1. Kipsplaadid

1 sekund 20 punkti

Selle ülesande lahendamiseks on kõigepealt vaja panna tähele, et kui ülemise plaadi serv on pikem kui temaga samasuunaline alumise plaadi serv, siis peab ülemise plaadi selle serva vähemalt üks ots alumise plaadi ääre kohale rippuma jääma.

Edasi paneme tähele, et ülemise plaadi keeramisel 90° võrra muutub ülemise plaadi iga serv samasuunaliseks alumise plaadi nende servadega, millega ta enne samasuunaline ei olnud, aga 180° võrra keeramine viib meid esialgsesse olukorda tagasi. Seega on vaja vaadelda ainult ülemise plaadi kahte omavahel 90° võrra keeratud asendit. Mõlemat asendit on vaja vaadelda nii esimese plaadi teise peale kui ka teise plaadi esimese peale asetamisel. Kõigi nende tingimuste ükshaaval väljakirjutamine annabki failides `kipslah1.pas`, `kipslah1.c`, `kipslah1.cpp`, `kipslah1.java` ja `kipslah1.py` toodud lahendused.

Kuna esimese plaadi teise peale ja teise plaadi esimese peale panekul vajalikud kontrollid on samad, võib need vormistada eraldi funktsioonina, mida kutsutakse välja kaks korda, andes plaadid ette kord ühes, kord teises järjekorras, nagu on tehtud failides `kipslah2.pas`, `kipslah2.c`, `kipslah2.cpp`, `kipslah2.java` ja `kipslah2.py`.

Teine võimalus tingimuste korduvat väljakirjutamist vähendada on panna tähele, et kui plaadid üldse üksteise peale mahuvad, siis mahuvad nad kindlasti ka selles asendis, kus nende lühemad servad on samasuunalised. Seega võime kõigepealt mõlemad plaadid keerata näiteks sedapidi, et lühema serva pikkus on antud esimesena ja pikema serva pikkus teisena ning teha mahtumise kohta otsused selle asendi põhjal. Nii ongi tehtud failides `kipslah3.pas`, `kipslah3.c`, `kipslah3.cpp`, `kipslah3.java` ja `kipslah3.py` toodud lahendustes.

Muidugi on veel palju muid võimalusi samaväärseid tingimusi kirja panna. Näiteks Pyhtonis oleks võimalik plaatide servade pikkused järjestada kohe andmete lugemisel, kirjutades

```
map(int, f.readline().split())
```

asemel

```
sorted(map(int, f.readline().split()))
```

Testid

1. Esimene plaat ükskõik kumbapidi teise peale. Väiksem plaat ruudukujuline.
2. Esimene plaat otse teise peale. Mõlemal plaadil pikem külg enne.
3. Esimene plaat pöörates teise peale. Esimesel plaadil pikem, teisel lühem külg enne.
4. Teine plaat ükskõik kumbapidi esimese peale. Suurem plaat ruudukujuline.
5. Teine plaat otse esimese peale. Mõlemal plaadil lühem külg enne.
6. Teine plaat pöörates esimese peale. Esimesel plaadil lühem, teisel pikem külg enne.
7. Ükskõik kumb plaat otse teise peale. Mõlemal plaadil pikem külg enne.
8. Ükskõik kumb plaat pöörates teise peale. Esimesel plaadil lühem, teisel pikem külg enne.
9. Ei saa, esimene plaat liiga pikk. Esimesel plaadil pikem, teisel lühem külg enne.
10. Ei saa, teine plaat liiga pikk. Mõlemal plaadil lühem külg enne.

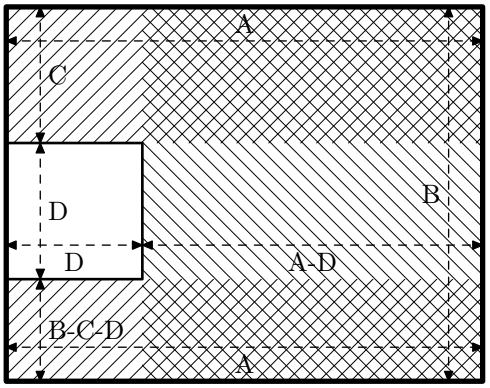
10 testi, à 2 punkti, kokku 20 punkti.

2. Kirjutuslaud

1 sekund 20 punkti

Selle ülesande lahendamiseks on kasulik kõigepealt tähele panna, et kui lauda ei ole lubatud paigutada viltu, peab ta asuma tervenisti ühes kolmest alloleval joonisel diagonaalselt viirutatud ristkülikust: mõõtudega $A \times C$ ukesealast põhja pool, mõõtudega $(A - D) \times B$ uksealast ida pool või mõõtudega $A \times (B - C - D)$ uksealast lõuna pool.

