Attālums kokā

Eiklīda algoritms. Šis uzdevums ir cieši saistīts ar plaši pazīstamo *Eiklīda algoritmu*. Šis algoritms atrod divu naturālu skaitļu lielāko kopīgo dalītāju (LKD). Vienkāršā algoritma versija ir šāda: ja dotie skaitļi ir x un y, tad vienā solī algoritms no lielāka skaitļa atskaita mazāko. Neizbēgami abi skaitļi kādā brīdī kļūst vienādi, kurā brīdī abu vērtība ir tieši vienāda ar $\mathrm{LKD}(x,y)$.

Šī algoritma ātrdarbība sliktākajā gadījumā var būt $O(\max(x,y))$. Tas var notikt, piemēram, ja $x=1$. Tad algoritms izpildīs $y-1$ soli, līdz abi skaitļi kļūst vienādi ar 1. Par laimi, Eiklīda algoritmam ir zināms paātrinājums: pieņemsim, ka $x<y$; tad tā vietā, lai y aizvietotu ar $y-x$, y aizvieto ar $y \pmod x$. Tādā rezultātā ar vienu operāciju tiek izpildīti visi soļi, kuros x nemaina savu vērtību. Var pierādīt, ka Eiklīda algoritms ar šādu paātrinājumu strādā laikā $O(\log(\max(x,y)))$.

Vispārīgie apsvērumi. Aplūkosim vienu skaitli kokā $\frac{x}{y}$, kas nav saknē, un tā vecāku. Skaitli, kas ir ierakstīts vecāka virsotnē, iegūst sekojošā veidā:

- ja $x>y$, tad vecāka skaitlis ir $\frac{x-y}{y}$;
- ja $x<y$, tad vecāka skaitlis ir $\frac{x}{y-x}$.

Ievērojam, ka naturāli skaitļi, kurus iegūstam vecāka skaitītājā un saucējā, ir tieši tie skaitļi, kurus mēs būtu ieguvuši, ja izpildītu vienu *Eiklīda algoritma* vienkāršās versijas soli uz skaitļiem x un y. Šis algoritms ir plaši pazīstama procedūra, kas diviem skaitļiem atrod to abu lielāko kopīgo dalītāju.

Līdz ar to uzdevums reducējas uz sekojošo uzdevumu: vajag atrast tādus naturālus skaitļus U un V, ka skaitlis $\frac{U}{V}$ ir zemākais priekštecis abiem skaitļiem $\frac{P}{Q}$ un $\frac{R}{S}$ Kalkina-Vilfa kokā; tad atbilde uzdevumam ir attālumu no $\frac{P}{Q}$ līdz $\frac{U}{V}$ un no $\frac{R}{S}$ līdz $\frac{U}{V}$ summa.

Atrisinājums. Pašā sākumā izdalīsim P un Q ar $\mathrm{LKD}(P,Q)$, un R un S ar $\mathrm{LKD}(R,S)$. Tādā veidā strādāsim ar nesaīsināmām daļām mūsu risinājumā.

Tad nesaīsināmai pozitīvai daļai $\frac{A}{B}$ definējam *īpašu priekšteču virkni*. Par $\frac{A}{B}$ *īpašu priekšteci* sauksim tādu priekšteci $\frac{A'}{B'}$, ka:

- ja $A=1$ vai $B=1$, tad $A'=B'=1$;
- citādi, ja $A>B$, tad $A'=A \pmod B$ un $B'=B$;
- citādi, ja $A<B$, tad $A'=A$ un $B'=B \pmod A$.

Par $\frac{A}{B}$ īpašu priekšteču virkni sauksim skaitļu virkni, ko iegūst, atkārtoti ņemot īpašos priekštečus (ieskaitot sākotnējo skaitli). Turklāt katram priekštecim arī izrēķinām, cik Eiklīda vienkāršā algoritma soļus vajag veikt, lai iegūtu šo priekšteci. Īpašo priekšteču virkne simulē Eiklīda paātrināta algoritma darbību (izņemot pēdējo soli, kad pieprasām, lai pēdējais pāris būtu $(1,1)$, nevis $(0,\mathrm{LKD}(A,B))$ vai $(\mathrm{LKD}(A,B),0)$, kā notiktu Eiklīda algoritma beigās – tas ir vajadzīgs, jo Kalkina-Vilfa kokā koka sakne ir $\frac{1}{1}$).

