Sisukord

1. Palindroom (palind)

1 sek 10 punkti

Idee: Tähvend Uustalu, teostus ja lahenduse selgitus: Ahto Truu

> Kas antud 4-elemendilisest arvujadast on võimalik ülimalt ühe elemendi muutmisega saada palindroom?

Olgu jada elemendid a, b, c, d. Siis on jada palindroomiks olemise tingimus $a = d$ ja $b = c$.

Kui need kaks võrdust kehtivad, on jada juba palindroom ja midagi muuta ei ole vaja.

Kui üks võrdus kehtib, aga teine mitte, siis saame teise võrduse kehtima panna, asendades ühe ebavõrdsetest elementidest teisega. Näiteks kui $a = d$, aga $b \neq c$, võime võtta b, c asemel kas b, b (saades jada a, b, b, d, mis on palindroom, kui $a = d$) või c, c (saades jada a, c, c, d, mis on ka palindroom, kui $a = d$).

Kui kumbki võrdus ei kehti, oleks nii ühe kui teise kehtima saamiseks vaja muuta vähemalt üht elementi, seega kokku vähemalt kaht elementi ja sel juhul jada ühe elemendi muutmisega palindroomiks teha ei saa.

Python

```
a, b, c, d = input().split()

if a == d and b == c:
  print("JAH")
  print(a, b, c, d)
elif a == d:
  print("JAH")
  print(a, b, b, d)
elif b == c:
  print("JAH")
  print(a, b, c, a)
else:
  print("EI")
```

C++

```
#include <iostream>

int main() {
  int a, b, c, d;
  std::cin >> a >> b >> c >> d;

  if (a == d and b == c) {
    std::cout << "JAH" << "\n";
    std::cout << a << " " << b << " "
        << c << " " << d << "\n";
  } else if (a == d) {
    std::cout << "JAH" << "\n";
    std::cout << a << " " << b << " "
        << b << " " << d << "\n";
  } else if (b == c) {
    std::cout << "JAH" << "\n";
    std::cout << a << " " << b << " "
        << c << " " << a << "\n";
  } else {
    std::cout << "EI" << "\n";
  }
}
```

2. Mägedes sõitmine (`soit`)

1 sek / 3 sek 20 punkti

Idee: Sandra Schumann, teostus: Kregor Ööbik, lahenduse selgitus: Ahto Truu

> Reisija alustab oma teekonda maapinnal seistes. Edasi on teada nii maapinna kõrgus merepinnast tema teekonna igal meetril kui ka reisija kõrguse muutus teekonna igal meetril. Leida, millistes teekonna punktides on reisija maapinnal, millistes maa kohal viaduktil ja millistes maa all tunnelis.
>
> Alamülesanded: teekonna pikkus 1 meeter, kuni 1 000 meetrit, kuni 100 000 meetrit.

Ülesande tekstis toodud tähistusi kasutades on maapinna kõrgus teekonna alguses H_0 ja edasi iga meetri järel $H_1, \ldots, H_N$ ning reisija kõrguse muutused teekonna igal meetril $D_1, D_2, \ldots, D_N$.

Tähistame nüüd reisija kõrguse k meetri läbimise järel R_k. Kuna on teada, et reisija alustab maapinnalt, peab kehtima $R_0 = H_0$. Edasi on teada, et esimese meetriga muutus reisija kõrgus D_1 võrra. Seega on esimese meetri läbimise järel tema kõrgus merepinnast $R_1 = H_0 + D_1$.

Järelikult piisab esimese alamülesande lahendamiseks $H_0 + D_1$ ja H_1 võrdlemisest. Kui $H_0 + D_1 = H_1$, on reisija endiselt maapinnal. Kui $H_0 + D_1 < H_1$, on reisija nüüd maapinnast madalamal ehk sisenenud tunnelisse. Kui $H_0 + D_1 > H_1$, on reisija nüüd maapinnast kõrgemal ehk tõusnud viaduktile. Täpselt sellised lahendused on toodud võistluse materjalide arhiivis failides `soit/solution/sol_1.py` ja `soit/solution/sol_1.cpp`.

