitate timp: $O(N^2 \cdot \log_2 A)$, deoarece gcd durează $O(\log_2 A)$
Complexitate memorie: $O(N)$

2.4.3 Subtask 3: $5 \cdot 10^8 < a_i \leq 10^9$

Observație 2. *Dacă alegem două numere x și y din șir, cu proprietatea că $x|y$, atunci $x = y$.*

Demonstrație 1. *Dacă cele două numere nu ar fi egale, ar însemna că* $y \geq 2 \cdot x$*, dar* $2 \cdot x > 2 \cdot 5 \cdot 10^8 = 10^9$*, deci* y *ar fi mai mare decât* 10^9*, ceea ce nu este posibil.*

Deci ne interesează subsecvențele cu toate elementele egale, așa că vom menține pentru fiecare poziție i câte numere din stânga sunt egale cu a_i. Fie această valoare $left_i$. Atunci, fiecare capăt dreapta i fixat va contribui cu $left_i$.

Complexitate timp: $O(N)$
Complexitate memorie: $O(1)$ (nu este necesar să reținem vectorul left)

2.4.4 Subtask 4: $1 \leq n \leq 50000$

Notație 1. *Vom fixa poziția minimului din subsecvență. Fie aceasta* i*.*

Observație 3. *Dacă* $[l, r]$ *este o subsecvență bună și* $l < i < r$*, atunci și* $[l + 1, r]$*,* $[l, r - 1]$ *vor fi subsecvențe bune. Deci vrem să găsim* l *minim și* r *maxim astfel încât* $gcd(l..r) = a_i$*.*

Numărul de secvențe bune în care a_i este minim va fi $(i - l + 1) \cdot (r - i + 1)$

Pentru acest subtask, este suficient să folosim un AINT pentru aflarea gcd-ului și să căutăm binar capetele stânga și dreapta.

Complexitate timp: $O(N \cdot \log^2 N \cdot \log_2 A)$
Complexitate memorie: $O(N)$

2.4.5 Subtask 5: Fără alte restricții

Pentru 100 de puncte trebuie optimizată căutarea capetelor stânga și dreapta. Aici avem mai multe metode:

- AINT: vom folosi soluția de la subtask-ul 4, cu observația că căutarea binară pe AINT se poate face în $O(\log_2 N)$.
- Sparse Table: vom folosi un sparse table pentru a menține pentru fiecare interval de lungime putere a lui 2 gcd-ul pe acel interval. Pentru căutarea binară, avem 2 variante:
  - pentru fiecare query putem compune intervalul din 2 intervale care se suprapun, similar cu rezolvarea query-urilor de tip minim pe interval (RMQ).
  - putem să facem o căutare binară similară cu căutarea binară pe AINT sau AIB.

Complexitate timp: $O(N \cdot \log N \cdot \log A)$
Complexitate memorie: $O(N)$ cu AINT, $O(N \cdot \log N)$ cu Sparse Table

2.4.6 Soluție alternativă

Autor soluție: Luca Ștefan Camburu

Asemănător cu soluțiile de la subtaskurile 4 și 5, încercăm să determinăm eficient capetele unei secvențe care să aibă minimul un element fixat. O primă margine pe care o putem determina este cea garantată de ultimul element la stânga mai mare decât a_i, respectiv primul element la dreapta mai mare decât a_i. Aceste margini se pot realiza cu o stivă ordonată sau un RMQ și o căutare binară. Pentru elemente egale consecutive, se poate realiza același algoritm, însă pentru perechi ordonate (a_i, i), argumentul fiind același ca la determinarea arborelui cartezian unui vector.

