DESCRIEREA SOLUȚIILOR
OLIMPIADA NAȚIONALĂ DE INFORMATICĂ
CLASA A XI/XII-A

Problema 1: Lupusor si Mielu

Propusă de: asist. doctorand Andrei-Costin Constantinescu, ETH Zűrich

Primul pas în rezolvarea problemei constă în înțelegerea mai îndeaproape a jocului jucat de către Lupușor și Mielu. Notăm cu $\mathcal{M} \neq \emptyset$ mulțimea indicilor cărților rămase în joc după prima mutare a lui Mielu. De asemenea, notăm cu $a_{\mathcal{M}} = \{a_i \mid i \in M\}$ și $b_{\mathcal{M}} = \{b_i \mid i \in M\}$ mulțimile valorilor a și, respectiv, b, ale cărților rămase în joc. Pentru o mulțime nevidă A, notăm cu $\min A$ și $\max A$ valoarea minimă și, respectiv, maximă, din mulțimea A. În acest caz, avem că:

Observatie 1. Mielu câștigă jocul dacă și numai dacă oricare ar fi $i, j \in \mathcal{M}$ avem $a_i < b_j$. Cu alte cuvinte, dacă și numai daca $\max a_{\mathcal{M}} < \min b_{\mathcal{M}}$.

Demonstrație. Cum Lupușor este viclean, acesta va face orice îi stă în putință să caștige odată ce Mielu a fixat mulțimea $\mathcal{M}$. Dacă există doi indici $i, j \in \mathcal{M}$ astfel încât $a_i > b_j$, atunci Lupușor îi poate alege și să câștige, Mielu pierzând. Observați că în acest caz avem $\max a_{\mathcal{M}} > \min b_{\mathcal{M}}$. Altfel, dacă $a_i < b_j$ oricare ar fi $i, j \in \mathcal{M}$, Lupușor va pierde, indiferent ce valori $i, j \in \mathcal{M}$ alege, deci Mielu câștigă. Observați că acest lucru se va întampla dacă și numai dacă $\max a_{\mathcal{M}} < \min b_{\mathcal{M}}$. □

Știind aceasta, putem face o a doua observație, destul de naturală la acest moment:

Observatie 2. Dacă Mielu alege $\mathcal{M}$ astfel încât în joc să rămână o carte $i \in \mathcal{M}$ cu $a_i > b_i$, atunci Mielu pierde.

Demonstrație. În acest caz Lupușor poate alege $i = j$ și să obțină $a_i > b_j$. □

Prin urmare, cărțile cu $a_i > b_i$ sunt în mare parte imateriale pentru rezolvarea problemei, ele trebuind mereu să fie eliminate pentru ca Mielu să aibă o șansă reală de a câștiga. Pentru simplitate, vom presupune în cele ce urmează $a_i < b_i$ pentru toate cărțile, atât la început, cât și după fiecare modificare dispusă de director. Tratarea cărților care au $a_i > b_i$ necesită numai modificări minore.

Date fiind acestea, cerința 1 a problemei cere să aflăm cardinalul maxim al unei mulțimi $\mathcal{M} \neq \emptyset$ cu proprietatea că $\max a_{\mathcal{M}} < \min b_{\mathcal{M}}$ (sau -1 dacă nu există), iar cerința 2 cere numărul de astfel de mulțimi. Pentru ambele cerințe poate fi util să privim perechile $(a_i, b_i)_{1 \leq i \leq N}$ drept intervale pe axa $\mathbb{R}$, însă acest lucru nu este neapărat necesar pentru rezolvarea problemei. Totuși, pe aceste considerente, vom numi numerele $(a_i)_{1 \leq i \leq N}$ *capete stânga*, iar numerele $(b_i)_{1 \leq i \leq N}$ *capete dreapta*, deoarece ele sunt capetele stânga și, respectiv, dreapta ale intervalelor închise $[a_i, b_i]_{1 \leq i \leq N}$. Amintim că $a_i < b_i$, deci intervalele sunt bine definite.

