Olimpiada Națională de Informatică, Baraj de selecție a lotului național de juniori Descrierea soluțiilor

Comisia științifică

April 17, 2025

Problema 1. Joc

Propusă de: prof. Emanuela Cerchez, Colegiul Național "Emil Racoviță" Iași

Soluţia 1

Vom reţine elementele triunghiului într-o matrice T cu N linii şi N coloane, deasupra diagonalei principale.

Cerința 1

Aşezarea elevilor pe scaune reprezintă o permutare de ordin N, pe care o vom reconstitui într-un vector *sol* cu N elemente, numerotate de la 1 la N. Pentru a reconstitui permutarea minimă din punct de vedere lexicografic:

- $T[1][N]$ reprezintă poziţia elementului minim (adică 1); deci, vom plasa în permutare elementul 1 pe poziţia $T[1][N]$.
- În stânga elementului 1 se vor afla $T[1][N] - 1$ elemente, iar în dreapta sa celelalte $N - T[1][N]$; pentru a obţine permutarea minimă din punct de vedere lexicografic, vom alege să plasăm în stânga cele mai mici $T[1][N] - 1$ elemente, adică valorile 2, 3, ..., $T[1][N]$, iar în dreapta celelalte.
- Am redus astfel problema la rezolvarea a două subprobleme de acelaşi tip, pe care le vom rezolva recursiv, ele fiind independente.

Putem formula o subproblemă la modul general astfel: "să se reconstituie secvenţa din permutarea *sol* de la poziţia *st*, până la poziţia *dr* (inclusiv), ştiind că în această secvenţă se vor plasa valori distincte cuprinse între valoarea *minim* şi valoarea *maxim*".

Funcţia reconstituire implementează recursiv acest procedeu:

```
void reconstituire(int st, int dr, int minim, int maxim)
{int pozmin;
 if (st<=dr)
     {pozmin=T[st][dr];
      sol[pozmin]=minim;
      reconstituire(st,pozmin-1,minim+1, minim+pozmin-st);
      reconstituire(pozmin+1,dr, minim+pozmin-st+1,maxim);
     }
}
```

Cerința 2

Pentru a număra câte modalităţi de reconstituire a permutării există procedăm într-un mod similar:

- plasăm elementul minim pe poziţia $pozmin = T[st][dr]$;
- au rămas $dr - st$ elemente din care trebuie să alegem $pozmin - st$ elemente pentru a fi plasate pe poziţiile $st \ldots pozmin - 1$; aceasta se poate face în combinări de $dr - st$ luate câte $pozmin - st$ moduri (în dreapta fiind automat plasate elementele rămase)

```
void numarare(int st, int dr)
{
 if (st<dr)
    {
     int pozmin=T[st][dr];
     combinari(dr-st,pozmin-st);
     numarare(st,pozmin-1);
     numarare(pozmin+1,dr);
    }
}
```

Se obţine un produs de combinări care trebuie calculat pe numere mari; pentru aceasta vom reţine într-un vector p descompunerea în factori primi a acestui produs ($p[x]$=puterea factorului prim x în produsul combinărilor).

```
void combinari(int n, int m)
{if (m>0 && m<n)
    {descompunere(n-m+1,n,1);
     descompunere(2,m,-1);    }
}
void descompunere(int st, int dr, int semn)
//descompune in factori primi produsul numerelor naturale de la st la dr
//semn=1 daca produsul este la numarator (puterile factorilor primi se aduna)
//semn=-1 daca produsul este la numitor (puterile factorilor primi se scad)
{int k, d, x;
 for (k=st; k<=dr; k++)
      {for (d=2, x=k; d*d<=x; d++)
            while (x%d==0)
                  {p[d]+=semn;  x/=d;}
       if (x>1) p[x]+=semn;
      }
}
```

Pentru a calcula rezultatul final este suficient să înmulţim la rezultat factorii primi la puterea corespunzătoare, fiind necesară doar o funcţie de înmulţire a unui număr mare cu un număr mic.