Kui vaatame edasi, siis näeme, et 2 meetri läbimise järel on reisija kõrgus merepinnast $R_2 = H_0 + D_1 + D_2$, 3 meetri läbimise järel $R_3 = H_0 + D_1 + D_2 + D_3$, j.n.e. Üldiselt on reisija kõrgus merepinnast k meetri läbimise järel seega $R_k = H_0 + \mathrm{SUM}(D_1, D_2, \ldots, D_k)$. Tema asendi maapinna suhtes saame endiselt R_k ja H_k võrdlemisega. Failides `sol_sum_loop.py` ja `sol_sum_loop.cpp` toodud lahendustes on $D_1, D_2, \ldots, D_k$ summeerimine tehtud ise kirjutatud kordustega, failides `sol_sum_std.py` ja `sol_sum_std.cpp` toodud lahendustes aga standardteegi funktsioone kasutades.

Nende lahenduste puudus on, et $D_1, D_2, \ldots, D_k$ summeerimine on umbes k tehet, üksõik, kas me teeme seda ise kirjutatud kordusega (kus see on ilmsem) või standardteegi funktsiooni kasutades (kus see paistab ühe käsuna ja kus selle töömaht võib kergemini kahe silma vahele jääda). Kui seda summat on vaja arvutada $k = 1, 2, \ldots, N$ jaoks, siis on töömaht kokku umbes $N^2/2$ tehet, mis suuremates testides jääb liiga aegalaseks.

Parema lahenduse saamiseks tuleb tähele panna, et me võime reisija asukoha kõrguse R_k avaldada ka teisiti: $R_k = R_{k-1} + D_k$. Nii saame iga järgmise R_k väärtuse arvutada eelmisest ainult ühe liitmisega ja kõigi $k = 1, 2, \ldots, N$ peale kokku kulub ainult N tehet. Nii ongi tehtud failides `sol.py` ja `sol.cpp` toodud lahendustes, mis töötavad piisavalt kiiresti kõigis testides (ja teeniks maksimumpunktid, kui keegi žürii lahenduste eest punkte annaks).

3. Kulude jagamine (`kulud`)

1 sek / 3 sek 30 punkti

Idee ja teostus: Heno Ivanov, lahenduse selgitus: Ahto Truu

> N sõpra tegid M ühist kulutust. Iga kulutuse eest maksis üks sõpradest. Leida minimaalse kogusummaga pangaülekannete komplekt, millega sõbrad saavad kulud võrdselt jagada.
>
> $N \leqslant 50\,000$, $M \leqslant 50\,000$.

Alamülesanne $M = 1$

Üks sõber maksis summa S, keegi teine ei maksnud midagi. Võrdselt jagades peaks iga sõbra kulu olema S/N. See tähendab, et kõik teised peavad maksnule just nii palju üle kandma. Siis on iga ülekande tegija kulutanud S/N ja maksnu pärast S kulutamist $(N-1) \cdot S/N$ tagasi saanud, ehk ka tema kulu on $S - (N-1) \cdot S/N = S/N$.

Alamülesanne $M = 2$

Kui mõlema kulu eest maksis sama sõber, siis on ta kokku kulutanud $S = S_1 + S_2$ ja teised keegi mitte midagi. Sel juhul sobib eelmise alamülesande lahendus.

Kui kulud tegid erinevad sõbrad, siis nimetame esimeseks maksnuks seda, kes kulutas vähem, ja teiseks maksnuks seda, kes kulutas rohkem. Lahendus sõltub siis sellest, kas esimese maksnu kulu oli täpselt S/N või alla või üle selle. (Mõtle järele, miks kunagi ei saa olla nii, et mõlemad maksnud kulutasid alla S/N!)