Cerința 1. Vom face următoarea observație, argumentabil mai puțin directă de această dată:

Observatie 3. Pentru o mulțime M avem că $\max a_{\mathcal{M}} < \min b_{\mathcal{M}}$ dacă și numai dacă intervalele închise $[a_i, b_i]_{i \in \mathcal{M}}$ au intersecția nevidă.

Demonstrație. Observația este o consecință directă a algoritmului de calcul a intersecției unor intervale pe axa $\mathbb{R}$. Mai exact, algoritmul calculează capătul stâng al intersecției ca fiind $\max a_{\mathcal{M}}$, iar capătul drept ca fiind $\min b_{\mathcal{M}}$. Excepție face cazul când $\max a_{\mathcal{M}} > \min b_{\mathcal{M}}$, caz în care intersecția este vidă. Amintim că în cele de mai sus capete de intervale sunt disjuncte două câte două. □

O altă observație utilă este următoarea:

Observatie 4. O mulțime de intervale $[a_i, b_i]_{i \in \mathcal{M}}$ se intersectează dacă și numai dacă există un număr întreg x astfel încat $a_i \leq x \leq b_i$ oricare ar fi $i \in \mathcal{M}$.

Demonstrație. Definitia intersecției este echivalentă cu existența unui număr real x care respecta proprietatea. Deoarece capetele intervalelor sunt numere intregi, dacă numarul x respectă proprietatea, atunci și numărul $\lfloor x \rfloor$ respectă proprietatea, de unde rezultă concluzia. □

Definim șirul $s_x = |\{i : 1 \leq i \leq N \text{ si } a_i \leq x \leq b_i\}|$, reprezentând pentru fiecare numar x câte intervale din cele N conțin x. Observațiile 3 și ne dau acum următoarea observație:

Observatie 5. Mărimea celei mai mari mulțimi $\mathcal{M}$ astfel încat Mielu să câștige este dată de maximul din șirul s. Mai exact, dacă maximul s_x se atinge pentru o valoare x, atunci mulțimea $\mathcal{M}$ este determinată de intervalele ce conțin valoarea x.

Astfel, problema se poate rezolva în următorul mod:

(1) Se calculează șirul s înainte de orice modificari. Acest lucru se poate face în timp liniar folosind Smenul lui Mars.
(2) Se calculează maximul din șirul s pentru a obține răspunsul înainte de modificări.
(3) Pentru fiecare modificare care elimină un interval $[a, b]$ și adaugă în loc un interval $[c, d]$:
   (a) Se scade 1 din valorile $s_a, s_{a+1}, \ldots, s_b$.
   (b) Se adună 1 la valorile $s_c, s_{c+1}, \ldots, s_d$.
   (c) Se recalculează maximul din șirul s pentru a răspunde din nou la întrebare.

Aceste operații se pot implementa eficient folosind arbori de intervale,[1] folosind tehnica „lazy update".[2] Arborii de intervale se pot implementa atât recursiv cât și iterativ. Complexitatea soluției este $O((N+M)\log(N+M))$ și este suficientă pentru a rezolva toate testele de evaluare cu $C = 1$.

Cerința 2. Și pentru această cerință vom privi cărțile drept intervale $[a_i, b_i]_{1 \leq i \leq N}$. Observațiile făcute anterior conduc la următoarea:

Observatie 6. Numărul de mulțimi $\mathcal{M}$ pentru care Mielu castigă este determinat de numărul de submulțimi nevide ale lui $\{1, 2, \ldots, N\}$ care au intersecția intervalelor asociate nevidă.

Din păcate, nu este suficient să numărăm pentru fiecare număr întreg x câte submulțimi de intervale îl conțin pe x, deoarece pentru o submulțime pot exista mai multe valori x conținute în intersecția intervalelor. Din acest motiv, este nevoie de un truc: dorim ca pentru fiecare valoare x să numărăm câte submulțimi de intervale îl conțin pe x și au intersecția un interval de forma $[x, x']$, unde $x < x'$ si x' poate fi arbitrar. Cu alte cuvinte, dorim să numărăm pentru fiecare x câte submulțimi de intervale respectă $x = \max a_{\mathcal{M}} < \min b_{\mathcal{M}}$. Remarcăm următoarele:

