TABĂRA DE PREGĂTIRE A LOTULUI NAȚIONAL DE INFORMATICĂ
ORĂȘTIE, 24-29 MAI 2024
BARAJ 4
DESCRIEREA SOLUȚIILOR

COMISIA ȘTIINȚIFICĂ

Problema 1: Farfurii

Propusă de: Prof. Zoltan Szabo, Inspectoratul Școlar Județean Mureș

Soluția 1 - prof. Zoltan Szabo

Problema *farfurii* este o adaptare a problemei *turnurilor din Hanoi*. În problema clasică toate discurile sunt pe o singură tijă ordonate strict descrescător. La această variantă avem trei deosebiri majore:

- farfuriile nu formează un singur teanc;
- farfuriile nu sunt ordonate;
- farfuriile pot avea dimensiuni egale.

Cazul general se obține prin includerea completă a celor trei condiții de mai sus, însă prin păstrarea doar a câtorva dintre condiții, se obțin câteva probleme particulare cu un nivel de dificultate mai scăzut, aducând un punctaj parțial corespunzător. Menționăm, că pentru fiecare caz, pot exista mai multe moduri de rezolvare, soluția nu este unică. Avem următoarele cazuri:

(1) Toate farfuriile au dimensiuni egale (4 puncte) În acest caz toate farfuriile de pe celelalte două coloane diferite de coloana C se vor muta pe coloana C.
(2) Există doar două dimensiuni diferite de farfurii (11 puncte)
   (a) Toate farfuriile sunt pe o singură coloană (5 puncte) Avem la dispoziție două coloane libere, deci toate elementele le vom separa după dimensiunea lor pe cele două coloane: cele *mici* pe o coloană și cele *mari* pe o altă coloană. Apoi vom construi coloana ordonată de farfurii pe coloana C. În funcție de coloana de destinație, se vor obține trei moduri diferite de mutare a farfuriilor, însă fără a avea un grad de dificultate ridicat.
   (b) Farfuriile sunt dispersate pe mai multe coloane (6 puncte) În acest caz, coloana 1 va fi punctul de colectare a farfuriilor *mici*, iar coloana 2 va fi punctul de colectare a farfuriilor *mari*. În primul pas, toate farfuriile *mici* din vârful coloanei 2 se vor muta pe coloana 1. Astfel am obținut proprietatea, că prima coloană are în vârf o valoare de farfurie *mic*, iar coloana 2 are în vârf o farfurie *mare*. Coloanele 1 și 2 nu sunt ordonate, dar vârful lor permite plasarea tuturor farfuriilor de pe coloana 3 sortate pe cele două mărimi. Astfel putem elibera coloana 3. Urmează să sortăm elementele de pe coloana 2, mutând toate mărimile *mari* pe coloana 3, iar cele mici pe coloana 1. Elementele sunt separate: cele *mici* pe coloana 2 și cele *mari* pe coloana 3. Mutarea lor pe coloana C se va realiza ca la cazul precedent.
(3) Există doar trei dimensiuni de farfurii (12 puncte) În acest caz, coloana 1 va colecta farfuriile *mici*, coloana 2 farfuriile *mijlocii*, iar coloana 3 farfuriile *mari*. În primul pas vom muta eventualele farfurii *mici* din vârful coloanei 2 pe coloana 1. În pasul al doilea vom muta din vârful coloanei 3, toate farfuriile *mici* pe coloana 1, iar farfuriile *mijlocii* pe coloana 2. Am obținut proprietatea ca fiecare dintre cele trei coloane are în vârf: coloana 1 farfurie *mic*, coloana 2 farfurie *mijlocie*, coloana 3 farfurie *mare*. În continuare vom separa elementele neordonate de sub vârf pe cele trei coloane. Începem cu coloana

1. Verificăm care este prima valoare de sub coloana constantă *mic*. Dacă această valoare este *mijlocie*, atunci coloana de valori *mici* se va muta pe coloana 3, iar valoarea *mijlocie* pe coloana 2, iar dacă valoarea este *mare*, atunci coloana de valori *mici* se va muta pe coloana 2, iar valoarea *mare* pe coloana 3. După care coloana de valori *mici* se mută înapoi pe coloana 1. Acest procedeu se va relua până când toate valorile diferite de valoare *mic* vor ajunge pe coloanele 2 sau 3. Cu un algoritm asemănător vom muta valorile *mici* și *mari* de pe coloana 2, apoi vom relua un algoritm asemănător și pentru coloana 3, având în vedere ca niciodată să nu punem farfurii mai mari peste farfurii mai mici.

