LATVIJAS 22. INFORMĀTIKAS OLIMPIĀDES III POSMA UZDEVUMU ĪSI ATRISINĀJUMI
Ventspils, 2009. gada 11.-12. marts

1. "LOKOMOTĪVES"

Skaidrs, ka grupu skaits beigās būs vienāds ar grupu pirmo lokomotīvju skaitu. Tāpat skaidrs, ka, ja pirms kādas lokomotīves X ir kaut viena lokomotīve Y, pie kam $Y < X$, tad lokomotīve X nevar būt grupas pirmā lokomotīve. Savukārt, ja pirms lokomotīves X nevienas šādas lokomotīves nav, tad visas priekšā esošās lokomotīves kustas ātrāk par X, X tās nevar noķert un ir pirmā lokomotīve savā grupā.

Noteikums "visas lokomotīves, kas atrodas pirms X, brauc ātrāk par to" ir ekvivalents nosacījumam, ka X ir lēnāka par pašu lēnāko no tām lokomotīvēm, kas atrodas tai priekšā.

No šejienes uzreiz arī seko risināšanas algoritms:

1. Inicializācija.
   Min := N+1; – kārtējais mazākais lokomotīves numurs
   NGroups := 0; – grupu skaita skaitītājs
2. Apstrādes cikls.
   Virzāmies lokomotīvju masīvā no labās puses uz kreiso – t.i., sākot ar braukšanas virzienā pašu pirmo lokomotīvi.
   Ja kārtējās lokomotīves numurs X ir mazāks par Min, tad aizvietojam Min ar X un par viens palielinām grupu skaitītāju NGroups.

Viegli ievērot, ka uzdevuma sarežģītība ir O(N).

2. "SLIEŽU CEĻI"

Uzdevumu var atrisināt ar dinamiskās programmēšanas palīdzību.

Visiem i no 2 līdz N aprēķināsim šādus lielumus:

a[i] – mazākais tādu dublējošo sliežu ceļu kopgarums, kuriem

1) no katras no pirmajām i stacijām iziet kaut viens dublējošais sliežu ceļš;
2) stacijas i un i−1 ir savienotas ar <u>dublējošo</u> sliežu ceļu.

b[i] – mazākais tādu dublējošo sliežu ceļu kopgarums, kuriem

1) no katras no pirmajām i−1 stacijām iziet kaut viens dublējošais sliežu ceļš;
2) stacijas i un i−1 ir savienotas ar <u>parasto</u> sliežu ceļu.

Viegli redzēt, ka visiem $i \geq 3$:

$b[i] = a[i-1]$,

$a[i] = \min(a[i-1], b[i-1]) +$ (attālums no (i−1)-ās līdz i-tajai stacijai).

Bez tam:

$b[2] = +\infty$ (šajā uzdevumā $+\infty$ vietā var izmantot 1 000 000 001, jo tik daudz dublējošos sliežu ceļus uzbūvēt tik un tā neizdosies),

a[2] = attālums no 1. līdz 2. stacijai.

Aprēķināsim secīgi a[i] un b[i] visām i vērtībām: i = 3, 4, 5, ..., N.

Meklētā atbilde atrodas elementā a[N].

Ievērosim, ka ieviest masīvus a un b nav nepieciešams – pilnīgi pietiek saglabāt tikai pēdējo un priekšpēdējo vērtību pāri. Arī glabāt visus attālumus starp stacijām nevajag – pietiek saglabāt pēdējo nolasīto attālumu.

3. “MONĒTU STABIŅI”

Aizvietosim katru no dotajiem skaitļiem a[i] ar b[i] = a[i]−K.

Uzdevuma piemērā dotās skaitļu virknes 3 1 6 4 6 vietā iegūsim masīvu b: −1 −3 2 0 2.

Apzīmēsim S[j] = b[1]+b[2]+...+b[j], j = 1, 2, ..., N.

Aplūkotajā piemērā iegūsim masīvu S: −1 −4 −2 −2 0

Pamēģināsim “pagriezt” sākotnējo masīvu b un pavērosim, kas notiks ar masīvu S:

sākuma pozīcija	masīvs b	masīvs S
1	-1 -3 2 0 2	-1 -4 -2 -2 0
2	-3 2 0 2 -1	-3 -1 -1 -2 0
3	2 0 2 -1 -3	2 2 4 3 0
4	0 2 -1 -3 2	0 2 1 -2 0
5	2 -1 -3 2 0	2 1 -2 0 0

Nav brīnums, ka S[N] vienmēr ir 0 – tā ir visu doto skaitļu summa (katrs no tiem gan sākumā tika samazināts par K).

Brīnums slēpjas kur citur – mazākais gājienu skaits tiek iegūts, ja sāk līdzināšanu no trešā stabiņa. Tieši šajā (un tikai šajā!) gadījumā visi S elementi ir nenegatīvi. Un šajā gadījumā pats process apstājās pēc N-tā gājiena.

Mazliet padomājot, kļūs skaidrs, ka nekā brīnumaina šeit nav. S[j] – tas ir monētu daudzums, kas tiek pārcelts uz nākamo stabiņu. S[j] < 0 nozīmē, ka mēs pārnesam “parādu”, t.i., kaut vienā no stabiņiem, kam jau veikta līdzināšana, pietrūkst viena vai vairākas monētas un tāpēc kopējais gājienu skaits būs lielāks par N.

