Descrierea Soluțiilor
Olimpiada Societății pentru Excelență și Performanță în Informatică, Etapa Națională
Clasele XI-XII

1 Problema SaseG

Propunător: Costin-Andrei Oncescu, University of Oxford

Pentru subtaskul 1, trebuie afișat insuși inputul: dacă toate persoanele sunt în final infectate, un scenariu posibil este ca ele erau infectate și inițial, și deci orice om se poate să fi fost infectat la început. Dacă niciun om nu este infectat la final, atunci nimeni nu se poate sa fi fost infectat la început.

Pentru subtaskurile 2 și 6, este suficient să încercăm toate scenariile inițiale posibile (sunt 2^N astfel de scenarii, ce pot fi generate prin backtracking), să verificăm pentru fiecare dacă duce la scenariul din input (simularea celor N întâlniri efectuându-se conform protocolului în $O(M)$), și să stocăm pentru fiecare persoană dacă există vreun scenariu inițial plauzibil în care ea este infectată/neinfectată.

Pentru subtaskurile 3 si 7, putem în continuare să determinăm toate scenariile inițiale posibile, însă de data aceasta trebuie să facem observația cum că o persoană care este la final neinfectată nu poate fi infectată nici la început, și deci setul de persoane infectate inițial va fi mereu inclus în cel de persoane infectate la final. Astfel, numărul de scenarii pe care trebuie sa le incercăm coboară la 2^18.

Fie S_i mulțimea oamenilor care ar fi infectați la final dacă la început doar persoana i ar fi infectată. Aceasta poate fi calculată prin simulare efectivă în $O(N + M)$ pentru fiecare i fixat. Observăm că dacă inițial sunt infectate persoanele $i_1, i_2, \ldots$, atunci la final vor fi infectate exact persoanele din mulțimea $S_{i_1} \cup S_{i_2} \cup \ldots$. Fie S mulțimea persoanelor infectate la final. În mod evident, nu se poate ca o persoana i pentru care $S_i \not\subseteq S$ să fie infectata inițial pentru ca atunci ar exista o persoana $x \in S_i \setminus S$ care este infectată indirect de i dar care trebuie să nu fie infectată la final. Pe de altă parte, scenariul în care toți oamenii i pentru care $S_i \subseteq S$ sunt infectați inițial trebuie sa fie unul valid, pentru că prin construcție cineva care nu este in S nu poate fi infectat, și deci singura problemă ar fi daca exista cineva in S care nu este infectat in acest fel. Dar acesta este scenariul care maximizează infectările, deci daca exista o soluție (cum ne este garantat), aceasta

este una. Astfel, putem deduce că o persoana i se poate să fi fost infectată inițial dacă și numai dacă $S_i \subseteq S$ (și, in particular, scenariul în care toate aceste persoane sunt infectate este unul valid). Putem calula toate mulțimile S_i în $O(N(N+M))$, astfel rezolvând subtaskul 4. În continuare, numim o persoana i infectabilă dacă și numai dacă $S_i \subseteq S$.

Pentru subtaskul 8, este suficient să calculam S_i pentru fiecare om i și cnt_i = numărul de j-uri infectabile astfel încât $i \in S_j$ (numărul de persoane diferite care-l pot infecta indirect pe i). O persoană i se poate să nu fi fost infectată inițial doar dacă aceasta nu este infectabilă sau celelalte persoane infectabile j pot "acoperi" întregul S. Pentru a verifica aceasta, este suficient să vedem dacă există vreun $j \in S_i$ cu $cnt_j = 1$. Asta ar implica că i este indispensabil in infectarea lui j. Putem verifica asta parcurgand S_i, obținând o complexitate totală de $O(N(N+M))$.