Soluţia 2 - prof. Adrian Panaete

Pentru a evita abordarea recursivă de mai sus se poate simula recursivitatea utilizând o stivă în care se vor memora intervalele de care se ocupă fiecare apel recursiv. Obsevăm că în ambele cerințe recursivitatea funcționează în modul următor

- un apel recursiv se va face pe un interval de indici $[st, dr]$;
- apelul izolează o poziție mi a intervalului $[st, dr]$ pe care o tratează individual (în moduri diferite in funcție de cerință)
- se apelează funcția recursivă pe intervalul $[st, mi - 1]$ (dacă acesta nu este vid)
- se apelează funcția recursivă pe intervalul $[mi + 1, dr]$ (dacă acesta nu este vid).

Putem observa că fiecare procesare se referă la intervale de indici $[st, dr]$ și recursivitatea va procesa aceste intervale într-o anumită ordine.

În loc să folosim o funcție recursivă putem folosi o stivă în care să adăugăm și/sau să eliminăm intervale astfel încât să procesăm toate intervale din soluția recursivă, iar procesarea să se realizeze în aceeași ordine ca în soluția recursivă.

Pentru a realiza acest lucru se folosește următorul algoritm:

- Adăugăm în stivă intervalul $[1, n]$.
- Cât timp există elemente în stivă:
  - Scoatem intervalul $[st, dr]$ din vârful stivei.
  - Identificăm și procesăm poziția mi.
  - Adăugăm în stivă intervalul $[mi + 1, dr]$ (dacă nu este vid).
  - Adăugăm în stivă intervalul $[st, mi - 1]$ (dacă nu este vid).

În rest, toate detaliile de implementare sunt la fel ca în soluția recursivă.

Problema 2. succes

Propusă de: stud. Alin Răileanu, Facultatea de Informatică, Universitatea "Alexandru Ioan Cuza" Iași

Pentru început, vom defini funcțiile:

- $\text{sc}(i, j)$ - factorul de succes al secvenței delimitate de indicii i și j
- $\text{insc}(i, j)$ - factorul de insucces al secvenței delimitate de indicii i și j
- $U(i, j)$ - mulțimea obținută prin reuniunea mulțimilor din secvența delimitată de indicii i și j.

Subtask 1 - $1 \leq N \leq 100$ - 12 puncte

Se vor testa toate secvențele șirului de mulțimi prin selectarea în manieră brută a celor două capete i și j $(1 \leq i \leq j \leq N)$, iar apoi prin calcularea celor 2 valori $sc(i, j)$ și $insc(i, j)$.

Cei doi factori pot fi calculați în complexitate timp $O(N \times M)$, iar selectarea tuturor secvențelor în $O(N^2)$.

Complexitatea finală va fi $O(N^3 \times M)$.

Subtask 2 - $100 < N \leq 1000$ - 13 puncte

Similar primului subtask, se vor testa toate secvențele șirului de mulțimi prin selectarea în manieră brută a celor două capete i și j $(1 \leq i \leq j \leq N)$, iar cei 2 factori se pot actualiza în $O(M)$ la fiecare schimbare de capăt dreapta.

Complexitatea finală va fi $O(N^2 \times M)$.

Subtask 3 - Numerele din A și B sunt pozitive - 20 puncte

Pentru acest caz, *sc* și *insc* vor fi crescătoare pentru un capăt stânga setat (1).

Totodată, faptul că cele 2 șiruri au doar numere pozitive garantează că $insc(i, j) \leq insc(i+1, j)$.

Deci, dacă $insc(i, j) \leq K$, atunci și $insc(i + 1, j) \leq K$ (2).

Din (1) și (2) rezultă că putem aplica tehnica **"Two Pointers"** pentru acest caz, optimizând soluția de la subtask-ul 2.

Complexitatea finală va fi $O(N \times M)$.

Subtask 4 - Restricții inițiale - 55 puncte

Spre deosebire de subtask-ul precedent, funcțiile *sc* și *insc* nu mai sunt monotone.

Totuși, funcția U este monotonă pentru un capăt stânga setat, întrucât operația de reuniune este monotonă (1).