- Dacă nu există nici un interval cu capatul stânga egal cu x, atunci răspunsul pentru acest x este 0.
- Dacă un astfel de interval există, fie indicele său $\mathcal{I}$, atunci el este unic, și trebuie obligatoriu să facă parte din submulțimea $\mathcal{M}$ pentru ca să avem $x = \max a_{\mathcal{M}} < \min b_{\mathcal{M}}$. Acest lucru rezultă deoarece capetele intervalelor sunt distincte două cate două.
- Odată presupus că $\mathcal{I} \in \mathcal{M}$, condiția $x = \max a_{\mathcal{M}} < \min b_{\mathcal{M}}$ se poate rescrie astfel: oricare ar fi $i \in \mathcal{M} \setminus \{\mathcal{I}\}$ trebuie să avem $a_i < a_{\mathcal{I}} < b_i$, unde amintim că $a_{\mathcal{I}} = x$. Cu alte cuvinte, în afară de intervalul $\mathcal{I}$, celelalte intervale din $\mathcal{M}$ trebuie să conțină valoarea x.
- Notând cu S_x mulțimea intervalelor ce respectă $a_i < x < b_i$ observăm că fiecare astfel de interval poate fi luat în multimea $\mathcal{M}$ independent de celelalte, deci în total sunt $2^{|S_x|-1} = 2^{s_x-1}$ modalități de a alege submulțimea $\mathcal{M}$ astfel încat $\max a_{\mathcal{M}} = x$. Valoarea -1 din exponent corespunde cu tratementul special aplicat intervalului $\mathcal{I}$.

Date fiind acestea, o primă soluție, în complexitate pătratică, este prezentată în cele ce urmează. Soluția, implementată corect, ar trebui să obțină toate mai puțin ultimele 20 de puncte asociate cerinței. După fiecare modificare (și înainte de modificări) se procedează astfel:

[1] www.infoarena.ro/problema/arbint, infoarena.ro/arbori-de-intervale
[2] https://codeforces.com/blog/entry/18051

- Se calculează valorile s_x folosind Șmenul lui Mars.
- Pentru fiecare valoare x se calculează g_x ca fiind 1 dacă există un (unic) interval $[a_i, b_i]$ respectând $a_i = x$, și 0 altfel.
- Răspunsul la întrebare se calculează ca fiind $\sum_x g_x 2^{s_x - 1} = \frac{1}{2} \sum_x g_x 2^{s_x}$, în timp liniar.

Pentru a optimiza soluția, în primul rând observăm că valorile g_x se schimbă în maxim două locuri după fiecare modificare, deci ele pot fi menținute cu efort constant la fiecare modificare. De asemenea, este suficient să lucrăm cu valorile $p_x = 2^{s_x}$, după cum se va vedea în cele ce urmează, lucru care duce la formula mai simplă de calcul a răspunsului $\frac{1}{2} \sum_x g_x p_x$.

Date fiind acestea, putem reformula soluția anterioară în felul următor:

(1) Se calculează șirul p_x înainte de orice modificări, în timp liniar. Acest lucru se poate face tot cu Șmenul lui Mars dacă schimbăm operațiile de $+1$ și -1 cu $\times 2$ și $/2$. Amintim că operația de împărțire necesită cunoașterea conceptului de invers modular.[3]
(2) Se calculează răspunsul înainte de orice modificări ca $\frac{1}{2} \sum_x g_x p_x$.
(3) Pentru fiecare modificare care elimină un interval $[a, b]$ și adaugă în loc un interval $[c, d]$:
   (a) Se setează pe 0 valoarea g_a și pe 1 valoarea g_c.
   (b) Se împart la 2 valorile $p_a, p_{a+1}, \ldots, p_b$.
   (c) Se înmulțesc cu 2 valorile $p_c, p_{c+1}, \ldots, p_d$.
   (d) Se recalculează răspunsul ca fiind $\frac{1}{2} \sum_x g_x p_x$.