Kui esimese maksnu kulu oli täpselt S/N, siis on ta kulutanud täpselt oma osa ja kõik ülejäänud peavad tegema ülekanded teisele maksnule eelmise alamülesandega sama loogika järgi. Kui esimese maksnu kulu oli alla S/N, siis peab tema ka teisele maksnule nii palju üle kandma, et tema kulu oleks kokku S/N.

Kui mõlemad maksnud kulutasid üle S/N, siis läheb lahendus keerulisemaks, sest siis peavad osad mittemaksnud tegema ülekande esimesele ja osad teisele maksnule ja võib juhtuda, et üks mittemaksnu peab tegema ülekanded mõlemale maksnule. Seda võiks lahendada nii, et arvestada välja, mitu mittemaksnut peavad tegema S/N summas ülekanded esimesele ja mitu teisele maksnule ja siis vajadusel lisada ühe mittemaksnu väiksemad ülekanded kahele maksnule.

See kõik kokku läheb päris keeruliseks ja seda niisugusel kujul nelja punkti nimel programmeerida pigem ei tasu ära.

Alamülesanne $N \leqslant 100$

Küll aga võiks eelmise alamülesande läbimõtlemine aidata teha selle alamülesande lahendamiseks vajalikku tähelepanekut: igaüks, kes maksis vähem kui S/N, peab puuduva osa juurde maksma, ja igaüks, kes maksis rohkem, peab üle makstud osa tagasi saama.

Maksjate ja saajate paari panemiseks on kõige lihtsam kõik sõprade paarid läbi vaadata ja iga kord, kui leiame ühe sõbra, kes on maksnud liiga vähe ja ühe, kes on maksnud liiga palju, lasemegi vähem maksnul teisele ülekande teha. Ülekande summat piiravad seejuures kaks tegurit: (a) kui palju vähem maksnul on vaja juurde maksta ja (b) kui palju rohkem maksnul on vaja tagasi

taasa. Ülekanne ei tohi neist kummastki suurem olla, sest muidu läheb osa raha valesse kohta ja seda peab pärast uuesti liigutama. Siis pole kõigi ülekannete kogusumma enam vähim võimalik.

Üle- ja puudujääkide arvestamiseks on kõige lihtsam pidada iga sõbra kohta meeles tema tehtud kulude saldot. Alguses andmete sisse lugemisel liidame iga tehtud kulu selle maksnu saldole. Hiljem ülekandeid tehes liidame üle kantud summa maksja saldole (sest see on tema jaoks kulu) ja võtame selle saaja saldolt maha (sest tema kulud on nüüd selle võrra väiksemad).

Lisapiiranguteta alamülesanne

Eelmises lõigus kirjeldatud lahendus annab kõigis testides õiged vastused, aga kulutab suurematest testides liiga palju aega, sest vaatab läbi kõik sõprade paarid, mida $N = 50\,000$ sõbra korral on $N^2 = 50\,000^2 = 2\,500\,000\,000$.

Parema lahenduse saamiseks paneme tähele, et ülekande tegemisel võib maksjaks olla ainult selline sõber, kelle tehtud kulude summa on liiga väike, ja saajaks ainult selline sõber, kelle tehtud kulude summa on liiga suur. Üks võimalus oleks jagada sõprade nimekiri kaheks: saajad ja maksjad. Teine võimalus on käia üht nimekirja läbi parallelselt kahe järjehoidjaga, millest esimene peatub ainult maksjatel ja teine ainult saajatel.

Vaadates failides `sol_4.py` ja `sol_4.cpp` olevaid `while`-kordusi võime panna tähele, et korduse igal läbimisel juhtub üks kolmest asjast: (a) järjehoidja x liigub edasi, sest selle näidatav sõber pole maksja, (b) järjehoidja y liigub edasi, sest selle näidatav sõber pole saaja või (c) tehakse ülekanne. Veel võime tähele panna, et (a) järjehoidja x liigub edasi maksimaalselt N korda, sest siis läheb ta nimekirjast välja ja kordus lõpeb, (b) järjehoidja y liigub edasi maksimaalselt N korda, sest siis läheb ta nimekirjast välja ja kordus lõpeb, (c) ülekannete arv ei saa olla üle N, sest iga ülekandega väheneb ühe võrra kas maksjate arv või saajate arv või isegi mõlemad. Seega ei saa selle korduse läbimiste arv kuidagi olla suurem kui $3 \cdot N$, mis maksimaalses testis on $3 \cdot 50\,000 = 150\,000$ ja mahub lahedasti ajalimiitidesse.