Pentru îmbunătățirea marginilor, pentru fiecare element a_i, se poate realiza o dinamică:

Definiție 1. *Fie $left_i$ = care este indicele cel mai mic pentru care se respectă proprietatea că toate elementele cu indici de la $left_i$ la i sunt divizibile cu a_i.*

Notație 2. *Notăm cu p cea mai din dreapta poziție înainte de i în care apare a_i.*

Notație 3. *Să notăm cu l cel mai mic indice astfel încât $p \leq l \leq i, a_i \mid a_j, \forall\ j \in [l, i]$*

Recurența va fi următoarea:

$$\text{left}_i = \begin{cases} \text{left}_p & \text{dacă } l = p \\ \text{l} & \text{altfel} \end{cases}$$

Pentru determinarea ultimei apariții a valorii a_i, se poate utiliza map sau unordered-map. Aceeași idee se poate utiliza și pentru dreapta.

Vom demonstra că complexitatea va fi $O(n \log_2 \text{MAXVAL})$

Demonstrație 2. *O să încercăm să determinăm câte operații se realizează în partea brut, întrucât partea de programare dinamică este $O(n)$ sau $O(n \log_2 n)$ după map/unordered-map. Încercăm să calculăm în câte secvențe între valori egale se găsește o poziție i. Dacă această secvență este completă (aproximativ $O(n)$), atunci este clar că toate acele elemente sunt divizibile cu a_i și diferite de a_i. Astfel, este clar că toate acele elemente sunt cel puțin dublul lui a_i. Intrucât $a_i \leq 10^9$, se pot face cel mult $\log_2 a_i$ parcurgeri.*

Complexitate timp: $O(N \log_2 \text{MAXVAL})$ Complexitate memorie: $O(N)$

2.4.7 Cod sursă

Soluție de 100 cu AINT
Soluție de 100 cu Sparse Table
Soluție alternativă de 100

2.5 McNugget

Autori: Horia Andrei Boeriu, Traian Mihai Danciu

2.5.1 Subtask 1: $1 \leq n, q \leq 40$, $1 \leq v_i \leq 1000$

La fiecare query vom trece prin fiecare subsecvență și vom calcula frecvența fiecărei valori din ea. Apoi, pentru valorile care apar de minim k ori vom verifica dacă au un divizor comun diferit de 1 cu x (gcd-ul lor cu x este mai mare strict decât 1).

Complexitate timp: $O(Q \cdot N^3 \cdot \log_2 \text{MAXVAL})$
Complexitate memorie: $O(N + \text{MAXVAL})$

2.5.2 Subtask 2: $1 \leq n, q \leq 1000$

Pentru că k nu se schimbă la niciun query, putem precalcula la început pentru fiecare număr în câte subsecvențe apare de minim k ori. Putem face acest lucru trecând prin pozițiile numărului din vector.

Notație 1. *Să notăm cu $secv_i$ această contribuție calculată mai sus, pentru valoarea i.*

Acum, în loc să adunăm contribuția fiecărei subsecvențe putem aduna contribuția fiecărui număr care are gcd cu x mai mare strict decât 1.

Deci la fiecare query vom testa gcd-ul numerelor distincte din vector cu x și vom aduna contribuția celor care au un divizor comun diferit de 1 cu acesta.

Complexitate timp: $O(Q \cdot N \cdot \log_2(\text{MAXVAL}))$
Complexitate memorie: $O(N + \text{MAXVAL})$

2.5.3 Subtask 3: $1 \leq q \leq 10^5$

Ne vom folosi de precalcularea de la subtaskul 2 (contribuția fiecărei valori), dar în plus, vom mai calcula cu un ciur pentru fiecare număr suma contribuțiilor multiplilor săi.

Notație 2. *Notăm cu sum_i suma contribuțiilor multiplilor lui i.*

Așa că putem să folosim principiul includerii și excluderii (PINEX), iar la fiecare query vom trece prin divizorii lui x care sunt liberi de pătrate și vom aduna sau scădea contribuția multiplilor fiecărui divizor, în functie de paritatea numărului de factori primi din descompunere.

Ca să trecem prin divizorii liberi de patrate, trebuie doar să îl descompunem pe x în factori primi și să trecem prin fiecare combinație de factori primi distincți.

Această soluție poate obține și 100 de puncte cu o implementare eficientă.