Pentru subtaskul 5 putem observa că daca ne uităm la intâlniri de la coada la cap, putem fi pesimiști cu privire la numărul de infectați, întotdeauna presupunând cel mai rău scenariu care sa nu infecteze pe nimeni care nu este infectat la final. Putem menține astfel un vector b_i care este 1 dacă în scenariul nostru curent persoana i este infectată, și 0 altfel. Când procesăm o întalnire (nu uitați! de la coadă la cap) între persoanele x și y, dacă vreuna din acestea știm ca va fi sanătoasă după intalnire, înseamnă ca ambele erau sănătoase înainte, și deci actualizam vectorul b. Dacă ambele erau bolnave nu putem deduce nimic. Observați că, pe de-o parte, suntem pesimiști: dacă putem presupune că cineva e infectat, o facem, dar pe de altă parte, scenariul nostru niciodata nu duce la o persoana infectată care nu este infectată in input (acesta este un invariant menținut). Practic, la finalul aceste procesări (cu complexitate $O(N+M)$), vom avea mulțimea maximala de oameni infectați la început care sa garanteze totuși ca nimeni nu se va infecta dacă nu era infectat în input. Această definiție demonstreaza echivalența cu soluția precedentă și deci determină exact persoanele ce pot fi infectate inițial.

Pentru subtaskul 9, determinăm mai întâi persoanele infectabile in timp liniar urmând soluția subtaskului 5. Apoi încercam sa aflăm pentru fiecare j infectat în final dacă există o unică persoană infectabilă i astfel încât $j \in S_i$. Soluția subtaskului 8 ne spune practic că pentru fiecare astfel de j trebuie marcat acel unic i ca fiind necesar să fie infectat inițial (dacă n-ar fi, atunci nici j n-ar fi infectat la final). Pe de altă parte, acestea vor fi singurele persoane care nu pot fi neinfectate inițial (din logica subtaskului 8). Putem, deci, încerca sa calculăm T_i mulțimea tuturor j-urilor infectabile care îl pot infecta direct sau indirect pe i. La fiecare întâlnire intre x si y, atât T_x cât și T_y devin $T_x \cup T_y$. Inițializarea se face cu $T_i = \{i\}$ pentru fiecare i infectabil și $T_i = \emptyset$ altfel. Pentru a face această soluție sa mearga în timp liniar, observați că este suficient să ținem până la 2 elemente din T_i în loc de întregul T_i pentru că ne interesează doar mulțimile de maxim un element. La o reuniune, putem păstra oricare doua elemente dacă avem de ales (din moment ce orice mulțime care va fi reunită cu aceasta va avea minim 2 elemente si nu ne vor mai interesa elementele în sine, ci doar că nu este doar unul). Astfel, obținem complexitate liniară.

2 Problema Bisortare

Propunător: Adrian Panaete, Colegiul Național August Treboniu Laurian , Botoșani

Vom numi o permutare care îndeplinește condiția de ordonare din enunț pentru o poziție dată K, permutare K-bisortată. Dacă în plus o astfel de permutare se obține cu un număr minim de interschimbări, vom avea o K-bisortare optimă. Cu alte cuvinte valoarea din cerință, $Best_K$, nu este altceva decât numărul minim de interschimbări prin care se poate obține o permutare K-bisortată optimă.

Pentru fiecare valoare $v = p_i$ situată pe o poziție i, calculăm două valori asociate lui v care se vor dovedi extrem de utile în rezolvarea problemei:

- st_v = numărul valorilor mai mici decât v situate la stânga lui v în permutarea inițială (sau echivalent numărul pozițiilor $j < i$ pentru care $p_j < p_i$).
- dr_v = numărul valorilor mai mici decât v situate la dreapta lui v în permutarea inițială (sau echivalent numărul pozițiilor $j > i$ pentru care $p_j < p_i$).

Se observă imediat că $st_v + dr_v = v - 1$ deci pentru o valoare v este suficient să calculăm st_v din care se poate deduce imediat dr_v.

Evident că oricare două valori trebuie interschimbate cel mult o dată. Când interschimbăm două valori trebuie să contorizăm interschimbarea și vom conveni să o contorizăm pentru valoarea mai mare.

Să considerăm o poziție K fixată și q o permutare K-bisortată fixată (dar nu neapărat optimă).

Se observă că pentru orice poziție $i < K$ valoarea $v = q_i$ are toate valorile situate la stânga strict mai mari deci toate valorile strict mai mici decât v vor fi situate la dreapta lui v. Înseamnă că numărul de interschimbări pe care le contorizăm la valoarea v este cel puțin st_v și ne putem asigura că putem atinge acest minim dacă niciodată nu vom interschimba v cu o valoare mai mică și situată în permutarea inițială în dreapta lui v. Analog pentru orice valoare pe o poziție $i > k$ vom contoriza exact dr_v interschimbari.

