1. Pildiraam

1 sekund 10 punkti

Selle ülesande lahendamiseks on vaja kõigepealt tähele panna, et kui mõlema pildi asendid on fikseeritud, siis määrab raami kõrguse see, mille kõrgus on suurem, ja raami laiuse see, mille laius on suurem.

Edasi tuleb märgata, et ülesande tekstis toodud tingimus, mille kohaselt raami saab seinale riputada nii püsti kui ka rõhtsalt, tähendab sisuliselt seda, et pilte võib raami sisse panna nii otse kui ka 90° kaupa pööratuna. Kuna ristkülikut kaks korda 90° võrra pöörates saame esialgse ristküliku tagasi, on vaja pildi ja raami suhete uurimisel kaaluda ainult pildi algasendit ja 90° võrra pööratud asendit.

Eeltoodu põhjal võib tunduda, et ülesande lahendamiseks tuleks vaadelda nelja varianti (mõlemad pildid algasendis, esimene algasendis ja teine pööratud, esimene pööratud ja teine algasendis, mõlemad pööratud), aga tegelikult piisab ka kahest esimesest, sest kolmas on samaväärne teisega ja neljas esimesega (jällegi, kui arvestame, et raami võib pärast pildi sinna sisse panekut seinal pöörata).

Seega on üks võimalik lahendus leida, millised peaks olema raami mõõdud, kui mõlemad pildid tuleks sinna panna esialgses orientatsioonis ja millised peaks raami mõõdud olema, kui üks kahest pildist tuleks sinna panna pööratult (kuna pilt peaks seinal õigetpidi olema, siis tegelikult pööraks me muidugi raami pildi ümber, aga nende üksteise sisse mahtumise seisukohalt see asja ei muuda).

Selline lahendus ongi toodud failis `raamlah1.pas`.

Veidi lühema lahenduse saame, kui paneme tähele, et kahest eelmises lahenduses eraldi vaadeldavast paigutusest annab alati parema tulemuse see, kus pildid on omavahel seatud nii, et nende pikemad server on paralleelsed (me justkui vaatleks alati pildi pikemat serva kõrguse ja lühemat laiusena).

Selline lahendus on toodud failis `raamlah2.pas`.

Testid

1. Mõlemal pildil lühem külg enne. Esimene pilt pikem, aga kitsam.
2. Mõlemal pildil lühem külg enne. Esimene pilt lühem ja kitsam.
3. Mõlemal pildil pikem külg enne. Esimene pilt lühem, aga laiem.
4. Mõlemal pildil pikem külg enne. Esimene pilt pikem ja laiem.
5. Esimesel pildil lühem, teisel pikem külg enne. Esimene pilt pikem ja laiem.
6. Esimesel pildil lühem, teisel pikem külg enne. Esimene pilt pikem, aga kitsam.
7. Esimesel pildil pikem, teisel lühem külg enne. Esimene pilt lühem ja kitsam.
8. Esimesel pildil pikem, teisel lühem külg enne. Esimene pilt lühem, aga laiem.
9. Mõlemad pildid ruudukujulised, esimene suurem.
10. Mõlemad pildid ruudukujulised, esimene väiksem.

10 testi, à 1 punkt, kokku 10 punkti.

2. Pikkuse järjekorras

1 sekund 20 punkti

Üsna ilmne võimalus selle ülesande lahendamiseks on pidada andmete lugemisel jooksvat edetabelit kolmest seni pikimast õpilasest, võrrelda iga uut pikkust nendega ja vastavalt võrdlemiste tulemusele kas jätta uus kandidaat kohe kõrvale või lisada ta edetabelisse sobivale kohale (ja tõrjuda senine pikkuselt kolmas tabelist välja).

Kuna edetabeli pikkus on väike fikseeritud arv, võib vajalikud võrdlused ja tabelisse lisamise omistamised käsitsi välja kirjutada, nagu on tehtud failis `pikklah1.pas` toodud lahenduses. Teine võimalus on kirjutada võrdlemised ja omistamised korduslause abil. Siis on lihtne üldistada lahendus 3-kohaliselt edetabelilt k-kohalisele, nagu on tehtud failis `pikklah2.pas`, mille saab ühe konstandi muutmisega panna leidma ka näiteks pikkuselt teist või neljandat õpilast.