(4) Toate farfuriile sunt ordonate pe o singură coloană (22 puncte)
   (a) Farfuriile sunt ordonate strict crescător și se vor muta pe o altă coloană (2 puncte) În acest caz, toate elementele, printr-o simplă parcurgere se vor muta pe coloana *C*.
   (b) Farfuriile sunt ordonate strict crescător și se vor muta pe aceeași coloană (4 puncte) Toate elementele, printr-o simplă parcurgere se vor muta pe o altă coloană, apoi cu algoritmul turnurilor din Hanoi se vor remuta farfuriile pe coloana inițială.
   (c) Farfuriile sunt ordonate strict descrescător, și se vor muta pe o altă coloană (4 puncte) În acest caz, se aplică algoritmul turnurilor din Hanoi pentru a muta farfuriile pe coloana *C*.
   (d) Farfuriile sunt ordonate crescător și se vor muta pe o altă coloană (2 puncte) Există și elemente egale. În acest caz, la fel ca la subpunctul 4.*a*., toate elementele, printr-o simplă parcurgere se vor muta pe coloana *C*.
   (e) Farfuriile sunt ordonate crescător și se vor muta pe aceeași coloană (5 puncte) Există și elemente egale. Toate elementele, printr-o simplă parcurgere se vor muta pe o altă coloană, apoi cu algoritmul turnurilor din Hanoi modificat să fie funcționabil și pentru numere egale se vor remuta farfuriile pe coloana inițială.
   (f) Farfuriile sunt ordonate descrescător, și se vor muta pe o altă coloană (5 puncte) În acest caz, se aplică algoritmul turnurilor din Hanoi modificat să fie funcționabil și pentru dimensiuni egale pentru a muta farfuriile pe coloana *C*. Modificarea adusă agoritmului se poate realiza în mai multe feluri, de exemplu, pentru fiecare element folosim o pereche pentru valoare și frecvență, astfel obținem doar elemente învecinate în șir cu valori distincte. O altă modalitate este de a folosi algoritmul turnurilor din Hanoi în care tipărirea se realizează în mod repetat cât timp elementul din vârful stivei este aceeași valoare.

(5) Toate farfuriile formează două coloane crescătoare de la bază spre vârf (14 puncte)
   (a) Farfuriile sunt ordonate strict crescător pe două coloane și se vor muta pe a treia coloană (2 puncte) În acest caz se realizează o interclasare a celor două șiruri.
   (b) Farfuriile sunt ordonate strict crescător pe două coloane și se vor muta pe una dintre coloanele inițiale (4 puncte) Teancul de farfurii se obține prin metoda interclasării pe a treia coloană, apoi cu algoritmul turnurilor din Hanoi se vor remuta farfuriile pe coloana dorită.
   (c) Farfuriile sunt ordonate crescător pe două coloane și se vor muta pe a treia coloană (3 puncte) În acest caz se realizează o interclasare a celor două șiruri, ținând cont că există și elemente egale.
   (d) Farfuriile sunt ordonate crescător pe două coloane și se vor muta pe una dintre coloanele inițiale (5 puncte) Prin metoda interclasării se obține teancul de farfurii pe a treia coloană, apoi cu algoritmul turnurilor din Hanoi adaptat și pentru numere egale se vor remuta farfuriile pe coloana dorită.