Am demonstrat astfel că numărul minim de interschimbări este suma ST a valorilor st_v pentru toate valorile situate pe poziții mai mici decât K adunată cu suma DR valorilor dr_v pentru toate valorile situate pe poziții mai mari decât K.

Trebuie să alegem acum valorile puse în stânga poziției K și cele rămase în dreapta lui K astfel încât suma $ST + DR$ să fie minimă. Pentru asta este suficient să observăm că indiferent unde apare o valoare v în soluția finală ea va adăuga $\min(st_v, dr_v)$ la soluție. Suplimentar, dacă o valoare are $st_v \leq dr_v$ dar sunt nevoit să o pun în dreapta mai adug $dr_v - st_v$ la soluție și dacă are $st_v \geq dr_v$ dar sunt nevoit să o pun în stânga mai adug $st_v - dr_v$.

În esență observația principală este că, cu cât diferența $st_v - dr_v$ este mai mică cu atât este mai bine să punem valoarea v pe partea stângă și cu cât este mai mare cu atât e mai bine să o punem pe partea dreaptă. Cum mereu pe pozitia K vom avea valoarea 1 și $st_1 = dr_1 = 0$ este suficient să sortez toate valorile $v \geq 2$ crescător după diferența $st_v - dr_v$ și să mă asigur că primele $K - 1$ (după sortare) vor fi la final în stânga poziției K iar celelalte $N - K$ valori vor fi în dreapta poziției K.

Pentru a rezolva cerința 2 se poate folosi prima cerință. Inițial pentru $K = 1$ soluția este suma tuturor valorilor dr_v. Pentru $K = 2, 3 \ldots N$ se adugă la soluția inițială $st_v - dr_v$ în ordinea sortării cu

semnificația că trecem succesiv pe partea stângă cea mai convenabilă valoare la momentul respectiv.

Observații utile pentru rezolvarea subtask-urilor:

- Pentru a rezolva subtask-urile cu $C = 1, K = 1$ este suficient să afișăm numărul de inversiuni ale permutării inițiale.
- Pentru a rezolva subtask-urile cu $C = 1, K = N$ este suficient să afișăm $\frac{n(n-1)}{2}$ minus numărul de inversiuni ale permutării inițiale.
- Pentru a rezolva subtask-urile cu $N \leq 3000$ este suficient să calculăm st_v direct prin parcurgerea permutării până la poziția valori v și contorizarea valorilor mai mici decât v.
- Pentru a rezolva subtask-urile cu $C = 1$ se poate rezolva cazul $C = 2$.
  Din șirul $Best_1, Best_2, \ldots Best_N$ se selectază valoarea $Best_K$ dorită.
- Pentru a rezolva subtask-urile cu $N \leq 100\ 000$ este necesar să utilizăm o metodă eficientă de calcul a valorilor st_v. O astfel de metodă ar fi să utilizăm un arbore indexat binar în care se adună în ordine crescătoare o unitate la poziția fiecărei valori. Astfel pentru fiecare i valoarea lui st_i este suma parțială până la poziția lui i după care adunăm unitatea la poziția lui i.
- Pentru a rezolva subtask-ul cu $N \leq 8$ este suficientă o soluție de tip backtracking: un BFS pe stări, unde stările sunt reprezentate de toate permutările de lungime N (deci $N!$ stări în total) iar tranzițiile sunt reprezentate de interschimbări în permutare.
- Pentru a rezolva subtask-ul cu $N \leq 18$ este necesară o soluție de tip backtracking mai eficientă: se observă că putem itera valorile din permutare de la N la 1 și la fiecare pas să alegem dacă valorea curentă este dusă la început sau la final folosind interschimbări (deci 2^N stări în total).
- Pentru a rezolva subtask-ul cu $N \leq 60$ este suficientă o dinamică în $O(N^4)$, iar pentru a rezolva subtask-ul cu $N \leq 200$ este suficientă o dinamică în $O(N^3)$. Aceste două soluții sunt lăsate ca exercițiu pentru cititor.

3 Problema Cuvinte

Propunător: Alexandra Udristoiu, Universitatea din Bucuresti

Se observă pentru orice cuvânt i din cele N inițiale, dacă există cel puțin un alt cuvânt j astfel încât j este prefix al lui i, i poate fi eliminat din șirul de cuvinte fără a influența rezultatul. Astfel, dacă un cuvânt c nou format îl are ca prefix pe i, atunci și j va fi un prefix al lui, iar dacă c este un prefix al lui i, atunci c va fi fie un prefix al lui j, fie j va fi un prefix al său, deci cuvântul c nu ar putea fi format nici după eliminarea lui i.

