Descrierea soluţiilor

Problema 1. mnp

Autor: prof. Adrian Panaete, Colegiul Naţional “A.T.Laurian” Botoşani

Soluția cu backtracking - 20-25 de puncte

Se generează cu un algoritm de tip *backtracking* toate șirurile corecte.

Soluţia cu programare dinamică - 40-50 de puncte

Se construiește tabloul multi-dimensional `dp`, în care

`dp[i][j][k][last]` = numărul de șiruri de lungime `i+j+k` format din `i` litere A, `j` litere B și `k` litere C, ultima literă fiind `last`. Last va fi 0 pentru 'A', 1, pentru 'B', respectiv 2 pentru 'C'.

Inițializare:

```
dp[1][0][0][0] = dp[0][1][0][1] = dp[0][0][1][2] = 1
```

Relaţii de recurență:

```
dp[i][j][k][0] = dp[i][j-1][k][1] + dp[i][j][k-1][2]
```

celelalte două recurențe sunt asemănătoare.

Soluția se găsește în sum(`dp[m][n][p][ch]`, `ch`=0, 1,2)

Complexitate $O(n^3)$

Soluţia O(n) - 100 de puncte

Se observă că dacă am avea doar două litere, de exemplu doar literele A și B (p=0), aceste litere trebuie să alterneze adică am avea doar soluții de forma :

```
ABAB...ABAB cu m=n
BABA...BABA cu m=n
ABAB...BABA cu m=n+1
BABA...ABAB cu n=m+1
```

Mai precis
- dacă `m=n` există două soluții
- dacă `m=n+1` există o soluție
- dacă `n=m+1` există o soluție
- dacă `|m-n|>1` nu exista soluții

Pentru cazul când avem toate cele 3 litere procedăm astfel: împărțim soluțiile în 4 tipuri
1: cele care încep și se termină cu C
2: cele care încep cu C și se termină cu A sau B
3: cele care încep cu A sau B și se termină cu C
4: cele care încep cu A sau B și se termină cu A sau B

Dar se observă că:
- dacă la o soluție tip 2 adăugăm un C la sfârșit, obținem o soluție de tipul 1 pentru numerele `m`, `n`, `p+1`

- dacă la o soluție tip 3 adăugăm un C la început, obținem o soluție de tipul 1 pentru numerele `m`, `n`, `p+1`
- dacă la o soluție de tipul 4 adăugăm cu C la început și un C la sfârșit obținem o soluție de tipul 1 pentru numerele `m`, `n`, `p+2`.

Fie `solC(m, n, p)` = numărul soluțiilor care încep și se termină cu C.

Pe baza observațiilor de mai sus deducem că soluția finală este:

```
solC(m,n,p)+2*solC(m,n,p+1)+solC(m,n,p+2)
```

Am redus problema la problema determinării numărului de soluții de tip 1. Fie pentru această problemă, `p` = numărul de valori C. Se formează `p-1` intervale, între fiecare doi de C consecutivi, acestea trebuie să conțină cel puțin o literă A sau B. Pe fiecare astfel de interval trebuie ca numărul de A să difere de numărul de B cu cel mult o unitate.

- Fie X numărul de intervale în care numărul de A este egal cu numărul de B

- Fie Y numărul de intervale în care există un A în plus

- Fie Z numărul de intervale în care există un B în plus

Pentru X, Y, Z fixate avem următoarele observații:
- $X + Y + Z = p - 1$
- $d = Y - Z = m - n$

Se observă astfel că putem exprima două dintre valorile X, Y, Z în funcție de a treia.

Vom considera, fără a reduce din generalitate că $m \geq n$ (dacă $m < n$, atunci schimbăm între ele m cu n și A cu B). Avem $Y = Z + d$ și $X = p - 1 - Y - Z = p - 1 - d - 2Z$.

Pentru fiecare alegere a lui $Z \geq 0$ procedăm astfel:

Calculăm $Y = Z + d$ și $X = p - 1 - d - 2Z$

Deoarece vrem să obținem $X \geq 0$ avem $2Z \leq p - d - 1 = p + n - m - 1$

Facem observația că dacă $p - d - 1 < 0$ atunci nu avem cum să obținem soluții.

Altfel: alegem o distribuire posibilă a celor X, Y, Z intervale de cele trei tipuri

Avem permutări cu repetiţie deci:

- soluția se înmulţeşte cu

$$\frac{(X+Y+Z)!}{X!\cdot Y!\cdot Z!}$$

Pentru cele X intervale stabilim dacă începem cu AB sau cu BA și punem în fiecare interval primele două litere

- soluția se înmulteşte cu 2^X => am folosit câte un A și câte un B.