4. Kaustad (`kaustad`)

1 sek / 3 sek 40 punkti

Idee ja teostus: Andres Alumets, lahenduse selgitus: Andres Alumets ja Tähvend Uustalu

> Tahame navigeerida kaustade hierarhias. Tahame liikuda ühe teatud failini, selleks peame i-ndal tasemel valima järjekorras K_i-nda kausta; kokku on sel tasemel M_i kausta. Lisaks on igal tasemel kaustadest ühe sammu võrra üleval nupp, mis võimaldab sel tasemel kaustade järjekorra ümber pöörata. Kursori ühe sammu liigutamine võtab 1 sekund. Leia kiireim viis soovitud failini jõuda.
>
> $N \leqslant 10^5$, $K_i, M_i \leqslant 10^9$ iga i korral.

Arvutame igal taseme kohta välja, mitu sekundit on minimaalselt kulunud sellel tasemel sobiva kaustani jõudmiseks, juhul kui:

- sel tasemel kaustade järjekorda ei pööratud;
- sel tasemel pöörati kaustade järjekord.

Esiteks vaatame, kuidas neid väärtuseid arvutada esimesel tasemel.

- Kui kaustade järjekorda ei pöörata, peame liikuma 1. positsioonilt K_1. positsioonile. Selleks kulub $K_1 - 1$ sekundit.
- Kui kaustade järjekorda pööratakse, peame liikuma 1. positsioonilt järjekorramuutmisnupule. Selleks kulub 1 sekund. Pärast pööramisnupu klõpsamist on soovitud kaust $M_1 - K_1 + 1$. positsioonil. Sama palju sekundeid kulub sellele positsioonile liikumiseks: kokku $M_1 - K_1 + 2$ sekundit.

Olgu T_i i-ndal tasemel soovitud kausta indeks pärast seda, kui sel tasemel kaustade järjestus ümber pööratakse. On selge, et $T_i = M_i - K_i + 1$.

Iga järgmise taseme kohta arvutame need väärtused eelmise taseme põhjal. Vaatleme mingit konkreetset taset; olgu e_i ja e_p minimaalne ajakulu, et jõuda eelmisel tasemel soovitud kaustani vastavalt nii, et viimasel tasemel ei pöörata ja nii, et pööratakse. Võtame eesmärgiks arvutada välja samad väärtused praeguse taseme jaoks, tähistame need vastavalt p_i ja p_p.

Et arvutada p_i paneme tähele, et:

- Kui eelmisel tasemel pööret ei tehtud, siis kulus meil praeguse tasemeni jõudmiseks e_i sekundit ja kursor asub hetkel positsioonil K_{i-1}. Peame liigutama kursori positsioonile K_i; kokku kulub $e_i + |K_{i-1} - K_i|$ sekundit.
- Kui eelmisel tasemel tehti pööre, siis kulus meil praeguse tasemeni jõudmiseks e_p sekundit ja kursor asub positsioonil T_{i-1}. Analoogiliset eelmisega kulub kokku $e_p + |T_{i-1} - K_i|$ sekundit.

Et tahame p_i väärtust võimalikult väiksena hoida valime selle variandi, mis vähem aega võtab: $p_i = \min(e_i + |K_{i-1} - K_i|, e_p + |T_{i-1} - K_i|)$.

Nüüd arvutame analoogiliselt p_p.