Keeltes, kus arvujada sorteerimine on standardvarustuses, võib ka kogu jada ära sorteerida. Tavaliselt sorteeritakse andmed vaikimisi kasvavasse järjekorda. Siis tuleks väljastada sorteeritud jada tagantpoolt kolmas element, nagu on tehtud failis `pikklah3a.py` toodud lahenduses. Teine võimalus on jada pärast sorteerimist ümber pöörata, nagu on tehtud failis `pikklah3b.py` toodud lahenduses.

Kui edetabeli pikkus on oluliselt väiksem kogu sisendjada pikkusest, siis on sorteerimisel põhinev lahendus natuke vähem efektiivne kui ainult vajaliku edetabeli hoidmisel põhinev, aga selle ülesande andmemahtude juures pole see vahe oluline.

Testid

1. $N = 3$. Minimaalne test, kõik pikkused unikaalsed. 2 punkti.
2. $N = 3$. Minimaalne test, kõik pikkused võrdsed. 2 punkti.
3. $N = 5$. Väike test, teine ja kolmas pikkus võrdsed. 2 punkti.
4. $N = 5$. Väike test, kolmas ja neljas pikkus võrdsed. 2 punkti.
5. $N = 10$. Pikkused antud kasvavas järjekorras. 2 punkti.
6. $N = 10$. Pikkused antud kahanevas järjekorras. 2 punkti.
7. $N = 1000$. Mõõduka suurusega juhuslik test. 4 punkti.
8. $N = 10\,000$. Maksimaalse suurusega juhuslik test. 4 punkti.

8 testi, kokku 20 punkti.

3. TopSwops

1 sekund 30 punkti

Selle ülesande lahendamiseks on kõige lihtsam mängu simuleerida ja loendada, mitu käiku tuleb teha, enne kui 1 paki pealmiseks saab.

Failis `topslah1.pas` toodud lahendus teeb seda ilmutatud kujul, kui paki pealmise osa ümberpööramiseks kasutatakse ilmutatud kordust. Sellise lahenduse saab kirjutada igas keeles.

Failis `topslah1.cpp` toodud lahendus kasutab ära C++ standardteegis olevat funktsiooni massiivi (või selle osa) kohapeal ümberpööramiseks. Selle funktsiooni realisatsioonis on muidugi Pascali lahenduse omale sarnane kordus.

Failis `topslah1.py` toodud lahendus kasutab ära asjaolu, et Pythonis on olemas standardsed operatsioonid jadade tükeldamiseks ja nende tükkide edaspidi või tagurpidi kokkukleepimiseks. See lahendus teeb uute jadade loomisega tõenäoliselt küll natuke rohkem tööd kui puhtalt elementide ühes massiivis ümberpaigutamiseks kuluks, aga kuna need operatsioonid on standardteegis arvatavasti realiseeritud riistvarale lähedasemalt (ja seega efektiivsemalt) kui Pythonis endas kirjutatud korduslause tulemus, siis võib arvata, et selline lahendus ei kuluta kokkuvõttes rohkem aega kui Pascali variandi Pythonisse ümber kirjutamise saadu. Muidugi võib igaüks kodus selle arvamuse kinnitamiseks või ümber lükkamiseks ise katseid teha.

Selle mängu leiutas 1973. aastal J. H. Conway ja üldiselt pole tänaseni kuigi täpselt teada, kui pikk võib N kaardiga mäng maksimaalselt olla. 1974. aastal tõestas D. E. Knuth, et N kaardiga mäng jõuab mistahes seisust alustades lõpule hiljemalt F_{N+1} sammuga (kus F_i on Fibonacci jada element number i). 2006. aastal tõestasid L. Morales ja I. H. Sudborough, et on lõpmatu arv N väärtusi, mille jaoks maksimaalse mängu pikkus on vähemalt

$$\frac{29 \cdot N^2 - 718 \cdot N + 5265}{96}.$$

Testid

1. $N = 1$. Minimaalne test, vastus 0. 2 punkti.
2. $N = 2$. Peaaegu minimaalne test, vastus 0. 2 punkti.
3. $N = 2$. Peaaegu minimaalne test, vastus 1. 2 punkti.
4. $N = 4$. Väike lihtne test, vastus 4. 3 punkti.
5. $N = 4$. Väike lihtne test, vastus 4. 3 punkti.
6. $N = 21$. Pika mänguga test, vastus 91. 4 punkti.
7. $N = 33$. Pika mänguga test, vastus 154. 4 punkti.
8. $N = 97$. Pika mänguga test, vastus 1029. 5 punkti.
9. $N = 289$. Peaaegu maksimaalne (mitte eriti juhuslik) test, vastus 9775. 5 punkti.