Ja S[j] ≥ 0 visiem j, tad nevienu brīdi mēs nepalikām “parādā” – vai nu katrā gājienā notika monētu pārcelšana, vai arī (ja S[j] = 0) stabiņā jau bija nepieciešamais monētu skaits. Vienmēr S[N] = 0, kas nozīmē, ka, ja iepriekšējo darbību laikā nav “parādu”, tad būs nepieciešami ne vairāk kā N gājieni.

Vai vienmēr būs iespējams atrast tādu masīva b pagriešanu, ka visi masīva S elementi būs nenegatīvi? Īsā atbilde ir: jā, vienmēr.

Ievērosim, ka visi masīvi S ir ļoti līdzīgi: katru no tiem var iegūt no iepriekšējā, veicot pagriešanu par vienu vienību, bet pēc tam visi masīva S elementi tiek izmainīti par vienu un to pašu skaitli. Tas arī saprotams – mēs taču summējam vienus un tos pašus skaitļus no masīva b.

Šis novērojums uzreiz ļauj mums panākt situāciju, ka visi S elementi ir nenegatīvi: pagriezīsim masīvu S tā, lai pats mazākais skaitlis tajā nonāktu pēdējā vietā. Pēdējā vietā vienmēr atrodas 0 un šī 0 ir mazākais skaitlis, kas atrodas pēc pagriešanas iegūtajā masīvā S, t.i., pārējie skaitļi nav mazāki par 0.

Lieliski – esam ieguvuši veidu, kā atrast tādu sākuma stabiņu, kas ļauj pabeigt līdzināšanu ne vairāk kā N gājienos: aizpildām masīvu S, atrodam tajā mazāko elementu S[p] un sākam līdzināšanu, sākot ar stabiņu, kas seko aiz p-tā.

Vai līdzināšanu nevar pabeigt mazāk soļos? Iespējams, ka var, lai gan ne vienmēr. Tas ir iespējams gadījumos, ja pagriešanas rezultātā iegūtais masīvs S beidzas nevis ar vienu, bet vairākām nullēm. Iespējams, ka nulles un nenulles skaitļi masīvā ir pamīšus. Precīzāk: ja masīvs S beidzas ar j nullēm, tad būs nepieciešams N−j+1 gājiens.

Atrisinājumu modificēsim šādi:

1. aprēķinām masīvu b un S elementu vērtības;

2. mainīgajā s_{min} saglabājam mazākā masīva S elementa vērtību; visu masīva S elementu vērtības palielinām par s_{min};
3. iegūtajā masīvā S atrodam garāko nuļļu fragmentu (pēc kārtas esošo elementu virkni). Ja šādi fragmenti ir vairāki, tad izvēlamies jebkuru no tiem. Pie tam nevajag aizmirst, ka masīvs ir apļveida – pēc pēdējā elementa seko pirmais.
4. Ja atrastais nuļļu fragments beidzas pozīcijā P, tad kā atbildi izvadām skaitli P+1 (ja P = N, tad izvadām 1).

4. "INTERESANTIE VĀRDI"

Vispirms nomainīsim doto vārdu pret leksikogrāfiskajā kārtībā nākamo vārdu. To izdarīt nav grūti, tas ir gandrīz tikpat vienkārši kā palielināt naturālu skaitli par 1. Rezultātā iegūsim kādu vārdu S.

Saskaitīsim atšķirīgo burtu skaitu vārdā S. Ja tas pārsniedz K, tad vārds S mums der, un to arī izvadām.

Ja turpretī S satur vairāk par K atšķirīgiem burtiem, tad kaut kas ir papildus jādara. Proti, jāmaina, t.i., jāpalielina vārds S.

Sanumurēsim vārda S burtus pēc kārtas no kreisās uz labo pusi. Apzīmēsim i-to burtu ar s_i. Tad $S = s_1s_2s_3...s_N$.

Virzīsimies no kreisās uz labo pusi, skaitot atšķirīgo burtu skaitu.

Pieņemsim, ka K-tais burts pirmoreiz parādās P-tajā pozīcijā, t.i., starp burtiem $s_1s_2s_3...s_{P-1}$ ir atrodami (K−1) atšķirīgi burti, bet burts s_P atšķiras no visiem iepriekšējiem.

Analoģiski, pieņemsim, ka (K+1)-ais burts pirmoreiz parādās Q-tajā pozīcijā, t.i., starp burtiem $s_1s_2s_3...s_{Q-1}$ ir atrodami K atšķirīgi burti, bet burts s_Q atšķiras no visiem iepriekšējiem.

Apzīmēsim ar M lielāko no burtiem $s_1s_2s_3...s_{Q-1}$, bet ar m – mazāko no šiem burtiem.

Skaidrs, ka no kreisās puses pirmajai vārda S maiņai ir jānotiek ne vēlāk par Q-to pozīciju. Turklāt no kreisās puses pirmajam izmanītajam burtam ir jākļūst lielākam.