JAH/EI vastuse leidmine taandub seega põhikooli rühma kipsplaatide paigutamise ülesande juures toodud ideedele.



Laua koordinaatide leidmiseks on lisaks vaja teha üsna vähe. Kui kaks ristkülikut mahuvad üksteise sisse, siis saab sisemise alati nihutada välimise nurka. Seega võime laua loodenurga koordinaatideks alati väljastada vastava toaosa loodenurga koordinaadid. Kagunurga koordinaatide leidmiseks on lisaks vaja jätta meelde, kas laud mahtus toaosasse otse või pöörates. Selle põhjal saame teada, kumb laua küljepikkus on lõppasendis põhja-lõuna ja kumb ida-lääne suunaline.

Failides `laudlah1.pas`, `laudlah1.c`, `laudlah1.cpp`, `laudlah1.java` ja `laudlah1.py` toodud lahendused nii töötavadki. Muidugi võiks ka neid lahendusi struktureerida mitmel erineval moel.

Testid

1. Laud mahub pööramata väikese varuga uksest põhja poole.
2. Laud mahub pöörates väikese varuga uksest põhja poole.
3. Laud mahub pööramata väikese varuga uksest lõuna poole.
4. Laud mahub pöörates väikese varuga uksest lõuna poole.
5. Laud mahub pööramata väikese varuga uksest ida poole.
6. Laud mahub pöörates väikese varuga uksest ida poole.
7. Laud mahub pööramata täpselt uksest põhja poole.
8. Laud mahub pöörates täpselt uksest põhja poole.
9. Laud mahub pööramata täpselt uksest lõuna poole.
10. Laud mahub pöörates täpselt uksest lõuna poole.
11. Laud mahub pööramata täpselt uksest ida poole.
12. Laud mahub pöörates täpselt uksest ida poole.
13. Ukseala ei olegi ($D = 0$); ruudukujuline laud mahub ruudukujulisse tuppa.
14. Ukseala ulatub idaseinani ($D = A$); laud mahub põhja või lõunasse.
15. Uks otse vastu põhjaseina ($C = 0$); laud mahub lõunasse või itta.
16. Uks otse vastu lõunaseina ($C + D = B$); laud mahub põhja või itta.
17. Ei saa; laua pikkus ületab toa pikkuse.
18. Ei saa; laua laius ületab toa laiuse.
19. Ei saa; toa pikkus ida-lääne suunas; uksest põhja ja lõuna pool ruumi vähem kui laua laius; uksest ida pool ruumi rohkem kui laua laius, aga laua pikkus ületab toa laiuse.
20. Ei saa; toa pikkus põhja-lõuna suunas; uksest ida pool ruumi vähem kui laua laius; uksest põhja ja lõuna pool ruumi rohkem kui laua laius, aga laua pikkus ületab toa laiuse.

20 testi, à 1 punkt, kokku 20 punkti.

3. Tabelarvutus (põhikooli variant)

1 sekund 40 punkti

Selle ülesande lahendamiseks tasub kõigepealt peast välja arvutada mõned tähiste ja numbrite vastavused:

A...Z on 1...26	See on ülesande tekstis antud.
AA...AZ on 27...52	AA on kohe Z järel ja AA...AZ arv peab olema sama, mis A...Z arv, sest esimene A on konstant ja teine täht käib terve tähestiku läbi.
xA...xZ on $26 \cdot x + 1 \ldots 26 \cdot x + 26$	xA on kohe $(x-1)$Z järel ja xA...xZ arv peab olema sama, mis A...Z arv.

Tähtedest arvuks teisendamise reegli saame eelneva põhjal üsna vahetult. Kui sisend on ühetäheline, siis on vastuseks selle tähe järjekorranumber tähestikus. Kui sisend on kahetäheline, siis on vastuseks esimese tähe järjekorranumber korda 26 pluss teise tähe järjekorranumber.

Arvust tähtedeks teisendamine on vaid veidi keerulisem. Muidugi on vastus ühetäheline, kui veeru järjekorranumber N ei ületa 26, ja kahetäheline, kui ületab. Kui N ületab 26, siis peab esimese tähe järjekorranumber tähestikus olema $(N-1)$ div $26+1$ (kus div tähistab täisarvulist jagamist; näiteks 39 div 26 = 1, aga mitte 1,5). Teise tähe järjekorranumber tähestikus on $(N-1) \bmod 26 + 1$ (kus mod tähistab jäägi leidmist; näiteks $39 \bmod 26 = 13$).

Soovitame vaadata ka selle ülesande algajate rühma variandi lahendust, mis ei eelda, et veeru tähis on ülimalt kahetäheline.

Testid

Testid 1–8 on ühetähelised, testides 9–20 on kahetähelised tähised.