9 testi, kokku 30 punkti.

4. Copy-Paste

1 sekund 40 punkti

Nimetame klahvivajutuste Ctrl-A, Ctrl-C ja Ctrl-V jada, mis algsest lausest N koopiat teeb, *N-kordistavaks jadaks*. Ülesandes otsitakse seega antud arvu N jaoks lühimat N-kordistavat jada. Tähistame klahvivajutusi Ctrl-A, Ctrl-C ja Ctrl-V lühiduse mõttes tähtedega A, C ja V.

Kõigepealt veendume, et lühim N-kordistav jada koosneb mingist arvust (0 või enamast) järjestikustest klahvivajutuste jadadest kujul ACV…V, kus V esineb kaks või enam korda.

Kui jadas esineb V, siis kusagil eespool peab esinema C, sest lõikepuhvri sisu alghetkel on määramata, aga me tahame jada, mis annaks soovitud tulemuse alati.

A-le ei saa lühimas jadas järgneda V, sest sellega asendaksime kogu teksti lõikepuhvri sisuga; kuna puhvris on teksti mingi varasem seis, saaksime kogu vahepealse tegevuse eemaldamisega lühema jada, mis annab sama tulemuse. Samamoodi ei saa A lühimas jadas esineda kaks korda järjest ega jada viimase klahvivajutusena. Järelikult peab A-le alati järgnema C.

Ilmselt ei saa ka C lühimas jadas esineda kaks korda järjest. Samuti ei saa talle eelneda V ega saa C olla jada esimene täht, muidu kopeeriksime tühja sõne ja sellest ei oleks kasu. Järelikult C-le eelneb alati A, seega tähed A ja C esinevad alati paaris AC.

Paarile AC peab lühimas jadas järgnema V, sest kui paarile AC järgneb teine AC või esineb paar AC jada lõpus, siis võime ta jadast eemaldada. Kolmikule ACV peab järgnema lühimas jadas veel vähemalt üks V, sest kui talle järgneks ACV või ta asuks jada lõpus, siis võiks ta lihtsalt ära jätta.

Seega ei jäägi muud üle, kui et lühim jada peab olema kujul ACV…VACV…V … ACV…V.

Olgu nüüd $n_1, \ldots, n_k$ V-tähtede arvud osajadades ACV…V, millest lühim N-kordistav jada koosneb. Ilmselt on V-tähte m korda sisaldav jada ACV…V m-kordistav, seega peavad arvud $n_1, \ldots, n_k$ rahuldama tingimust $n_1 \cdot \ldots \cdot n_k = N$ ja eelneva põhjal peavad kõik tegurid n_i olema ühest suuremad.

Kuna V-tähte m korda sisaldava jada ACV…V pikkus on $m+2$, saame ülesande ümber sõnastada nii: leida arvu N selline tegurdus ühest suuremate täisarvude korrutiseks $n_1 \cdot \ldots \cdot n_k = N$, et summa $(n_1 + 2) + \ldots + (n_k + 2)$ oleks võimalikult väike (nulli liidetava summaks loeme 0 ja nulli teguri korrutiseks 1 — seega kehtib eelnev jutt ka juhul $N = 1$).

Lihtne viis selle mõtte realiseerimiseks on toodud failis `acvvlah1.cpp`, kus vaadatakse läbi arvu N kõik tegurdused kujul $N = a \cdot b$, kus $a \leqslant \sqrt{N}$ (suuremaid a väärtusi pole mõtet vaadelda, kuna siis saaksime uuesti sisuliselt samad tegurdused, vaid tegurite järjekord erineks) ja kummagi poole lahendamiseks kasutatakse rekursiivselt sama algoritmi. Kuigi sellistel algoritmidel (kus ühe suurema juhu lahendamiseks tehakse mitu rekursiivset pöördumist väiksemate juhtude poole) on üldiselt kalduvus olla ebaefektiivsed, töötab see lahendus siiski piisavalt efektiivselt, sest pöördumisel väiksema juhu poole väheneb argumendi väärtus mitte ühe (või mingi muu väikese konstandi) võrra, vaid kordades.