Pirmais gadījums: $s_Q < M$. Tas nozīmē, ka mēs varam nemainīt pirmos (Q−1) burtus, bet jāpalielina s_Q, turklāt jāpalielina pēc iespējas mazāk. Starp pirmajiem (Q−1) burtiem ir K atšķirīgi, tāpēc s_Q jānomaina pret kādu no šiem burtiem. Atradīsim starp šiem burtiem vismazāko, kas lielāka par s_Q, un nomainīsim s_Q pret šo burtu. Iegūtais vārds būs leksikogrāfiski lielāks par S, neatkarīgi no tā, kādi burti tam būs pa labi no Q-tās pozīcijas. Tāpēc no (Q+1)-ās līdz pēdējai pozīcijai jāliek mazākais iespējamais burts – un tas ir m.

Otrais gadījums: $s_Q > M$. Tas nozīmē, ka jāpalielina kāds burts, kas atrodas pa kreisi no s_Q. Taču mēs gribam, lai no kreisās puses pirmā maiņa notiktu pēc iespējas vēlāk – pēc iespējas vairāk pa labi. Tāpēc virzīsimies no Q-tās pozīcijas pa kreisi, kamēr atradīsim burtu, kas mazāks par M. Uzreiz atzīmēsim, ka virzīties pa kreisi nepieciešams tikai līdz pozīcijai P, jo pa kreisi no P ir atrodami tikai (K−1) atšķirīgi burti, un situācija radikāli mainās – mums rodas iespēja ieviest jaunu burtu.

Aplūkosim atsevišķi abus variantus:

2.1. gadījums. Starp burtiem s_{P+1}, s_{P+2}, ..., s_{Q-1} ir burts, kas mazāks par M, teiksim, s_R. Šī situācija gandrīz sakrīt ar pirmā gadījuma situāciju: pa kreisi no s_R ir K

atšķirīgi burti. Nomainām s_R pret vismazāko no tiem burtiem, kas lielāki par s_R, bet burtus, kas ir pa labi no s_R, nomainām pret burtiem m.

2.2. gadījums. Visi burti s_{P+1}, s_{P+2}, ..., s_{Q-1} sakrīt ar M. Tātad jāpalielina s_P. Pa kreisi no šī burta ir (K−1) atšķirīgi burti, tāpēc mēs varam droši nomainīt s_P pret alfabēta nākamo burtu. Tāds burts noteikti ir: s_P nevar būt alfabēta pēdējais burts "Z", jo $s_P \le M < s_Q$. Visās pozīcijās, kas atrodas pa labi no P-tās pozīcijas, ir jāliek vismazākais iespējamais burts. Ja starp pirmajiem P burtiem pēc s_P maiņas palika (K−1) atšķirīgs burts, tad visās pozīcijās pa labi jāliek pirmais alfabēta burts "A". Ja turpretī starp pirmajiem P burtiem joprojām ir K atšķirīgi burti, tad pozīcijās pa labi jāliek burts m. Izņēmums ir tad, ja pirms burta s_P palielināšanas starp burtiem s_1, s_2, ..., s_P vienīgais vismazākais burts m bija P-tajā pozīcijā. Tādā gadījumā tagad vismazākais burts starp burtiem s_1, s_2, s_3, ..., s_P ir nākamais alfabēta burts pēc m, tāpēc pozīcijās pa labi jāliek šis burts.

Aplūkosim šīs pēdējās iespējas piemēru. Vārds S ir RORERRULA, K = 3. Tad P = 4, Q = 7, m = "E", M = "R", s_Q = "U". Starp burtiem s_{P+1}, s_{P+2}, ..., s_{Q-1} visi burti sakrīt ar M, tāpēc jāpalielina s_P. Pēc s_P palielināšanas starp burtiem s_1, s_2, ..., s_P vismazākais ir "F", tāpēc pa labi jāliek burti "F". Iegūstam vārdu RORFFFFFF.

5. "SPRIGANĀS GOVIS"

Uzdevums patiesībā ir uzdevuma "Sliežu ceļi" nedaudz grūtāka versija.

Pieņemsim, ka kādu žoga fragmentu mēs nenomainām (jeb – nedublējam sliežu ceļu). Tad tālāk iegūstam precīzi uzdevuma "Sliežu ceļi" versiju – žoga fragmenti ir izvietoti vienā virknē (nevis pa apli).

Bet kuru žoga fragmentu nenomainīt? Mēģinot visas N iespējas pie dotajiem ierobežojumiem nevaram atļauties. Taču varam ievērot, ka no trim pēc kārtas esošiem fragmentiem nekad nav vērts nomainīt visus trīs – ja nomainām pirmo un trešo, tad vidējo fragmentu nomainīt nav nepieciešams.

Tātad, aplūkosim pirmos trīs fragmentus un pamēģināsim visas trīs iespējas: neņemt pirmo fragmentu, neņemt otro fragmentu un neņemt trešo fragmentu. Katrā no variantiem atrodam optimālo atbildi kā uzdevuma "Sliežu ceļi" risinājumā. Beigās izvadām mazāko no trim vērtībām.

6. "KUĢĪŠI"

Vispirms atgādināsim dažas definīcijas.

Grafu sauc par *sakarīgu*, ja starp katrām divām tā virsotnēm ir vismaz viens ceļš.

Nākamās trīs definīcijas ir savā starpā ekvivalentas (to nepierādīsim, lai arī šāds pierādījums nav pārāk sarežģīts):

1. *koks* ir saistīts grafs, kas nesatur ciklus;
2. *koks* ir grafs, starp kura katrām divām virsotnēm ir tieši viens tās savienojošs ceļš;
3. *koks* ir saistīts grafs, kurā šķautņu skaits ir tieši par vienu mazāks nekā virsotņu skaits.