Testides 1–4 testitakse piirjuhte A, Z, 1, 26.

Testides 9–16 testitakse piirjuhte AA, ZZ, 27, 702, AZ, 52, ZA, 677.

Ülejäänud testid on juhuslikud.

20 testi, à 2 punkti, kokku 40 punkti.

4. Tabelarvutus (algajate variant)

1 sekund 40 punkti

Lahenduse leidmisel aitab tähelepanek, et veerutähised on sisuliselt 26-süsteemi arvud, kus numbritena on kasutusel tähed A…Z. Ainus mittestandardne asjaolu on, et tähtede väärtused pole 0…25, nagu positsioonilises 26-arvusüsteemis tavaks, vaid 1…26.

Tähtedest arvuks teisendamine käib nagu positsioonilises arvusüsteemis arvu väärtuse leidmine ikka. Kõige lihtsam on seda programmeerida Horneri skeemi abil: kõigepealt võtame $N := 0$; veerutähise V iga tähe V_i korral (vasakult paremale) omistame $N := N \cdot 26 + V_i$.

Arvust tähtedeks teisendamisel tuleb, nagu iga teise positsioonilise arvusüsteemi puhul, tähele panna, et järjekorranumbri 26-ga mittejaguva osa saame paigutada vaid parempoolseimasse tähte. Erinevusena standardsest arvusüsteemist tuleb 26-ga jaguva veerunumbri korral parempoolseks täheks võtta väärtusele 26 vastav Z, sest väärtusele 0 vastavat märki meil ei ole. Kompaktse valemiga saab märkida, et veerutähise parempoolseilm märk peab olema see, mille järjekorranumber tähestikus on $(N-1) \bmod 26 + 1$. Viimasest märgist vasakule jääv tähiseosa peab esitama kogu ülejäänud väärtust, mis tähendab, et pärast omistamist $N := (N-1) \operatorname{div} 26$ saame tähise eelviimase märgi järjekorranumbri kätte jälle valemiga $(N-1) \bmod 26+1$. Sarnaselt jätkates leiame ka ülejäänud tähed.

Alternatiivne lahendus oleks hakata genereerima kõikidele järjekorranumbritele vastavaid tähekombinatsioone alates vastavusest $1 = A$ ja lõpetada, kui oleme jõudnud sisendis antud arvu või tähiseni. Kuna jooksvalt on vaja ainult viimast leitud tähekombinatsiooni, siis mälupuudust ei teki. Küll aga jääks selline lahendus suuremates testides ajahätta.

Testid

Testides 1–4 testitakse ühetähelisi piirjuhte A, Z, 1, 26.

Testides 7–8 testitakse piirjuhte AA, 27.

Testid 9–10 on maksimaalsed 1 000 000 000, CFDGSXL.

Testid 11–14 on maksimaalsed piirjuhud AAAAAAA, ZZZZZZ, 321 272 407, 321 272 406.

Ülejäänud on testid on juhuslikud.

20 testi, à 2 punkti, kokku 40 punkti.

5. Telefoninumbrid (põhikooli variant)

1 sekund 40 punkti

Kõige lihtsam viis selle ülesande lahendamiseks on iga telefoninumbrit vahetult iga teisega võrrelda, nagu ongi tehtud failides `tel1lah1.pas`, `tel1lah1.c`, `tel1lah1.cpp`, `tel1lah1.java` ja `tel1lah1.py`.

Jääb veel välja mõelda, kuidas kontrollida, kas üks telefoninumber on teise prefiks või ei ole.

Selle osa juurde asudes tuleks kõigepealt tähele panna, et telefoninumbrite arvudena sisselugemine võib infot kaotada. Nimelt on 1, 01, 001 j.n.e. arvudena küll võrdsed, aga telefoninumbritena erinevad. Seega tuleks andmed sisse lugeda kindlasti tekstidena, mitte arvudena. See teeb ka edasise töö oluliselt mugavamaks.

Igas keeles oleks võimalik kontrollida, kas sõne `s1` on sõne `s2` prefiks vahetu märkhaaval võrdlemisega umbes nii:

```
if length(s1) > length(s2) then
  on_prefiks = false
else
  on_prefiks = true
  for i = 1 to length(s1)
    if not (s1[i] = s2[i]) then
      on_prefiks = false
    endif
  endfor
endif
```

Peaaegu igas keeles on võimalik kirjutada see kontroll alamsõne eraldamise funktsiooni abil:

```
if length(s1) > length(s2) then
  on_prefiks = false
elseif s1 = substring(s2, 1, length(s1))
  on_prefiks = true
else
  on_prefiks = false
endif
```

Samuti on peaaegu igas keeles võimalik kirjutada see kontroll alamsõne otsimise funktsiooni abil:

```
if find(s1, s2) = 1 then
  on_prefiks = true
else
  on_prefiks = false
endif
```

Sageli pole standardsed alamsõne otsimise funktsioonid pikkade tekstide korral kuigi efektiivsed, aga 10-kohaliste telefoninumbrite puhul see küll takistuseks saada ei tohiks.