Complexitate timp: $O(N + \text{MAXVAL} * \log \text{MAXVAL} + Q * 2^{\text{NRFACT}})$
Complexitate memorie: $O(N + \text{MAXVAL})$

2.5.4 Soluție completă

Observație 1. *Răspunsul pentru un query va fi suma valorilor $sum_d \cdot coef_d$ pentru toți divizorii d ai lui x.*

$$\text{coef}_d = \begin{cases} -1 & \text{dacă } d \text{ are număr par de factori primi} \\ 0 & \text{dacă } d \text{ nu este liber de pătrate} \\ +1 & \text{dacă } d \text{ are număr impar de factori primi} \end{cases}$$

Observație 2. *$coef_d = -\mu(d)$, unde μ reprezintă funcția Möbius*

Mai departe, vom precalcula răspunsurile, așa că vom avea:

$$\text{answer}_i = \sum_{d|i} -\mu(d) \cdot \text{sum}_d$$

Complexitate timp: $O(N + \text{MAXVAL} \cdot \log(\text{MAXVAL}) + Q)$
Complexitate memorie: $O(N + \text{MAXVAL})$

2.5.5 Cod sursă

Soluție cu PINEX, care ia 85
Soluție cu PINEX, implementată elegant, care ia 100
Soluție cu Möbius, care ia 100

2.6 McProgres

Autori: Ștefan Vîlcescu, Traian Mihai Danciu, Radu Vasile

Convenție 1. *Un singur element formează o progresie aritmetică. Vom trata separat acest caz.*

2.6.1 Subtaskul 1

Putem să sortăm șirul și să verificăm dacă toate diferențele între două elemente consecutive sunt egale, însă noi vom face o altă soluție, care ne va ajuta la soluția completă.

Observație 1. *O subsecvență $[l, r]$ ar fi progresie aritmetică daca ar fi sortată crescător dacă respectă următoarele condiții:*

- $\frac{max-min}{r-l} = ratie,\ ratie \in \mathbb{N}^*$*, unde max și min sunt maximul, respectiv minimul șirului (ratie va fi rația progresiei);*
- *Pentru fiecare $i \in [l, r]$ există un x astfel încât $v_i = min + ratie \cdot x$, adică $gcd(|v_l - v_{l+1}|,\ |v_{l+1} - v_{l+2}|,\ ...) = ratie$, unde gcd înseamnă cel mai mare divizor comun;*
- *Elementele trebuie să fie distincte.*

Vom verifica brut toate aceste condiții (sortând subsecvența, ca să aflăm ușor maximul, minimul și dacă sunt distincte).

Complexitate: $O(q \cdot n \cdot \log_2(n))$ timp, $O(n)$ memorie

2.6.2 Subtaskul 2

Vom calcula prev_i = cel mai mare $j < i$ astfel încât $v_i = v_j$. Dacă nu există un astfel de j, avem $\text{prev}_i = 0$. Astfel, toate elementele din subsecvența $[l, r]$ sunt distincte dacă $\text{prev}_i < l,\ \forall\ i \in [l, r]$, adică $\text{maxprev} < l$, unde $\text{maxprev} = \max(\text{prev}_l, \text{prev}_{l+1}, ..., \text{prev}_r)$.

Vom folosi AINT sau RMQ pentru a afla maximul subsecvenței, minimul subsecvenței, maximul prev-urilor din subsecvență și gcd=ul diferențelor de elemente conseuctive.

Complexitate: $O(q \cdot \log_2(n) \cdot \log_2(\text{MAXVAL}))$ timp, $O(n)$ memorie dacă folosim AINT, respectiv $O(q \cdot \log_2(\text{MAXVAL}))$ timp, $O(n \cdot \log_2(n))$ memorie dacă folosim RMQ, deoarece operația gcd(a, b) durează $O(\log_2(a))$. Soluția dată folosește AINT.