Uzdevumā dotajam grafam ir N virsotnes un (N+1) šķautne. Līdz ar to šis grafs nav koks. Toties tas ir "gandrīz" koks – tajā ir tikai divas "liekas" šķautnes un, kad tās tiks izmestas, tad grafs pārvērtīsies par koku.

Atradīsim kādu šķautni (piemēram, starp virsotnēm V un W), kuru izmetot, grafs joprojām paliek sakarīgs. Tas nozīmē, ka virsotnes V un W ir savienotas ar kādu ceļu, kas sastāv no neizmestajām šķautnēm, t.i., izmestā šķautne VW bija kāda dotajā grafā esoša cikla sastāvdaļa.

Pēc šķautnes VW izmešanas grafā palika N šķautnes – tikpat cik virsotņu. Tas nozīmē, ka atlikušais grafs nav koks – t.i., tajā ir vismaz viens cikls. Ja izmetīsim jebkuru šķautni, kas ietilpst šajā ciklā, grafs joprojām būs sakarīgs un tas kļūs par koku (šķautņu skaits būs N−1).

Tātad divu šķautņu izmešanas process izskatās šādi:

1. solis: atrodam grafā kādu ciklu un izmetam jebkuru šķautni, kas tajā ietilpst;

2. solis: atrodam atlikušajā grafā ciklu (ievērosim, ka šāds cikls bija arī sākotnējā grafā) un izmetam jebkuru šķautni, kas ietilpst šajā ciklā.

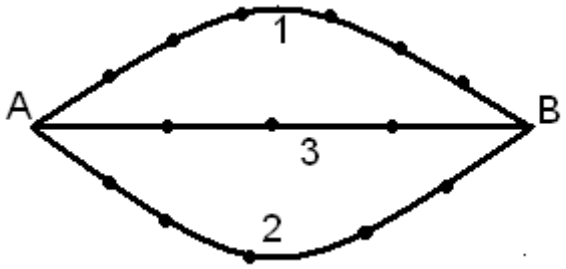
Kā šie divi cikli var būt savstarpēji izvietoti? Iespējami divi principiāli atšķirīgi gadījumi:

1.gadījums. Cikliem nav kopīgu šķautņu.

Tad viena no izmestajām šķautnēm pieder vienam, bet otra – otram ciklam. Pavisam šādu šķautņu pāri iespējams izvēlēties $L_1 \cdot L_2$ variantos, kur L_1 – šķautņu skaits pirmajā ciklā, bet L_2 – šķautņu skaits otrajā ciklā.

2.gadījums Cikliem ir vismaz viena kopīga šķautne.

Tādā gadījumā cikli savstarpēji var būt izvietoti tikai šādi (attēlotas tikai šķautnes, kas ietilpst kaut vienā ciklā):



Ar cipariem 1, 2 un 3 attēloti trīs ceļi, kas savieno kādas divas grafa virsotnes A un B.

Skaidrs, ka jebkādā veidā varam izvēlēties pa vienai šķautnei uz diviem atšķirīgiem ceļiem un grafs joprojām būs saistīts.

Tādejādi, kopējais dažādo variantu skaits ir $L_1 \cdot L_2 + L_1 \cdot L_3 + L_2 \cdot L_3$, kur L_k – šķautņu skaits k-tajā ceļā no A uz B, k = 1,2,3.

Ciklu atrašanai izmantosim standarta algoritmu meklēšanai dziļumā (Depth First Search, DFS). Meklēšanas laikā katrai virsotnei glabāsim tās priekšteci (to virsotni, no kuras mēs atnācām uz aplūkojamo, meklējot dziļumā). Līdzko tiks atrasts pirmais cikls – pazīme, ka iestājusies šī situācija, ir mēģinājums ieiet jau apciemotā virsotnē V – šajā ciklā ietilpstošās virsotnes kaut kādā veidā ir jāatzīmē. Pēc tam izmetam no šī grafa kādu šķautni (piemēram, VW, kur W ir V priekštecis, meklējot dziļumā) un turpinām meklēšanu dziļumā. Kad līdzīgā veidā būs atrasts otrs cikls, atzīmēsim visas virsotnes tajā un meklēšanu pārtraucam.

Pieņemsim, ka C_1 virsotnes pieder tikai pirmajam no atrastajiem cikliem, C_2 virsotnes – tikai otrajam, bet C_3 virsotnes – abiem atrastajiem cikliem.

Ja C_3 ir 0 vai 1, tad tas nozīmē, ka ir 1. gadījums – cikliem nav kopīgu šķautņu. Šajā gadījumā $L_1 = C_1$, $L_2 = C_2$.

Ja $C_3 > 1$, tad ir 2. gadījums – cikliem ir vismaz viena kopīga šķautne. Precīzāk, kopējo šķautņu skaits ir $C_3 - 1$. Atzīmēsim, ka formulas $L_1 \cdot L_2 + L_1 \cdot L_3 + L_2 \cdot L_3$

simetrijas dēļ nav svarīgi, kādā secībā ceļi no A uz B ir numurēti un kuru no tiem uzskatām par pirmo, kuru par otro, bet kuru – par trešo. Uzskatot, ka ceļš, kas kopīgs abiem cikliem, ir trešais, iegūsim: $L_3 = C_3 - 1$, $L_1 = C_1 + 1$, $L_2 = C_2 + 1$.