Süsteemides, kus on olemas regulaaravaldiste või muude mallide sobitamise funktsioonid, võiks kontrolli ka nende abil kirjutada.

Lisaks on mõneski standardteegis olemas spetsiaalne `startswith` või `beginswith` funktsioon, mis täpselt vajalikku tingimust kontrollib. Siis on põhiline trikk see funktsioon dokumentatsioonist õigel ajal üles leida.

Lõpetuseks tasub tähele panna, et kõigi numbripaaride ükshaaval võrdlemine on tegelikult üsnagi ebaefektiivne ja suuremate andmemahtude korral jääks selline lahendus ajahätta sõltumata sellest, kui efektiivselt me ühe paari kontrollimise realiseerime. Paremaid ideid on näha selle ülesande algajate rühma variandi lahendustes.

Testid

1. $N = 0$. Minimaalne test. Vastus `EI`.
2. $N = 1$. Peaaegu minimaalne test. Vastus `EI`.
3. $N = 5$. Väike lihtne test. Vastus `EI`.
4. $N = 7$. Mõned numbrid erinevad ainult algusnullide võrra. Vastus `EI`.
5. $N = 1000$. Maksimaalne juhuslik test. Vastus `EI`.
6. $N = 6$. Väike lihtne test, lühem number eespool. Vastus `JAH`, ühene.
7. $N = 6$. Väike lihtne test, lühem number tagapool. Vastus `JAH`, ühene.
8. $N = 9$. Erijuht: kaks numbrit on täpselt samad. Vastus `JAH`, ühene.
9. $N = 100$. Mõõdukas juhuslik test. Vastus `JAH`, mitteühene.
10. $N = 1000$. Maksimaalne juhuslik test. Vastus `JAH`, mitteühene.

10 testi, à 4 punkti, kokku 40 punkti.

6. Telefoninumbrid (algajate variant)

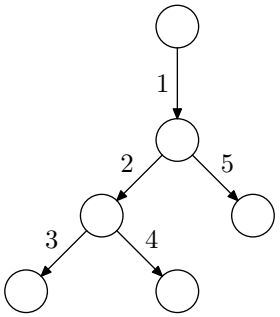
1 sekund 40 punkti

Nagu selle ülesande põhikooli variandi lahenduse selgituses juba mainitud, jääb kõigi numbripaaride vahetul kontrollimisel põhinev lahendus suuremate andmemahtude korral ajahätta.

Lihtne võimalus võrreldavate paaride arvu oluliselt vähendada on telefoninumbrid tähestiku järjekorda seada. Number, mis langeb kokku teise algusega, on järjestatud nimekirjas vahetult selle teise ees. (Järelemõtlemiseks: mis juhtub, kui üks telefoninumber on mitme teise numbri prefiks?)

Sellel ideel põhinevad lahendused on toodud failides `tel2lah1.pas`, `tel2lah1.c`, `tel2lah1.cpp`, `tel2lah1.java` ja `tel2lah1.py`, kus kasutatakse kõikjal programmeerimissüsteemidega kaasas olevaid sorteerimisprotseduure, aga sama hästi töötaks ka "käsitsi" programmeeritud sorteerimisega lahendused (muidugi eeldusel, et sorteeritakse korrektselt ja piisavalt efektiivselt).

Veel efektiivsem võimalus seda liiki ülesannete lahendamiseks on kasutada prefiksipuu (ingl *trie*, *prefix tree*) nimelist andmestruktuuri. Prefiksipuu on puu, mille juurele vastab tühisõne ja mille igal elemendil võib olla alluv vaadeldava tähestiku iga märgi jaoks. Prefiksipuu igale elemendile vastab sõne, mis on saadud juurest temani viival teel olevate märkide järjest lugemisega. Näiteks telefoninumbrite 123, 124, 15 esitus prefiksipuuna on selline:



Peaks olema üsna selge, et n-kohalise telefoninumbri lisamiseks sellisesse puusse tuleb teha n operatsiooni: alustame juurest ja liigume telefoninumbri iga märgi põhjal jooksvast elemendist sellele märgile vastavasse alluvasse. Kui mõnel sammul elementi, kuhu liikuda, ei ole, siis lisame selle puusse.

Üldjuhul on sõnede prefiksipuus hoidmisel vaja iga elemendi juures meeles pidada, kas see on kasutusel ainult abivahendina allpool olevate elementide juurde pääsemiseks või leidub meie andmekogus sõne, mis selles tipus lõpeb.