Siiski on võimalik ka efektiivsem lahendus. Paneme tähele, et kui $n_1 \cdot \ldots \cdot n_k = N$ on otsitav tegurdus (s.t. $n_1, \ldots, n_k$ on V-tähtede arvud mingi lühima N-kordistava jada osajadades ACV…V), siis iga $i \in \{1, \ldots, k\}$ korral peab lühim n_i-kordistav osajada olema kujul ACV…V: vastasel juhul saaksime lühema N-kordistava jada, kui asendaksime i-nda osajada ACV…V lühima n_i-kordistava jadaga. Võime ka eeldada, et ühegi n_i korral ei juhtu, et jada kujul ACV…V on üks mitmest lühimast n_i-kordistavast osajadast (sest iga sellist tegurit n_i sisaldava tegurduse $n_1 \cdot \ldots \cdot n_k = N$ saab asendada sama lühikese lahendusega, mis ei sisalda tegurit n_i). Seega kerkib küsimus: missuguste arvude N korral on ainus lühim N-kordistav jada kujul ACV…V?

Ilmselt on otsitav lühim N-kordistav jada kujul ACV...V, kui N on algarv, sest algarvulise N ainus võimalik tegurdus ühest suuremate arvude korrutiseks on $n_1 = N$. Kui aga N on kordarv, s.t. $N = a \cdot b$, kus a ja b on ühest suuremad täisarvud, siis üks N-kordistav jada on kindlasti ACV...VACV...V, kus esimeses osas on a V-tähte ja teises osas b V-tähte. Selleks, et ainus lühim N-kordistav jada oleks kujul ACV...V, peab tema pikkus olema väiksem kui eeltoodud jada pikkus, ehk $N + 2 < (a + 2) + (b + 2)$ ehk $N - a - b + 1 < 3$ ehk $(a - 1)(b - 1) < 3$.

Kui $a \geqslant 4$ või $b \geqslant 4$, siis viimane võrratus ilmselt ei kehti, seega jäävad läbi vaadata üksnes juhud $a = b = 3$, $a = b = 2$ ning juht, kus üks arvudest a ja b on 2 ja teine 3. Võrratus kehtib vaid teisel ja kolmandal juhul. Järelikult ainsad kordarvud N, mille korral ainus lühim N-kordistav jada on kujul ACV...V, on $2 \cdot 2 = 4$ ja $2 \cdot 3 = 6$.

Ülesandes otsitav tegurdus $N = n_1 \cdot \ldots \cdot n_k$ saab koosneda niisiis üksnes algarvudest ning arvudest 4 ja 6. Ilmselt igas sellises tegurduses on arvu N kolmest suuremad algtegurid, nii et jääb vaid selgitada, kuidas tegurdada arvu N see osa, mille algtegurid on 2 ja 3.

Seepärast vaatlemegi nüüd juhtu $N = 2^k 3^l$, kus k ja l on mittenegatiivsed täisarvud. Arvu N teguriteks on meil neli kandidaati: 2, 3, 4 ja 6. Olgu nende tegurite kordsused otsitavas tegurduses vastavalt m_2, m_3, m_4 ja m_6, s.t. vaadeldav tegurdus on $N = 2^{m_2} 3^{m_3} 4^{m_4} 6^{m_6}$. Antud tegurdusele vastava klahvivajutuste jada pikkus on $L = 4m_2 + 5m_3 + 6m_4 + 8m_6$.

Võrdusest $2^k 3^l = 2^{m_2} 3^{m_3} 4^{m_4} 6^{m_6}$ saame tingimused $k = m_2 + 2m_4 + m_6$ ja $l = m_3 + m_6$. Seega $L = m_2 + 3k + 5l$. Et liidetav $3k + 5l$ ei sõltu tegurduse valikust, saimegi tingimuse arvu N tegurdamiseks: arv N tuleb esitada arvude 2, 3, 4 ja 6 korrutisena nii, et tegurit 2 esineks võimalikult vähe.