7. "SKAITĻU FRAGMENTI"

Skat. vecākās grupas uzdevuma "Skaitļu fragmenti – 2" risinājumu.

8. "FOLKLORISTS AUSTRIS"

Šajā risinājumā kā sinonīmus lietosim "virsotne X" un "virsotne ar numuru X", kas vispārīgā gadījumā nav gluži viens un tas pats.

Vispirms atrisināsim palīguzdevumu: dotai virsotnei V (V > 1) atrast tās priekšteci W, kas ar šķautni ir saistīta ar V un ir spēkā sakarība W < V.

Ar R apzīmēsim pēdējās virsotnes numuru tajā līmenī, kurā atrodas virsotne V (šo līmeni sauksim par pašreizējo līmeni), ar S – pēdējās virsotnes numuru iepriekšējā līmenī, kurā atrodas virsotne W.

Katrai iepriekšējā līmeņa virsotnei grafā atbilst A pēcteči.

Pašreizējā līmenī ir R−S virsotnes. Virsotnes S pēcteči ir virsotnes no R−(A−1) līdz R; virsotnes S−1 pēcteči – virsotnes no R−(2A−1) līdz R−A; virsotnes S−2 pēcteči – virsotnes no R−(3A−1) līdz R−2A utt.

Viegli ievērot, ka virsotņu no R−((K+1)A−1) līdz R−KA priekštecis ir virsotne S−K. Citiem vārdiem, virsotnes R−X priekštecis ir virsotne $S - \lfloor X/A \rfloor$, t.i., virsotnes V priekštecis ir virsotne $S - \lfloor (R - V)/A \rfloor$.

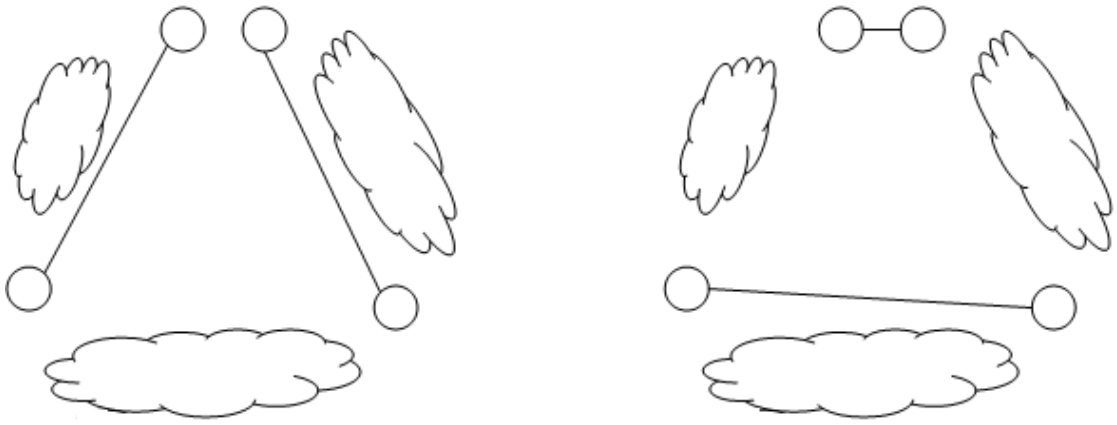
Uzskatīsim, ka dotās virsotnes ir V_1 un V_2. Pakāpeniski pielietojot palīguzdevuma risinājumu, izveidosim ceļu no V_1 līdz pirmajai virsotnei: $a_1, a_2, ..., a_P$ ($a_1 = V_1$, $a_P = 1$, katra nākamā virsotne sarakstā ir iepriekšējās priekštece). Analoģiski atradīsim ceļu no virsotnes V_2 līdz pirmajai virsotnei: $b_1, b_2, ..., b_Q$.

Iespējams, ka dažas pēdējās virsotnes šajos ceļos ir kopīgas $a_P = b_Q$, $a_{P-1} = b_{Q-1}$, ..., $a_{P-i} = b_{Q-i}$, bet $a_{P-i-1} \neq b_{Q-i-1}$.

Tas nozīmē, ka ceļš no V_1 uz V_2 ir šāds: $a_1, a_2, ..., a_{P-i-1}, a_{P-i}, b_{Q-i-1}, b_{Q-i-2}, ..., b_1$.

9. "KRĀSAINIE PUNKTI"

Pirmais un svarīgākais novērojums: ja divi vienas krāsas punkti ir blakus, tad eksistē korekts atrisinājums, kurā šie divi punkti ir savienoti. Lai to pierādītu, pieņemsim, ka mums ir korekts atrisinājums, kurā katrs no šiem diviem punktiem ir savienots ar kādu citu punktu (skat. zīmējumu pa kreisi). Tad varam attēlā redzamos divus nogriežņus nomainīt pret citiem diviem (skat. zīmējumu pa labi), un rezultātā šis variants būs korekts un blakusesošie punkti būs savienoti.



Tātad, tiklīdz sastopam divus vienādas krāsas punktus blakus, tā varam tos savienot (jeb – izmest).

Realizējot šo pieeju pa tiešo, var gadīties, ka iegūsim neefektīvu (kvadrātisku) risinājumu.

