OLIMPIADA NATIONALA DE INFORMATICA, FAZA JUDETEANA, 2023, CLASELE 11-12

Problema 1: Parcare

Propusă de: stud. Bogdan Sitaru

Pregătită de: stud. Alexandra Nicola, stud. Tudor Magirescu

Subtask 1 (24 puncte). Deoarece fiecare maşină stă exact o secundă, iar timpii de sosire şi plecare sunt distincţi, deducem că fiecare maşină va fi singură în parcare, deci se poate afişa locul 1. Dacă ultima maşină are timpul de plecare cel mult egal cu T atunci se va afişa timpul său de sosire urmat de $N-1$ de valori -1, altfel se vor afişa N de -1.

Complexitate: $O(N+M)$.

Subtask 2 (26 puncte). Se va atribui câte un loc de parcare fiecărei maşini în ordinea sosirii şi se va afişa locul respectiv. După sosirea tuturor maşinilor se vor elibera locurile de parcere ale maşinilor al căror timp de plecare este cel mult T şi se va afişa configuraţia parcării.

Complexitate: $O(N+M)$.

Subtask 3 (26 puncte). Se va procesa fiecare maşină în ordinea sosirii. Mai întâi se vor elibera locurile de parcare ale maşinilor care au timpul de plecare cel mult egal cu timpul actual. Apoi se va căuta secvenţial un loc de parcare pentru maşina tocmai sosită. La final se vor elibera locurile de parcare ale maşinilor cu timpul de plecare mai mic sau egal cu T şi se va afişa configuraţia parcării.

Complexitate: $O(N \cdot M)$.

Subtask 4 (24 puncte). Pentru a şti în fiecare moment dacă există locuri libere în parcare se va folosi o structură de heap (*set*) în care se vor reţine perechi de forma (t, j), unde t reprezintă secunda la care locul j va fi liber. Iniţial în heap toate perechile vor fi de forma $(0, j)$ pentru orice j de la 1 la N. Pentru fiecare maşină sosită la secunda s_i cu plecare la p_i, dacă perechea din heap (t, j), cu t minim, verifică condiţia $t < s_i$, atunci se elimină din heap această pereche, se afişează locul j disponibil pentru maşină şi se adaugă în heap perechea (p_i, j). În caz contrar se va afişa -1 deoarece nu există loc de parcare pentru maşina curentă. La final se eliberează locurile de parcare pentru maşinile care au timpul de plecare cel mult T şi se afişează configuraţia parcării.

Complexitate: $O(M \cdot log(N))$.

Soluție alternativă: Deoarece la fiecare moment de timp se întâmplă maxim un eveniment (venirea/pleacarea unei mașini), putem reține pentru fiecare secundă infomațiile ce descriu acest eveniment. Astfel, pentru fiecare secundă reținem una sau două valori: 0 - niciun eveniment, 1 - venirea unei mașini, 2 x - pleacarea mașinii de pe locul de parcare x. În același timp, vom menține locurile de parcare libere într-o stivă/coadă, pentru a putea găsi in timp constant un loc de parcare liber sau a vedea dacă parcarea este plină. Astfel, putem itera prin secunde de la 1 la T și procesa evenimentele. La venirea unei mașini la timpul s_i care va sta până la timpul p_i, verificăm dacă stiva este goală, caz în care afișăm -1. În caz contrar, vom obține un loc liber de parcare x (din vârful stivei), pe care îl vom elimina din stivă, și îl vom aloca mașinii curente, după care vom updata configurația parcării și informațiile despre evenimentul de plecare al mașinii la timpul p_i. La plecarea unei mașini, vom updata configurația parcării și vom insera noul loc gol de parcare în vârful stivei. După procesarea evenimentelor din intervalul de timp $[1, T]$, afișăm configurația parcării.

Complexitate: $O(N+M)$.

Problema 2: Turcane

Propusă de: prof. Mihai Bunget, drd. Andrei-Costin Constantinescu

Subtask 1 (26 puncte). Avem de calculat numărul minim de sărituri pentru o matrice cu o linie. Dacă $N-1$ este divizibil cu P rezultatul este $\frac{N-1}{P}$, altfel este $\left[\frac{N-1}{P}\right]+1$, unde $[x]$ reprezintă partea întreagă a lui x.

Complexitate: $O(1)$.

Subtask 2 (15 puncte). Avem de calculat numărul minim de sărituri pentru o matrice cu două linii. Pentru aceasta se va determina minimul dintre numărul de sărituri obţinut în fiecare din următoarele situaţii:

- se efectuează o săritură a calului, dacă este posibil, ajungând în pătrăţelul (2,3), după care se aplică formula de la subtask 1;
- se efectuează o săritură pe diagonală, dacă este posibil, pentru N=2;
- se efectuează o săritură pe verticală, apoi pe orizontală, dacă este posibil, pentru N=2.

