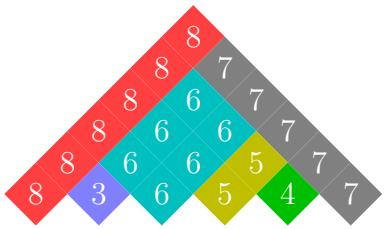
Latvijas 28.informātikas olimpiādes trešā posma uzdevumu atrisinājumi

Latvijas informātikas olimpiāde

Maksimumi

Aplūkosim pirmo rindu $a_1, a_2, \ldots, a_n$. Katrai pozīcijai i atradīsim skaitļus l un r: l ir lielākā pozīcija pa kreisi no i tāda, ka $a_l > a_i$, vai 0, ja tāda skaitļa nav; r ir mazākā pozīcija pa labi no i tāda, ka $a_i \leq a_r$, vai $n+1$, ja tāda skaitļa nav. Tad var ievērot, ka beigu tabulā taisnstūris, kas sākas a_i un kura izmēri ir $(i-l) \times (r-i)$ satur tikai skaitļus a_i (pirms atbilstošie taisnstūri no a_l un a_r tam «nogriež sānus»).

Piemērā 1. att. skaitlim 6 (pozīcijā 3) tuvākais skaitlis no kreisās puses, kas lielāks par to, ir 8 (pozīcijā 1), bet tuvākais skaitlis no labās puses, lielāks par to, ir 7 (pozīcijā 6). Tāpēc 6 aizpilda taisnstūri ar izmēriem $(3-1) \times (6-3) = 2 \times 3$.



1. att. Tabulas piemērs.

Aprēķināt uzdevumā prasīto var lineārā laikā, izmantojot steku. Lai atrastu skaitļus l, apstrādāsim pirmo rindu no kreisās puses uz labo. Stekā glabāsim pozīcijas, kas potenciāli vēl var kalpot kādam neapstrādātam skaitlim par tuvāko skaitli, kas lielāks par to, no kreisās puses. Pozīcijas stekā ir augošā secībā no lejas uz augšu. Pieņemsim, ka apstrādājam jauno skaitli pozīcijā i un stekā augšā ir pozīcija j. Ja $a_j \leq a_i$, tad a_j nevar būt tuvākais skaitlis, kas lielāks par aplūkojamo skaitli no kreisās puses ne skaitlim a_i, ne jebkuram citam skaitlim a_k pa labi no i. Tāpēc no steka varam ņemt ārā augšējo elementu j, kamēr $a_j \leq a_i$. Ja pēc tā steks nav tukšs, tad pozīcija steka augšā ir vajadzīgais l pozīcijai i. Ja steks ir tukšs, tad l pozīcijai i ir 0. Visbeidzot stekā ieliekam augšā i un turpinām procedūru.

Līdzīgi atrodam arī skaitļus r (apstrādājot skaitļus no labās puses uz kreiso ar steku). Vienīgi jāņem vērā, ka pirmajā rindā var būt arī vienādi skaitļi: uzskatīsim, ka divu vienādu skaitļu gadījumā lielākais ir skaitlis pa labi. Tas jau tika ņemts vērā iepriekš, definējot l un r.

Tagad zinām, cik reizes katrs skaitlis atkārtojas tabulā. Atkarībā no katra skaitļa paritātes saskaitām pāra un nepāra skaitļu skaitu tabulā.

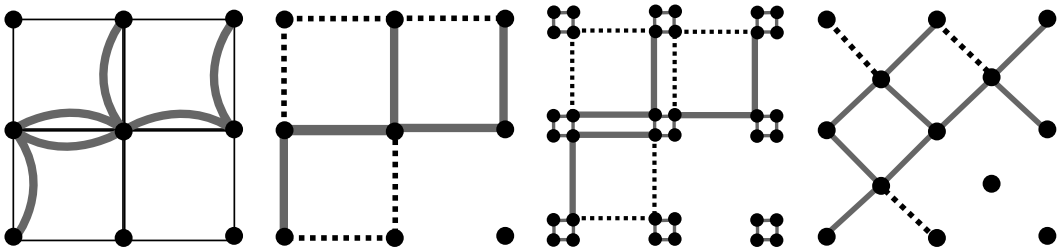
Darbības laiks: $\mathcal{O}(n)$ **Tēmas:** steks

Vai kontakts?