Și în acest caz operațiile de mai sus se pretează la un atac cu arbori de intervale folosind tehnica lazy update. Complexitatea timp este,și de această dată, $O((N+M)\log(N+M))$, suficientă pentru punctaj maxim pe testele de evaluare cu $C = 2$. Un detaliu de implementare inedit necesar funcționării acestei abordări este menținerea unei variabile reprezentând suma $\sum_{x=\ell}^{r} g_x p_x$ pentru fiecare nod reprezentând un interval $[\ell, r]$ al arborelui de intervale.

Problema 2: Schema și Investițiile

Propusă de: student Matei Tinca, Vrije Universiteit Amsterdam

Subtask 1: Ordinea proiectelor dată în fișierul de intrare este cea optimă. Putem simula efectiv investițiile pe care le face Dorel. Le parcurgem pe rând, iar când a_i este mai mic sau egal decât G, putem să scădem din G pe a_i.

Subtask 2: $N \leq 7$. Putem aplica un algoritm de generat toate permutările posibile. Astfel, pentru fiecare permutare aplicăm algoritmul descris în subtask-ul 1 și luăm soluția maximă. Pentru această soluție este important în implementare cum se generează permutările. Acestea trebuie generate în $O(N!)$, nu în $O(N^N)$

Subtask 3: Șirul a este format din două valori distincte care se repetă, în orice ordine. Putem să ne fixăm în câte proiecte investește Dorel din prima valoare și în câte proiecte investește Dorel din a doua valoare. După ce scădem din G suma investită, trebuie să verificăm să nu investim cu banii rămași în proiectele rămase. Practic, trebuie să vedem dacă $G_{rămas} < x$ dacă nu a investit în toate proiectele cu valoare x și dacă $G_{rămas} < y$ dacă nu a investit în toate proiectele cu valoare y.

Subtask 4: $N \leq 80$. Din subtaskul anterior, putem face următoarea observație: fie m minimul dintre a_i specifice pentru proiectele în care nu investim. Atunci răspunsul va fi mai mic decât m. Presupunând că această condiție nu se respectă, atunci noi vom investi automat în unul din proiectele în care am ales să nu investim și ajungem la o contradicție.

Astfel, putem să folosim următorul raționament pentru a afla răspunsul: ne fixăm indicele proiectului cu costul cel mai mic în care nu investim. Astfel, știm că vom investi în toate proiectele cu cost mai mic decât cel fixat, iar din proiectele cu cost mai mare, noi trebuie să alegem pentru fiecare dacă investim în el sau nu, astfel încât la sfârșit să rămânem cu o sumă de bani mai mare sau egală cu 0 și mai mică strict decât costul proiectului fixat. Suma rămasă la final va fi un posibil răspuns. Astfel, noi vom alege maximul dintre toate răspunsurile posibile.

[3] Inversul modular al lui 2 este 500000004.

Astfel, o soluție pe care o putem face este să fixăm fiecare proiect ca fiind minim. Din G scădem toate proiectele cu cost mai mic. Pentru cele mai mari, putem folosi tehnica programării dinamice pentru problema rucsacului. Dinamica ne spune pentru fiecare sumă S_{mari} dacă poate fi obținută ca suma unei submulțimi de elemente. Astfel, putem itera prin toate valorile lui S_{mari} și să verificăm dacă se respectă condiția $G - S_{mici} - S_{mari} < minim$.

În soluția acestei probleme, fixăm fiecare proiect ca fiind minim, și după aplicăm rucsac pe proiectele cu cost mai mare. Complexitatea acestei soluții va fi $O(NG)$.

Un caz particular la care trebuie să avem grijă este faptul că uneori nu putem fixa minimul acela. Asta înseamnă că investim automat în toate proiectele, și trebuie să afișăm $G - a_1 - a_2 - \ldots - a_N$.

Subtask 5: $N \leq 2000$ **și** $0 \leq a_1 + a_2 + \ldots + a_N \leq 150$**.** Putem folosi un algoritm de backtracking similar cu soluția de la subtask-ul 3. Pentru fiecare valoare distinctă, alegem în câte proiecte investim, iar la sfârșit verificăm dacă suma rămasă este mai mică decât proiectele în care nu investim.