Complexitate: $O(1)$.

Subtask 3 (7 puncte). Avem de calculat numărul minim de sărituri pentru o matrice cu trei linii. Pentru aceasta se va determina minimul dintre numărul de sărituri obţinut în fiecare din următoarele situaţii:

- se efectuează două sărituri ale calului, dacă este posibil, ajungând în pătrăţelul (3,5), după care se aplică formula de la subtask 1;
- se efectuează una sau două sărituri pe diagonală, dacă este posibil, pentru a ajunge pe linia a treia, apoi se aplică formula de la subtask 1;
- se efectuează una sau două sărituri pe verticală, dacă este posibil, pentru a ajunge pe linia a treia, apoi se aplică formula de la subtask 1;
- se efectuează o săritură a calului combinată cu o săritură pe verticală sau diagonală pentru a ajunge pe linia a treia, apoi se aplică formula de la subtask 1.

Complexitate: $O(1)$.

Subtask 4 (7 puncte). Se foloseşte programarea dinamică. Vom nota cu $d_{i,j}$ numărul minim de sărituri necesare pentru a ajunge în pătrăţelul (i,j). Astfel se iniţializează $d_{1,1}=0$ şi $d_{i,j}=m+n$ pentru orice indici i,j. Se parcurge matricea d şi pentru fiecare poziţie (i,j) se fac actualizările:

- $d_{i,j+k}=min(d_{i,j+k},d_{i,j}+1)$, pentru fiecare $1\leq k\leq P$ cu $j+k\leq N$;
- $d_{i+k,j}=min(d_{i+k,j},d_{i,j}+1)$, pentru fiecare $1\leq k\leq Q$ cu $i+k\leq M$;
- $d_{i+k,j+k}=min(d_{i+k,j+k},d_{i,j}+1)$, pentru fiecare $1\leq k\leq R$ cu $i+k\leq M, j+k\leq N$;
- $d_{i+1,j+2}=min(d_{i+1,j+2},d_{i,j}+1)$ şi $d_{i+2,j+1}=min(d_{i+2,j+1},d_{i,j}+1)$, când indicii nu depăşesc dimensiunile matricei.

Complexitate: $O(M\cdot N\cdot(M+N+min(M,N)))$.

Subtask 5 (8 puncte). Se foloseşte programarea dinamică. Vom nota cu $d_{i,j}$ numărul minim de sărituri necesare pentru a ajunge în pătrăţelul (i,j). Astfel se iniţializează $d_{1,1}=0$ şi $d_{i,j}=m+n$ pentru orice indici i,j. Se parcurge matricea d şi pentru fiecare poziţie (i,j) se fac actualizările:

- $d_{i,j+k}=min(d_{i,j+k},d_{i,j}+1)$, pentru $k=min(P,N-j)$;
- $d_{i+k,j}=min(d_{i+k,j},d_{i,j}+1)$, pentru $k=min(Q,M-i)$;
- $d_{i+k,j+k}=min(d_{i+k,j+k},d_{i,j}+1)$, pentru $k=min(R,M-i,N-j)$;
- $d_{i+1,j+2}=min(d_{i+1,j+2},d_{i,j}+1)$ şi $d_{i+2,j+1}=min(d_{i+2,j+1},d_{i,j}+1)$, când indicii nu depăşesc dimensiunile matricei.

Complexitate: $O(M\cdot N)$.

Soluţie alternativă: Se foloseşte programarea dinamică. Astfel se iniţializează $d_{1,1}=1$ şi $d_{i,j}=m+n$ pentru orice indici i,j în rest. Se parcurge matricea d şi pentru fiecare poziţie (i,j) diferită de poziţia $(1,1)$ se actualizează $d_{i,j}=min(a,b,c,e,f)$, unde $a=min(d_{i,j-k},/1\leq k\leq$

P *cu* $1 \leq j-k$, $b = min(d_{i-k,j}, /1 \leq k \leq Q$ *cu* $1 \leq i-k$, $c = min(d_{i-k,j-k}, /1 \leq k \leq R$ *cu* $1 \leq i-k, j-k$, $e = d_{i-1,j-2}$ şi $f = d_{i-2,j-1}$. Pentru a calcula a, b, c în $O(1)$ se foloseşte *deque* pe fiecare linie, coloană şi diagonală.

Complexitate: $O(M \cdot N)$.

O soluţie asemănătoare, dar care nu trece toate testele, este ca în locul structurii *deque* să folosim *set*.

Complexitate: $O(M \cdot N \cdot log(max(M, N)))$.