Var ievērot, ka, ja rūtiņa satur slēdzi, tad iespējams savienot jebkurus divus stūrus. Lai to izdarītu, vajag ne vairāk kā vienu pagriezienu. Var izveidot grafu, kurā katram kontaktam atbilst virsotne un kontaktu rūtiņu malām šķautnes, skatīt 2b. attēlu. Pielietojot jebkuru grafa apstaigāšanas algoritmu iespējams pārbaudīt vai atbilde ir -1, 0 vai lielāka par 0. Šķautnēm piešķir garumu: ja kontakti ir savienoti sākotnējā konfigurācijā, šķautnes garums ir 0; ja tos var savienot veicot vienu pagriezienu (attēlots ar pārtrauktu līniju) šķautnes garums ir 1. Tā kā šķautnes garumā 1 atbilst vienam pagriezienam, tad īsākais ceļš grafā atbilst mazākajam

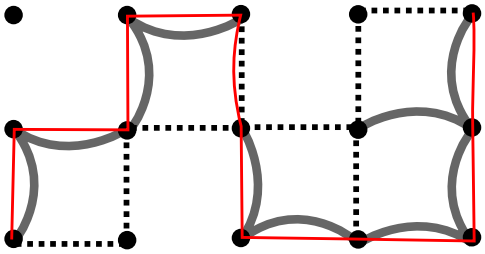
Latvijas 28.informātikas olimpiādes trešā posma uzdevumu atrisinājumi

Latvijas informātikas olimpiāde



(a) (b) (c) (d)

2. att. Dažādi kontaktu grafa veidi.



3. att. Neiespējams ceļš.

pagriezienu skaitam, kas nepieciešams stūru savienošanai. Pastāv dažādi algoritmi, lai atrastu īsāko ceļu grafā ar dažādiem šķautņu garumiem. Piemēram, Dijkstras algoritms, bet vairums no tiem strādā ar laika sarežģītību $\mathcal{O}(m \log n)$ (kur n ir grafa virsotņu skaits, bet m ir šķautņu skaits). Tā kā grafs satur tikai šķautnes garumā 0 un 1, tad īsāko ceļu var atrast lineārā laikā izmantojot 0-1 BFS algoritmu. 0-1 BFS ir līdzīgs parastajam BFS. Ja šķautnes garums ir 1, tad sasniedzamā virsotne tiek pievienota rindas beigās, kā parasti. Bet, ja šķautnes garums ir 0, tā tiek pievienota rindas sākumā. Tādējādi tiek nodrošināts, ka ātrāk sasniedzamās virsotnes tiek apstrādātas vispirms. Šādu rindu sauc par deku, jeb rindu ar diviem galiem (angliski *deque* no *double-ended queue*). Tā vietā var izmantot arī divas rindas vai stekus.

Lai iegūtu veicamos pagriezienus, nepieciešams atjaunot īsāko ceļu, pa kuru var sasniegt beigu kontaktu. To var iegūt, katrā virsotnē saglabājot norādi uz iepriekšējo virsotni. Pēc tam, sākot ar pēdējo virstoni un ejot atpakaļ, iegūst ceļu. Gadījumā, ja šķautnes garums ir 1, nepieciešams rotēt blakus esošo slēdzi. Var rasties divi galvenie sarežģījumi: kā noteikt, kuram slēdzim pieder šķautne, kā izvairīties no slēdža pagriešanas divreiz.

Pirmo var atrisināt, pārbaudot rūtiņas pa kreisi un pa labi no šķautnes, bet tas ir sarežģīti. Var viegli nokļūdīties, kā arī speciāli jāapstrādā malējās šķautnes, kam ir tikai viena blakus rūtiņa. To var atrisināt vienkāršāk, nedaudz pārveidojot grafu. Interpretācija 2b. visvairāk atbilst uzdevuma formulējumam un aizņem mazāk atmiņas, bet 2c. un 2d. ļauj vieglāk atrisināt iepriekš minētās problēmas.

Grafā 2c. katrs kontakts ir pārveidots par četriem, kas ir savienoti ar šķautnēm garumā 0. Tādējādi šķautņu un virsotņu piederība kontaktiem ir acīmredzama. Šāda grafa glabāšanai šķautnes nav speciāli jāglabā, jo no visām virsotnēm var pārvietoties 4 virzienos, ja tādējādi netiek pamests laukums.

Līdzīgī grafā 2d. skaidri saprotams, kuram kontaktam pieder virsotnes, kas atrodas rūtiņu centros. Rotācijas virziena noteikšana ir nedaudz sarežģītāka, jo jāapskata divas secīgas šķautnes, nevis viena, kā

tas ir 2b. un 2c..