(6) Toate farfuriile au dimensiuni diferite (18 puncte) Se referă la cazul general cu toate dimensiunile de farfurii distincte două câte două. Studiem valorile elementelor din vârf. Toate elementele fiind distincte, vom avea câte un element minim, maxim și mijlociu. Știind că cel mai mic element se potrivește peste celelalte două, încercăm să maximizăm adâncimea coloanei care conține vârful minim, având toate elementele mai mici decât

elementul mijlociu. O astfel de coloană respectă proprietățile jocului Hanoi clasic, și vom aplica astfel: - coloana cu vârful minim, fără elementul de la bază se mută pe coloana care conține maximul - elementul de la baza coloanei minimului se mută pe coloana care continue elementul mijlociu. După ce s-au efectuat pașii de mai sus, configurația nouă devine mai "bună" decât configurația anterioară, eliberând un nou element de sub coloana ce a fost mutată, iar elementele mutate păstrează proprietatea Hanoi în continuare. Aceste mutări sunt ireversibile, deci nu putem ajunge la ciclu infinit. Astfel pas cu pas se reconstruiește toată coloana. În cazul în care coloana cu toate elementele s-a format pe o altă tijă decât cea din problemă, mai apelăm încă un Hanoi clasic și problema va fi rezolvată complet.

După cum se știe, algoritmul turnurilor din Hanoi cu n discuri necesită $2^n - 1$ mutări, deci turnul nostru se va construi în cel mai nefavorabil caz în $2^1+2^2+2^3+ ... + 2^n=2^{n+1}$-1 mutari, deci complexitatea algoritmului este $O(2^n)$.

(7) Fără restricții suplimentare (19 puncte) Se referă la cazul general când există dimensiunile de farfurii egale. Algoritmul este similar cu cel descris la punctul 6, aplicând modificarea algoritmului turnurilor din Hanoi pentru numere egale, descrisă la punctul 4.f.

Soluția 2 - stud. Dumitru Ilie

Vom pleca de la explicația soluției optime a problemei turnurilor din Hanoi. Vrem să mutăm cele N discuri de pe turnul A pe turnul C într-un număr minim de mutări. Pentru a face asta va trebui mai întâi să mutăm cele $N-1$ discuri mai mici de pe turnul A pe turnul B, apoi vom muta cel mai mare disc pe turnul C și apoi toate discurile de pe turnul B pe turnul C. Pentru a muta cele $N-1$ discuri vom folosi același raționament, recursiv, mai puțin în cazul în care nu trebuie să mutăm nici un disc ($N = 0$).

Acum să ne întoarcem la problema noastră. Pentru a muta toate discurile, în ordine crescătoare pe un anume turn, să îl numim *target*, vrem să poziționăm cel mai mare disc la baza lui. Întâi trebuie să știm unde se află cel mai mare disc. Avem 2 cazuri:

(1) Discul cel mai mare este pe un alt turn decât *target*. În acest caz vrem să mutăm toate discurile de deasupra lui, și discurile de pe turnul *target* pe cel de-al treilea turn. Apoi vrem să mutăm discul cel mai mare pe turnul *target* și apoi vrem să mutăm toate discurile înapoi pe *target*. Observăm un tipar recursiv.

(2) Discul cel mai mare este pe turnul *target*. În acest caz avem două subcazuri. Fie este deja la baza turnului *target*, fie nu.
   (a) Dacă este la bază putem să îl ignorăm complet (fiind cel mai mare putem pune orice alt disc deasupra lui) și putem rezolva problema pentru $N-1$ discuri.
   (b) Dacă însă nu este la bază, avem sub el discuri mai mici. Ce vom face este să îl mutăm pe unul dintre turnurile auxiliare, urmat de a scoate toate discurile de pe turnul *target* și a le muta pe alt disc auxiliar. Acum putem muta cel mai mare disc pe *target* și apoi toate celelalte peste el. Am presupus totuși că putem muta discul cel mai mare pe un turn auxiliar. Dacă nu îl putem muta instant, vom muta tot ce e deasupra lui și tot ce e pe primul turn auxiliar pe cel de-al doilea turn auxiliar iar acum acesta poate fi mutat.