Niisiis üldjuhul, kus N on positiivne täisarv, tuleb N esitada algarvude ning arvude 4 ja 6 korrutisena sedasi, et tegurit 2 esineks võimalikult vähe. See on saavutatav nii, et jagame arvu N kõigepealt 4-ga niipalju kui võimalik, siis 6-ga niipalju kui võimalik, seejärel allesjäänud osa esitame algtegurite korrutisena (tõepoolest, sellise algoritmi puhul esineb tegurit 2 lahendis üks kord, kui N ei jagu kolmega ja algtegur 2 esineb paaritus astmes, ning muudel juhtudel lahendis tegurit 2 ei esine).

Eelneva tõttu võimegi olla kindlad, et failis `acvvlah2.pas` toodud lahendus annab alati õige vastuse.

Testid

1. $N = 1$. Minimaalne test; 0 klahvivajutust. 2 punkti.
2. $N = 2$. Peaaegu minimaalne test; 4 klahvivajutust. 2 punkti.
3. $N = 3$. Peaaegu minimaalne test; 5 klahvivajutust. 2 punkti.
4. $N = 4$. Erijuht, tegurdamine ei tasu ära; 6 klahvivajutust. 2 punkti.
5. $N = 6$. Erijuht, tegurdamine ei tasu ära; 8 klahvivajutust. 2 punkti.
6. $N = 8$. Erijuht, ilma tegurdamiseta ja $2 \cdot 4$ tegurdamisega on võrdsed, aga $2 \cdot 2 \cdot 2$ tegurdamisega halvem; 10 klahvivajutust. 2 punkti.
7. $N = 9$. Esimene tegurdamisega test; 10 klahvivajutust. 2 punkti.
8. $N = 10$. Tegurdamisega; 11 klahvivajutust. 2 punkti.
9. $N = 12$. Tegurdamisega, aga $3 \cdot 4$, mitte $2 \cdot 6$ või $2 \cdot 2 \cdot 3$; 11 klahvivajutust. 2 punkti.
10. $N = 16$. Tegurdamisega, aga kindlasti $4 \cdot 4$; 12 klahvivajutust. 2 punkti.
11. $N = 125$. Tegurdamisega $5 \cdot 5 \cdot 5$; 21 klahvivajutust. 3 punkti.
12. $N = 127$. Algarv, 129 klahvivajutust. 3 punkti.
13. $N = 25\,005$. Tegurdamisega $3 \cdot 5 \cdot 1667$. 1681 klahvivajutust. 3 punkti.
14. $N = 80\,220$. Tegurdamisega $3 \cdot 4 \cdot 5 \cdot 7 \cdot 191$; 220 klahvivajutust. 3 punkti.
15. $N = 999\,983$. Maksimaalne algarvuline test; 999 985 klahvivajutust. 4 punkti.
16. $N = 1\,000\,000$. Maksimaalne tegurdamisega test; 60 klahvivajutust. 4 punkti.

Kokku 40 punkti.

5. Minotauros

50 punkti

Selle ülesande lahendamiseks on kõige lihtsam leida labürindi kõigi ruutude kaugused Minotaurose asukohast ja siis väljastada see ruut, mille kaugus on maksimaalne.

Kauguste leidmist võime alustada tähelepanekust, et Minotaurose asukoharuudu kaugus Minotaurosest on loomulikult 0. Edasi võime labürindikaarti korduvalt läbi vaadata ja iga läbivaatuse käigus uurida iga ruudu kohta, mille kaugust me juba teame, kas sealt tema naaberruutudesse edasi liikudes oleks võimalik pääseda mõnda ruutu, millesse me varem üldse jõudnud pole, või avastada mõne ruudu jaoks lühem tee. Kui me labürindi mingi läbivaatamisega enam ühtki uut või lühemat teed ei leia, siis ilmselt ei muudaks ka järgmised enam midagi ja võimegi otsimise lõpetada.

Sellel ideel põhinev lahendus on toodud failis `minolah1.pas`. Kuna selles lahenduses suurenevad leitud teede pikkused terve kaardi iga läbivaatusega ühe võrra, peaks olema selge, et L sammu kaugusel oleva ruudu leidmiseks teeb see lahendus umbes $L \cdot N \cdot M$ operatsiooni. Kuna maksimaalne võimalik käigupikkus $N \times M$ ruudust koosnevas labürindis on umbes $1/2 \cdot N \cdot M$, võib see lahendus $N \times M$ labürindi töötlemisel kulutada kuni $1/2 \cdot N^2 \cdot M^2$ operatsiooni.