Otrā problēma redzama 3. attēlā. Sarkanā ceļa garums sakrīt ar īsāko, bet nav iespējams pagriezt slēdžus tā, lai to savienotu. Šāda situācija rodas, ja ceļš iet pa rūtiņas pretējām malām un vismaz vienai no tām atbilst šķautne garumā 1. Drīkst būt ceļš, kas pārvietojas pa rūtiņas pretējām malām, ja tas izmanto savienojumus no vairākiem kontaktiem. No šādiem ceļiem viegli izvairīties, izmantojot grafu 2d.. Ja rūtiņu centrālā virsotne tiek apmeklēta ne vairāk kā vienu reizi, tad nav iespējams pārvietoties pa rūtiņas pretējām malām. Cits veids ir minimizēt ne tikai skaitu šķautnēm 1, bet arī šķautnēm 0. To var viegli īstenot, ja īsākais ceļš tiek meklēts ar Dijkstras algoritmu. Ceļa garumu var aprakstīt ar skaitļu pāri (šķautņu ar svaru 1 skaits, šķautņu ar svaru 0 skaits). Skaits šķautnēm ar svaru 0 tiek salīdzināts tikai tad, ja šķautņu skaits ar svaru 1 sakrīt. Lietojot 0-1 BFS grafiem 2b. un 2c., nepieciešams pārbaudīt, vai virsotne ir sasniegta ar šāda gara ceļu, pirms ievietošanas rindā, nevis pēc ceļa izņemšanas no rindas.

Darbības laiks: $\mathcal{O}(n)$ **Tēmas:** grafi, 0-1 BFS, Dijkstras algoritms

Summa

Izmantosim dinamiskās programmēšanas metodi. Skaitlim i-tajā rindā j-tajā kolonnā $a[i][j]$ sarēķināsim vērtību $dp[i][j]$ – maksimālo summu, ko var iegūt no tabulas pirmajām i rindām tā, ka izvēlē tiek iekļauts $a[i][j]$. Šo lielumu var aprēķināt pēc formulām:

$$dp[1][j] = a[i][j]$$
$$dp[i][j] = a[i][j] + \max_{\substack{k=1 \\ k \neq i}}^{m} dp[i-1][k]$$

Tad atbilde ir $\max_{j=1}^{m} dp[n][j]$.

Tomēr šīs formulas tieša realizācija strādā laikā $\mathcal{O}(nm)$. Var ievērot, ka priekš $dp[i][j]$ izrēķināšanas pietiek glabāt tikai divus lielākos skaitļus starp $dp[i-1][1], dp[i-1][2], \ldots, dp[i-1][m]$: pieņemsim, ka tie ir $dp[i-1][a]$ un $dp[i-1][b]$, pie tam $dp[i-1][a] \geq dp[i-1][b]$. Tiešām,

- ja $j \neq a$, tad $dp[i][j] = a[i][j] + dp[i-1][a]$;
- ja $j = a$, tad $dp[i][j] = a[i][j] + dp[i-1][b]$.

Darbības laiks: $\mathcal{O}(n+m)$ **Tēmas:** dinamiskā programmēšana

Kāršu spēlmaņi

Katram spēlētājam uz rokas ir tieši $cnt = \left\lfloor \frac{b+m}{s} \right\rfloor$ kārtis. Tāpēc visi spēlētāji, kas nosauca summu, lielāku par cnt, noteikti melo. Pieņemsim, ka tādu spēlētāju skaits ir k. Atmetīsim tos un aplūkosim visus pārējos spēlētājus, kuru nosauktās summas ir $a[1], a[2], \ldots, a[s-k]$.

Saskaitīsim visu šo $s-k$ spēlētāju nosaukto skaitļu summu $sum = a[1] + a[2] + \ldots + a[s-k]$. Pāri palikušo kāršu skaits ir $rem = (b+m) \pmod{s}$. Šķirosim trīs gadījumus:

- $sum > b$. Tā kā summa pārsniedz kopējo vieninieku skaitu, tad noteikti kāds melo. Lai minimizētu meļu skaitu, visizdevīgāk ir uzskatīt par meli spēlētāju ar lielāko $a[i]$. Sakārtosim $a[]$ dilstošā secībā. Kamēr $sum > b$, ņemsim spēlētāju ar lielāko $a[i]$ un no sum atņemsim $a[i]$. Tad var uzskatīt, ka $a[i]$ ir melis un viņam var būt līdz pat 0 kārtīm ar vieninieku.