2.6.3 Subtaskul 3

Calculul maximului subecvenței, minimului subsecvenței, *gcd*-ului șirului de diferențe nu se modifică cu mult dacă folosim AINT (trebuie doar să implementăm și operația $v_{\text{poz}} = \text{val}$).

Pentru a afla dacă elementele sunt distincte, vom folosi aceeași abordare ca la problema clasică Distinct Values Queries II de pe CSES. Putem să modificăm vectorul prev doar la pozițiile necesare dacă menținem într-un set pentru fiecare valoare indicii la care apare acea valoare.

Complexitate: $O(q \cdot \log_2(n) \cdot \log_2(\text{MAXVAL}))$ timp, $O(n)$ memorie

2.6.4 Soluție completă

Mai departe, trebuie doar să menținem și valorile normalizate. Pe noi nu ne interesează ca după normalizare, valorile să își păstreze ordinea, așa că putem să normalizăm online.

Complexitate: $O(q \cdot \log_2(n) \cdot \log_2(\text{MAXVAL}))$ timp, $O(n)$ memorie

Observație 2. *Există și soluții cu hashing, cu complexitate $O(q \cdot \log_2(n))$ timp și $O(n)$ memorie, care pot obține punctajul maxim, însă există contraexemple pentru fiecare.*

2.6.5 Cod sursă

Soluție de 21 - subtaskul 1
Soluție de 42 - subtaskul 2
Soluție de 7 - subtaskul 3
Soluție de 100 - completă

3 Ziua 2

3.1 BigMac

Autor: Traian Mihai Danciu

3.1.1 Soluție completă

Notație 1. *Notăm cu neg numărul de elemente negative.*

Avem două cazuri: $k >$ neg sau $k \leq$ neg

Cazul 1. *$k >$ neg. Vom face toate elementele care inițial erau negative să fie pozitive. Ne-au rămas $k -$ neg operații.*

Subcaz 1.1. *$k -$ neg este par. Putem face toate operațiile rămase pe un singur element, iar acestea se vor anula pentru că $-(-x) = x$. Răspunsul va fi suma valorilor absolute din tot șirul.*

Subcaz 1.2. *$k -$ neg este impar, dar există cel puțin un 0. Vom face toate operațiile rămase pe acel 0, iar acesta nu se va modifica ($-0 = 0$).*

Subcaz 1.3. *$k -$ neg impar și nu există niciun 0. Vom irosi $k -$ neg $- 1$ operații pe orice element (deoarece este număr par) și ne va mai rămâne o operație. Când facem o operație pe un element, înseamnă că vom împărți șirul într-un sufix și un prefix. Este optim să facem operația fie pe primul, fie pe ultimul, ca să avem prefixe sau sufixe cât mai lungi.*

Cazul 2. *$k \leq neg$. Vom strânge toate elementele negative într-un șir. Vom lua secvențe de câte k elemente negative (nu neapărat consecutive în șir, dar consecutive în șirul de elemente negative). Acum, va trebui doar să vedem dacă suma valorilor absolute din secvența $[st + 1, dr - 1]$ este maximă, unde st și dr reprezintă indicii primelor elemente negative la stânga, respectiv la dreapta. Dacă nu mai sunt elemente negative la stânga sau la dreapta, $st = 0$ sau $dr = n + 1$, respectiv.*

Complexitate timp: $O(n)$
Complexitate memorie: $O(n)$

3.1.2 Cod sursă

Soluție de 100

3.2 Fillet-o-Fish 1

Autori: Radu Vasile, Traian Mihai Danciu, Horia Andrei Boeriu

3.2.1 Subtaskul 2

Definiție 1. *Definim dp_i = suma maximă care se poate obține din prefixul $[1, i]$ dacă i este păstrat.*

Recurența este următoarea:

$$\mathrm{dp}_i = v_i + \max(\mathrm{dp}_j), \text{pentru } j \in [0, i-1], i \text{ poate să stea lângă } j$$

Vom itera prin toate valorile ale lui j de la 1 la $i - 1$ și vom verifica dacă i poate să stea lângă j.