Cum cardinalul maxim al unei mulțimi este M și (1), rezultă că funcția U va avea cel mult M schimbări de rezultat pentru un capăt stânga setat.

Cum funcțiile *sc* și *insc* își schimbă rezultatul doar atunci când și funcția U își schimbă rezultatul, o bună optimizare pentru soluția de la subtask-ul 2 este să parcurgem pentru un capăt stânga setat doar capetele dreapta care aduc o schimbare de rezultat pentru funcția U.

Putem observa că pentru un indice i, funcția $U(i, j)$ își modifică rezultatul doar atunci când mulțimea de la poziția j conține un element care nu se găsește în niciuna dintre mulțimile de la i la $j - 1$.

Astfel, putem parcurge mulțimile de la N la 1 și să reținem tabloul *last* cu semnificația:

$$last[x] = \begin{cases} j_{(i \leq j \leq N)}, & j \text{ este cel mai mic indice al unei mulțimi parcurse care conține elementul } x \\ N + 1, & \text{dacă nu există un astfel de indice.} \end{cases}$$

În continuare, la fiecare iterație vom actualiza valorile din *last*, apoi vom parcurge numerele de la 1 la M în ordinea crescătoare a valorilor din *last* și vom testa rezultatele *sc* și *insc*, având în vedere doar stările valide.

Pentru parcurgerea în ordinea dorită, valorile pot fi sortate sau se poate utiliza următorul "trick":

- punem la început în sortare toate valorile care nu se găsesc în mulțimea de la indicele curent, în ordinea în care se găseau în sortarea finală de la indicele precedent
- punem la final în sortare elementele mulțimii de la indicele curent.

Pentru soluția cu sortare, complexitatea timp va fi $O(N \times M \times log(M))$, iar în urma optimizării, complexitatea obținută va fi $O(N \times M)$.

Mai există și soluții $O(N \times M \times log(N))$ bazate pe precalculări și căutare binară pe rezultat.

Problema 3. Vnoroc

Propusă de: stud. Victor Botnaru, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie POLITEHNICA București

Soluţia 1

Observaţii iniţiale:

- Numărul 1 este singurul număr natural care nu are divizori mai mari decât 1, prin urmare toate valorile egale cu 1 vor fi eliminate din șir de la început (şi contorizate ca elemente eliminate).
- Numărul 0 este singurul număr natural care este divizibil cu orice alt număr natural nenul. Vom partiţiona șirul dat în secvenţe de valori care nu conţin zerouri şi vom rezolva problema pentru aceste secvenţe, zerourile fiind „punţi de legătură" între soluţiile obţinute pentru aceste secvenţe (utilizând zerourile vom putea concatena aceste soluţii, inserând zerourile pe poziţiile corespunzătoare).

Vom analiza în continuare secvenţe de valori > 1.

- Dacă secvenţa conține doar cifrele 2, 3, 5, 7 (cifre prime), oricare două elemente diferite nu au divizori comuni diferiți de 1. Numărăm câte cifre de 2, 3, 5 respectiv 7 avem în șirul inițial. Soluția va fi alcătuită din cifra care apare de cele mai multe ori. În caz de egalitate, luăm cifra cea mai mică, pentru a face șirul minim lexicografic.
- Dacă secvenţa conţine doar cifrele 2, 3, 4, 5, 7, soluția problemei nu se schimbă mult, deoarece 4 poate fi asimilat cu 2, în ceea ce priveşte divizorii. Dacă dorim să reconstituim soluţia minim lexicografică, trebuie să adăugăm încă o verificare. În cazul în care pentru o secvenţă dată, numărul maxim de apariţii este deţinut de cifrele 3 la egalitate cu numărul de apariţii ale cifrelor 2 sau 4, atunci trebuie să verificăm dacă primul element par este 2 sau 4: dacă este 2, vom alege să utilizăm cifrele 2 şi 4; dacă primul element este 4, este preferabil să alegem cifrele egale cu 3.
- Dacă secvenţa conține toate cifrele > 1, observăm că cifra 6 este din nou o cifră specială, fiind o punte de legătură între secvenţe de cifre egale cu 3 şi secvenţe de cifre 2 sau 4. Pentru 5 şi 7 vom contoriza numărul de apariţii. Pentru 2, 3, 4, vom partiţiona din nou secvenţa în subsecvenţe delimitate de cifra 6, vom rezolva problema pentru fiecare subsecvenţă şi