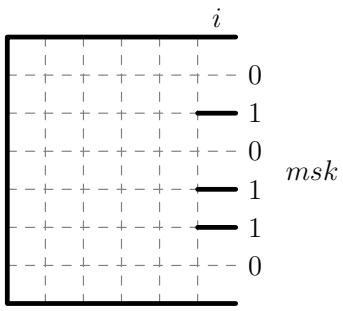
- $sum < b - rem - k \cdot cnt$. Tā kā starp pāri palikušajām kārtīm var būt lielākais rem vieninieki, un katram no k jau atrastajiem meļiem var būt ne vairāk par cnt vieniniekiem, tad pārējiem $s - k$ spēlētājiem noteikti kopā ir vismaz $b - rem - k \cdot cnt$ vieninieki. Ja sum ir mazāka par šo skaitli, tad noteikti kāds melo. Lai minimizētu meļu skaitu, visizdevīgāk ir uzskatīt par meli spēlētāju ar mazāko $a[i]$. Sakārtosim $a[]$ augošā secībā. Kamēr $sum < b - rem - k \cdot cnt$, ņemsim spēlētāju ar vismazāko $a[i]$ un pieskaitīsim sum vērtību $cnt - a[i]$. Tad uzskatām, ka $a[i]$ ir melis un viņam var būt līdz pat cnt kārtīm ar vienīnieku.
- $b - rem - k \cdot cnt \leq sum \leq b$. Šajā gadījumā var sadalīt vieniniekus tā, lai nebūtu pretrunas, un meļu skaits var būt 0.

Darbības laiks: $\mathcal{O}(s \log s)$ **Tēmas:** rijīgie algoritmi, kārtošana

Cik veidos?

Sākumā pagriezīsim taisnstūri tā, lai augstums nebūtu garāks par platumu, $a \leq b$. Tas nozīmē, ka $a \leq 10$. Vēl risinājumā uzskatīsim, ka nesadalīts taisnstūris $a \times b$ arī ir veids, kā sagriezt taisnstūri – tad, izvadot atbildi, atņemsim no sarēķinātās vērtības 1.

Aplūkosim vienu taisnstūra kolonnu. Tās stāvokli var aprakstīt ar bitmasku garumā $a - 1$, kur j-tais bits ir 1 tad un tikai tad, ja gar j-to vertikālu rūtiņu malu no augšas šajā kolonnnā tiek veikts griezums, skat. 4. att. Ar dinamisko programmēšanu rēķināsim vērtības $dp[i][msk]$ – cik veidos korekti var veikt griezumus taisnstūrī no pirmajām i kolonnām tā, lai i-tajā griezumi atbilstu bitmaskai msk. Ievērosim, ka līdz tam i-tās kolonnas labējās malas griezumus neapskatām.

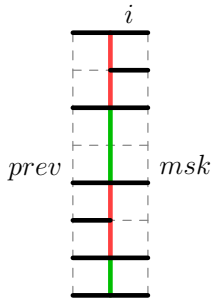


4. att. Kolonas maska.

Pirmajai kolonnai ir spēkā $dp[1][msk] = 1$ katram msk. Kā izrēķināt $dp[i][msk]$ citiem $i > 1$? Pārlasām masku $(i-1)$-jai kolonnai $prev$. Lai q ir veidu skaits, kā pareizi var salikt griezumus abu kolonnu kopējā malā. Tad veidu skaits, lai korekti sagrieztu taisnstūra pirmās i kolonnas tā, lai $(i-1)$-ā kolonna atbilstu maskai $prev$, bet i-tā atbilstu maskai msk, ir $dp[i-1][msk] \cdot q$. Šo skaitli pieskaitīsim $dp[i][msk]$.

Atlika uzzināt skaitli q priekš maskām $curr$ un msk. Ievērosim, ka divas secīgas grieztas vertikālas rūtiņu malas $(i-1)$ kolonnā nozīmē, ka rūtiņās starp tām no kreisās puses turpinās taisnstūris. Šis taisnstūris var vai nu izbeigties $(i-1)$-ajā kolonnā, vai nu turpināties i-tajā kolonnā. Bet tas var turpināties tad un tikai tad, ja i-tajā rindā abas vertikālas rūtiņu malas tajā pašā augstumā arī ir nogrieztas, un starp tām nav

nogriezta neviena cita. Pretējā gadījumā taisnstūrim noteikti ir jāizbeidzas $(i-1)$-ajā kolonnā. Piemēram, 5. att. varam paturpināt divus taisnstūrus (zaļas malas), bet izbeigt noteikti vajag trīs citus taisnstūrus (sarkanas malas). Lai k ir tādu vietu skaits, kur taisnstūris var turpināties no $(i-1)$-ās kolonnās uz i-to. Tad katrā no k šādām vietām var vai nu izbeigt, vai nu paturpināt taisnstūri. Tāpēc veidu skaits, kā korekti sagriezt kolonnu kopējo malu, ir $q = 2^k$.



5. att. DP pāreja.