Kuna selles ülesandes piisab meil vastuse leidmiseks esimesest paarist, kus üks number on teise prefiks, võiksime konflikti olemasolu kontrollida jooksvalt iga telefoninumbri puusse lisamisel. Prefiksipuu kasutamise illustreerimise huvides failis `tel2lah2.pas` toodud lahendus seda ei tee ja hakkab konflikte otsima alles pärast kõigi telefoninumbrite puusse paigutamist.

Testid

1. $N = 0$. Minimaalne test. Vastus `EI`. 2 punkti.
2. $N = 1$. Peaaegu minimaalne test. Vastus `EI`. 2 punkti.
3. $N = 5$. Väike lihtne test. Vastus `EI`. 2 punkti.
4. $N = 7$. Mõned numbrid erinevad ainult algusnullide võrra. Vastus `EI`. 2 punkti.
5. $N = 1000$. Mõõdukas juhuslik test. Vastus `EI`. 2 punkti.
6. $N = 50\,000$. Maksimaalne juhuslik test. Vastus `EI`. 8 punkti.
7. $N = 6$. Väike lihtne test, lühem number eespool. Vastus `JAH`, ühene. 2 punkti.
8. $N = 6$. Väike lihtne test, lühem number tagapool. Vastus `JAH`, ühene. 2 punkti.
9. $N = 9$. Erijuht: kaks numbrit on täpselt samad. Vastus `JAH`, ühene. 2 punkti.
10. $N = 100$. Mõõdukas juhuslik test. Vastus `JAH`, mitteühene. 2 punkti.
11. $N = 1000$. Mõõdukas juhuslik test. Vastus `JAH`, mitteühene. 2 punkti.
12. $N = 50\,000$. Maksimaalne juhuslik, lühem number eespool. Vastus `JAH`, ühene. 6 punkti.
13. $N = 50\,000$. Maksimaalne juhuslik, lühem number tagapool. Vastus `JAH`, ühene. 6 punkti.

Kokku 40 punkti.

7. Arvujada

1 sekund 20 punkti

Esmapilgul tundub see ülesanne triviaalne: arvutame $M = A_1 \cdot A_2 \cdot \ldots \cdot A_N$ ja sealt edasi väljastame iga i korral $M/A_i \bmod 10^8$. Failis `jadalah0a.pas` toodud lahenduse testimine toob siiski välja paar tõrkekohta: esiteks võib M olla standardsetesse täisarvutüüpidesse mahtumiseks liiga suur ja teiseks võib A_i hulgas olla nulle.

Nulliga jagamise veast võiks üle saada, kui korrutada M arvutamisel kokku ainult nullist erinevad sisendjada elemendid ja lisaks loendada nullide arv. Kui sisendis nulle pole, võime iga i korral endiselt väljastada $M/A_i \bmod 10^8$. Kui sisendis on üks null, näiteks A_j, siis väljastame $i = j$ korral $M \bmod 10^8$ ja kõigi muude i korral 0. Kui sisendis on rohkem kui üks null, koosneb kogu väljund nullidest.

Liiga suurte M väärtustega toimetulek on veidi keerulisem. Kindlasti ei piisa M asemel lihtsalt $M \bmod 10^8$ kasutamisest, sest arvude endi asemel võime nende mooduleid liita, korrutada ja teatava ettevaatuse korral ka lahutada, aga jagamine nii ei tööta.

Valede vastuste vältimsieks võib arvutada ülipikkade arvudega. Paraku näitab nii standardteekidele toetuvate `jadalah0b.java` ja `jadalah0b.py` kui ka omakirjutatud ülipikkade arvude aritmeetikat kasutava `jadalah0b.pas` testimine, et suuremate sisendfailidega jäävad nad kõik ajahätta. See pole tegelikult üllatav, sest maksimaalses testis on M ligi $(10^8)^{50\,000}$ ja need lahendused teevad selle ligi 400 000-kohalise arvuga umbes 100 000 jagamistehet.

Veel üks võimalus on arvutada iga $B_i \bmod 10^8$ välja eraldi, hoolitsedes kogu aeg jooksvalt selle eest, et vahetulemused liiga suureks ei kasvaks. Failis `jadalah0c.pas` toodud lahendus väldib sellega küll nii nulliga jagamist kui ka ületäitumise tõttu valede vastuste andmist, kuid teeb N-elemendise sisendjada töötlemisel umbes N^2 tehet ja jääb samuti suuremates testides ajahätta.