- Kui eelmisel tasemel pööret ei tehtud, siis kulus praeguse tasemeni jõudmiseks e_i sekundit ja kursor asub positsioonil K_{i-1}. Peame liigutama kursori ümberpööramisnupuni, selleks kulub K_{i-1} sekundit. Nüüd on soovitav kaust positsioonil T_i ja kursori sinna liigutamiseks kulub sama palju sekundeid. Kokku läheb aega $e_i + K_{i-1} + T_i$ sekundit.
- Kui eelmisel tasemel tehti pööre, siis analoogiliselt kõikide eelmistega kulub kokku $e_p + T_{i-1} + T_i$ sekundit.

Jällegi võtame miinimumi: $p_p = \min(e_i + K_{i-1} + T_i, e_p + T_{i-1} + T_i)$.

Vastuseks on lihtsalt $\min(p_i, p_p)$, kus vaatame viimasele tasemele vastavaid väärtuseid. Iga taseme kohta arvutame välja kaks väärtust, mille välja arvutamiseks kulub $O(1)$ aega. Kuna tasemeid on N tükki, siis on läheb kokku $O(N)$ aega.

Implementeerides tuleb arvestada, et vastus ei pruugi mahtuda 32-bitiste täisarvude sisse. Seega tasub kasutada 64-bitiseid (nt. C++ `long long`).

5. Kontrollsumma (`summa`)

2 sek 60 punkti

Idee: Heno Ivanov, teostus ja lahenduse selgitus: Peeter Aleksander Randla

Kontrollsumma on summa kujul

$$(A_1 \cdot K_1 + A_2 \cdot K_2 + \ldots + A_N \cdot K_N + A_{N+1} \cdot K_1 + \ldots) \bmod 10,$$

kus K on pikkusega N ja perioodiline. Antud on must kast, mis võtab sisendiks jada A ja väljastab selle kontrollsumma. Leia K väärtused ja N. Teada on, et $N \leqslant 1\,000$.

Tähistagu $M = 1\,000$ maksimaalset võimalikku N väärtust. Esiteks märkame, et N väärtuse leidmiseks piisab leida jada K esimesed $2M$ liiget. Selle formaalne tõestamine on keerulisem (kes tahab, võib seda koduülesandena proovida), aga $2M$ liikme hulgas leidub K juba vähemalt 2 korda terves ulatuses, seega saab veenduda, et proovitud N väärtus on õige. Samas leidub jadasid, mille puhul on pikkuse teada saamiseks kindlasti vaja vähemalt $2M-2$ väärtust (näiteks jadad $(1, 1, \ldots, 1, 2)$ pikkusega 999 ja $(1, 1, \ldots, 1, 2, 1)$ pikkusega 1000 ühtivad esimese 1997 liikme juures).

Selleks, et leida K väärtusi, võib esimese katsena anda sisendisse päringuid kujul $A_1 = (1)$, $A_2 = (0, 1)$, $A_3 = (0, 0, 1)$ jne. Siis taanduvad kontrollsummas kõik liikmed peale ühe välja ja i-ndas päringus alles jääv liige ongi lihtsalt K_i. Selleks, et leida $2M$ jada K väärtust, on kokku vaja testimissüsteemile umbes $(2M)^2/2 = 2\,000\,000$ väärtust saata, mis mahub veel ajalimiidi sisse.

Sellise lahenduse eest saab aga vaid 75% punktidest, kuna see kasutab päringutes nulle.

Üks võimalik viis päringutes nullide vältimiseks on esiteks pärida jada $A_0 = (1, 1, 1, \ldots, 1)$ (pikkusega $2M$) kontrollsummat, olgu see S_0. Edasi pärime jada $A_1 = (2, 1, 1, \ldots, 1)$, siis $A_2 = (1, 2, 1, \ldots, 1)$ jne kuni $A_{2M} = (1, 1, 1, \ldots, 2)$. Olgu nende kontrollsummad $S_1, S_2, \ldots, S_{2M}$. Siis

$$S_i - S_0 = (K_1 + \ldots + 2K_i + \ldots + K_{2M}) - (K_1 + \ldots + K_i + \ldots + K_{2M}) = K_i$$

Seega K_i väärtuse saab kätte ühe lahutustehtega. Tuleb tähele panna, et kuna kontrollsumma arvutatakse mod 10, siis tuleb ka lahutustehe teha mod 10.