Visbeidzot atbilde ir vienāda ar summu pa visām iespējamām beigu kolonnas maskām, $\sum_{msk} dp[b][msk]$. Jāatceras, ka visas darbības ir jāņem pēc moduļa 10^9+7.

Novērtēsim šī risinājuma sarežģītību taisnstūrim ar izmēriem $a \times b$. Pavisam dažādu $dp[i][msk]$ ir $b \cdot 2^a$. Lai sarēķinātu vienu $dp[i][msk]$ vērtību, pirmkārt vajadzīgs pārlasīt iepriekšējās kolonnas masku $prev$, tādu skaits ir 2^a. Savukārt, lai uzzinātu skaitli k priekš viena pāra $curr$, msk, vajadzīgs iziet cauri abām maskām, kas abas ir garumā a. Tāpēc risinājums strādā laikā $\mathcal{O}(b \cdot 2^a \cdot 2^a \cdot a) = \mathcal{O}(a \cdot b \cdot 4^a)$. Tas strādā pietiekoši efektīvi, lai atrisinātu visus 100 testus dažu sekunžu laikā.

Darbības laiks: $\mathcal{O}\left(\max(a,b) \cdot \min(a,b) \cdot 4^{\min(a,b)}\right)$ **Tēmas:** dinamiskā programmēšana, bitmaskas

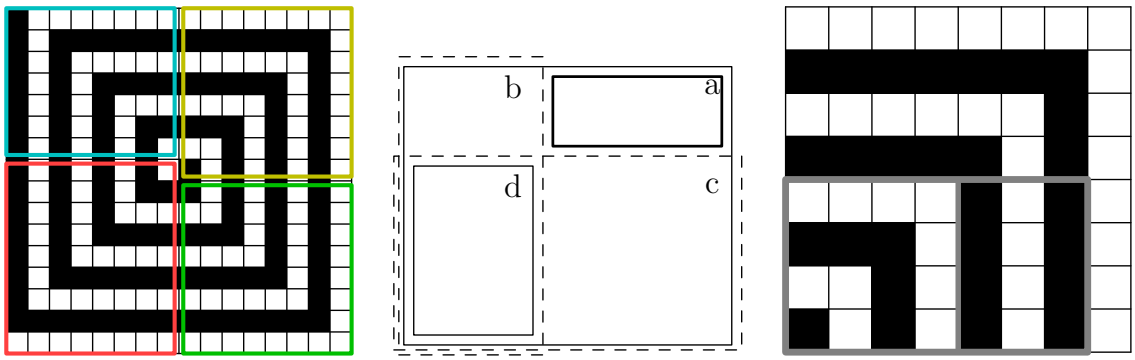
Spirāle

Ja taisnstūrī sākotnēji ir a melnas rūtiņas un b baltas rūtiņas, tad atbilde ir $\min(a,b)$. Lai iegūtu balto un melno rūtiņu skaitu, pietiek saskaitīt melno rūtiņu skaitu. Balto rūtiņu skaitu var iegūt, atņemot no laukuma melno rūtiņu skaitu. Spirāli var sadalīt četrās līdzīgās daļās, kā parādīts 6a. attēlā. Melno rūtiņu skaitu var skaitīt katrā ceturtdaļā atsevišķi. Melno rūtiņu skaitīšanu taisnstūrī $m(x_1, y_1, x_2, y_2)$ var vienkāršot par melno rūtiņu skaitu summu četriem taisnstūriem, kuriem viens stūris ir $(0,0)$, kā parādīts 6b. attēlā.

$$m(x_1,y_1,x_2,y_2) = m(0,0,x_2,y_2) - m(0,0,x_2,y_1-1) - m(0,0,x_1-1,y_2) + m(0,0,x_1-1,y_1)$$

Tas vienkāršo uzdevumu uz melno rūtiņu saskaitīšanu taisnstūrī, kam viens stūris ir rūtiņā $(0,0)$, tā platums ir w, bet augstums - h. Spirāles ceturtdaļa ir simetriska taisnei $x = y$, tāpēc var pieņemt, ka $w \geq h$. Šo taisnstūri var tālāk sadalīt kvadrātā ar izmēru $h \times h$ un taisnstūrī ar vertikālām melnām, skat. 6c.. Aprēķinot melno rūtiņu skaitu vertikālajās līnijās, jāņem vērā kvadrāta malas garuma paritāte.

$$h \cdot \left\lfloor \frac{1 - (h \mod 2) + w - h}{2} \right\rfloor$$



(a) (b) (c)

6. att. Risinājums.

