OLIMPIADA NAȚIONALĂ DE INFORMATICĂ, ETAPA JUDEȚEANĂ
CLASA A VIII-A
DESCRIEREA SOLUȚIILOR

COMISIA ȘTIINȚIFICĂ

Problema 1: Candidați

Propusă de: Csaba-György Pătcaș

Pentru a rezolva prima subproblemă, putem marca într-un tablou unitățile de timp în care un partid a fost la guvernare și într-un alt tablou cele în care un politician a făcut voluntariat. Limitele permit realizarea acestui lucru într-un mod naiv, sau folosind tablouri de diferențe.

Combinând ideea de mai sus cu normalizare ar trebui să se mai poată obține puncte adiționale.

Pentru a putea rezolva problema eficient, trebuie să unim intervalele politicienilor, ceea ce se poate realiza în $O(KlogK)$ pentru un anumit politician. Parcurgem intervalele în ordine crescătoare după capătul stâng. La fiecare pas, ne ținem capetele noului interval format: $[l, r]$, și avem următoarele cazuri:

- Dacă intervalul curent este inclus în $[l, r]$, intervalul format nu se modifică.
- Dacă se intersectează, fără ca intervalul curent să fie inclus în $[l, r]$, atunci capătul drept al intervalului curent devine noul r.
- Altfel, intervalul format până acum nu va mai putea fi extins, deoarece următoarele vor avea capete stânga și mai mari. De aceea, vom păstra intervalul $[l, r]$, iar noul interval format devine intervalul curent.

Cerința 1 se rezolvă în complexitate $O(MK)$ prin sortarea intervalelor și aplicarea tehnicii two pointers.

Pentru a rezolva cerința 2, parcurgem cele trei subpuncte din enunț pentru calcularea preferinței unui politician.

(1) Obținem valoarea p_i prin aplicarea metodei greedy: în fiecare pas alegem dintre intervalele politicienilor cel cu capătul dreapta maxim dintre cele care au capătul stâng mai mic sau egal cu ultimul punct acoperit (adică capătul drept al intervalului precedent ales). Pentru un anumit partid, complexitatea este $O(MK)$, dacă considerăm toate intervalele asociate politicienilor sortate.

O variantă mai ineficientă pentru a obține punctajul aferent subproblemei 3 se bazează pe metoda programării dinamice.

(2) Dacă în pașii precedenți am unit intervalele politicienilor, valoarea q_j va fi numărul de intervale rămase pentru politicianul j minus 1.

(3) Numărul de distribuire a celor p_i puncte de putere în q_j intervale este dată de formula $C^{p_i}_{p_i+q_j-1}$, cunoscută și sub numele de *stars and bars*. Pentru a putea deterimina eficient combinările, precalculăm factorialele modulo $10^9 + 7$ și inverșii lor modulari.

Prin rezolvarea cerințelor 1 și 2, obținem o matrice de $N \times M$ cu preferințele partidelor și o matrice de $M \times N$ cu preferințele politicienilor. Obținerea acestor matrici este cea mai costisitoare parte a problemei, realizarea lor făcându-se în complexitate $O(NMK + MK \log(MK))$. Dacă am rezolvat eficient cele două cerințe aplicând metodele descrise mai sus, pentru 40% din cerința 3 trebuie doar să afișăm suma celor două matrici modulo $10^9 + 7$.

Pentru restul punctelor de la cerința 3 aplicăm următorul algoritm de tip greedy. Pentru matricea corespunzătoare preferințelor partidelor sortăm fiecare linie. Într-o primă fază pentru fiecare partid încearcă să candideze politicianul preferat. Dacă un politican este propus de mai multe partide, el alege partidul pe care îl preferă cel mai mult.

În urma primei faze pot rămâne partide fără candidat. Pe acestea le punem într-o coadă din care pe urmă scoatem pe rând partidele. Pentru partidul actual scos, trecem la următoarea preferință. Dacă politicianul corespunzător nu candidează momentan pentru niciun partid, sau preferă partidul actual mai mult decât cel pentru care candidează momentan, el va candida în continuare pentru partidul actual. În acest caz vechiul partid pentru care a candidat politicianul va fi reintrodus în coadă.