Efektiivse lahenduse saamiseks paneme tähele, et sisendjada pikkuse suhtes lineaarse ajaga ja ainult väikesi täisarve kasutades on võimalik leida kõigi selle jada prefiksite ja sufiksite korrutiste moodulid ($B1_i = A_1 \cdot A_2 \cdot \ldots \cdot A_{i-1} \bmod 10^8$ ja $B2_i = A_{i+1} \cdot A_{i+2} \cdot \ldots \cdot A_N \bmod 10^8$) ning edasi arvutada iga i jaoks väljastatav vastus $B1_i \cdot B2_i \bmod 10^8$ konstantse ajaga. Nii ongi tehtud failis `jadalah1.pas` toodud lahenduses.

Veel väärib märkimist, et kuigi nii sisend- kui väljundjada elemendid on väikemad kui 10^8 ja jäävad seega 32-bitiste täisarvude määramispiirkonda, tuleb nende korrutiste hoidmiseks enne moodulite arvutamist kasutada 64-bitiseid arvutüüpe.

Testid

1. $N = 1$. Minimaalne test. 1 punkt.
2. $N = 5$. Väike lihtne test. 2 punkti.
3. $N = 5$. Sisendjada üks element on null. 2 punkti.
4. $N = 25$. Lühike jada, aga elementide korrutis on suur arv. Pikkade arvude aritmeetikal põhinev $O(N^2)$ lahendus töötab, hooletu lühikeste arvude aritmeetikal põhinev lahendus saab vale vastuse. 5 punkti.
5. $N = 50\,000$. Pikk jada, aga elementide korrutis on väike arv. Igasugune $O(N)$ lahendus töötab, $O(N^2)$ lahendus jääb ajahätta. 5 punkti.
6. $N = 50\,000$. Pikk jada, suured arvud. 5 punkti.

Kokku 20 punkti.

8. Takistid

1 sekund 40 punkti

Kõige lihtsam oleks seda ülesannet lahendada kõigi variantide läbivaatusega, proovides igale värvusele seada vastavusse iga numbrit ja vaadata, kas sisendis antud värvuskoode nii tõlgendades saame täpselt sisendis antud väärtused.

Selline lahendus oleks aga liiga aeglane — esimesele värvusele tuleks proovida 10 väärtust, järgmisele 9 väärtust iga esimese värvuse väärtuse kohta j.n.e. Kokku tekiks 10 värvuse korral $10! = 3\,628\,800$ varianti, millest igaühega tuleks mõlemad sisendjadad läbi vaadata, et nende vastavust kontrollida. Osutub, et variantide arvu saab oluliselt vähendada, kui eelnevalt analüüsida, millise tähe kohale milline number saab sobida.

Failis `takilah1.pas` toodud lahendus asendab edasise töötlemise hõlbustamiseks väärtuse lõpus olevad nullid vastava kümne astmega (näiteks 320000 asendatakse 324-ga). Siis saavad igale värvusele vastata ainult sellised numbrid, mis esinevad täpselt samades positsioonides kui see värvus (näiteks kui A esineb nii esimesel kui viimasel kohal, aga number 3 ainult esimesel kohal, siis ei saa A-le 3 vastata).

Samuti sorteerib programm andmete sisselugemise järel nominaalid, et saaks hilisemal nominaalide olemasolu kontrollimisel kasutada kahendotsingut, mis on oluliselt kiirem nominaalide nimekirja lineaarsest läbivaatusest.

Seejärel hakkab programm rekursiivselt proovima värvidele järjest numbrilisi väärtusi, arvestades eelnevalt koostatud võimalike vastavuste tabeliga. Kui kõik värvused on määratud, siis asendab programm järjest värvuskoodide nimekirjas tähed leitud numbritega ja kontrollib iga koodi jaoks, kas nominaalide nimekirjas on vastav takisti olemas.

Sobiva kombinatsiooni leidmisel programm väljastab selle ja lõpetab tegevuse.

Testid

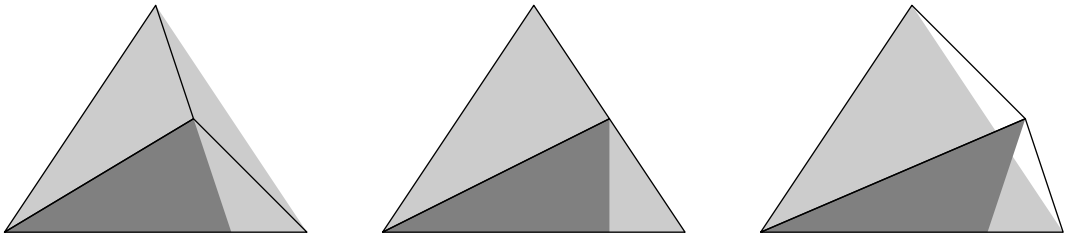
1. $N = 1$. Takistus 0, nii tüvenumbrid kui astmenäitaja nullid. 4 punkti.
2. $N = 1$. Takistus 0, tüvenumbrid nullid, astmenäitaja erineb. 4 punkti.
3. $N = 1$. Ühekohaline takistus. 4 punkti.
4. $N = 1$. Takistus üle 2^{32}. 4 punkti.
5. $N = 3$. Kasutatud tähed pole järjest tähestiku algusest. 4 punkti.
6. $N = 7$. AAA, BBB, ..., GGG $= 11 \cdot 10^1$, $22 \cdot 10^2$, ... $77 \cdot 10^7$. Vastuseks sobib ükskõik milline vastavus $\{\mathrm{A} \ldots \mathrm{G}\} \leftrightarrow \{1 \ldots 7\}$. 4 punkti.
7. $N = 10$. Väike juhuslik test. 4 punkti.
8. $N = 100$. Suur juhuslik test. 4 punkti.
9. $N = 200$. Suur juhuslik test. 4 punkti.
10. $N = 500$. Suur juhuslik test. 4 punkti.

