1. Sudoku kontrollija

Üks võimalus selle ülesande lahendamiseks on kontrollida vahetult kõiki ruudupaare igas reas, igas veerus ja igas plokis. Kui leiame kontrollimise käigus mõne sellise paari, kus mõlemas ruudus on sama number, olemegi leidnud vastuolu. Sellel põhimõttel töötavad failides `sklah1a.pas` ja `sklah1b.pas` toodud lahendused.

Teine võimalus on vaadata rea, veeru või ploki kontrollimisel ruudupaaride asemel läbi üksikud ruudud ja pidada iga võimaliku numbri kohta meeles, kas see on juba esinenud või mitte. Sellel põhimõttel töötavad failides `sklah2a.pas` ja `sklah2b.pas` toodud lahendused.

Teoreetiliselt on teine lahendus esimesest natuke efektiivsem. Nimelt vaatab esimene lahendus igas reas (mida on 9 tükki), veerus (neid on samuti 9) ja 3×3 plokis (neidki on 9) läbi $9 \cdot 8/2 = 36$ ruudupaari, ehk kokku tuleb teha $3 \cdot 9 \cdot 36 = 972$ kontrolli. Teine lahendus seevastu vaatab igas reas, veerus ja plokis läbi 9 ruutu, tehes seega kokku $3 \cdot 9 \cdot 9 = 243$ kontrolli ehk täpselt 4 korda vähem tööd. Muidugi on need arvud selle ülesande andmemahtude juures nii väikesed, et vahe reaalselt mõõdetav ei ole.

Üldisemalt teeks esimene lahendus $n \times n$ sudoku korrektsuse tuvastamiskes $3 \cdot n \cdot \frac{n(n-1)}{2} = O(n^3)$, teine aga $3 \cdot n \cdot n = O(n^2)$ kontrolli. Kuna juba $n \times n$ tabeli sisselugemiseks kulub vähemalt $O(n^2)$ operatsiooni, on teine lahendus sellega nii efektiivne kui üldse võimalik.

Testid

1. Õige “reaaleluline” sudoku. 3 punkti.
2. Õige “süstemaatiline” sudoku. 3 punkti.
3. Vead ridades 7 ja 8 (2-kordsed). 3 punkti.
4. Vead veergudes 5 ja 6 (2-kordsed). 3 punkti.
5. Vead ridades 2, 5 ja 8 ning plokkides 1:3, 2:3, 3:3 (3-kordsed). 3 punkti.
6. Vead kõigis plokkides (read-veerud õiged). 3 punkti.
7. Vead plokkides 2:2, 2:3, 3:2, 3:3 (read-veerud õiged). 3 punkti.
8. Vead plokkides 2:1, 2:2, 3:1, 3:2 (read-veerud õiged). 3 punkti.
9. Vigane, kuid ridade, veergude ja plokkide summad on võrdsed. 3 punkti.
10. Vigane, kuid ridade, veergude ja plokkide korrutised on võrdsed. 3 punkti.

Kokku 30 punkti.

2. XBox

Esimene võimalus selle ülesande lahendamiseks on lähtuda vahetult ülesande püstituses toodud skeemist:

1. lahutada mälupesa number bittideks;
2. kombineerida uuesti arvudeks eraldi paaritutel positsioonidel olevad bitid (X-koordinaat) ja paarispositsioonidel olevad bitid (Y-koordinaat);
3. arvutada otsitava naaberpunkti koordinaadid;
4. lahutada uued koordinaadid bittideks;
5. kombineerida saadud bitid uuesti arvuks (naaberpunkti mälupesa number).

Failis `xblah1a.pas` toodud lahendus teeb seda puhtmatemaatiliselt, eraldades arvudest bitte korduva kahega jagamise teel ja moodustades bittidest uusi arve korduva kahega korrutamise teel. (Selle kohta lisaks lugemiseks võiks otsida märksõnu "Horneri skeem" või "Horner's scheme" või "Horner scheme".)

Failis `xblah1b.pas` toodud lahendus kasutab sama algoritmi kompaktsemaks üleskirjutamiseks ära asjaolu, et arvuti mälus hoitakse arve niikuinii kahendkujul ja seetõttu on paljudes keeltes olemas spetsiaalsed tehted bittidega opereerimiseks: bitinihked ja bitikaupa loogikatehted.

Loogikatehted on näiteks "loogiline JA" (tähistatakse sageli '$\wedge$'; tehte $A \wedge B$ tulemus on tõene siis ja ainult siis, kui A ja B on mõlemad tõesed), "loogiline VÕI" (tähistatakse sageli '$\vee$'; tehte $A \vee B$ tulemus on väär siis ja ainult siis, kui A ja B on mõlemad väärad) ja "loogiline EI" (tähistatakse sageli '$\neg$'; tehte $\neg A$ tulemus on tõene, kui A on väär, ja väär, kui A on tõene).

