Tabăra de pregătire a lotului național de informatică - juniori, Craiova 9-14 mai
Baraj 1
Descrierea soluțiilor

Comisia științifică

May 13, 2025

Problema 1. Lemmings

Propusă de: prof. Gheorghe-Eugen Nodea, Centrul Județean de Excelență Gorj, Târgu-Jiu

Toate soluțiile de mai jos necesită abilitatea de a calcula cantitatea de hrană disponibilă într-un interval de camere. Putem precalcula această informație într-un vector de lungime n (sau dublat la $2n$, pentru a simplifica tratarea circularității). Notăm 1 pe pozițiile unde există hrană și 0 în rest, apoi calculăm sume parțiale. Acum cantitatea de hrană din intervalul $[st, dr]$ este diferența între sumele parțiale la pozițiile dr și $st - 1$.

Această tehnică adaugă un factor de $O(n)$ la timp și la memorie. Pentru glorie virtuală, putem evita acest factor dacă calculăm informația în $O(m + k + t)$, interclasând vectorii de hrană și lemingi. Nu putem calcula hrana din orice interval arbitrar, dar putem calcula o informație care ne interesează, și anume: pentru intervalul $[st, dr]$ dintre doi lemingi, câtă hrană este consumată în cele patru scenarii posibile?

1. Lemingul stâng pornește spre stânga, iar cel drept spre stânga.
2. Lemingul stâng pornește spre stânga, iar cel drept spre dreapta.
3. Lemingul stâng pornește spre dreapta, iar cel drept spre stânga.
4. Lemingul stâng pornește spre dreapta, iar cel drept spre dreapta.

Subtask 1

Cînd $k \leq 20$ putem evalua toate cele 2^k posibilități de orientare. Pentru fiecare posibilitate putem calcula hrana consumată în $O(k)$, iar o soluție în $O(2^k \cdot k)$ bine scrisă va trece toate testele.

Putem și să evaluăm posibilitățile în ordinea codului Gray, caz în care între două posibilități evaluate consecutiv trebuie să schimbăm direcția unui singur leming. Astfel obținem complexitatea $O(2^k)$, dar această complexitate nu este necesară pentru obținerea punctelor pe subtask.

Subtask 2

Dacă distanța între oricare doi lemingi (consecutivi) este mai mare de $2t$, atunci fiecare leming are „teritoriul" său în care doar el ajunge la hrană. Deci este suficient să calculăm, pentru fiecare leming, maximul dintre hrana disponibilă la stânga și la dreapta, iar răspunsul este suma acestor maxime.

Acest subtask poate ajuta la verificarea corectitudinii codului de sume pe interval, cu circularitate.

Subtask 3

Dat fiind că în intervalul $[1, t]$ nu există nici lemingi, nici hrană, putem considera vectorul liniar. Algoritmul corect este programarea dinamică descrisă mai jos, iar subtaskul poate ajuta la verificarea codului fără complexitatea suplimentară dată de circularitate.

Subtaskul 4

Soluțiile optime necesită reducerea problemei de la forma circulară la forma liniară. Pentru aceasta, este suficient să încercăm ambele orientări pentru unul dintre lemingi, să zicem primul. Apoi putem trata zona de $n - t$ camere rămase ca pe un vector normal (liniar). Calculăm rezultatul pentru acest vector și adăugăm hrana consumată de primul leming. Răspunsul este maximul dintre cele două variante.

Pentru forma liniară, o primă abordare definește $H[i][j][st/dr][st/dr]$ ca fiind cantitatea maximă de hrană pe care o pot mânca lemingii $i, i+1, \ldots, j$ în condițiile în care lemingul i pornește spre stânga / dreapta, iar lemingul j pornește spre stânga / dreapta. Practic, $H[i][j]$ este un cvadruplet.

Putem calcula recurent H fie încercând să împărțim $H[i]][j]$ în două porțiuni $H[i][r]$ și $H[r][j]$ pentru fiecare $r \in [i+1, j-1]$, fie extinzând intervalul $[i, j-1]$ cu o poziție la dreapta. De exemplu,

$$H[i][j][st][st] = \max \begin{cases} H[i][r][st][st] + H[r][j][st][st] \\ H[i][r][st][dr] + H[r][j][dr][st] \end{cases}$$

Observăm că lemingii i și j își păstrează orientările, iar pentru lemingul k încercăm ambele orientări. Cazul de bază este $j - i = 1$, care este un singur interval între doi lemingi consecutivi.

Acest algoritm duce la complexitatea $O(k^3)$ sau $O(k^2)$.