Pentru valoarea minimă vom face analog.

Complexitate timp: $O(N^2)$
Complexitate memorie: $O(N^2)$

3.2.2 Soluție completă

În loc să iterăm prin toate valorile lui j, vom itera doar prin cele care ni se dau la intrare, folosind std::vector.

Pentru valoarea minimă vom face analog.

Complexitate timp: $O(N + M)$
Complexitate memorie: $O(N + M)$

3.2.3 Cod sursă

Soluție de 40
Soluție de 100

3.3 Fillet-o-Fish 2

Autori: Radu Vasile, Traian Mihai Danciu, Horia Andrei Boeriu

3.3.1 Subtaskul 2

Soluția este asemănătoare cu cea de la Fillet-o-Fish 1.

Definiție 1. *Definim dp_i = suma maximă care se poate obține din prefixul $[1, i]$ dacă i este păstrat.*

Recurența este următoarea:

$$\mathrm{dp}_i = v_i + \max(\mathrm{dp}_j), \text{pentru } j \in [0, i-1], i \text{ poate să stea lângă } j$$

Vom itera prin toate valorile ale lui j de la 1 la $i-1$ și vom verifica dacă i poate să stea lângă j.

Pentru valoarea minimă vom face analog.

Complexitate timp: $O(N^2)$
Complexitate memorie: $O(N^2)$

3.3.2 Soluție completă

Vom menține un AINT cu valorile dp-ului.

Vom itera prin valorile lui j date la intrare (cu std::vector) și vom seta în AINT valoarea $-$INFINIT.

Pentru a afla maximul dp-urilor, doar vom da query la AINT.

Există și o soluție care folosește std::set sau std::multiset, însă aceasta s-ar putea să nu intre în timp.

Complexitate timp: $O((N+M) \cdot \log_2 N)$
Complexitate memorie: $O(N+M)$

3.3.3 Cod sursă

Soluție de 40
Soluție cu std::multiset de 40
Soluție de 100

3.4 Mc-A-Mc Mc

Autor: Ștefan Vîlcescu

3.4.1 Soluție completă

Observație 1. *Indiferent de cum permutăm direcțiile, tot la aceleași coordonate de final ajungem (fie acestea l, c). Astfel, pentru fiecare interogare, vrem să vedem dacă putem forma suma k pe orice drum de la $(1, 1)$ până la (l, c).*

Definiție 1. *Fie $dp[i][j]$ o mască unde bitul s este activat dacă putem forma suma s mergând pe un drum de la coordonatele $(1, 1)$ până la (i, j).*

$$\mathrm{dp}_{i,j,s} = \begin{cases} 1, & \text{dacă } \mathrm{dp}_{i-1,j,s-\mathrm{mat}_{i,j}} = 1 \text{ sau } \mathrm{dp}_{i,j-1,s-mat_{i,j}} \\ 0, & \text{altfel} \end{cases}$$

Această soluție nu intră în limita de memorie, având complexitatea memoriei de $O(n \cdot m \cdot \mathrm{SUM})$. Putem să o micșorăm la $O(m \cdot \mathrm{SUM})$ dacă ținem doar o linie a dp-ului, dar asta ar însemna că ar trebui să procesăm întrebările offline, sortându-le după linia coordonatei de final.

Observație 2. *Putem să optimizăm dp-ul, dacă îl ținem cu un bitset, recurența devenind $dp_j = (dp_j \mid dp_{j-1}) \ll mat_{i,j}$.*

Complexitate timp: $O(Q \cdot \log_2 Q + N \cdot M \cdot \frac{\mathrm{SUM}}{w})$
Complexitate memorie: $O(Q + N \cdot M + M \cdot \frac{\mathrm{SUM}}{w})$