Efektīvs risinājums ir – ielasīt dotos punktus pa vienam un likt tos (precīzāk, punktu krāsu numurus) stekā:

1. Ja kārtējā punkta krāsas numurs sakrīt ar pēdējo stekā esošo krāsas numuru, tad šos punktus varam savienot un izmest no steka šo krāsas numuru.
2. Ja turpretī kārtējā punkta krāsas numurs nesakrīt ar pēdējo stekā esošo krāsas numuru, tad kārtējo punktu ieliekam stekā.

Tā kā korekts risinājums vienmēr eksistē, tad beigās steks būs tukšs.

Šāds algoritms darbojas lineārā laikā.

Aplūkosim uzdevuma formulējumā doto otro piemēru (2 3 2 3 3 2 7 7 3 2 4 4). Iekavās pie krāsu numuriem pierakstīts punkta numurs.

Nr	Punkta krāsas numurs	Steks	Izvads
	-	tukšs	
1	2	$2^{(1)}$	
2	3	$2^{(1)}$ $3^{(2)}$	
3	2	$2^{(1)}$ $3^{(2)}$ $2^{(3)}$	
4	3	$2^{(1)}$ $3^{(2)}$ $2^{(3)}$ $3^{(4)}$	
5	3	$2^{(1)}$ $3^{(2)}$ $2^{(3)}$	4 5
6	2	$2^{(1)}$ $3^{(2)}$	3 6
7	7	$2^{(1)}$ $3^{(2)}$ $7^{(3)}$	
8	7	$2^{(1)}$ $3^{(2)}$	7 8
9	3	$2^{(1)}$	2 9
10	2	tukšs	1 10
11	4	$4^{(11)}$	
12	4	tukšs	11 12

10. "KĀRTIS"

Šajā uzdevumā grūtākais ir efektīvi realizēt kāršu pārvietošanu, tāpēc jāatrod kādā veidā glabāt kāršu izvietojumu. No uzdevuma nosacījumiem iespējams izsecināt, ka kāršu kaudzīšu apvienošanai un sadalīšanai ir jānotiek ātri. Šādas darbības ar nepieciešamo ātrdarbību atbalsta divvirzienu saistītais saraksts, jo pievienot vienu sarakstu otram galā un „sagriezt" vienu sarakstu divos var veikt ar dažām operācijām. Tāpēc katru kāršu kaudzīti aprakstīsim ar divvirzienu saistīto sarakstu, kur katrai kārtij glabāsim apakšējās un virsējās kārts numuru. Kāršu kaudzīšu pārvietošanas laikā pārrēķināsim kaimiņu kāršu numurus tikai tām kārtīm, kuras būs pārceļamo kaudzīšu pašā augšā un apakšā.

Tā kā maksimālais kāršu skaits viena testa laikā nemainās, tad šos sarakstus varam glabāt divos masīvos `prev` un `next`, kuru izmērs ir vienāds ar maksimālo kāršu skaitu. Ja zem i-tās kārts ir vēl kāda kārts, tad tās numuru glabājam masīva `prev` i-tajā elementā, taču, ja i-tā kārts ir kaudzītes apakšējā kārts, tad masīva i-tajā elementā glabājam "−1" – skaitli, kas nav sastopams kā kārts numurs. Līdzīgi masīva `next` i-tajā elementā glabājam tās kārts numuru, kas atrodas tieši virs i-tās kārts vai "−1", ja i-tā kārts ir pašā kaudzītes augšā. "−1" vērtība tiek lietota, lai, aplūkojot katrai

kārtij atbilstošo `prev` un `next` vērtību, varētu viennozīmīgi noteikt, vai tā ir augšējā vai apakšējā kārts kaudzītē, vai nē.

Pēc katra kāršu pārcelšanas pieprasījuma kaudzīšu skaits var palielināties par vienu, samazināties par vienu vai nemainīties. Kaudzīšu skaita izmaiņas atkarīgas no tā, kurā vietā savā kaudzītē atrodas katra norādītā kārts (skat. tabulu). Sākumā kaudzīšu skaits ir zināms – N. Ja zina, cik kāršu kaudzītes bija pirms gājiena, un prot atpazīt katru no trim kaudzīšu skaita izmaiņu gadījumiem, tad viegli var izrēķināt, cik kaudzītes ir pēc katra gājiena. Kaudzīšu skaitu var glabāt atsevišķā mainīgajā, kura vērtību pēc pieprasījuma arī izvada, neveicot nekādus papildus aprēķinus.

1.kārts	2.kārts	Kaudzīšu skaita izmaiņa
Virsējā kārts	Apakšējā kārts	−1
Virsējā kārts	Nav apakšējā kārts	0
Nav virsējā kārts	Apakšējā kārts	0
Nav virsējā kārts	Nav apakšējā kārts	+1

11. "MUCAS"

Uzskatīsim, ka mums ir divas mucu kopas – doto mucu kopa un to mucu kopa, kas izvēlētas pārvešanai. Katra muca var atrasties tikai vienā no kopām. Sākumā visas mucas ir doto mucu kopā, bet pārvešanai izvēlēto mucu kopa ir tukša. Parādīsim, kā mucas no vienas kopas var pārvietot uz otru, par katru no mucām nosakot, vai tā ir derīga pārvešanai vai nē.