Selline lahendus aga peab viimase alamülesande läbimiseks testimissüsteemile saatma kokku ligikaudu $(2M)^2 = 4\,000\,000$ väärtust, mis on liiga aeglane. Samas kuna progamm ei tea jooksmise ajal seda, millises alamülesandes hetkel ollakse, võib juhtuda, et punkte ei saada isegi varasematest alamülesannetest, kuna programm üritab siiski lahendada ka viimast alamülesannet. Seega tuleb alamülesannetest punktide saamiseks käsitsi oma programmis päringute arvu vähendada.

Kiiremini saab siis, kui teha päringud kujul $A_1 = (1)$, $A_2 = (1, 1)$, $A_3 = (1, 1, 1)$, jne. Olgu vastavad kontrollsummad $S_1, S_2, S_3, \ldots$ Siis $K_1 = S_1$ ja edasi iga i korral $K_i = S_i - S_{i-1}$, kuna jälle kõik varasemad kontrollsumma liikmed taanduvad välja (ka siin tuleb tähele panna, et tehted toimuvad mod 10). Selline lahendus kasutab $(2M)^2/2 = 2\,000\,000$ väärtust, mis on viimase alamülesane läbimiseks piisavalt kiire.

Viimaseks vaatame, kuidas esimesest $2M$ K väärtusest leida perioodi tegelik pikkus N. Kuna M on nii väike, siis võib selle jaoks lihtsalt kõik võimalikud N väärtused läbi proovida ja kontrollida, kas antud pikkus klapib jadaga või mitte, s.t. kas kõik väärtused leitud jadas alates väärtusest $N + 1$ on samad mis esimesed N väärtust. Kui seda teha kasvavas järjekorras (s.t. alustades $N = 1$-st), siis leitakse esimesena kõige lühem võimalik periood, ja piisab seda kasutada.

6. Metroovõrgu tsoonid (`metroo`)

1 sek / 3 sek 80 punkti

Idee, teostus ja lahenduse selgitus: Tähvend Uustalu

> Tasandile on joonistatud graaf N tipu ja M servaga. Jaota graafi tipud võimalikult mitmeks mittetühjaks tsooniks nii, et iga k korral on tsoonidest $1, 2, \ldots, k$ moodustuv graaf sidus. Tsoonid peavad olema sõõrid, millest igaühe keskpunkt asub punktis $(0, 0)$.
>
> $N, M \leqslant 3 \cdot 10^5$.

Esimese asjana olgu öeldud, et sellistes ülesannetes ja programmeerimisvõistlustel üldiselt on alati mõistlik opereerida täisarvudega nii palju kui vähegi võimalik. Näiteks on punkti (x, y) kaugus punktist $(0, 0)$ Pythagorase teoreemi järgi teatavasti $\sqrt{x^2 + y^2}$. Enamuse täisarvuliste x ja y korral ei ole see kaugus täisarv. Tegu on mingi reaalarvuga, mille täpseks esituseks on üldiselt vaja kirja panna lõpmata palju kahendkohti (või kümnendkohti). Seega tekivad nendega opereerides paratamatult ümardamisvead. See ei tähenda, et selliste arvudega opereerimine täiesti kasutu oleks — kaugel sellest — aga võistlusprogrammeerimise kontekstis kipuvad ümardamisvead nii palju kuhjuma, et vastus muutub ebatäpseks või täiesti kasutuks.

Kõige parem on seega ruutjuure välja arvutamist ja ujukomaarvudega opereerimist täielikult vältida. Selles ülesandes on see õnneks võimalik. Näiteks näeme peagi, et peame sorteerima punktid selle järgi, kui kaugel nad on keskpunktist. Selleks ei ole meil vaja arvutada punktide *kaugusi* keskpunktist — piisab, kui arvutame *kauguste ruudud*. Punktide $(0, 0)$ ja (x, y) kauguse ruut on $x^2 + y^2$, mis on alati täisarv. See on ka põhjus, miks lahendusena palutakse väljastada mitte ringjoonte raadiused, vaid raadiuste ruudud.