Procedeul se termină când coada devină vidă, ceea ce clar se va întâmpla după cel mult $O(NM)$ pași. Menționăm că se obține o soluție corectă și în cazul în care schimbăm rolul partidelor și al politicienilor în algoritmul de mai sus.

Algoritmul de mai sus a fost descris în 1962 de Gale și Shapley pentru problema mariajului stabil[1].

Problema 2: Minish

Propusă de: Alexandru Ispir

În problema aceasta, se consideră un șir $v_1, \ldots, v_{26}$, inițializat cu 0, pe care putem face următoarele operații:

(i) să setăm $v_i := x$
(ii) să actualizăm $v_i := v_i + v_j$ (unde pentru $i = j$ operația este de fapt $v_i := 2v_i$)

Scopul este să ajungem la un șir țintă, unde numerele sunt cel mult 2^{31}, folosind cel mult o operație de tipul $v_i = x$ și cel mult 330 de operații per total.

Prima observație este să abordăm problema *invers*. (Această metodă este de fapt utilă în foarte multe probleme.) Așadar, vom începe de la șirul $v_1, \ldots, v_{26}$ țintă, și vom încerca să creăm un șir care conține doar un singur element nenul, folosind următoarele operații:

(a) $v_i := v_i - v_j$ (unde $i \neq j$)
(b) $v_i := v_i/2$, cu condiția ca v_i să fie par

Acestea corespund cazurilor $i \neq j$ și $i = j$ din operația de tipul (ii) de mai sus.

Scopul este să reducem la fiecare pas cât mai mult numerele, căci vrem să transformăm toate numerele mai puțin unul egale în 0. Considerând câteva exemple, putem observa că: foarte des va fi bine să scădem din maxim al doilea maxim, sau să împărțim maximul la doi. Această soluție obține în jur de 96 de puncte. Singurul caz neacoperit este atunci când maximul este impar — atunci vom vrea să scădem din el un număr impar. Dacă nu avem un număr impar, trebuie să împărțim un alt număr la 2 până el devine impar. Așadar ajungem la următorul algoritm:

- Cât timp șirul $v_1, \ldots, v_{26}$ conține cel puțin două numere nenule:
  (1) Fie $v_{\max} = \max_i v_i$, $v_{\min} = \min_i v_i$ și $v_{\max 2}$ al doilea cel mai mare număr din șir.
  (2) Dacă $v_{\max 2} \geq v_{\max} \ / \ 2$, aplicăm operația $v_{\max} := v_{\max} - v_{\max 2}$.
  (3) Altfel, dacă $v_{\max}$ e par, îl împărțim la 2.
  (4) Altfel, dacă $v_{\min}$ este impar, aplicăm operația $v_{\max} := v_{\max} - v_{\min}$ și apoi îl împărțim pe $v_{\max}$ la 2.
  (5) Altfel îl împărțim pe $v_{\min}$ la 2.

Se poate observa că, în practică, soluția asta folosește o singură atribuire și aprox. 230 de operații, suficient pentru a obține un punctaj maxim. În continuare va urma demonstrația faptului că soluția aceasta folosește cel mult 330 de operații.

Definim pentru fiecare stare $v_1, \ldots, v_{26}$ un *potențial*, notat cu $\Phi(v_1, \ldots, v_{26})$. Vom defini acest potențial în așa fel încât (i) să fie mereu non-negativ, și (ii) dacă facem t operații, potențialul să

[1] https://en.wikipedia.org/wiki/Stable_marriage_problem

scadă cu cel puțin t. Potențialul pe care îl vom folosi va fi următorul:

$$\Phi(v_1, \ldots, v_{26}) = 2 \cdot \log_2 v_{\max} + \log_2 v_{\min} + \alpha \cdot \ln v_{\max} + \beta \cdot \sum_{i=1}^{26} \ln v_i.$$