Piemēram, skaitlim $\frac{7}{31}$ īpašo priekšteču virkne (un soļu skaiti) būtu

$$\frac{8}{31} \xrightarrow{4} \frac{8}{3} \xrightarrow{2} \frac{2}{3} \xrightarrow{1} \frac{2}{1} \xrightarrow{1} \frac{1}{1}$$

Ievērosim, ka pēdējais skaitlis šajā virknē vienmēr būs $\frac{1}{1}$. Ievērosim arī, ka Eiklīda algoritma darbības laikā abu skaitļu lielākais kopīgais dalītājs nemainās; un tā kā $\mathrm{LKD}(A, B) = 1$, tad katrs īpašais priekštecis arī būs nesaīsināmā daļa.

Tālāk pierādīsim sekojošo apgalvojumu: ja $\frac{P}{Q}$ un $\frac{R}{S}$ ir divas nesaīsināmās daļas, tad to zemākais kopējais priekštecis $\frac{U}{V}$ Kalkina-Wulfa kokā ir īpašais priekštecis kādai no šīm divām daļām.

Pierādījums: pieņemsim no pretējā, ka $\frac{U}{V}$ nav īpašais priekštecis nevienai no abām daļām. Daļa $\frac{U}{V}$ nevar būt vienāda ar $\frac{1}{1}$, jo tas ir īpašs priekštecis abās virknēs. Tāpēc ir vismaz viens skaitlis, kas ir priekštecis pašai daļai $\frac{U}{V}$, nosauksim to par $\frac{X}{Y}$. Paņemsim augstākos īpašos $\frac{P}{Q}$ un $\frac{R}{S}$ priekštečus tādus, kuriem $\frac{X}{Y}$ vēl ir priekštecis. Pieņemsim, ka $X < Y$ (gadījums $X > Y$ ir analoģisks). Tad pēc īpašo priekšteču definīcijas, jāizpildās $Y = Q$ un $X = P \pmod{Y}$, kā arī $Y = S$ un $X = R \pmod{Y}$. Bet tad vai nu $\frac{P}{Q}$ ir priekštecis $\frac{R}{S}$, vai nu $\frac{R}{S}$ ir priekštecis $\frac{P}{Q}$. Tā ir pretruna, jo sanāk, ka vai nu $\frac{U}{V} = \frac{P}{Q}$, vai nu $\frac{U}{V} = \frac{R}{S}$, un tā kā abi skaitļi $\frac{P}{Q}$ un $\frac{R}{S}$ ir arī īpaši priekšteči savās virknēs, tad $\frac{U}{V}$ ir vienāds ar kādu no īpašiem priekštečiem.

No šī apgalvojuma seko arī algoritms uzdevumam.

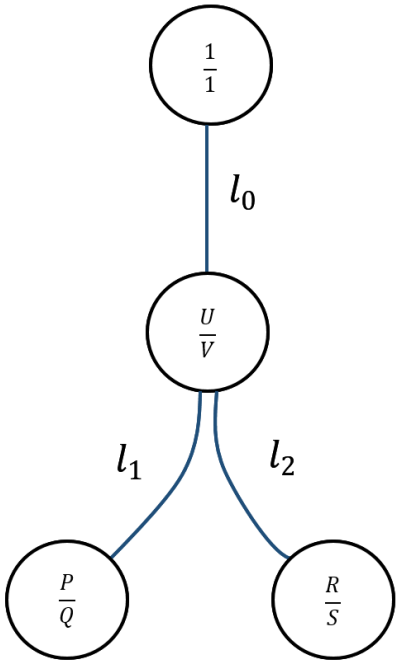
1. Atrodam abiem skaitļiem $\frac{P}{Q}$ un $\frac{R}{S}$ īpašo priekšteču virknes.
2. Katram no viena skaitļa īpašiem priekštečiem pārbaudām, vai tas var būt arī otrā skaitļa īpašais priekštecis. Kā to izdarīt? Ja tā ir tiesa, tad tam vai nu jābūt arī otrā virknē, vai nu tam jābūt kāda otrās virknes skaitļa priekštecim. Piemēram, kādiem skaitļiem $\frac{8}{3}$ vai būt priekštecis? Tie ir vai nu skaitļi formā $\frac{8}{3+8k}$, vai nu skaitļi formā $\frac{8+3k}{3}$ (kur k ir naturāls). Piemēram, ja $\frac{P}{Q} = \frac{8}{31}$ un $\frac{R}{S} = \frac{25}{3}$, tad šo skaitļu īpašo priekšteču virknes ir