Ne rămân m-X de A și n-X de B.

Pentru cele Y intervale punem prima literă care este A.

Ne rămân m-X-Y de A.

Pentru cele Z intervale punem prima literă care este B.

Ne rămân n-X-Z de B

De remarcat că ne-au rămas m-X-Y de A, n-X-Z de B și R = m-X-Y = n-X-Z deoarece m-n=Y-Z

Au mai rămas R valori de A și R valori de B. Acestea pot fi adaugate pe oricare dintre cele p-1 intervale astfel:
- dacă ultima literă a intervalului este A, adăugăm BA
- dacă ultima literă a intervalului este B, adăugăm AB

Astfel pe fiecare interval i dintre cele p-1 vom avea posibilitatea să facem a[i] adaugări. Dar numărul total de adăugări este R.

a[1] + a[2] + … + a[p-1] = R

Numărul de soluții pentru X, Y, Z fixate trebuie înmulțit cu scrieri ale lui R ca sumă de p-1 numere naturale.

Observație: numărul de partiții a unui număr N=R ca sumă de M=p-1 termeni este combinări de N+M-1 luate câte cate N.

- soluția trebuie înmulțită cu combinări(N+M-1, N) unde M=p-1 si N=R=m-X-Y
- pentru fiecare X, Y, Z fixate și determinate în funcție de Z, la soluție trebuie adunat

$$\frac{(X+Y+Z)!}{X!\cdot Y!\cdot Z!}\cdot 2^X\cdot C_{M+N-1}^{N}$$

unde M=p-1 și N=R=m-X-Y=n-X-Z

Complexitatea finală obținută este O(Z) = O((p + n - m - 1) / 2).

Optimizarea complexităţii

Deoarece putem interschimba între ele valorile m, n, p, se poate deduce că pentru a avea soluție pentru un triplet (m, n, p) trebuie verificate simultan condițiile

`p + n ≥ m + 1`

`m + p ≥ n + 1`

`n + p ≥ m + 1`

Se pot reordona valorile m, n, p astfel încât 2Z=n+p-m-1 să fie minim posibil. Mai precis, dacă alegem m≥n, p se obține complexitatea cea mai bună. Aceasta optimizare nu este necesară!

Problema Ostasi

Autor: stud. Rotaru Răzvan-Alexandru, Facultatea de Informatică, Universitatea "Al. I. Cuza" Iaşi

În cadrul acestui editorial vom utiliza notarea **mex**, care reprezintă cel mai mic număr care nu apare într-o mulţime.

Pentru a simplifica enunţul problemei, vom înlocui povestea soldaţilor cu o reprezentare ce implică adăugarea numerelor în cele `n` mulţimi (fiecare mulţime reprezentând o trupă).

Subtask-ul 1 (`n≤1000, q≤1000, 10 puncte`)

Pentru a rezolva acest subtask este suficient să simulăm operaţiile pe rând. Având în vedere că $q \leq 1000$, numărul maxim de elemente adăugate în acelaşi grup este ≤ 1000 deoarece orice comandă de tip `2` adaugă maxim un element într-o mulţime. Utilizând această informaţie, mex-ul maxim al oricărei mulţimi este ≤ 1001, astfel numărul maxim de elemente care trebuie reţinute pentru fiecare mulţime este ≤ 1000 şi are sens să păstrăm numai numerele ≤ 1000 din acea mulţime. Din asta rezultă că numărul maxim de numere care trebuie să fie reţinute pentru toate mulţimile, în total, este $\leq 1000*1000$, cu proprietatea că ele sunt mai mici decât `1000`. Pentru asta vom putea utiliza un vector caracteristic pentru fiecare mulțime, pentru a avea evidența tuturor numerelor relevante din toate mulțimile.

- Pentru a gestiona o interogare de tipul `1` este suficient să parcurgem vectorul caracteristic al mulţimii interogate de la `1` la `1000` şi să afişăm indexul primei poziţii egale cu `0` sau `1001` în cazul în care toate numerele apar cel puţin o dată în mulţimea respectivă (`O(n)`).
- Pentru a gestiona o interogare de tipul `2` parcurgem mulţimile care sunt modificate şi adăugăm elementul corespunzător în mulţime (`O(n)`).

Complexitatea finală este `O(n*q)`.