3.4.2 Cod sursă

Soluție de 100

3.5 McMartin

Autor: Rareș Hanganu

3.5.1 Subtask 1: $1 \leq n, q \leq 500$

Fie

$$sp_i = \sum_{p=1}^{i} a_p$$

Pentru fiecare query, vom parcurge toate perechile (i, j) pentru care $l \leq i < j \leq r$, iar condiția ca perechea să fie *martin* este:

$$a_i + a_j \equiv sp_{j-1} - sp_i \pmod{k}$$

Complexitate timp: $O(Q \cdot N^2)$
Complexitate memorie: $O(N)$

3.5.2 Subtask 2: $1 \leq k \leq 50$

Observație 1. *O pereche (i, j) este martin dacă și numai dacă:*

$$a_i + a_j \equiv sp_{j-1} - sp_i \pmod{k}$$

Putem rescrie relația ca:

$$a_i + sp_i \equiv sp_{j-1} - a_j \pmod{k}$$

Pentru fiecare poziție i vom calcula:

$$\text{cnt}_{i,r} = \begin{cases} 0, & \text{dacă } sp[i-1] - a[i] \not\equiv r \pmod{k} \\ \#\left\{j < i \mid (a[j] + sp[j]) \bmod k = r\right\}, & \text{altfel} \end{cases}$$

După aceea, vom construi sume parțiale, astfel încât $\text{cnt}_{i,r}$ devine:

$$\sum_{p=1}^{i} \text{cnt}_{p,r}$$

Pentru un query (l, r), vom fixa r', valoarea comună a celor două expresii: $(a_i + sp_i) \bmod k$ și $(sp_{j-1} - a_j) \bmod k$.

Pentru acest r' fixat, vom calcula $cnt_{r,r'} - cnt_{l-1,r'}$, din care trebuie să scădem numărul de elemente din interval care au $(sp_{j-1} - a_j) \bmod k = r'$, înmulțit cu numărul de poziții $j < l$ pentru care $(a_j + sp_j) \bmod k = r'$.

Complexitate timp: $O((N + Q) \cdot K)$

Complexitate memorie: $O(N \cdot K)$

3.5.3 Subtask 3: Fără alte restricții

Pentru a răspunde eficient la query-uri, vom folosi algoritmul lui Mo.

Pentru a putea efectua operațiile de extindere/contractare a intervalului, vom ține doi vectori de frecvență: unul pentru valorile $(a_i + sp_i) \bmod k$ din interval, iar altul pentru valorile $(sp_{i-1} - a_i) \bmod k$ din interval.

Complexitate timp: $O((N + Q) \cdot \sqrt{N})$

Complexitate memorie: $O(N + K + Q)$

3.5.4 Soluție alternativă

Autor soluție: Luca Ștefan Camburu

Combinăm soluția de la subtaskul 2 cu una care rezolvă $K > 50$. Facem o baleiere și pentru un i fixat și determinăm condiția pentru j în funcție de sumele parțiale (restul pe care trebuie să-l aibă $(s_i + a_i) \bmod K$). Menținem pentru fiecare rest un vector specific al tuturor pozițiilor de tip j care să aibă $(s_j + a_j) \bmod K = \text{rest}$. Se poate menține o structură de date care face update punctual cu creșterea unei poziții și query pe interval (arbore indexat binar sau arbore de intervale). Există cel mult $\frac{N}{K}$ valori pe grupă de rest, așadar au loc $N \cdot \frac{N}{K} \cdot \log_2 N$ operații. Dacă se alege minimul dintre cele 2 complexități ($O((N+Q)K)$ și $O(\frac{N^2}{K} \log_2 N + Q \log_2 N)$), se ajunge la complexitatea de $O(N\sqrt{N} \log_2 N)$, suficient pentru a trece testele.