Subtaskul 5

Pentru punctaj maxim putem descrie o recurență mai simplă. Fie $x_1, x_2, \ldots x_k$ pozițiile lemingilor. Fie $S[x, y]$ cantitatea de hrană din intervalul închis $[x, y]$, cu convenția că, dacă $x > y$, atunci $S[x, y] = 0$. Fie $H[i][st/dr]$ cantitatea maximă de hrană consumată de primii i lemingi în condițiile în care lemingul i pornește spre stânga, respectiv spre dreapta. Atunci iau naștere patru cazuri similare, deoarece pentru fiecare orientare a lemingului i vom analiza cele două orientări posibile pentru lemingul $i - 1$.

$$H[i][st] = \max \begin{cases} H[i-1][st] + S[\max\{x_{i-1}, x_i - t\}, x_i] \\ H[i-1][dr] + S[\max\{x_{i-1} + t, x_i - t\}, x_i] \end{cases}$$

În cuvinte, lemingul i va consuma hrană spre stânga, nu mai mult de t poziții și fără a depăși lemingul $i - 1$ sau hrana consumată de acesta. Similar,

$$H[i][dr] = \max \begin{cases} H[i-1][st] + S[x_i, \min\{x_i + t, x_{i+1}\}] \\ H[i-1][dr] + S[x_i, \min\{x_i + t, x_{i+1}\}] \end{cases}$$

În cuvinte, lemingul i va consuma hrană spre dreapta, nu mai mult de t poziții și fără a depăși lemingul $i + 1$. Subliniem că nu ne preocupăm de situația în care lemingul $i + 1$ se îndreaptă spre stânga. Vom trata acest caz (lemingii i și $i + 1$ se îndreaptă unul spre celălalt) din perspectiva lemingului $i + 1$.

Motivul pentru care această recurență este corectă este că lemingul i nu poate fi influențat de decizia lemingilor $1, 2, \ldots, i - 2$. Lemingul $i - 1$ îi blochează pe toți aceștia, indiferent în ce orientare se află.

Acest algoritm are complexitatea $O(k)$, iar complexitatea totală este $O(k + m + t)$ sau $O(k + m + t + n)$ în funcție de implementarea sumelor pe interval.

Problema 2. Mutare

Propusă de: prof. Ciprian Chește, Liceul Tehnologic „Grigore C. Moisil", Buzău

Soluția 1 „brute-force"

Se construiește o structură de date care poate memora starea fiecărui segment al unui indicator și apoi se extinde această structură pentru mai multe indicatoare. Dacă un segment este aprins, în structură se va memora 1 și respectiv 0 dacă segmentul este stins. Se determină apoi numărul de cifre ale lui N, pe care să-l notăm cu nrcifre și se transformă fiecare cifră a numărului N într-un element al structurii. Se parcurg apoi toate numerele care au exact nrcifre și se păstrează doar acelea care diferă față de N printr-o singură mutare. Pentru a rezolva această problemă trebuie realizate funcții specifice pentru:

- comparare a două indicatoare
- comparare a două numere sub forma lor echivalentă din structură

- transformarea unui număr din forma sa structurată în număr de tip long long

Pentru a compara două numere sub forma lor echivalentă din structura se poate folosi operatorul XOR. Soluția poate obține aproximativ 40 puncte.

Soluția 2 „perechi de cifre”

O altă variantă, mai eficientă, se poate realiza prin parcurgerea tuturor perechilor $1 \leq i \leq j \leq nrcifre$ având în vedere că se face o singură mutare, mutare care stinge un segment pe poziția i și-l aprinde pe poziția j. Pozițiile i și j pot fi chiar egale, caz în care mutarea unui segment se face în cadrul aceluiași indicator. Sunt necesare și alte funcții suplimentare față de varianta anterioară cum ar fi o funcție care să testeze dacă un indicator la care s-a aprins un segment reprezintă sau nu, un număr. Ordinul de complexitate al soluției este $O(nrcifre^2)$. Soluția obține 100 puncte.

Problema 3. Wall-E

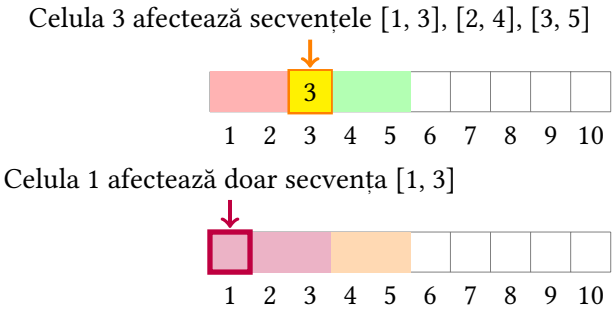
Propusă de: stud. Rareș-Andrei Cotoi, Facultatea de Matematică și Informatică UBB Cluj-Napoca

O abordare posibilă este de a genera folosind metoda backtracking toate soluțiile obținute prin alocarea energiei în oricare dintre cele N celule, cu respectarea condiției date. O astfel de soluție obține aproximativ 20 de puncte, fiind o soluție viabilă doar în cazul unui N foarte mic.

Pentru a rezolva problema într-un mod eficient, vom folosi o abordare de tip Greedy. Încercăm să identificăm pentru o valoare potențială T, dacă este posibil să distribuim energia suplimentară astfel încât toate secvențele să aibă cel puțin energia T. Dacă putem atinge cel puțin suma T pentru fiecare secvență de lungime L, dar nu putem atinge cel puțin suma $T + 1$, răspunsul final este T. Știind că T este suma elementelor dintr-o secvență de lungime L, remarcăm faptul că $T \in [0, sumaElem + S]$, unde $sumaElem$ este suma tuturor elementelor din celulele inițiale. Așadar, T poate fi determinat optim folosind căutare binară pe intervalul $[0, sumaElem + S]$.