Avem deci un algoritm recursiv care încearcă să mute mereu discul maxim. Se poate demonstra că în cazul problemei originale, acest algoritm este chiar optim (din punctul de vedere al numărului de mutări). În cazul problemei noastre avem de făcut decizii legate de care turn auxiliar este folosit cu ce scop. Aceste decizii influențează numărul de mutări făcute, dar pentru restricțiile mici ale problemei nu este nevoie să le facem într-un mod inteligent, orice alegere este suficient de bună.

Ce facem dacă avem valori egale?

Aproape nimic diferit ("If it works don't fix it" - citat din popor). Singura diferență constă în alegerea discului maxim, deoarece acum există mai multe "discuri maxime". O soluție care

merge foarte bine este să alegem primul disc maxim găsit, în ordine din vârful turnului spre bază.

Complexitatea finală a acestei soluții este de aproximativ la $O(N \cdot 2^N)$. Probabil că implementări mai rapide există, dar aceasta este suficient de bună.

Pentru cei doritori de provocări puteți încerca să rezolvați optim problema (din punctul de vedere al numărului de mutări) pentru orice caz.

Problema 2: Numbers

Propusă de: prof. Dan Pracsiu, Liceul Teoretic "Emil Racoviță", Vaslui

Subtask 1 - $O(n!)$ - 20p

Deoarece n este foarte mic, atunci punctajul maxim pentru acest subtask se poate obține cu metoda backtracking.

Subtask 2 - $O(n * 2^{15})$ - 80p

Deoarece n este mai mic sau egal cu 200, atunci vom utiliza metoda programării dinamice pe stări exponențiale. Construim pentru aceasta matricea dp cu n linii și 2^{15} coloane în care $dp[i][mask]$ = suma maximă care se poate obține din primele i numere, unele dintre ele fiind rotite cu k cifre, unde valorile k folosite sunt marcate cu 1 în $mask$.

Parcurgem fiecare număr din șirul a și pentru fiecare $a[i]$ avem opțiunea să nu-l rotim, deci îl adunăm la sumă chiar pe $a[i]$, sau să-l rotim cu j poziții, unde j este setat cu 0 în $mask$, caz în care la sumă putem adăuga valoarea lui $a[i]$ rotită la stânga cu j cifre, iar bitul j se setează cu 1 în noua mască. Deci

$dp[i][mask] = max(dp[i-1][mask] + a[i], dp[i-1][mask|2^j] + RotateLeft(a[i], j))$

unde prin $|$ se înțelege operatorul OR pe biți, iar $RotateLeft(a[i], j)$ este funcția care returnează valoarea lui $a[i]$ rotită de j ori.

Soluția problemei se găsește în $max(dp[n][i], i = 1..2^{15} - 1)$

Subtask 3 - 100p

Se observă că sunt în total cel mult 14 numere care se vor roti. În consecință, pentru fiecare $j = 1..14$, vom reține în $L[j]$ cele mai mari 14 numere care se pot obține din șirul de numere prin rotirea cu exact j cifre la stânga. În acest fel, vom putea lua în considerare numai 14 * 14 = 196 de numere. Rețineți faptul că listele $L[j]$, cu $j = 1..14$, nu sunt neapărat disjuncte.

În continuare se poate aplica pentru cele cel mult 196 de numere ideea de la subtaskul 2.

Problema 3: NYSE

Propusă de: Stud. Andrei Onuț, Yale University, S.U.A.

Prima observație.

Putem să cumpărăm și să vindem (în orice ordine) o acțiune dacă: $X \leq Y$, unde X este prețul la care cumpărăm și Y este prețul la care vindem. În urma acestei tranzacții, se obține un profit (net) de $(Y - X) \geq 0$ dolari. Prin urmare, dacă avem două prețuri, notate cu a și b, alegem să cumpărăm la $\min(a, b)$ și să vindem la $\max(a, b)$ – profitul, în această situație, este egal cu $\max(a, b) - \min(a, b)$ dolari.

Soluție pentru primul subtask.