Subtask-ul 2 (`1000<n≤200000, 1000<q≤200000`, Toate comenzile de tip `2` sunt plasate înaintea comenzilor de tip `1` şi `val=1` pentru toate comenzile de tip `2`, `10 puncte`)

Având în vedere că toate comenzile de tip `2` sunt înaintea celor de tip `1`, ordinea execuţiei lor nu influenţează rezultatul final. Deoarece `val` este egal cu `1`, pentru a adăuga numărul `nr` în mulțimea cu indicele `S` este suficientă comanda "`2 st dr 1`", unde `st = S - nr + 1` și $dr \geq S$.

Pentru fiecare mulțime `i` ($1 \leq i \leq n$) se definește un vector auxiliar V_i astfel încât V_i`[j]` reprezintă numărul de intervale care încep la `j` și îl conțin pe `i`. Diferența dintre V_i și V_{i+1} este rezultată din intervalele care încep la `i+1` (se adaugă) şi celor care se termină la `i` (se scad). Parcurgând mulțimile de la `1` la `n` se obține configurația fiecărui V_i.

Valoarea `mex` a mulțimii `i` este ultima poziție `j` ($j \leq i$) pentru care V_i`[j] = 0`. Pentru a trece eficient de la V_i la V_{i+1} se folosește o metodă inspirată de algoritmul *șmenul lui Mars*: se stochează pentru fiecare capăt poziția de start a intervalului și variabila `val` a comenzii.

Se utilizează două structuri auxiliare: `cnt[i]` reține numărul de puncte de start la poziția `i`, iar `right[i]` stochează punctele de final de la poziția `i`. Astfel, la trecerea de la V_i la V_{i+1} se adaugă `cnt[i+1]` la V_{i+1} pe poziția i+1 iar pentru fiecare punct din `right[i]` se scade `1` din V_i la poziția capătului de început.

Observație: Pentru orice `j` ($1 \leq j \leq i$), avem $0 \leq V_{i+1}[j] \leq V_i[j]$. Dacă V_i`[j]` este `0`, atunci va rămâne `0` și în V_{i+1}, V_{i+2}, etc., ceea ce face ca valorile mex să fie monoton crescătoare (`poz[i] ≤ poz[i+1]`). Se actualizează `poz[i+1]` ca `max(poz[i+1], j)` atunci când o poziție `j` trece de la nenulă în V_i la `0` în V_{i+1}.

Notăm cu `poz[i]` poziția ultimului `0` care se află pe o poziţie mai mică decât `i` pentru toate intervalele care îl conțin pe `i`.

Complexitatea finală este `O(q*log(n))`.

Subtask-ul 3 (`1000<n≤200000, 1000<q≤200000,` Toate comenzile de tip `2` sunt plasate înaintea comenzilor de tip `1`, fără alte restricţii, 20 puncte)

Observaţia esenţială pentru acest subtask este că un interval poate adăuga acelaşi element în aceeaşi mulţime dacă şi numai dacă valoarea st-val este egală între două comenzi de `tip 2`. Din cauza inconsistenței valorilor pe care le poate lua val, metoda anterioară nu mai funcționează, întrucât valorile vectorului poz nu mai sunt monoton crescătoare (adică nu este întotdeauna adevărat că `poz[i] ≤ poz[i+1]`). Din această cauză, vom reține în doi vectori auxiliari: left[i] pentru punctele de start de la poziția i și right[i] pentru punctele de final de la poziția i.

Astfel, la trecerea de la V_i la $V_{(i+1)}$ se procedează astfel: pentru fiecare punct din left[i+1] se adaugă `1` la $V_{(i+1)}$ pe poziția "`capătul de început - val + 1`" (pentru acel interval), iar pentru fiecare punct din right[i] se scade `1` din V_i pe poziția "`capătul de început - val + 1`" (pentru acel interval).

Pentru a face tranziția de la V_i la $V_{(i+1)}$ se va utiliza un arbore indexat binar sau un arbore de intervale, pe care se va efectua o căutare binară pentru a găsi cel mai lung șir de valori nenule din V_i cu capătul drept în `i`. Update-ul se va realiza doar atunci când frecvența unui anumit update trece de la `1` la `0` sau de la `0` la `1`. Vom căuta astfel care este cel mai lung interval care se termină cu `i` și are suma elementelor egală cu lungimea intervalului.

Complexitatea finală este `O(q*log(n))` folosind o metodă eficientă de căutare binară pe arborele indexat binar (https://codeforces.com/blog/entry/61364) sau `O(q*log(n)*log(n))` pentru metoda mai lentă.