$$\frac{8}{31} \xrightarrow{4} \frac{8}{3} \xrightarrow{2} \frac{2}{3} \xrightarrow{1} \frac{2}{1} \xrightarrow{1} \frac{1}{1}$$

$$\frac{17}{3} \xrightarrow{5} \frac{2}{3} \xrightarrow{1} \frac{2}{1} \xrightarrow{1} \frac{1}{1}$$

Tad $\frac{8}{3}$ ir priekštecis $\frac{17}{3}$, jo $\frac{17}{3} = \frac{8+3\cdot 3}{3}$. Šajā gadījumā $\frac{8}{3}$ ir arī zemākais kopējais priekštecis.

3. Izvēlamies no visiem īpašajiem priekštečiem, kuri ir priekšteči abiem skaitļiem $\frac{P}{Q}$ un $\frac{R}{S}$, zemāko – tas arī ir meklētais $\frac{U}{V}$. Attālumu no saknes var izrēķināt, izmantojot soļu skaitu virkni īpašo priekšteču virknēs.
4. Lai iegūtu attālumu starp $\frac{P}{Q}$ un $\frac{R}{S}$, saskaitām visu soļu skaitu abās virknēs, un atņemam divreiz attālumu no saknes līdz $\frac{U}{V}$. To ilustrē 42. zīmējums:



1. zīm.

Attālums starp $\frac{P}{Q}$ un $\frac{R}{S}$ ir vienāds ar

$$l_1 + l_2 = (l_1 + l_0) + (l_2 + l_0) - 2l_0$$

Risinājuma izpildes laika sarežģītība. Atkarībā no realizācijas izpildes laika sarežģītība ir vai nu $O(\log(\max(x, y)))$, vai arī $O(\log(\max(x, y)))^2)$. Abi varianti ir pietiekoši ātri.

Virknes fragments

Uzdevuma risinājumā izmantosim asociatīvu masīvu (piemēram, *map* valodā C++) A, kura elementu atslēgas ir veseli skaitļi un vērtības - nenegatīvi veseli skaitļi. Pievienosim A elementu $A[0] = 0$. Aprēķināsim pirmo $k(1 \leq k \leq N)$ virknes locekļu vērtību summu s_k un katrai no tām pārbaudīsim, vai A neatrodas elements $A[s_k - S]$.

Ja šāds elements atrodas, tad esam atraduši derīgu fragmentu, kas sākas ar virknes locekli, kura indekss ir $A[s_k - S] + 1$, bet beidzas ar k-to virknes locekli. Tātad attiecīgā fragmenta garums ir $k - A[s_k - S]$. Ja šis fragments ir garāks par iepriekš atrastajiem, tad jāsaglabā informācija par to.

Ja A nesatur elementu $A[s_k]$, tad pievienojam to: $A[s_k] = k$.

Elementa esamības asociatīvajā masīvā pārbaude un jauna elementa pievienošana notiek logaritmiskā laikā, tādēļ kopējā risinājuma izpildes laika sarežģītība ir $O(NlogN)$.

Kastu tornis

Lai gan uzdevuma noteikumos ir minēti trīs atšķirīgi kastes novietošanas veidi (uzsēšanās, uzkāršanās un nosegšana), varam būtiski atšķirīgo veidu skaitu samazināt uz diviem - vai kastes pamats sasniedz kādu no iepriekšējām kastēm (uzsēšanās vai nosegšana) vai - nē (uzkāršanās).