Subtask 6 (11 puncte). Se foloseşte metoda backtracking pentru a genera toate drumurile de la pătrăţelul de coordonate $(1, 1)$ la pătrăţelul (M, N).

Complexitate: $O(2^{M+N+min(M,N)})$.

Subtask 7 (12 puncte). Se foloseşte programarea dinamică. Vom nota cu $d_{i,j}$ numărul de moduri în care se poate ajunge în pătrăţelul (i, j). Se iniţializează $d_{1,1} = 1$, se parcurge matricea d şi pentru fiecare poziţie (i, j) se fac actualizările:

- $d_{i,j+k} += d_{i,j}$, pentru fiecare $1 \leq k \leq P$ cu $j + k \leq N$;
- $d_{i+k,j} += d_{i,j}$, pentru fiecare $1 \leq k \leq Q$ cu $i + k \leq M$;
- $d_{i+k,j+k} += d_{i,j}$, pentru fiecare $1 \leq k \leq R$ cu $i + k \leq M, j + k \leq N$;
- $d_{i+1,j+2} += d_{i,j}$ şi $d_{i+2,j+1} += d_{i,j}$, când indicii nu depăşesc dimensiunile matricei.

Complexitate: $O(M \cdot N \cdot (M + N + min(M, N)))$.

Subtask 8 (14 puncte). Se foloseşte programarea dinamică. Vom nota cu $d_{i,j}$ numărul de moduri în care se poate ajunge în pătrăţelul (i, j). Se iniţializează $d_{1,1} = 1$, se parcurge matricea d şi pentru fiecare poziţie (i, j) se fac actualizările:

- $d_{i,j} += \sum d_{i,j-k}$, pentru fiecare $1 \leq k \leq P$ cu $1 \leq j - k$;
- $d_{i,j} += \sum d_{i-k,j}$, pentru fiecare $1 \leq k \leq Q$ cu $1 \leq i - k$;
- $d_{i,j} += \sum d_{i-k,j-k}$, pentru fiecare $1 \leq k \leq R$ cu $1 \leq i - k, 1 \leq j - k$;
- $d_{i,j} += d_{i-2,j-1}$ şi $d_{i,j} += d_{i-1,j-2}$, când indicii nu depăşesc dimensiunile matricei.

Pentru aceasta se vor calcula dinamic sumele parţiale ale elementelor matricei d, pentru fiecare linie, coloană şi diagonală. Complexitate: $O(M \cdot N)$.

Problema 3: Veri

Propusă de: stud. Vlad-Adrian Ulmeanu

Subtask 1 (30 puncte). Tot graful este un ciclu. Primul drum are lungimea n. Următorul drum are lungimea $(n + A - S) \mod n$, analog pentru ultimul drum.

Complexitate: $O(n)$.

Subtask 2 (50 puncte). Observația principală a problemei implică recunoașterea formei drumului optim:

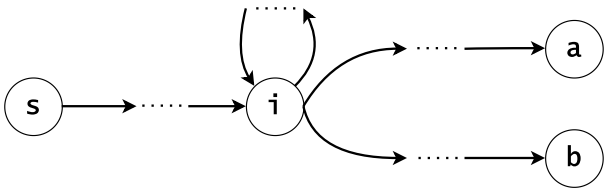

Figure 1. Trebuie să existe un nod i astfel încât drumurile $s..i$, $i..a$ și $i..b$ au lungime minimă și ne putem folosi de cel mai scurt ciclu care începe și se termină în i.

Toate constrângerile de mai sus pot fi rezolvate cu algoritmul Roy-Floyd, cu două mențiuni:

- $d_{i,i}$ va fi folosit pentru ciclul de lungime minimă care începe și se termină în i, $\forall i \in \{1, .., n\}$.
- Pentru reconstruirea oricărui drum, trebuie să reținem o matrice suplimentară p. Dacă am ameliorat lungimea drumului $i..j$ cu ajutorul nodului z, atunci actualizăm $p_{i,j} \leftarrow z$. Când vrem să reconstruim drumul $i..j$ de lungime minimă, construim recursiv și unim drumurile $i..p_{i,j}$ și $p_{i,j}..j$.

Trebuie să reconstruim 4 drumuri, fiecare în $O(n^2)$. Complexitatea totală a algoritmului este $O(n^3)$.

Testele nu sunt grupate, iar o parte din ele din acest subtask au fost create special astfel încât să faciliteze o soluție ce folosește backtracking doar pentru a închide cicluri din S. Acestea au următoarea formă: în lipsa orientărilor pe muchii, graful este un arbore cu 1 sau 2 muchii în plus.