Melno rūtiņu skaits kvadrātā veido aritmētisko progresiju.

$$k = \left\lfloor \frac{h+1}{2} \right\rfloor$$ melno joslu skaits

$$m = k \cdot \frac{1+1+4\cdot(k-1)}{2}$$ melno rūtiņu skaits kvadrātā

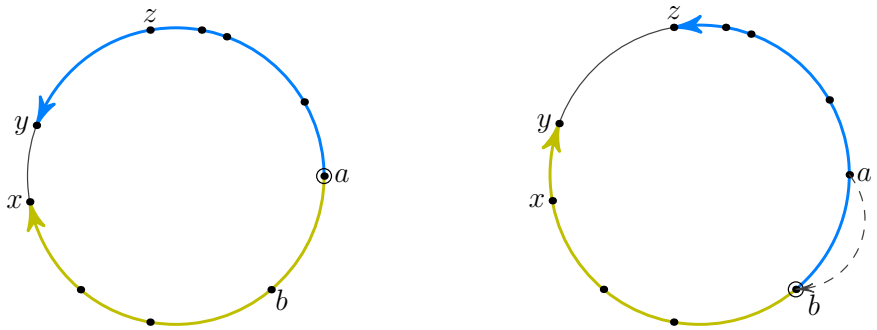
Darbības laiks: $\mathcal{O}(1)$ **Tēmas:** ad hoc, parciālsummas

Jaunā osta

Iztēlosimies pilsētas kā punktus uz riņķa tā, lai attālumi atbilstu dotajiem. Lietosim tā saukto divu norāžu metodi. Pārlasām centra virsotni $center$ pulksteņrādītāja virzienā. Uzturēsim divas norādes $left$ un $right$. Norāde $left$ ir tālākais punkts, uz kuru no punkta $center$ ir izdevīgāk virzīties pulksteņrādītāja virzienā. Norāde $right$ ir tālākais punkts, uz kuru no punkta $center$ ir izdevīgāk virzīties pretēji pulksteņrādītāja virzienam. Piemēram, 7a. att., ja $center = a$, tad $left = x$ un $right = y$. Uzturēsim arī ceļa garumus no $center$ līdz $left$ un $right$, lai tie ir $distLeft$ un $distRight$, attiecīgi. Uzturēsim arī punktu skaitu kreisajā un labajā ceļos attiecīgi mainīgajos $cntLeft$ un $cntRight$. Visbeidzot glabāsim arī kopējo attālumu summu no $center$ līdz visiem punktiem, uz kuriem izdevīgāk kustēties pa kreisi vai pa labi mainīgajos $totalLeft$ un $totalRight$, attiecīgi. Tad virsotnei $center$ atbilde ir vienāda ar $totalLeft + totalRight$.

Sākumā par $center$ izvēlamies kādu no punktiem. Atrodam punktus $left$ un $right$, ejot no $center$ uz abām pusēm, kamēr $distLeft$ un $distRight$ nepārsniedz pusi no riņķa garuma. Tajā laikā arī saskaitām kopējos ceļa garumus $totalLeft$ un $totalRight$ (piemēram, pabīdot norādi $left$ vienu punktu uz priekšu, pieskaitām pie $totalLeft$ jauno ceļa garumu $distLeft$), kā arī rēķinām pareizos $cntLeft$ un $cntRight$. Šī risinājuma daļa tiek izpildīta lineārā laikā.

Tagad pārbīdam norādi $center$ par vienu punktu pulksteņrādītāja virzienā. Piemēram, 7b. att. pabīdām to no a uz b. Tagad nepieciešams pārrēķināt visus pārējos mainīgos. Pieņemsim, ka attālums starp blakusesošiem diviem punktiem u un v ir $dist(u,v)$. Pirmkārt, dēļ tā, ka jaunais $center$ pārvietojas no a uz b, ir sekojošas izmaiņas (šādā secībā):



(a) Sākumstāvoklis. (b) Stāvokļa maiņa.

7. att. Divas norādes.

- $distLeft$ samazinās par $dist(a,b)$;
- $distRight$ palielinās par $dist(a,b)$;
- $totalLeft$ samazinās par $dist(a,b) \cdot cntLeft$;
- $totalRight$ palielinās par $dist(a,b) \cdot cntRight$;
- $cntLeft$ samazinās par 1;
- $cntRight$ palielinās par 1.