10 testi à 4 punkti, kokku 40 punkti.

9. Kolmnurgad

40 punkti

Lihtne viis kontrollida, kas üks kolmnurk mahub teise sisse: paigutame kolmnurgad nii, et nende alused kattuvad ja aluste vasakud otspunktid langevad kokku. Kui tumeda kolmnurga alus ei ole pikem kui heleda oma ja tumeda kolmnurga vasaku haara tõus ei ole suurem kui heleda oma ja kahe alloleval joonisel musta piirjoonega märgitud kolmnurga pindalade summa ei ole suurem kui heleda oma, siis mahub tumedam kolmnurk heledama sisse.



Eelkirjeldatud kontrolli positiivne tulemus on alati lõplik, kuid negatiivne võib olla eksitav, sest kolmnurgad võivad mahtuda üksteise sisse mõnes muus orientatsioonis.

Võistlejate koostatud testandmete hindamiseks kasutatud programm proovis mõlemas kolmnurgas aluseks kordamööda kõiki servi ja vaatles igas asendis lisaks ka võimalust heleda kolmnurga haarad omavahel vahetada. Selle käigus kontrollis hindamisprogramm järgmist:

1. Kas on test, kus kumbki kolmnurk ei mahu teise sisse?
2. Kas on test, kus esimene kolmnurk mahub teise sisse? Kas on test, kus teine kolmnurk mahub esimese sisse?
3. Kas on test, kus üks kolmnurk mahub teise sisse nii, et sisemist pole vaja pöörata? Kus sisemist on vaja pöörata vastupäeva? Päripäeva?
4. Kas on test, kus üks kolmnurk mahub teise sisse nii, et välimist pole vaja pöörata? Kus välimist on vaja pöörata vastupäeva? Päripäeva?
5. Kas on test, kus üks kolmnurk mahub teise sisse nii, et kumbagi neist pole vaja vertikaalteje ümber peegeldada? Kus "välimist" on vaja peegeldada?
6. Kas on testid, kus "sisemise" kolmnurga mõlemad haarad on laugemad kui "välimise" omad? Eraldi juhud, kus "sisemine" mahub ja kus ei mahu "välimise" sisse?
7. Kas on testid, kus "sisemise" kolmnurga mõlemad haarad on järsemad kui "välimise" omad? Eraldi juhud, kus "sisemine" mahub ja kus ei mahu "välimise" sisse?
8. Kas on testid, kus "sisemise" kolmnurga vähemalt üks haar on paralleelne "välimise" omaga? Eraldi juhud, kus "sisemine" mahub ja kus ei mahu "välimise" sisse?
9. Kas on testid, kus "sisemise" kolmnurga üks haar on laugem ja teine järsem kui "välimise" oma? Eraldi juhud, kus "sisemine" mahub ja kus ei mahu "välimise" sisse?
10. Kas on test, kus kolmnurgad on kongruentsed?
11. Kas on test, kus mõlemad kolmnurgad on kõdunud?
12. Kas on test, kus ühe kolmnurga kaks külge on pikemad kui teise kaks külge ja esimene mahub teise sisse?
13. Kas on test, kus ühe kolmnurga kõik küljed on lühemad kui teise omad ja esimene ei mahu teise sisse?
14. Kas sisendandmete hulgas on esindatud nii võrdkülgsed, võrdhaarsed kui ka erikülgsed kolmnurgad?

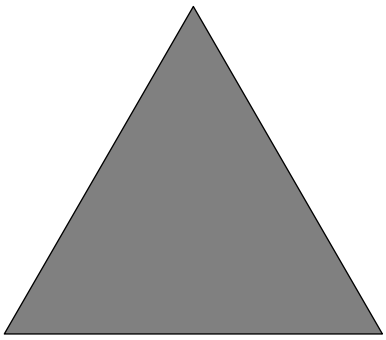
15. Kas sisendandmete hulgas on esindatud nii sirgnurksed, nürinurksed, täisnurksed kui ka teravnurksed kolmnurgad? (Ülesande püstitus keelas kõdunud kolmnurgad, mille mõne külje pikkus on null, kuid ei keelanud kõdunud kolmnurki, mille pikima külje pikkus on täpselt kahe ülejäänu pikkuste summa.)