Pieņemsim, ka kādai mucai j ir spēkā sakarība $G_j/M_j \geq P/Q$ un šī muca līdz šim nav iekļauta pārvešanai izvēlēto mucu kopā. Ja pārvešanai izvēlēto mucu kopā pirms šīs mucas nevienas citas mucas nebija, tad šo mucu droši var iekļaut šajā kopā, jo tā atbilst pārvešanas kritērijam. Savukārt, ja šajā kopā kādas mucas jau ir, tad kopējā šajā kopā esošo mucu šķidruma attiecība pret kopējo tilpumu pēc uzdevuma noteikumiem ir $G_\Sigma/M_\Sigma \geq P/Q$. Ja pārvešanai izvēlētu arī mucu j, tad summārā šķidrumu un tilpuma proporcija būtu: $(G_j+G_\Sigma)/(M_j+M_\Sigma) \geq (M_jP/Q+M_\Sigma P/Q)/(M_j+ M_\Sigma) = P/Q$

Tātad mucu j pārvešanai izvēlēto mucu kopai var pievienot **jebkurā** gadījumā. Visas šādas mucas (kurām šķidruma/tilpuma attiecība ir vismaz P/Q) sauksim par *labām* mucām, bet pārējās – par *sliktām*.

Tātad visas labās mucas var pievienot pārvešanai izvēlēto mucu kopai.

Aplūkosim atlikušās mucas un mēģināsim noteikt, vai kādu no tām ir iespējams pārvest.

Tā kā mucu tilpumi un piepildījums var atšķirties, tās nepieciešams sakārtot. Vienkāršākais kārtošanas kritērijs (pēc mucas šķidruma/tilpuma attiecības) izrādās derīgs tikai īpašos gadījumos (piemēram, kad visu mucu tilpumi ir vienādi), bet vispārīgā gadījumā tas neraksturo dažādo slikto mucu piemērotību aizvešanai.

Piemēram, ja P=3 un Q=4, bet piecu mucu piepildījums/šķidruma daudzums ir 17/24, 3/3, 1/2, 2/3 un 5/8, tad starp tām ir tikai viena labā muca (3/3), bet pārējās pēc iepriekšminētās proporcijas (neaugošā secībā) jāsakārto šādi: 17/24 > 2/3 > 5/8 > 1/2.

Bet, pievienojot pārvešanai izvēlēto mucu kopai kaut vai tikai vienu "labāko" mucu (17/24), iegūsim kopējo šķidruma daudzuma un mucu kopējā tilpuma attiecību (17+3)/(24+3)=20/27, kas ir mazāka par 3/4. Tātad varētu rasties iespaids, ka nevienu no šīm sliktajām mucām pievienot nevar (jo citām šajā nozīmē jābūt vēl "sliktākām"). Bet tā nav. Šajā piemērā var pārvest ne tik vien vienu, bet pat divas sliktās mucas (2/3 un 1/2), tātad izvēlētais kārtošanas kritērijs nav izvēlēts pareizi.

Mēģināsim citādāk. Pēc tam, kad pievienotas labās mucas, iespējams, ka tām proporcija G_Σ/M_Σ ir lielāka par P/Q. Tas nozīmē, ka ir "lieks" šķidruma daudzums, kas varētu ļaut pievienot arī kādu slikto mucu. Katrai sliktajai mucai noskaidrosim, kāds mazākais šķidruma daudzums nepieciešams, lai šo mucu padarītu par labu. Katrai mucai i šis daudzums ir vienāds ar $M_iP/Q\text{-}G_i$. Aprēķinot šo šķidruma daudzumu katrai mucai, iepriekšējā piemērā iegūsim: 1 (mucai 17/24), 0.5 (1/2), 0.25 (2/3), 1 (5/8). Starp katrām divām mucām par labāku uzskatīsim to mucu, kurai nepieciešamais pievienojamā šķidruma daudzums ir mazāks. Sakārtojot mucas pēc šī kritērija, iegūsim ("$\Rightarrow$" nozīmēs "nav sliktāka par"):

$2/3 \Rightarrow 1/2 \Rightarrow 17/24 \Rightarrow 5/8$ (pēdējās divas mucas varēja arī mainīt vietām, jo to papildināšanai nepieciešams vienāds šķidruma daudzums).

Kad mucas šādi ir sakārtotas, viegli pamanīt, ka mucas jāņem pēc kārtas, sākot ar labākajām (neņemot mucas pēc kārtas, neko labāku neiegūsim), kamēr vien summārā proporcija šķidrums/kopējais_tilpums nekļūst mazāka par P/Q.

Nobeigumā dažas piezīmes par realizāciju.

Veicot kārtošanu, mucām i un j nepieciešams salīdzināt, kura no vērtībām $M_iP/Q-G_i$ un $M_jP/Q-G_j$ ir lielāka. Lai varētu darboties tikai ar veseliem skaitļiem, varam pareizināt abas šīs vērtības ar Q un salīdzināt M_iP-G_iQ ar M_jP-G_jQ.

Labās mucas sākumā speciāli varam nemeklēt – tām M_iP-G_iQ vērtības būs negatīvas un pēc sakārtošanas tās atradīsies pirms sliktajām mucām.