O a doua observație este că putem forma cuvintele în orice ordine, deci pentru a număra mai ușor, vom considera cuvintele în ordine crescătoare după lungimea x_i. Vom calcula num_i, numărul de moduri de a forma al i-lea cuvânt, considerând că am format înaintea lui primele $i - 1$ cuvinte. Rezultatul final va fi produsul num_i, pentru $1 \leq i \leq M$. num_i va fi calculat scăzând din numărul total de cuvinte de lungime x_i formate din primele K litere, care va fi egal cu K^{x_i}, numărul de cuvinte care au ca prefix un cuvânt inițial sau unul din primele $i - 1$ formate și numărul de cuvinte care sunt prefixele unui cuvânt inițial.

Numărul de cuvinte care sunt prefixe ale unui cuvânt inițial se poate precalcula numărând prefixele distincte d_i pentru fiecare lungime posibilă. Pentru subtaskul doi, acest lucru se poate face calculând pentru fiecare cuvânt j de lungime len_j un număr p, reprezentând prefixul de lungime maximă care a apărut deja printre primele $j-1$ cuvinte. Apoi, putem adăuga la numărul de prefixe distincte cele de lungime de la $p+1$ la $len_j - 1$. Prefixul de lungime len_j nu se va număra, pentru a nu scădea de două ori cuvintele care sunt de aceeași lungime cu cel pe care îl formăm.

Se observă că orice cuvânt i format ar putea să aibă ca prefix maxim un cuvânt dintre cele inițiale sau dintre primele $i-1$ formate (dacă ar fi cel puțin două cuvinte, atunci ar fi și ele unul prefixul altuia, ceea ce nu este posibil). Astfel, numărul de cuvinte care mai trebuie scăzut este: $\sum j = 1i - 1K^{x_i - x_j} + \sum j = 1NK^(x_i - len_j)$, dacă $len_j \leq x_i$, unde len_j reprezintă lungimea cuvântului inițial j.

Această soluție are complexitate O($N*S+M*(N+M)$) care obține punctajul pentru subtaskul 2.

Pentru subtaskul 3, vom calcula num_i din num_{i-1}. num_i va fi $(num_{i-1} + d_{x_{i-1}}) * K^{x_i - x_{i-1}} - d_{x_i}$ - numărul de cuvinte care ar fi prefixele unui cuvânt inițial cu $x_{i-1} < len_j \leq x_i$. Dacă sortăm len_i în ordine crescătoare și precalculăm puterile de K, putem scădea direct valorile doar pentru cuvintele care ne interesează, păstrând poziția la fiecare pas poziția ultimului cuvânt cu $len_j \leq x_i$. Această soluție are complexitate O($N * S + N * logN + M$).

Pentru soluția de 100 de puncte, vom elimina cuvintele inițiale care au prefixe și vom calcula d_i folosind un trie, în care vom adăuga inițial toate cuvintele și apoi îl vom parcurge. Astfel, complexitatea devine O($S + N * logN + M$), care poate ajunge la O($S + N + M$) cu ajutorul vectorilor de frecvență pentru len_i.

Echipa Setul de probleme pentru această rundă a fost pregătit de:

- prof. Adrian Panaete, Colegiul "A. T. Laurian", Botosani
- prof. Zoltan Szabó, Liceul Tehnologic "Petru Maior" Reghin / ISJ Mureș Tg. Mureș
- Adrian Emanuel Dicu, student Universitatea Politehnică din București
- Alexa Maria Tudose, student University of Oxford, University College
- Alexandra Maria Udristoiu, student Universitatea din București
- Andrei Constantinescu, student University of Oxford, Balliol College
- Bogdan Ciobanu, software engineer, Hudson River Trading
- Bogdan Sitaru, student University of Oxford, Hertford College
- Costin-Andrei Oncescu, student University of Oxford, St. John's College
- George Chichirim, student University of Oxford, Keble College
- Stelian Chichirim, student Universitatea din București
- Theodor Pierre Moroianu, student Universitatea din București