Ja aplūkosim tās kastes uz kurām ir iespējams uzlikt vēl kādu kasti, tad redzēsim, ka to malu garumi vienmēr ir sakārtoti augošā secībā no lejas uz augšu (varam pieņemt, ka sākotnējais pamats ir milzīgas kastes augša ar malas garumu, kas pārsniedz lielākās kastes malas garumu). Tās kastes, kas ir nosegtas ar lielāku kasti, vai arī kuras vairs nav pieejamas lielākas kastes uzkāršanās dēļ, nekādi vairs nevar ietekmēt nākamo kastu novietošanu.

Saglabāsim informāciju par kastēm, uz kurām vēl iespējams uzlikt vēl kādu kasti, saglabājot kastes malas garumu un tās augšējās skaldnes augstumu, skaitot no

sākotnējā pamata. Pēc katras kastes uzlikšanas torņa augstums vai nu pieaug, vai arī saglabājas iepriekšējais - tas nevar samazināties kastes uzlikšanas rezultātā.

Ciklā ielasām kārtējās kastes malas garumu $kastes\_mala$ un meklējam augstāko kasti, kura derētu pēc izmēra, lai uz tās šo kasti uzliktu. Meklējot šo kasti varam dzēst informāciju par kastēm, kas ir mazākas par $kastes\_mala$, jo, neatkarīgi no novietošanas veida, šīs kastes vairs nebūs pieejamas kā pamats kādai no nākamajām kastēm. Kad pirmā derīgā kaste atrasta, aprēķinām teorētisko torņa augstumu, ja šī kaste būtu novietota: $augstums_{teorētiskais} = augstums_{derīgs\ kā\ pamats} + kastes\_mala$. Ja šī teorētiskā torņa augstums ir lielāks vai vienāds ar torņa augstumu pirms kastes novietošanas, tad ir notikusi uzsēšanās vai nosegšana un teorētiskais augstums ir reālais torņa augstums. Papildinām ierakstu par šobrīd augstāko kasti (ar malas garumu $kastes\_mala$) ar informāciju par torņa augstumu $augstums_{teorētiskais}$.

Ja teorētiskā torņa augstums ir mazāks par torņa augstumu pirms kastes novietošanas, tas nozīmē, ka ir notikusi uzkāršanās un ieraksts par šobrīd augstāko kasti (ar malas garumu $kastes\_mala$) ir jāpapildina ar informāciju par to torņa augstumu, kāds tas bija pirms kastes novietošanas.

Pēc visu kastu apstrādes atliek izvadīt aprēķināto augstākās kastes līmeni, kas sakrīt ar torņa kopējo augstumu.

```
raksts: {mala: naturāls_skaitlis; augstums: naturāls_skaitlis}
tornis: raksts masīvs
tornis[0] ≔ {∞, 0}
skaits ≔ 0
visa_torņa_augstums ≔ 0
cikls
    ielasa kastes_mala
    cikls
          ja tornis[skaits].mala > kastes_mala
                pārtraukt
          citādi
                skaits ≔ skaits - 1
    teorētiskais_augstums ≔ tornis[skaits].augstums + kastes_mala
    ja teorētiskais_augstums > visa_torņa_augstums
          visa_torņa_augstums ≔ teorētiskais_augstums
    skaits ≔ skaits + 1
    tornis[skaits] ≔ {kastes_mala, visa_torņa_augstums}
izvada visa_torņa_augstums
```

Algoritma pierakstā izmantotā masīva vietā šajā uzdevumā dabīgi ir lietot datu struktūru steks.

Algoritma izpildes laika sarežģītība ir $O(N)$ - lineāra atkarībā no kastu skaita.

Mazākais taisnstūris

Saglabāsim informāciju par divām rūtiņām, kurām koordinātas x vērtības starp visām rūtiņām ir lielākās - t.i., rūtiņu ar vislielāko un otru lielāko x vērtību. Iespējams, ka šīs vērtības sakrīt - t.i., ir divas vai vairāk rūtiņas ar lielāko x vērtību. Šādā gadījumā saglabā informāciju par jebkurām divām rūtiņām ar šo īpašību.

Līdzīgi saglabā informāciju par divām “malējām” rūtiņām ar mazāko x vērtību, ar lielāko y vērtību un ar mazāko y vērtību.