Astfel, orice nod din graf poate fi vizitat de maxim 3 ori într-o aplicare a backtracking-ului, o dată normal și încă de două ori cu ajutorul muchiilor în plus. În condițiile problemei, putem explora maxim $n - 1 + (n - 3 + 1) + (n - 4 + 1) < 3n \in O(n)$ muchii pornind din S.

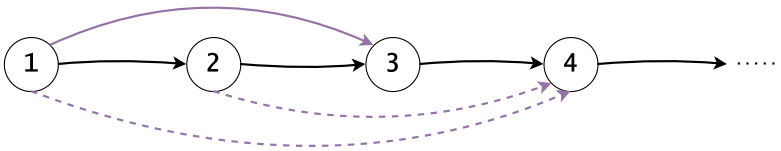

Figure 2. Nu contează dacă adăugăm muchia $1 \to 4$ sau $2 \to 4$, tot maxim $3n - 6$ muchii putem explora din 1.

Precalculăm pentru orice nod x valorile da_x si db_x, reprezentând numărul minim de muchii ce trebuie parcurse din x pentru a ajunge în A sau B (două BFS-uri pe graful transpus din A sau B).

Putem să folosim backtracking-ul pentru a ne plimba până la închiderea unui ciclu. Când am închis un ciclu, știm ce distanță am făcut din S până într-un Z anume (primul drum) și ne

folosim de vectorii *da* și *db* pentru a calcula ce distanță o să aibă celelalte două drumuri.

O soluție optimizată cu ideea de mai sus (tai ramura curentă a backtracking-ului când nu mai pot să scad $\max(t + t_a, t + t_b)$, chiar dacă aș reuși cumva să închid ciclul în nodul curent) poate să ia 62 de puncte.

Subtask 3 (20 puncte). Căutăm ciclul optim (*i..i*) astfel: oricum ar arăta ciclul, există un arbore BFS al grafului astfel încât ciclul să fie reprezentat de un lanț în jos din rădăcină, închis de un back edge înapoi în rădăcină.

Constrângerile ne permit să construim fiecare arbore BFS din graf (adică să facem n BFS-uri, unul pentru fiecare rădăcină posibilă), care vor lua $O(n(n+m)) \subset O(n^2)$, deoarece $m \leq 4n$.

Putem reconstrui orice drum în $O(n)$, deci soluția are complexitatea $O(n^2)$ pentru acest subtask. Pentru celelalte subtaskuri, soluția are complexitatea $O(nm) \subset O(n^3)$, încadrându-se în limita de timp.

Discuție.

- Nu putem să folosim doar arborele BFS cu rădăcina în S și să presupunem că unul dintre ciclurile închise de un back edge va fi parte și dintr-un raspuns corect. Exemplul 2 arată un caz în care orice ciclu corect folosește doar cross edge-uri din perspectiva lui S. O rezolvare simplă consideră toate rădăcinile posibile pentru arborele BFS (de fapt ce apare ca soluție la subtask-ul 3).
- Problema generării de teste pe care să nu treacă soluții cu backtracking (inclusiv cele care se opresc la un procent din TL și afișează cea mai bună soluție găsită) este discutabil mai grea decât rezolvarea problemei în sine.
  - Dacă folosim prea multe muchii, atunci este foarte probabil ca backtracking-ul să găsească în TL un ciclu îndeajuns de mic pentru răspunsul corect.
  - Pentru ultimele două subtaskuri, vrem grafuri care să aibă destule cicluri, toate de lungime relativ mare.
  - De asemenea, am vrea să construim grafuri care să posede natural muchii care ar deveni de tip cross în urma unui BFS din S.
- A fost folositor ultimul exemplu?

Echipa

Problemele pentru această etapă au fost pregătite de:

- Prof. Zoltan Szabo, Inspectoratul Şcolar Judeţean Mureş
- Drd. Andrei Costin Constantinescu, ETH Zurich
- Prof. Mihai Bunget, Colegiul Naţional "Tudor Vladimirescu", Târgu Jiu
- Stud. Bogdan Sitaru, University of Oxford
- Stud. Vlad-Adrian Ulmeanu, Universitatea Politehnica Bucureşti
- Prof. Mariana-Carmen Copilu, Liceul Teoretic "Nicolae Bălcescu", Medgidia
- Stud. Andrei Cotor, Universitatea "Babeş Bolyai", Cluj-Napoca
- Stud. Tudor-Ştefan Măgirescu, DELFT University of Technology, Delft
- Stud. Alexandra-Mihaela Nicola, DELFT University of Technology, Delft
- Stud. Costin-Andrei Oncescu, Harvard University, Cambridge
- Stud. Matei Tinca, Vrije Universiteit Amsterdam