vom concatena soluţiile inserând cifrele egale cu 6 pe poziţiile corespunzătoare. La final vom alege, evident, varianta pentru care numărul de apariţii este maxim, dar, din nou, la reconstituirea soluţiei minime din punct de vedere lexicografic trebuie să verificăm dacă numărul maxim de apariţii se obţine pentru cifra 5 la egalitate cu soluţia obţinută pentru cifrele 2, 3, 4, 6. Vom verifica dacă prima valoare care apare în soluţia cu 2, 3, 4, 6 este egală cu 6 (caz în care va fi preferabil să folosim cifrele 5, în soluţia minim lexicografică).

Soluţia 2

Rezolvăm problema prin metoda programării dinamice. Formulăm o subproblemă astfel: „să se determine lungimea celui mai lung subşir care începe la o poziţie mai mare sau egală cu i şi are proprietatea de a fi talisman noros ($1 \leq i \leq N$)".

Vom reţine lungimile în vectorul $d[]$, $d[i]$= lungimea celui mai lung subşir care începe la poziţia i şi are proprietatea de a fi talisman noros ($1 \leq i \leq N$).

În vectorul $next[]$, vom reţine informaţii pentru reconstituirea soluţiei, $next[i]$ = poziţia elementului care urmează după elementul de pe poziţia i, în cel mai lung subşir cu proprietatea de a fi talisman norocos minim lexicografic (sau 0 dacă nu există un astfel de element).

Rezolvăm subproblemele în ordinea descrescătoare a dimensiunii acestora, parcurgând vectorul de la N către 1.

Pentru a calcula cel mai lung subşir cu proprietatea de a fi talisman norocos care începe la poziţia i, încercăm să adăugăm valoarea $v[i]$ unui subșir maximal de la dreapta acestuia; pentru ca acest lucru să fie posibil, $v[i]$ trebuie să aibă un divizor comun > 1 cu elementul de început al subșirului. Cu alte cuvinte, pentru orice $j > i$, cu $(v[i], v[j])$ având un divizor comun > 1:

$d[i] = max(d[i], d[j] + 1)$;

În caz de egalitate, ținem minte în $next[i]$ cea mai mică valoare următoare pentru care subșirul construit cu aceasta să fie maximal. Vom folosi vectorul $next$ pentru a reconstrui soluția minimă lexicografică la final. După finalizarea dinamicii, este suficient să începem de la poziția i pentru care $d[i]$ e maxim și $v[i]$ e minim, să iterăm după regula $i = next[i]$, și să afișăm cifrele de pe toate pozițiile parcurse.

Soluția descrisă mai sus are complexitatea O(N^2), dar poate fi optimizată folosind următoarea observație: este suficient pentru fiecare cifră de la 0 la 7 să ne uităm doar la cea mai din stânga poziție pe care aceasta apare, pentru a găsi subșirul maximal care începe cu valoarea respectivă.

E ușor de văzut că acest fapt este adevărat, deoarece, pentru două poziții a și b, cu $a < b$ și $v[a] = v[b]$, subșirul maximal care începe în a are lungimea cel puțin egală cu cea a subșirului maximal care începe în b, plus 1.

Putem construi vectorul auxiliar $last\_pos$:

$last\_pos[c]$ = ultima poziție găsită în iterație a cifrei c.

Astfel, dinamica devine: $d[i] = max(d[i], d[last\_pos[c]] + 1)$, pentru fiecare c între 0 și 7 cu $(v[i], c)$ având un divizor comun >1;

La final, $last\_pos[v[i]] = i$.

Acum, în loc să facem N iterații pentru fiecare i, vom face maxim 8. Complexitatea finală este O(N).

Echipa

Problemele pentru această etapă au fost pregătite de:

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