Testides on N ja M väärtused kuni 500, mis tähendab, et halvimal juhul võiks sellise lahenduse tööaeg olla kümmekond minutit. Tegelikult töötab ta siiski mõne sekundiga, sest üheski testis pole Minotaurosest kaugeima talle kättesaadava ruudu kaugus üle 3000. Maksimaalse teepikkuse saavutamiseks peaks käigud labürindis olema ühe hargnemisteta ahela kujulised, aga testid on kõik hargnevate käikudega.

Märksa efektiivsema lahenduse saame, kui paneme tähele, et tegelikult on igal kaardi läbivaatusel mõtet uurida ainult neid ruute, millesse viiva tee me avastasime vahetult eelneval läbimisel. Tõepoolest, kõigist varem avastatud ruutudest nende naabritesse liikumise võimalusi oleme eelmistel läbimistel juba uurinud.

Sellel ideel põhinev lahendus on toodud failis `minolah2.cpp`. Osutub, et selles lahenduses satub labürindi iga läbitav ruut töödeldavate ruutude järjekorda maksimaalselt ühe korra. Kuna iga järjekorrast võetud ruudu töötlemiseks tehakse maksimaalselt neli operatsiooni (vaadeldakse selle ruudu naabreid, mida ei saa olla üle nelja), siis ei kuluta see lahendus $N \times M$ labürindi töötlemiseks kokku rohkem kui $4 \cdot N \cdot M$ operatsiooni, mis jääb isegi maksimaalses testis alla 1 000 000.

See lahendus vaatleb labürinti graafina, mille tippudeks on labürindi käiguruudud ja milles on servad üksteise naabriteks olevatele käiguruutudele vastavate tippude vahel. Sisuliselt on lahendus seega graafi läbimine laiuti[1], mille kasutamine lühimate teede leidmiseks graafi ühest tipust kõigisse teistesse on laiuti läbimise standardrakendus[2].

Selles ülesandes on omaette huvitav ka testandmete genereerimine. Kuidas genereerida juhuslikke labürinte, mis paistaks "normaalsed" või "mõistlikud", aga samas poleks liiga korrapärased? Kuna ülesanne oli antud lahtiste testidega, oleks väga korrapäraseid teste olnud liiga kerge käsitsi lahendada.

[1] `http://www.ttkool.ut.ee/comp/kaug/prog1/prog009.html#htoc108`
[2] `http://www.ttkool.ut.ee/comp/kaug/prog1/prog009.html#htoc109`

Testid

1. $N = 15$, $M = 15$. Käigud moodustavad juhusliku puu. Maksimaalne kaugus 23. Üks võimalik koht.
2. $N = 25$, $M = 27$. Käigud moodustavad juhusliku puu. Maksimaalne kaugus 62. Kaks võimalikku kohta.
3. $N = 17$, $M = 17$. Käigud moodustavad juhusliku tsüklitega võrgu. Maksimaalne kaugus 30. Kaks võimalikku kohta.
4. $N = 27$, $M = 29$. Käigud moodustavad juhusliku tsüklitega võrgu. Maksimaalne kaugus 108. Üks võimalik koht ja see pole käigu ots.
5. $N = 77$, $M = 99$. Käigud moodustavad juhusliku puu. Maksimaalne kaugus 409. Üks võimalik koht.
6. $N = 199$, $M = 199$. Käigud moodustavad juhusliku puu. Maksimaalne kaugus 832. Üks võimalik koht.
7. $N = 299$, $M = 299$. Käigud moodustavad juhusliku tsüklitega võrgu. Maksimaalne kaugus 850. Üks võimalik koht.
8. $N = 397$, $M = 397$. Käigud moodustavad juhusliku puu. Maksimaalne kaugus 1511. Üks võimalik koht.
9. $N = 399$, $M = 399$. Käigud moodustavad juhusliku puu, välja arvatud üks mittesidus ruut. Maksimaalne kaugus lõpmatu. Üks võimalik koht.
10. $N = 499$, $M = 499$. Käigud moodustavad juhusliku puu. Maksimaalne kaugus 2607. Kolm võimalikku kohta.

10 testi, à 5 punkti, kokku 50 punkti.