În fine, observăm că nu putem face mai multe operații pentru șirul $v_1, \ldots, v_{26}$ decât $\Phi(v_1, \ldots, v_{26})$ — care în cel mai rău caz ia $v_1, \ldots, v_{26} = 2^{31} - 1$, din motiv de monotonicitate, adică

$$3 \cdot 31 + (\alpha + 26 \cdot \beta) \cdot \ln 2^{31}.$$

Vom seta α, β în așa fel încât condițiile necesare pentru Φ să fie satisfăcute, și ca $3 \cdot 31 + (\alpha + 26 \cdot \beta) \cdot \ln 2^{31} \leq 330$ — este suficient ca $\alpha + 26 \cdot \beta \leq 11$.

Să considerăm fiecare caz al algoritmului, și să sintetizăm condițiile aferente asupra lui α, β ca Φ să scadă cu măcar numărul de operații.

- Dacă avem $v_{\max 2} \geq v_{\max}$ / 2, facem o operație: scădem pe $v_{\max 2}$ din $v_{\max}$. În cazul acesta observăm că numărul maxim devine $v_{\max 2}$, și numărul $v_{\max}$ scade cu $v_{\max 2}$. Să presupunem că $v_{\max 2} = x \cdot v_{\max}$, unde $x \in [\frac{1}{2}, 1]$. Considerând doar termenii $\alpha \ln v_{\max}$ și $\beta \cdot \ln v_{\max}$, observăm că Φ scade cu cel puțin

$$-\alpha \cdot \ln x - \beta \cdot \ln(1 - x).$$

Impunem ca $-\alpha \cdot \ln x - \beta \cdot \ln(1 - x) \geq 1$ pentru oricare $x \in [\frac{1}{2}, 1]$.

- Dacă $v_{\max}$ este par, îl împărțim cu 2. În cazul acesta, termenul $2 \cdot \log_2 v_{\max}$ se scade cu cel puțin 1.
- Dacă $v_{\max}$ este impar și $v_{\min}$ este și el impar, atunci îl scădem pe $v_{\min}$ din $v_{\max}$ și apoi îl împărțim pe $v_{\max}$ la 2. Termenul $2 \cdot \log_2 v_{\max}$ se reduce cu cel puțin 2.
- În ultimul caz, dacă $v_{\min}$ este par, îl împărțim la 2 — și termenul $\log_2 v_{\min}$ se scade cu 1.

Așadar, pentru a demonstra corectitudinea, trebuie să găsim $\alpha, \beta \geq 0$ astfel încât $\alpha + 26 \cdot \beta \leq 11$, și $-\alpha \cdot \ln x - \beta \cdot \ln(1 - x) \geq 1$ pentru oricare $x \in [\frac{1}{2}, 1]$. În mod particular, se pot lua $\alpha = 3, \beta = 0.3$.

Problema 3: Șirag

Propusă de: Livia Măgureanu și Tamio-Vesa Nakajima

Notăm cu σ numărul de caractere distincte care apar în șir (care poate fi aproximat la 26).

Se observă că putem varia numărul de caractere distincte care apar în fiecare bucată după taiere și să rezolvăm independent pentru fiecare dintre aceste valori.

Fie d o astfel de valoare ($1 \leq d \leq \sigma$). Pentru $1 \leq i \leq N$, definim dinamica $dp[i]$ = numărul de moduri de a tăia o subsecvență care se termină pe poziția i (și începe oriunde) în bucăți cu d caractere distincte. A se nota că în această dinamică se consideră și subsecvențe netăiate. Prin convenție $dp[0] = 0$.

Pentru $1 \leq i \leq N$, calculăm:

- l_i = cea mai mare poziție astfel încât numărul de caractere distincte din subsecvența $S_{l_i} \ldots S_i$ să fie $> d$
- r_i = cea mai mare poziție astfel încât numărul de caractere distincte din subsecvența $S_{r_i} \ldots S_i$ să fie $\geq d$

Se observă că $dp[i]$ se poate actualiza din oricare $l_i \leq j < r_i$ fie prelungind dinamica din $dp[j]$, fie considerând că începem o subsecvență nouă din poziția $j + 1$. Atfel obținem recurența: $dp[i] = \sum_{j=l_i}^{r_i - 1} dp[j] + (r_i - l_i)$.