Pentru fiecare sumă-candidat T, procesăm secvențele de lungime L de la stânga spre dreapta, iar pentru fiecare secvență care are nevoie de energie suplimentară (suma secvenței < T), adăugăm energie începând de la cea mai din dreapta celulă a secvenței, apoi continuăm spre stânga. Utilizăm această strategie deoarece o celulă care se află la poziția j afectează toate secvențele care o includ (de la cea care începe la poziția $j - L + 1$, până la cea care începe la poziția j). Astfel, prin adăugarea energiei în celula cea mai din dreapta a unei secvențe, maximizăm numărul de secvențe ulterioare care beneficiază de această creștere. De exemplu, dacă $N = 10$ și $L = 3$:
Pentru a calcula rapid sumele secvențelor inițiale, vom folosi un șir de sume parțiale sum, astfel:

1: $sum[1] \leftarrow \sum_{i=1}^{L} v[i]$
2: **for** $i \leftarrow 2$ to $n - L + 1$ **do**
3: $\quad j \leftarrow i + L - 1$
4: $\quad sum[i] \leftarrow sum[i-1] + v[j] - v[i-1]$
5: **end for**



La momentul parcurgerii secvențelor de lungime L, putem folosi o coadă cu dublu acces (sau o stivă) pentru menținerea pozițiilor i în care mai putem adăuga energie ($v[i] < X$). Pentru o sumă-candidat T, un algoritm de verificare a validității lui T este:

```
1: function putemAtingeEnergia(T)
2:     Inițializăm o coadă cu dublu acces dq cu pozițiile 1 până la L − 1
3:     for i ← 1, L − 1 do
4:         pus[i] ← 0
5:     end for
6:     R ← S                                          ▷ energia rămasă de distribuit
7:     P ← 0                                          ▷ energia suplimentară pe secvența curentă
8:     for st ← 1, n − L + 1 do
9:         dr ← st + L − 1
10:        dq.pushBack(dr)                            ▷ adăugăm poziția dr în dq
11:        P ← P − pus[st − 1]      ▷ elimin energia adăugată la poziția care iese din secvență
12:        if dq.front() = st − 1 then
13:            dq.popFront()                          ▷ eliminăm poziția care iese din secvență
14:        end if
15:        if sum[st] + P < T then
16:            D ← T − P − sum[st]                    ▷ deficitul pentru secvența curentă
17:            while dq.size() > 0 AND D > 0 AND R > 0 do
18:                w ← dq.back()
19:                A ← min(D, min(R, X − pus[w]))     ▷ cantitatea de energie de adăugat
20:                D ← D − A
21:                R ← R − A
22:                P ← P + A
23:                pus[w] ← pus[w] + A
24:                if pus[w] = X then
25:                    dq.popBack()                   ▷ eliminăm poziția dacă am atins limita X
26:                end if
27:            end while
28:            if D > 0 then
29:                return false          ▷ nu putem atinge energia T pentru această secvență
30:            end if
31:        end if
32:    end for
33:    return true                                    ▷ toate secvențele pot avea cel puțin energia T
34: end function
```

Complexitatea totală a algoritmului este $O((N+S) \cdot \log(suma\_inițială + S))$, unde căutarea binară are complexitatea $O(\log(suma\_inițială + S))$ și $O(N+S)$ (în cel mai rău caz) reprezintă timpul de verificare a validității unei sume-candidat.

Echipa

Problemele pentru această etapă au fost pregătite de:

- Prof. Adrian Panaete, Colegiul Național „A.T. Laurian” Botoșani

- Prof. Ciprian Chește, Liceul Tehnologic „Grigore C. Moisil” Buzău
- Instr. Cristian Frâncu, Nerdvana București
- Instr. Cătălin Frâncu, Nerdvana București
- Stud. Andrei Boacă, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza”, Iași
- Prof. Mihai Bunget, Colegiul Național Tudor Vladimirescu, Târgu-Jiu
- Prof. Gheorghe-Eugen Nodea, Centrul Județean de Excelență Gorj, Târgu-Jiu
- Prof. Emanuela Cerchez, Colegiul Național „Emil Racoviță” Iași
- Stud. Alin Răileanu, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza”, Iași
- Stud. Victor Botnaru, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie Politehnica București
- Prof. Ionel-Vasile Piț-Rada, Colegiul Național Traian, Drobeta Turnu Severin
- Stud. Răzvan Alexandru Rotaru, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza”, Iași
- Stud. Rareș-Andrei Cotoi, Universitatea Babeș-Bolyai, Cluj, Facultatea de Matematică și Informatică
- Stud. Giulian Buzatu, Facultatea de Matematică-Informatică, Universitatea București
- Prof. Dan Pracsiu, Liceul Teoretic Emil Racoviță, Vaslui
- Prof. Marinel Șerban, Colegiul Național „Emil Racoviță” Iași
- Stud. Petruț-Rareș Gheorghieș, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie Politehnica București
- Stud. Ioan-Cristian Pop, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie Politehnica București