Complexitatea acestei soluții va fi $O((f_1+1)(f_2+1)(f_3+1)\ldots(f_N+1))$ unde f_i reprezintă câte proiecte există care au costul i. Putem observa că acest produs este suficient de mic pentru a intra în limita de timp.

Subtask 6: Fără restricții suplimentare. Observăm faptul că la soluția de la subtask-ul 4 este ineficient faptul că aplicăm rucsac de N ori. Putem sorta elementele descrescător și să aplicăm rucsacul o singură dată. Fixăm indicele i ca fiind minimul în care nu investim. Astfel, toate elementele din stânga lui i sunt mai mari, deoarece am sortat vectorul descrescător. Așa că dacă ne facem dinamica pe prefixe de elemente, noi am calculat pentru fiecare i toate aranjamentele posibile ale elementelor mai mari decât el.

Complexitatea acestei soluții va fi $O(NG + N\log N)$.

Bonus. Aflați profitul maxim pe care îl poate obține Dorel dacă poate să își aleagă o submulțime de proiecte pe care le poate ordona cum vrea. Restricțiile problemei sunt cele din problema originală.

Problema 3: Regate si Alianțe

Propusă de: student Stelian Chichirim, Universitatea din București

În această problemă se dă un graf cu N noduri și M muchii bidirecționale, unde fiecare muchie i, $1 \leq i \leq M$, are un cost c_i, iar fiecare nod j, $1 \leq j \leq N$, are un cost r_j. Costul ca un nod X să intre într-o alianță cu un nod Y, este minimul dintre r_X și, costul c_i al muchiei i, unde $1 \leq i \leq M$, astfel încât dacă muchia i este ștearsș din graf, nodul X nu mai este în aceeași componentă conexă cu nodul Y.

În alte cuvinte, $Cost(X, Y)$, este minimul dintre r_X și costul muchiilor critice care ne despart nodul X de nodul Y când muchia respectivă este ștearsă din graf.

Primul pas în rezolvarea problemei este să o reducem de la a o rezolva pe graf în a o rezolva pe arbore. Vom identifica fiecare muchie critică din graf (i.e. o muchie care, dacă este ștearsă din graf, numărul de componente conexe ale grafului rezultat crește cu unu). După ce am identificat aceste muchii critice, le ștergem din graf, și calculăm componentele conexe din acest graf rezultat. Apoi, compresăm fiecare componentă conexă într-un nod și fixăm ponderea p al acestui nod ca fiind egală cu numărul de noduri din componenta conexă respectivă. Vom trage o muchie i de cost c_i între componenta compresată A si componenta compresată B, dacă exista un nod a din componenta conexă reprezentată de A și un nod b din componenta conexă reprezentată de B, astfel încât muchia i unește nodurile a și b.

Graful indus de aceste componente conexe compresate este *arbore*. Acest arbore se mai numește și *bridge tree* și poate fi construit în complexitate timp $O(N + M)$.

Am redus problema la: Se dă un arbore cu N noduri, unde fiecare muchie i, $1 \leq i \leq N-1$, are un cost c_i și fiecare nod j, $1 \leq j \leq N$, are o pondere p_j. Notând cu $m_{x,y}$ valoarea minimă a oricărei muchii pe lanțul simplu din arbore care unește x cu y, pentru fiecare nod k din graful

inițial al problemei, astfel încât x_k este nodul compresat din arbore din care k face parte, costul ca acesta să fie într-o alianță perfectă este:

$$\left(\sum_{nod=1}^{N} p_{nod} \min(r_k, m_{nod,x_k})\right) - r_k.$$

Următorul pas al problemei este să scăpăm de valorile de pe noduri, respectiv șirul r. Pentru aceasta, în graful inițial, pentru fiecare nod i, adăugăm nodul i' cu ponderea p egală cu 0, și muchia (i, i') de cost r_i. Vom face arborele compresat pe acest nou graf, iar componentele conexe compresate formate din nodurile adăugate, i', vor avea ponderea p egală cu 0. Utilizând acest arbore, pentru fiecare nod k din graful inițial al problemei, astfel încât k' este nodul compresat din arbore ce conține nodul k' adăugat anterior, costul ca acesta să fie într-o alianță perfectă este:

$$\left(\sum_{nod=1}^{N} p_{nod} \min(m_{nod,k'})\right) - r_k.$$

Pentru a calcula această sumă pentru fiecare nod din arbore, vom sorta muchiile arborelui în ordine descrescătoare. Apoi, vom menține o pădure de mulțimi disjuncte (DSU) pentru nodurile arborelui și vom face sume parțiale ("șmenul lui Mars") pe arborele pădurilor.

Deci, parcurgem muchiile în ordine descrescătoare, fie c costul muchiei curente, și când trebuie să unim nodul x de nodul y, dacă acestea nu sunt în aceeași componenta în DSU, luăm rădăcina fiecărei componente. Vom nota cu X rădăcina componentei din care face parte x și cu Y rădăcina componentei din care face parte y.

Putem observa că, pentru toate nodurile din componenta lui X trebuie să adăugăm valoarea $c \cdot cnt_Y$, iar, pentru toate nodurile din componenta lui Y trebuie să adăugăm valoarea $c \cdot cnt_X$; unde cnt_X este suma ponderilor nodurilor din componenta lui X. Pentru a face asta, fără pierderea generalității, presupunem că Y va deveni tatăl lui X în DSU. Adăugăm valoarea $c \cdot cnt_X$ în Y, iar în X, adăugăm $c \cdot cnt_Y$ și scădem valoarea curentă din Y. La final, propagăm fiecare valoare din arborele DSU în subarborele acestuia. Răspunsul pentru nodul i, va fi valoarea din arborele DSU a nodului i' minus r_i.

Complexitatea de timp finală este $O((N+M)\log(N+M))$, dată de sortarea valorilor muchiilor. În afară de sortare, complexitatea timp este $O((N+M)\log^*(N+M))$.

Subtask 1: $N \leq 2000$ **și** $M = N-1$**, iar regatele și muchiile vor forma un lanț, în ordinea** $1, 2, \ldots, N$**.** Putem observa că în acest caz avem un arbore, mai exact, doar un lanț (în formula de mai sus $p_{nod} = 1$, pentru orice nod).

Pentru a calcula suma descrisă mai sus, ne putem fixa un nod și iterăm prin toate subsecvențele care încep in acest nod și toate subsecvențele care se termină în acest nod, menținând în același timp minimul valorilor muchiilor din subsecvență. Complexitatea de timp finală este $O(NM)$.

Subtask 2: $N \leq 200$ **și** $M \leq 400$**.** Pentru acest subtask, ne putem fixa fiecare nod și fiecare muchie. Ștergem muchia fixată și actualizăm răspunsul pentru nodul fixat și orice alt nod care nu mai este în aceeași componentă conexă cu nodul fixat în urma ștergerii muchiei. Complexitatea de timp finală este $O(NM(N+M))$.

Subtask 3: $N \leq 2000$ **și** $M \leq 2000$**.** Pentru acest subtask, putem calcula arborele compresat în complexitate timp $O(N(N+M))$. Apoi, ne putem fixa fiecare nod ca fiind rădăcina arborelui si calculăm suma de mai sus. Complexitatea de timp finală este $O(N(N+M))$.

Subtask 4: Toate numerele din șirul c **sunt egale.** Deoarece costul muchiilor este egal, este de ajuns să calculăm arborele compresat. Pentru un nod i, suntem obligați să luăm costul r_i pentru nodurile care fac parte din aceeași componentă în arborele compresat, iar pentru celelalte noduri j, $Cost(i,j) = min(r_i, c)$, unde c este costul muchiilor. Complexitate de timp finală este $O(N+M)$.

Subtask 5: $M = N-1$**.** Pentru acest subtask, putem face solutia generală a problemei, doar că nu mai este nevoie să calculăm *bridge tree-ul* grafului.

Subtask 6: Fără restricții suplimentare. Descrierea solutiei pe cazul general este prezentată mai sus.