Cum prețul acțiunii este constant pentru întreaga perioadă de N zile ($p_1 = p_2 = \ldots = p_N$), înseamnă că nu se poate obține un profit pozitiv (adică mai mare strict decât 0). Prin urmare, profitul maxim care se poate obține la finalul fiecărei zile i este de 0 (dolari). De vreme ce $i \in \{1, 2, \ldots, n\}$, înseamnă că profitul maxim care se poate obține la finalul primei zile este tot de 0 dolari.

În cadrul fiecărei întrebări, dacă valoarea x citită este egală cu 0, deducem că $j = 1$. Altfel (când $x > 0$), $j = -1$.

A doua observație.

Să presupunem că prețurile unei acțiuni pentru patru zile distincte sunt: a, b, c, respectiv d și că limita de tranzacționare din fiecare zi este egală cu 1. Fără a pierde din generalitate, putem asuma că $a \leq b \leq c \leq d$. Astfel, strategia care rezultă în profitul maxim în acest scenariu este să cumpărăm (câte o acțiune) la prețurile a și b și să vindem (câte o acțiune) la prețurile c și d. Prin urmare, obținem un profit de $(c + d) - (a + b)$ dolari – se cumpără două acțiuni și se vând tot două acțiuni, deci ne-am închis pozițiile.

Prin contradicție, să presupunem că nu aceasta ar fi strategia optimă. Astfel, să presupunem, de exemplu, că strategia optimă ar fi să cumpărăm la prețurile a și c și să vindem la prețurile b și d. Astfel, se obține un profit egal cu $(b + d) - (a + c)$. Am presupus că aceasta este soluția optimă, astfel că $b + d - a - c > c + d - a - b$, dacă folosim informațiile din paragraful precedent. Această inegalitate este echivalentă cu $b > c$, care contrazice $b \leq c$. În mod similar, se pot analiza și restul de patru strategii rămase, obținând, din nou, contradicții pentru fiecare caz analizat. Așadar, strategia descrisă în paragraful anterior este într-adevăr optimă.

Soluție pentru al doilea subtask.

Să notăm cu S_i șirul nevid, indexat începând cu poziția 1, cu i elemente, care reprezintă sortarea în ordine non-descrescătoare a primelor i elemente din șirul p, pentru fiecare zi i: $1 \leq i \leq N$. Astfel: $S_{i,1} \leq S_{i,2} \leq \ldots \leq S_{i,i}$. Având în vedere că, la finalul seriei de tranzacții, dorim să avem un număr egal de acțiuni cumpărate și vândute, să notăm cu $k = \lfloor \frac{i}{2} \rfloor$ – alegem să cumpărăm și să vindem (în orice ordine) de câte k ori.

Aplicând în mod inductiv observația anterioară, este optim să cumpărăm la prețurile: $S_{i,1}, \ldots, S_{i,k}$ și să vindem la prețurile: $S_{i,i-k+1}, \ldots, S_{i,i}$. Să notăm sumele cu $l_{sum}[i] = S_{i,1} + \ldots + S_{i,k}$ și $u_{sum}[i] = S_{i,i-k+1} + \ldots + S_{i,i}$. Prin urmare, profitul maxim $best[i]$ care se poate obține după primele i zile, închizând pozițiile, este egal cu $(u_{sum}[i] - l_{sum}[i])$.

Fiecare element $best[i]$ se poate calcula în complexitatea $O(i \cdot \log(i))$, prin sortarea primelor i elemente. Astfel, complexitatea totală pentru calcularea șirului $best$ este de $O(N^2 \cdot \log(N))$.

Soluție pentru al patrulea subtask.

Întrucât diferența $max(p_i) - min(p_i) \leq 25$, înseamnă că șirul p conține cel mult 26 de valori distincte. Prin urmare, calcularea șirului $best$ poate fi efectuată mai rapid, într-o complexitate liniară de $O(n)$, folosind o tehnică similară, de exemplu, cu Counting sort – se menține un vector de frecvență.