12. "SKAITĻU FRAGMENTI – 2"

Pieņemsim, ka $D = 2^{p_2} \cdot 5^{p_5} \cdot c$, kur p_2 un p_5 – veseli nenegatīvi skaitļi, bet skaitlis c nedalās ne ar 2, ne ar 5, t.i., LKD(c, 10) = 1. Ar p apzīmēsim lielāko no skaitļiem p_2 un p_5: $p = \max(p_2, p_5)$, $b = 2^{p_2} \cdot 5^{p_5}$. Uzreiz atzīmēsim, ka $p_2 \le \log_2 D$, $p_5 \le \log_5 D$.

Ievērosim, ka skaitļi b un c ir savstarpēji pirmskaitļi − LKD(b, c) = 1; $D = b \cdot c$. Tāpēc skaitlis x dalās ar D tad un tikai tad, kad tas dalās gan ar skaitli b, gan ar c.

Aplūkosim divus gadījumus.

1. gadījums. Fragmenta garums nepārsniedz p.

Skaitlis p ir neliels – dotajiem ierobežojumiem tas nepārsniedz 30 ($30 < \log_2 2000000000 < 31$). Tāpēc visu fragmentu, kuru garumi nepārsniedz p, dalāmību ar D var ātri pārbaudīt "pa tiešo". Šādu fragmentu skaits nepārsniedz $N \cdot p$, tāpēc kopējais pārbaudei nepieciešamais laiks ir $O(N \cdot \log D)$.

2. gadījums. Fragmenta garums ir lielāks par p.

Ar S[j..k] apzīmēsim virknes S fragmentu no j-tā līdz k-tajam ciparam.

Pieņemsim, ka x = S[j..k], $k-j \ge p$.

Skaitlis x dalās ar $b = 2^{p_2} \cdot 5^{p_5}$ tad un tikai tad, kad skaitlis S[k−p+1..k] (kuru veido skaitļa x pēdējie p cipari) dalās ar p. To var viegli redzēt, ja skaitli x pieraksta formā $x = S[i..k-p] \cdot 10^p + S[k-p+1..k]$.

Skaitlis x dalās ar c (kas ir savstarpējs pirmskaitlis ar 10) tad un tikai tad, ja skaitļiem S[j..N] un S[k+1..N] ir vienādi atlikumi, tos dalot ar c. Tiešām,

$$S[j..N] = x \cdot 10^{N-k} + S[k+1..N] \text{ jeb}$$

$$x \cdot 10^{N-k} = S[j..N] - S[k+1..N].$$

No šīs sakarības un tā, ka LKD(c, 10) = 1, uzreiz seko nepieciešamais.

Tātad, nepieciešams saskaitīt, cik ir tādu fragmentu S[j..k], kuriem vienlaicīgi ir spēkā šādas sakarības:

1. $k-j \geq p$
2. $S[j] \neq 0$
3. S[k−p+1..k] dalās ar b
4. S[j..N] un S[k+1..N] ir vienādi atlikumi, tos dalot ar c. Uzskatām, ka S[N+1..N] = 0 – ar to tiek aplūkots arī gadījums, kad k = N.

Aizpildīsim masīvu tail: šī masīva i-tais elements saturēs informāciju, vai fragments S[i−p+1..i] dalās ar b, $i \geq p$. Šo procesu var paātrināt, ja ievēro, ka S[i−p+1..i] dalās ar b tad un tikai tad, ja S[1..i] dalās ar b.

Aizpildīsim masīvu r: r[i] – tas ir atlikums, dalot S[i..N] ar c.

Nepieciešams atrast tādus skaitļu j un k pārus, ka r[j] = r[k+1].

Katrai r[i] vērtībai pievienosim i vērtību un sakārtosim šos pārus r[i] augšanas secībā. Ja $r[i_1] = r[i_2]$, tad kā pirmo liekam pāri ar mazāku i vērtību.

Tātad, sakārtotā virkne ir šāda: $r[j_1] = r[j_2] = \ldots = r[j_M]$, $j_1 < j_2 < \ldots < j_M$.

Katram m no 1 līdz M aprēķināsim tailcount[j_m] – tādu indeksu j_q skaitu, q = m+1, m+2, …, M, kuriem fragments $S[j_q-p \,..\, j_q-1]$ dalās ar b (ir izpildīts nosacījums tail[j_q–1]). To viegli izdarīt, virzoties masīvā $j_1, j_2, \ldots, j_M$ no labās puses uz kreiso.

Un, visbeidzot: pieņemam $j = j_1$, atrodam k – mazāko no indeksiem $j_1 < j_2 < \ldots < j_M$, kas lielāks par j+p, un, ja $S[j] \neq 0$, pieskaitām tailcount[k] rezultātam. Pēc tam j pārvietojam uz j_2 un attiecīgi pārvietojam k pa labi utt., kamēr k nesasniegs labo galu.

Šo procesu izpildām visām r[i] vērtībām.

Masīvu tail un r aizpildīšanai nepieciešamas O(N) darbības.

Kārtošanai, ja to veic ar efektīvu algoritmu, nepieciešamas O(N·logN) darbības.

Katram no indeksu nogriežņiem j_q, kur $r[j_q]$ vērtības – atlikumi, dalot ar c, ir vienādi, ir nepieciešamas trīs caurskates: viena caurskate tailcount aizpildīšanai, bet divas caurskates nepieciešamas j un k – kreisā un labā gala fragmentiem. Visiem nogriežņiem kopā nepieciešamas O(N) darbības.

Tātad, algoritma kopējā sarežģītība ir O(N·logD) + O(N) + O(N·logN), t.i., O(N·(logN + logD)).