Kokku 30 komponenti.

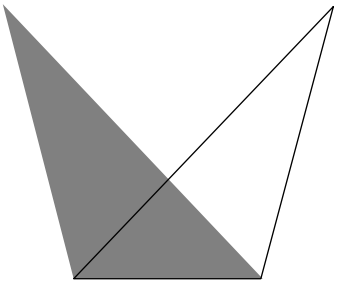
Näide

Näiteks järgmine 8 testist koosnev komplekt katab kõik eeltoodud juhud:

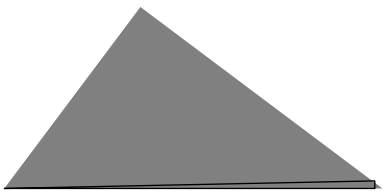
1. Märksõnad: võrdkülgne; teravnurkne; kongruentsed; sisemise ja välimise haarad paralleelsed, mahub; mahub teist peegeldamata; mahub teist pööramata; mahub esimest pööramata; esimene mahub teise sisse.



2. Märksõnad: erikülgne; nürinurkne; sisemise üks haar laugem, teine järsem, ei mahu; sisemise haarad järsemad, ei mahu; sisemise haarad laugemad, ei mahu; sisemise kõik küljed lühemad kui välimise küljed ja ei mahu; sisemise haarad laugemad, mahub; mahub teist peegeldades; mahub teist päripäeva pöörates; mahub esimest vastupäeva pöörates; teine mahub esimese sisse.



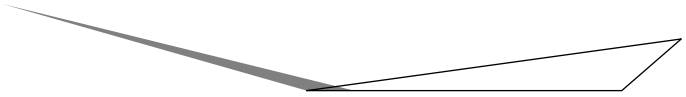
3. Märksõnad: erikülgne; täisnurkne; võrdhaarne; teravnurkne; sisemise üks haar laugem, teine järsem, ei mahu; sisemise haarad laugemad, ei mahu; sisemise haarad järsemad, ei mahu; sisemise kaks külge pikemad kui välimise kaks külge ja mahub; sisemise üks haar laugem, teine järsem, mahub; mahub teist peegeldades; mahub teist pööramata; mahub esimest pööramata; teine mahub esimese sisse.



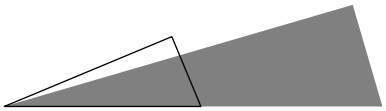
4. Märksõnad: võrdhaarne; sirgnurkne; kaks kõdunud kolmnurka; sisemise ja välimise haarad paralleelsed, mahub; mahub teist peegeldamata; mahub teist pööramata; mahub esimest pööramata; esimene mahub teise sisse.

---

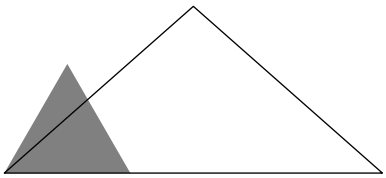
5. Märksõnad: erikülgne; nürinurkne; sisemise üks haar laugem, teine järsem, ei mahu; sisemise haarad järsemad, ei mahu; sisemise haarad laugemad, ei mahu; sisemise kaks külge pikemad kui välimise kaks külge ja mahub; sisemise haarad laugemad, mahub; mahub teist peegeldamata; mahub teist vastupäeva pöörates; mahub esimest päripäeva pöörates; esimene mahub teise sisse.



6. Märksõnad: erikülgne; täisnurkne; sisemise üks haar laugem, teine järsem, ei mahu; sisemise haarad laugemad, ei mahu; sisemise ja välimise haarad paralleelsed, ei mahu; sisemise haarad järsemad, ei mahu; sisemise üks haar laugem, teine järsem, mahub; mahub teist peegeldamata; mahub teist pööramata; mahub esimest pööramata; teine mahub esimese sisse.



7. Märksõnad: võrdkülgne; teravnurkne; võrdhaarne; nürinurkne; sisemise haarad järsemad, mahub; mahub teist peegeldamata; mahub teist pööramata; mahub esimest pööramata; esimene mahub teise sisse.



8. Märksõnad: võrdhaarne; nürinurkne; võrdkülgne; teravnurkne; sisemise üks haar laugem, teine järsem, ei mahu; sisemise haarad laugemad, ei mahu; sisemise kõik küljed lühemad kui välimise küljed ja ei mahu; sisemise haarad järsemad, ei mahu; ei mahu.