6. Multi-Ant

1 sekund 50 punkti

Ülesanne, mis tegeleb mingite objektide vaheliste seostega (sammud ja nende vahelised sõltuvused) on muidugi graafiülesanne.

Üks võimalus selle ülesande lahendamiseks on panna tähele, et esimesena saame paralleelselt teha parajasti kõik need sammud, mis ühestki teisest ei sõltu, teisena need, mis sõltuvad ainult neist, mis said esimesena tehtud j.n.e. Kuigi see tundub sarnane Minotaurose ülesande lahendusega, on siin üks oluline erinevus: kui Minotaurose ülesandes otsisime lühimaid teid Minotaurose asukohast kõigisse teistesse ruutudesse ja seetõttu lisasime iga tipu töödeldavate järjekorda selle esmakordsel avastamisel, siis selles ülesandes ei saa me ühtki sammu alustada enne kui kõik selle eeldused (mitte ainult üks eeldus) on lõpetatud.

Seega võime graafi laiuti läbimise sarnast algoritmi kasutada ainult juhul, kui peame iga sammu kohta meeles, mitu tema eeldust on veel tegemata. Selleks tuleks sisendis antud sõltuvuste info "ümber pöörata", nagu ongi tehtud failides `mantlah1a.cpp` ja `mantlah1b.cpp`. Neist lahendustest teine on suurtes testides natuke efektiivsem samamoodi, nagu Minotaurose ülesande teine lahendus on esimesest efektiivsem. Kuna andmemahud on selles ülesandes väiksemad (graafis on maksimaalselt 1000 tippu), siis see vahe programmi tööaega praktikas mõõdetavalt ei mõjuta.

Teine võimalus selle ülesande lahendamiseks on vaadelda sõltuvusseoseid samas suunas, nagu nad sisendis antud on: iga sammu enda saab ilmselt teha mitte varem kui kõige hilisema eelduse tegemisele järgneval ajaühikul. Selle idee otsene realisatsioon on toodud failis `mantlah2a.cpp`.

Nagu sageli juhtub rekursiivsete algoritmidega, mis teevad ülesande lahendamiseks pöördumisi mitme alamülesande poole, jääb see lahendus suuremates testides ajahätta. Nagu samuti sageli juhtub, aitab selle vastu naiivse rekursiive lahenduse muutmine mäluga rekursiivseks lahenduseks, nagu on tehtud failis `mantlah2b.cpp`.

Oma olemuselt on viimane lahendus lähedane graafi sügavuti läbimisele[3], mis pole väga üllatav, arvestades meie ülesande sarnasust topoloogilise sorteerimise ülesandega[4], mille lahendamine ongi sügavuti läbimise üks standardrakendusi.

[3] `http://www.ttkool.ut.ee/comp/kaug/prog1/prog009.html#htoc110`
[4] `http://www.ttkool.ut.ee/comp/kaug/prog1/prog009.html#htoc111`

Testid

1. $N = 1$. Minimaalne test, ainult üks samm. $T = 1$, vastus ühene. 2 punkti.
2. $N = 3$. Väike lihtne test, kõik sammud tuleb teha järjest. $T = 3$, vastus ühene. 3 punkti.
3. $N = 5$. Sõltuvusgraaf on puu, mõned paralleelsed sammud. $T = 3$, vastus peaaegu ühene. 5 punkti.
4. $N = 5$. Sõltuvusgraaf pole puu, mõned paralleelsed sammud. $T = 4$, vastus peaaegu ühene. 5 punkti.
5. $N = 5$. Sõltuvusgraaf pole puu, mõned paralleelsed sammud. $T = 4$, vastus pole ühene. 5 punkti.
6. $N = 50$. Mõõdukas juhuslik test. $T = 33$. 10 punkti.
7. $N = 300$. Suur juhuslik test. $T = 82$. 10 punkti.
8. $N = 1000$. Maksimaalne juhuslik test. $T = 125$. 10 punkti.

Kokku 50 punkti.

Märkus "vastus peaaegu ühene" tähendab, et iga sammu tegemise aeg on üheselt määratud, aga paralleelsete sammude nimed võib oma real väljastada mitmes erinevas järjekorras; märkus "vastus pole ühene" tähendab, et vähemalt ühe sammu tegemise aeg pole üheselt määratud.