Kad visos virzienos saglabāts pa divām “malējām” rūtiņām, varam aprēķināt, cik liels būtu atlikušās rūtiņas ierobežojošais taisnstūris, ja pašu malējo rūtiņu (tikai vienu!) mēģinātu atstāt ārpusē.

Aplūkosim gadījumu, ka gribam nogriezt rūtiņu ar maksimālo x vērtību. Varam iedomāties, ka griezienu veicam ar vertikālas taisnes palīdzību, ārpusē atstājot tieši vienu rūtiņu.

Ir iespējami vairāki gadījumi:

1) ja ar lielāko x vērtību ir vairākas rūtiņas, tad ar šāda grieziena palīdzību nevar nogriezt tikai vienu rūtiņu. Ierobežojošā taisnstūra laukums nemainās.

2) ir viena rūtiņa ar lielāko x vērtību un

a) šī rūtiņa ir malējā y virzienā (ar lielāko vai mazāko y vērtību). Ja šī ir vienīgā rūtiņa y virzienā ar lielāko (mazāko) vērtību, tad, nogriežot šo rūtiņu, tiks nogriezta arī malējā rūtiņa y virzienā un attiecīgi samazināsies ierobežojošā taisnstūra izmēri abos virzienos - gan x, gan y. Taisnstūra robežas attiecīgajos virzienos noteiks rūtiņas ar otru lielāko (mazāko) koordinātas vērtību.

b) šī rūtiņa nav malējā y virzienā - taisnstūra izmērs mainās tikai x virzienā un lielākā x vērtība ir otrai malējai rūtiņai.

Pēc tam analoģiski aprēķina ierobežojošā taisnstūra izmērus, ja tiek nogriezta viena rūtiņa katrā no atlikušajiem trim virzieniem.

Risinājuma izpildes laika sarežģītība ir $O(N)$.

Sēņošanas čempionāts

Par *s-nogabalu* sauksim nogabalu, kurā ir sēnes.

Sākumā “saspiedīsim” mežu masīvu - pārnumurēsim visas rindas un kolonnas tā, lai nebūtu neviena pilnīgi tukša rinda un kolonna (šo soli var arī nedarīt, bet, to izdarot, pēc tam ir vieglāk domāt un risināt uzdevumu). To darām sakārtojot visus s-nogabalus augošā secībā pēc kolonnas numura un piešķirot tām kolonnu numurus pēc kārtas (protams tiem s-nogabaliem, kam pirms tam bija vienādi numuri, arī pēc piešķiršanas ir jābūt vienādiem numuriem).

Lietosim datu struktūru, ko sauc par segmentu koku. Katra no koka lapām atbilst vienai kolonnai, un tajā glabāsim lielāko skaitu sēņu, kādu var salasīt, ja pēdējais apmeklētais s-nogabals atrodas šajā kolonnā. Segmentu koka virsotnēs glabāsim lielāko vērtību, kāda ir sastopama šīs virsotnes apakškokā.

Sakārtojam visus s-nogabalus pēc rindām augošā secībā un, ja rindas ir vienādas, tad pēc kolonnām augošā secībā. Tagad apstrādāsim nogabalus pa vienam pēc kārtas. Apstrādājot kārtējo s-nogabalu $[x, y]$, mums nepieciešams noskaidrot, cik ir lielākais iespējamais salasāmo sēņu skaits kolonnās $[1, x]$. To noskaidro ar segmentu koka palīdzību, jo no visiem šiem un tikai šiem s-nogabaliem mēs varam nokļūt s-nogabalā $[x, y]$. Pie iegūtās atbildes pieskaitām sēņu skaitu, kas ir s-nogabalā $[x, y]$ un iegūto vērtību ierakstām nogriežņu koka lapā kas atbilst x kolonnai un atjaunojam nogriežņu koka virsotņu vērtības.

Beigās noskaidrojam kāda ir lielākā ierakstītā vērtība nogriežņu koka lapās un tik sēnes arī būs iespējams savākt, ievērojot uzdevuma nosacījumus.

Risinājuma izpildes laika sarežģītība ir $O(NlogN)$.