3.5.5 Cod sursă

Soluție de 40
Soluție de 100
Soluție alternativă de 100

3.6 McArbore

Autor: Ștefan Vîlcescu

3.6.1 Soluție completă

Notație 1. *Notăm cu m numărul de frunze din arbore.*

Notație 2. *Notăm cu fr șirul de frunze în preordine (ordine în care le descoperim la DFS).*

Observație 1. *Frunzele din subarborele unui nod u vor fi puse într-un interval $[st, dr], 1 \leq st \leq dr \leq m$.*

Notație 3. *Fie $[st_u, dr_u]$ intervalul frunzelor din subarborele lui u*

Notație 4. *Definim $f_i = a_{fr_i}$, pentru $1 \leq i \leq m$ și $f_0 = f_{m+1} = 0$.*

Notație 5. *Fie b un șir de $m+1$ elemente, astfel încât, $b_i = f_{i+1} - f_i$ pentru $0 \leq i \leq m$*

Pentru fiecare nod u, dacă am aplica o operatie "aduna", operația de a aduna la șirul a numărul x pentru toate frunzele din subarbore, s-a transformat în a aduna x la $b_{st_u - 1}$ și $-x$ la b_{dr_u}.

Pentru fiecare nod u, tragem o muchie de la $st_u - 1$ la dr_u cu costul c_u, care va însemna operația de a activa nodul u.

Pentru a atinge costul minim, trebuie ca muchiile alese să formeze un arbore parțial de cost minim. Răspunsul va fi chiar costul acestui arbore parțial.

Vom demonstra că nu există strategie cu mai puțin strict decât m muchii.

Demonstrație 1. *Trebuie să arătăm de ce nu se poate mai puțin de m muchii alese (avem cele m noduri frunze și nodul auxiliar 0 în noul raf, având $m+1$ noduri). Putem să ne gândim la noduri ca un fel de mulțimi de numere, iar ca strategia să meargă, ar trebui să se îndeplinească două lucruri:*

- *Să nu existe două mulțimi la fel*
- *Să nu existe mulțimi vide*

Subdemonstrație 1.1. *Dacă ar exista două mulțimi la fel i și j, înseamnă că dacă am adăuga la a_i un x, îl vom adăuga și la a_j, având același efect și invers; acest lucru va funcționa doar dacă $a_i = a_j$, iar acest lucru nu este adevărat pentru toate șirurile a.*

Subdemonstrație 1.2. *Dacă există mulțimea vidă i, a_i va rămâne același, deci $a_i = 0$, ceea ce nu este adevărat pentru toate șirurile a.*

Astfel, dacă am trage muchiile $st - 1 \leftrightarrow dr$, ar trebui să existe cel puțin una care începe din i, $0 \leq i < m$.

Subdemonstrație 1.3. *Dacă nu ar exista o muchie pentru un nod u, se întâmplă două lucruri:*

- *Pentru $u > 0$, mulțimea nodului u va fi la fel ca mulțimea nodului $u - 1$.*
- *Mulțimea nodului 0 va fi vidă, deci nu se poate.*

Astfel, oricum am alege maxim $m - 1$ muchii, ajungem la o contradicție. Rezultă că trebuie să folosim minim m muchii.

Pentru m muchii, putem face astfel:

Alegem exact o muchie care începe din fiecare nod i, $0 \leq i < m$.

Vom demonstra că aceasta este o componentă conexă prin inducție:

Demonstrație 2. *Un singur nod este componentă conexă (nodul m).*

Acum, dacă știm că ultimele i noduri sunt conectate, pentru nodul $m - i$, cum tragem o muchie la un nod cu indice mai mare ca el, va fi conectat cu acel nod, iar acel nod mai mare este conectat la restul $\Rightarrow$ nodul $m - i$ este conectat cu toate celelalte noduri $\Rightarrow$ este componentă conexă.

Este suficient să alegem m muchii deoarece putem să mergem prin i-uri în ordine crescătoare și să alegem intervalul care începe din i și să adunăm $-a_i$, după ce au fost procesate celelalte intervale.

Cum noi vrem componenta conexă cu cost minim dintr-un graf de $m + 1$ noduri, alegând m, se va afla costul arborelui parțial de cost minim din graf.

Complexitate timp: $O(N \log_2 N)$ Complexitate memorie: $O(N)$

3.6.2 Cod sursă

Soluție de 100