Subtask-ul 4 (`1000<n≤200000, 1000<q≤200000`, Pentru toate comenzile de tip `2`, `val=1`, fără alte restricţii, `20 puncte`)

Putem să construim un arbore de intervale de minime în care comenzile de tip `2` reprezintă update-urile, iar comenzile de tip `1` sunt query-urile. Pentru fiecare comandă de forma "`2 st dr val`" se va actualiza valoarea nodului de pe poziția `st` cu maximul dintre vechea valoare a nodului și `dr`. Pentru a găsi mex-ul unei anumite mulțimi cu indexul `S`, vom căuta binar cea mai lungă secvență care are capătul drept egal cu `S` și pentru care minimul din interval este cel puțin `S`.

Complexitatea finală este `O(q*log(n)*log(n))`.

Subtask-ul 5 (`1000<n≤200000, 1000<q≤200000` Se garantează că nu există două comenzi de tipul `2` care să adauge un soldat cu același nivel de experiență în aceeași trupă)

Folosim observațiile de la subtask-ul 3 pentru a determina valorile mex-urilor fiecărei mulțimi `S`, unde `1 ≤ S ≤ n` (folosind căutare binară pe un arbore indexat binar). Parcurgem șirul de interogări în ordine inversă, astfel încât comanda de tipul `2 st dr val` își schimbă sensul: ea scoate elementul `val + (i − st)` din fiecare trupă `i` (`st ≤ i ≤ dr`). Știind că nu există o comandă de tip `2` care să adauge un număr ce apare deja în mulțimea respectivă, suntem convinși că atunci când scoatem un anumit soldat dintr-un anumit grup, parcurgând de la ultima interogare la prima, acel număr nu va fi scos niciodată în viitor.

Practic, acel număr reprezintă o posibilă limită superioară a mex-ului grupului din care este scos. Astfel, mex-ul grupului după eliminarea unui anumit element dintr-un grup este min(vechiul mex și elementul scos).

În acest mod, este suficient să reținem un singur număr pentru fiecare mulțime ca să putem răspunde la toate query-urile. Pentru a efectua comenzile de tip `2` eficient, vom folosi un arbore de intervale de minime lazy, care va reține pentru fiecare interval updatat valoarea minimă a celei mai din stânga mulțimi cuprinse

în interiorul nodului. Astfel, când coborâm în arbore pentru a găsi valoarea mex-ului unei anumite mulţimi, propagăm lazy-ul în ambii fii ai fiecărui nod prin care trecem, în mod asemănător cu o funcţie de gradul `1`.

Complexitatea finală este `O(q*log(n))`.

Subtask-ul 6 (`1000<n≤200000`, `1000<q≤200000`, Comenzile de `tip 2` sunt ordonate crescător după `val`*)*

Observaţia necesară pentru a face acest subtask este că, dacă `val` este crescător, singura modalitate ca un interval să adauge acelaşi element în aceeaşi mulţime este ca `st-val` să fie egal între două comenzi de tipul `2`, adică pentru toate intervalele care au `st-val` comun și `st` este crescător. Aceasta ne permite ca, la citire, în ordinea dată a comenzilor, să facem toate comenzile de tipul `2` să adauge numai elemente unice în mulţimi, salvând capătul dreapta maxim pentru fiecare mulţime de intervale cu `st-val` comun și modificând update-urile (atât intervalul cât și valoarea lui `val`) astfel încât ele să fie disjuncte. După modificarea intervalelor, subtask-ul 6 devine echivalent cu subtask-ul 5.

Complexitatea finală este `O(q*log(n))`.

*Subtask-ul 7 (*Fără restricţii suplimentare*)*

Observaţia esenţială pentru a rezolva acest subtask este că, dacă descompunem intervalele date de comenzile de tip `2` în mai multe intervale disjuncte pentru fiecare `st-val`, numărul maxim de intervale rezultate este mai mic sau egal cu `2*t`, unde `t` este cardinalul mulţimii iniţiale de intervale. Pentru a demonstra această afirmaţie este suficient să sesizăm că numărul de capete disjuncte ale intervalelor iniţiale se vor regăsi în maxim două intervale finale. Din aceasta rezultă că numărul de comenzi de tip `2` finale este `≤ 2*q`. În acest mod, problema devine echivalentă cu subtask-ul `5`.

Complexitatea finală este `O(q*log(n))`.