Otrkārt, jāatjauno norādes x un y. Bīdām norādi $left$ pulksteņrādītāja virzienā tik tālu, kamēr $distLeft$ nepārsniedz pusi no riņķa kopējā garuma; norādi $right$ tagad bīdām arī pulksteņrādītāja virzienā tik ilgi, kamēr $distRight$ ir vismaz puse no riņķa kopējā garuma. Paralēli atjaunojam visu mainīgo datus. Piemēram, zīmējumā pabīdot norādi $left$ no x uz y, bet $right$ no y uz z, ir sekojošas izmaiņas (šādā secībā):

- $distLeft$ palielinās par $dist(x,y)$;
- $distRight$ samazinās par $dist(y,z)$;
- $totalLeft$ palielinās par jauno $distLeft$;
- $totalRight$ samazinās par jauno $distRight + dist(y,z)$;
- $cntLeft$ palielinās par 1;
- $cntRight$ samazinās par 1.

Viens šāds solis prasa konstantu laiku. Tā kā kopā ir $n-1$ šādi soļi, tad šī risinājuma daļa arī patērē lineāru laiku. Jāievēro, ka kāds punkts var būt diametrāli pretējs punktam $center$; tāpēc jāuzmanās no tā, lai $left$ nesakristu ar $right$. Tādēļ uzskatīsim, ka $right$ ir tālākais punkts, līdz kuram izdevīgākais ceļš ir pa labi un ceļa garums līdz tam ir strikti mazāks par pusi no kopējā riņķa garuma. Jāpiemin arī, ka ja priekš $left$ vai $right$ nav kandidātu, tad tie var droši būt vienādi ar pašu $center$ (tad attiecīgi, piemēram, $cntLeft$ būtu 0, ja $left = center$, utml.).

Latvijas 28.informātikas olimpiādes trešā posma uzdevumu atrisinājumi

Latvijas informātikas olimpiāde

Darbības laiks: $\mathcal{O}(n)$ **Tēmas:** divas norādes

Kāršu jaukšana

Apzīmē kāršu skaitu $m = n^2 + 1$. Kārtis pārnumurē no 0 līdz n^2 un tālākās darbības veic pēc moduļa m. Kāršu kavu var raksturot ar diviem skaitļiem: a ir augšējās kārts skaitlis, un b ir starpība starp divām secīgām kārtīm. Kāršu kava veido aritmētisko progresiju pēc moduļa m. Tas nozīmē, ka kāršu vērtības, sākot skaitīt no augšējās, ir $v_i \equiv a + i \cdot b \pmod{m}$. Sākotnēji $a = 0$ un $b = 1$. Pie tam starpība starp pirmo un pēdējo kārtīm arī ir 1.

$$v_{n^2} + 1 \equiv a + b \cdot n^2 + 1 \equiv a + n^2 + 1 \equiv a \equiv v_0$$

Pēc A operācijas veikšanas augšējās kārts vērtība palielinās par $b \cdot k$. Starpība starp divām secīgām kārtīm nemainās.

$$a' = a + b \cdot k$$

Kārtis pēc B darbības:

$$\begin{array}{ccccc} v_0 & & & & \\ v_{n^2-n+1} & v_{n^2-n+3} & v_{n^2-n+3} & \cdots & v_{n^2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ v_{n+1} & v_{n+2} & v_{n+3} & \cdots & v_{2n} \\ v_1 & v_2 & v_3 & \cdots & v_n \end{array}$$

Atbilstošās vērtības:

$$\begin{array}{ccccc} a & & & & \\ a + (n^2 - n + 1)b & a + (n^2 - n + 2)b & a + (n^2 - n + 3)b & \cdots & a + n^2 b \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a + (n+1)b & a + (n+2)b & a + (n+3)b & \cdots & a + 2nb \\ a + b & a + 2b & a + 3b & \cdots & a + nb \end{array}$$

Pēc B darbības veikšanas divu secīgu kāršu starpība ir $(n^2 - n + 1)b \equiv -nb \pmod{m}$, augšējās kārts vērtība nemainās.

$$b' = (n^2 - n + 1)b$$

Lai noteiktu vērtību k-tajai kārti no augšas, jāaprēķina $a + (k-1)b + 1$. Izmantojot šīs formulas, var uzrakstīt vienkāršu risinājumu, bet uzrakstot to tiešā veidā starprezultāti neietilps 64 bitu mainīgajos. Maksimālās k un b vērība ir aptuveni 10^{12}. Līdz ar to $a + (k-1)b + 1$ vērtība var būt aptuveni 10^{24}, kas ir daudz vairāk, nekā $2^{64} \approx 1.8 \cdot 10^{19}$. Viens no veidiem, kā to atrisināt, ir uzrakstīt funkciju, kas reizina skaitļus pēc moduļa, sadalot vienu no tiem divnieka pakāpēs. Tad starprezultātu vērtības ir ne vairāk kā divas reizes lielākas par operandiem, un pietiek ar 64 bitu mainīgajiem. Cits veids ir ievērot, ka b iespējamās vērtības ir $1, n^2 + 1 - n, n^2, n$. Var aizvietot $n^2 + 1 - n$ un n^2 ar $-n$ un -1. Tad starprezultāti pēc absolūtās vērtības nav daudz lielāki par 10^{18}, kas ietilpst 64 bitu mainīgajā.