De vreme ce fiecare pereche de tranzacții cumpărare-vânzare, sau vânzare-cumpărare, rezultă într-un profit de cel puțin 0 dolari, înseamnă că $best[1] \leq best[2] \leq \ldots \leq best[N]$. Astfel, în cadrul unei întrebări, determinarea indicelui minim j al unei zile, în cazul în care există, se poate face prin căutare binară pe șirul $best$.

Complexitatea totală este de $O(N + Q \cdot \log(N))$.

Soluție pentru al cincilea subtask.

Similar cu soluția de la subtask-ul anterior, numărul maxim de elemente distincte din șirul p este de 10^5. Așadar, în loc să folosim un vector de frecvență, putem alege să calculăm șirul $best$ folosind Sqrt Decomposition, în complexitatea de timp $O(N \cdot \sqrt{M}) \leq O(N \cdot \sqrt{N})$, după ce am atribuit, în prealabil, fiecărui element din șirul p o valoare din mulțimea $\{1, 2, \ldots, M\}$ – unde M reprezintă numărul de valori distincte din p.

Soluție completă.

Ideea pentru soluția de 100 de puncte este să utilizăm o structură de date ce permite determinarea poziției minime y pentru care suma elementelor pe prefixul $[1..y]$ este mai mare sau egală decât o valoare fixată k, într-o complexitate de timp logaritmică, $O(\log(N))$; în cazul problemei noastre, pentru fiecare i, putem seta $k = \lfloor \frac{L_1+L_2+...+L_i}{2} \rfloor$, iar $y \in \{1, 2, \ldots, M\}$. Astfel, putem utiliza un Arbore de intervale. Căutarea binară pentru găsirea lui y și interogările (de sumă) în cadrul arborelui de intervale pot fi efectuate simultan, vizitând $\Theta(\log(N))$ noduri din arbore, per interogare (cu parametrul k).

Deci, complexitatea totală de timp este de $O(N \cdot \log(N) + Q \cdot \log(N)) \leq O((N + Q) \cdot \log(N))$.

Având în vedere numărul relativ mare de elemente N din șirul p, din motive practice, recomandăm sortarea utilizând o metodă pseudo-liniară, cum ar fi Radix sort.

Echipa

Problemele pentru acest baraj au fost pregătite de:

- Prof. Emanuela Cerchez, Colegiul Național "Emil Racoviță" Iași
- Prof. Adrian Panaete, Colegiul Naţional "August Treboniu Laurian", Botoşani
- Prof. Ciprian Chește, Liceul Tehnologic "Grigore C. Moisil", Buzău
- Prof. Ionel-Vasile Piț-Rada, Colegiul Naţional "Traian", Drobeta Turnu Severin
- Prof. Mihai Bunget, Colegiul Național "Tudor Vladimirescu", Târgu Jiu
- Lector dr. Paul Diac, Facultatea de Informatică, Universitatea "Alexandru Ioan Cuza" Iaşi
- Prof. Dan Pracsiu, Liceul Teoretic "Emil Racoviță", Vaslui
- Prof. Gheorghe Eugen Nodea, Centrul Județean de Excelență Gorj, Târgu Jiu
- Prof. Daniela Elena Lica, Centrul Județean de Excelență Prahova, Ploiești
- Stud. Ioan-Cristian Pop, Universitatea Politehnica București
- Stud. Dumitru Ilie, Facultatea de Matematică-Informatică, Universitatea București
- Stud. Giulian Buzatu, Facultatea de Matematică-Informatică, Universitatea București
- Stud. Mircea Măierean, Facultatea de Matematică și Informatică, Universitatea "Babeș-Bolyai", Cluj-Napoca
- Stud. Bogdan-Ioan Popa, Facultatea de Matematică-Informatică, Universitatea București
- Stud. Andrei Onuț, Yale University, S.U.A.
- Instr. Cristian Frâncu, Nerdvana Education, București
- Prof. Marinel Șerban, Colegiul Național "Emil Racoviță" Iași
- Prof. Zoltan Szabo, Inspectoratul Școlar Județean Mureș