Sorteerime punktid kauguse järjekorras, st. nii, et nullpunktile lähim punkt on kõige esimene. Käime punktid selles järjekorras läbi ja kontrollime pärast iga punkti u külastamist, kas juba läbi käidud punktidest moodustub sidus graaf. Kui moodustub, siis lisame vahetult tipust u väljapoole tsoonipiiri.

See on peaaegu lahendus, aga sellel on kaks viga:

- On võimalik, et mitu punkti asuvad nullpunktist sama kaugel. Sel juhul võib juhtuda, et lisame pärast nendest esimese külastamist tsoonipiiri. Sellesse tsooni kuuluvad aga ka ülejäänud sama kaugel olevad punktid, mida me sidususe kontrolli juures ei arvestanud. Selle vea vältimiseks lisame tsoonipiire ainult siis, kui järjekorras järgmise punkti kaugus nullpunktist on praeguse punkti kaugusest rangelt suurem.
- See lahendus lisab tsoonipiiri ka kõige lõppu. Ülesande teksti järgi ei ole seda vaja teha.

Võttes arvesse neid parandusi, saame uue lahenduse.

- Käime graafi tipud läbi kauguse järjekorras; iga u kohta:
  - Kui u ei ole järjekorras viimane ning järjekorras tipust u järgmine tipp on nullpunktist rangelt kaugemal kui u:
    - Kui läbi käidud tippudest moodustuv graaf on sidus:
      - Lisame vahetult tipust u väljapoole tsoonipiiri, st. kui u koordinaadid on (x, y), siis lisame ringi, mille raadiuse ruut on $x^2 + y^2 + 1$.

Lahendus on nüüd korrektne, aga ei ole selge, kuidas kontrollida, kas läbi käidud tippudest moodustuv graaf on sidus. Tuleb arvestada, et seda võib olla vaja teha N korda. Kui me iga kord hakkaksime uuesti sügavuti läbimist vmt. tegema, oleks keerukus $O(N^2)$, mis on liiga aeglane.

Aga saab ka paremini. On olemas andmestruktuur, mille abil on võimalik kiiresti teha järgmisi operatsioone:

- Lisa serv tippude u ja v vahele.
- Leia tippu u sisaldava sidususkomponendi suurus.

Seda andmestruktuuri on eesti keeles nimetatud *klassimetsaks* või *lõikumatute hulkade süsteemiks*. Inglise keeles kasutatakse väljendeid *disjoint set union* (DSU) ja *union-find*. Klassimetsade kohta saab lähemalt lugeda siit. Algoritm vastab igale päringule $O(\alpha(N))$ ajas. Siin α on nn Ackermanni funktsiooni pöördfunktsioon. Tegemist on väga aeglase kasvuga funktsiooniga: $\alpha(n) \leqslant 4$ iga $n \leqslant 10^5$ korral.

Paneme selle andmestruktuuri oma lahendusega kokku ja saamegi piisavalt kiire lahenduse.

- Käime graafi tipud läbi kauguse järjekorras; iga u kohta:
  - Tipu u iga naabri v kohta:
    - Kui v on juba töödeldud, lisame serva u ja v vahele.
  - Kui u ei ole järjekorras viimane ning järjekorras tipust u järgmine tipp on nullpunktist rangelt kaugemal kui u:
    - Kui tipu u sidususkomponendi tippude arv on võrdne läbi käidud tippude arvuga:
      - Lisame vahetult tipust u väljapoole tsoonipiiri, st. kui u koordinaadid on (x, y), siis lisame ringi, mille raadiuse ruut on $x^2 + y^2 + 1$.

Keerukus $O(N \log N)$: sorteerimine domineerib.