Întrucât problema ne cere să numărăm doar cazurile în care tăiem subsecvențele în cel puțin un loc, vrem să creștem răspunsul final doar în cazul în care prelungim un răspund deja existent. Prin urmare, în paralel cu calculul dinamicii, vom calcula răspunsul *ans*, inițial 0. Pentru fiecare $1 \leq i \leq N$ adunăm la răspund doar prima parte a recurenței, deci $ans = ans + \sum_{j=l_i}^{r_i - 1} dp[j]$.

În funcție de modul în care calculăm l_i, r_i și suma din recurență, complexitatea soluției poate varia de la $O(\sigma \cdot N^3)$ până la $O(\sigma \cdot N)$.

De exemplu o implementare naivă care pentru fiecare i caută liniar l_i și r_i și apoi calculează tot liniar suma, va avea complexitate timp de $O(\sigma \cdot N^2)$ – subtask 4. Calculând l_i și r_i cu tehnica two-pointers și suma cu ajutorul unor sume parțiale, obținem o complexitate optimă $O(\sigma \cdot N)$.

Soluție în $O(N^2)$. Calculăm inițial o valoare $t[i][j]$, ce reprezintă numărul de caractere diferite în subsecvența cu indicii $i, \ldots, j$. Algoritmul de calcul este următorul:

(1) Pentru $i = 1, \ldots, n$:
  (a) Fie $f_1, \ldots, f_\sigma$, inițial egale cu *fals*, unde f_x va reprezenta dacă apare caracterul x în subsecvența cu indicii $i, \ldots, j$.
  (b) Fie $v = 0$, numărul de valori diferite în subsecvența cu indicii $i, \ldots, j$.
  (c) For $j = i, \ldots, n$:
    (i) Dacă f_{S_j} este *fals*, o setăm la *adevărat* și incrementăm pe v.
    (ii) Setăm $t[i][j] = v$.

Definim dinamica $d[i][j]$, ce reprezintă "numărul de moduri de a selecta subsecvențe cu j caractere distincte, în subsecvența cu indicii $1, \ldots, i$, conținând neapărat elementul cu indicele i". Observația cheie e dacă noi setăm ultima subsecvență, fie ea cea cu indicii $i, \ldots, j$, atunci ea va influența doar starea $d[j][t[i][j]]$. Așadar, următorul algoritm funcționează.

(1) Pentru $j = 1, \ldots, n$:
  (a) Pentru $i = 1, \ldots, j$:
    (i) Incrementăm pe $d[j][t[i][j]$ cu $d[i-1][t[i][j]]$.

Complexitatea finală este de $O(N^2)$.

Echipa

Problemele pentru această etapă au fost pregătite de:

- Stud. Ariciu Toma, Universitatea Națională de Știință și Tehnologie POLITEHNICA București, Facultatea de Automatică și Calculatoare
- Stud. Dobleagă Alexandru, Constructor University, Bremen, Germania
- Stud. Ispir Alexandru, Universitatea București
- Prof. Lica Daniela, Centrul Județean de Excelență Prahova, Ploiești
- Instr. Măgureanu Livia, Universitatea București
- Drd. Nakajima Tamio-Vesa, Department of Computer Science, University of Oxford
- Stud. Onuț Andrei, Yale University, Statele Unite ale Americii
- Stud. Oprea Mihai-Adrian, ETH Zurich
- Prof. Pascu Olivia-Cătălina, Colegiul Național "Nichita Stănescu", Ploiești
- Prof. Pătcaș Csaba, Universitatea "Babeș-Bolyai", Cluj-Napoca
- Stud. Peticaru Alexandru, Universitatea Națională de Știință și Tehnologie POLITEHNICA București, Facultatea de Automatică și Calculatoare
- Stud. Verzotti Matteo-Alexandru, Universitatea București