Darbības laiks: $\mathcal{O}(d)$ **Tēmas:** skaitļu teorija

Latvijas 28.informātikas olimpiādes trešā posma uzdevumu atrisinājumi

Latvijas informātikas olimpiāde

Stikla domino

Veidosim grafu, kur virsotnes ir tabulas laukumi, bet šķautne starp divām virsotnēm ir tad, ja tās ir divas rūtiņas ar kopīgu malu, kurās pēc noteikumiem var ievietot vienu domino. Tā kā katrai virsotnei šajā grafā ir ne vairāk par diviem kaimiņiem, tad grafs sadalās ceļos un ciklos. Izmantojot kādu grafu apstaigāšanas algoritmu (BFS/DFS), var atrast visas šīs komponentes.

Pieņemsim, ka pavisam ir k komponentes. Lai a_i ir maksimālais domino skaits, ko pēc noteikumiem var izvietot i-tajā komponentē. Lai b_i ir veidu skaits, kā to izdarīt i-tajā komponentē. Tad atbildes uz uzdevumu būs attiecīgi $a_1 + a_2 + \ldots + a_k$ un $b_1 \cdot b_2 \cdot \ldots \cdot b_k$.

Aplūkosim, kā aprēķināt skaitļus a_i un b_i vienai komponentei. Šķirojam divus gadījumus:

- Komponente ir ceļš garumā len. Iespējami divi apakšgadījumi:
  - len ir pāra. Tad to var noklāt ar domino pilnībā, un $a_i = \frac{len}{2}$. Savukārt veidu skaits, kā to izdarīt, ir $b_i = 1$.
  - len ir nepāra. Tad var noklāt visu ceļu, izņemot vienu laukumu. Tāpēc $a_i = \frac{len-1}{2}$. Šis viens brīvs laukums sadala ceļu divos ceļos pāra garumā. Tāpēc tas var būt katrs otrais laukums, sākot no ceļa sākuma, un $b_i = \frac{len+1}{2}$.
- Komponente ir cikls garumā len. Var pierādīt, ka cikls ir noteikti pāra garumā: nokrāsojam laukumu šahā kārtībā, tad ciklā pēc kārtas ir balti un melni laukumi: tāpēc balto un melno laukumu skaits ir vienāds ciklā, un tam ir pāra garums. Līdzīgi kā ceļam pāra garumā, arī šajā gadījumā $a_i = \frac{len}{2}$ un $b_i = 1$.

Darbības laiks: $\mathcal{O}(nm)$ **Tēmas:** grafi, BFS, DFS

Basketbola spēle

Aplūkosim, kā atrast lielāko punktu starpību par labu mājinieku komandai: otrais jautājums ir analoģisks. Apzīmējām i-tās lapiņas skaitļu skaitu ar k_i. Katrai lapiņai atrodam uz tās uzrakstīto skaitļu summu s_i.

Moments, kad komanda iegūst lielāko pārsvaru, pieder tieši vienai lapiņai. Pieņemsim, ka tā ir lapiņa ar numuru i. Tad katrai citai lapiņai ar numuru $j \neq i$ tā ir vai nu pilnībā pirms, vai nu pilnīgi pēc lielākā pārsvara. Tāpēc, ja $s_j \geq 0$, tad izdevīgāk ir ievietot šo lapiņu pirms lapiņas i; ja $s_j < 0$, tad izdevīgāk ir ievietot to pēc lapiņas i. Savukārt i-tajā lapiņā jāatrod tāds skaitļu prefikss, kas dod lielāko skaitļu summu: lai šī summa būtu $pref_i$. Tādā gadījumā atbilde būs vienāda ar $pref_i + \sum_{j \neq i} \max(s_j, 0)$.

Šo izteiksmi var pārveidot uz $(pref_i - \max(s_i,0)) + \sum_j \max(s_j, 0)$. Sarēķinot visas vērtības $pref_i$ un vienu summu $\sum_j \max(s_j, 0)$ sākumā, fiksētam i šī formula dod atbildi konstantā laikā. Atliek izvēlēties labāko atbildi pāri visiem iespējamiem i no 1 līdz n.

Darbības laiks: $\mathcal{O}\left(\sum_{i=1}^{n} k_i\right)$ **Tēmas:** rijīgie algoritmi