Bitikaupa loogikatehted vaatlevad arvu kahendkuju iga bitti tõeväärtusena (0 = väär, 1 = tõene), rakendavad loogikatehteid arvude kohakuti olevatele bittidele ja konstrueerivad saadud bittidest uusi arve. Näiteks

$$12_{10} \wedge 10_{10} = 1100_2 \wedge 1010_2 = 1000_2 = 8_{10},$$
$$12_{10} \vee 10_{10} = 1100_2 \vee 1010_2 = 1110_2 = 14_{10},$$
$$\neg 10_{10} = \neg 1010_2 = 0101_2 = 5_{10}.$$

Kuna märke $\wedge$, $\vee$ ja $\neg$ arvutite klaviatuuridel ei ole, kasutatakse programmikeeltes loogikatehete tähistamiseks muid viise. Pascalis on $\wedge$, $\vee$ ja $\neg$ asemel kasutusel sõnad `AND`, `OR` ja `NOT`, mida võib kasutada nii tavaliste loogikatehete jaoks (tõeväärtustega) kui ka bitiloogika tehete jaoks (täisarvudega). C perekonna keeltes (C, C++, Java jne) on eraldi tähistused tavaliste loogikatehete ja bitiloogika tehete jaoks. Tõeväärtustega opereerimisel on $\wedge$, $\vee$ ja $\neg$ asemel kasutusel `&&`, `||` ja `!`, bitiloogika jaoks aga `&`, `|` ja `~`.

Bitinihked nihutavad arvu kõiki bitte kas vasakule (täites paremal tekkivad tühjad kohad nullidega) või paremale. Kuna kahendsüsteemis on iga positsiooni väärtus kaks korda suurem kui temast paremale jääva oma, tähendab arvu n biti võrra vasakule nihutamine tema korrutamist arvuga 2^n ja paremale nihutamine tema jagamist arvuga 2^n. Pascalis on vasakule ja paremale nihutamise tähistamiseks kasutusel sõnad `SHL` ja `SHR`, C perekonna keeltes `<<` ja `>>`.

Failis `xblah2.pas` toodud lahendus läheb kahendsüsteemi ärakasutamisega veel sammukese kaugemale. Nimelt takistab meil näiteks lähtepunkti alumise naabri mälupesa leidmist lihtsalt mälupesa numbrile ühe liitmise teel ainult asjaolu, et kui selle liitmise käigus peaks tekkima ülekanne, läheb see Y-koordinaadi järgmise biti asemel hoopis X-koordinaadi madalaimasse bitti. Kui see X-koordinaadi madalaim bitt oleks juhtumisi 1, tekiks ka seal ülekanne ja see teine

ülekanne läheks juba õigesse kohta — Y-koordinaadi järgmisse bitti. Seega olekski lahendus enne Y-koordinaadi suurendamist panna ajutiselt kõigi X-koordinaadi bittide väärtuseks 1, teha liitmine ja siis X-koordinaadi bittidele nende esialgsed väärtused tagasi panna.

Selleks ongi failis `xblah2.pas` toodud sisse konstandid `xmask` ja `ymask`: ühel on väärtus 1 kõigis X-koordinaadi, teisel kõigis Y-koordinaadi bittides.

Seega on avaldise `z or xmask` väärtus selline arv, kus Y-koordinaadi bitid on samad, mis muutujas `z`, aga kõigi X-koordinaadi omade väärtus on 1. Siis on `(z or xmask) + 1` väärtus selline, milles Y-koordinaadi bitid on otsitava naaberpunkti omad, aga X-koordinaadi bitid on tõenäoliselt rikutud. Avaldises `((z or xmask) + 1) and ymask` olev `and`-tehe kustutab need rikutud bitid ära ja alles jääb puhas Y-koordinaadi väärtus, aga jagatuna laiali ainult paarisarvulistel positsioonidel olevatesse bittidesse, nii nagu XBoxi aadressides olema peabki.

Analoogiliselt on `z and xmask` väärtus selline arv, kus Y-koordinaadi bitid on kõik nullitud ja kombineerimisel eelmises lõigus saadud tulemusega annabki `(((z or xmask) + 1) and ymask) or (z and xmask)` otsitava aadressi.

Selline keeruline avaldis võimaldab hoiduda korduslausetest ja asendada need bititehetega, mis on kahendsüsteemil põhinevates arvutites välkkiired. Olümpiaadil, kus kõigi kolme lahenduse tööajast lõviosa kulub sisend- ja väljundfailidega toimetamisele, pole see vahe muidugi mõõdetav, aga mängukonsoolis, mis peab igas sekundis töötlema kümneid miljoneid pikseleid, tasub nähtud vaev ennast kuhjaga ära.

Testid

1. Nihe üles, laenamist ei ole. 1 punkt.
2. Nihe üles, laenamine paari järgu võrra. 2 punkti.
3. Nihe üles, laenamine viimasest järgust. 2 punkti.
4. Nihe alla, ülekannet ei ole. 1 punkt.
5. Nihe alla, ülekanne paari järgu võrra. 2 punkti.
6. Nihe alla, ülekanne viimasesse järku. 2 punkti.
7. Nihe vasakule, laenamist ei ole. 1 punkt.
8. Nihe vasakule, laenamine paari järgu võrra. 2 punkti.
9. Nihe vasakule, laenamine viimasest järgust. 2 punkti.
10. Nihe paremale, ülekannet ei ole. 1 punkt.
11. Nihe paremale, ülekanne paari järgu võrra. 2 punkti.
12. Nihe paremale, ülekanne viimasesse järku. 2 punkti.
13. Nihe diagonaalis üles vasakule. 2 punkti.
14. Nihe diagonaalis alla vasakule. 2 punkti.
15. Nihe diagonaalis üles paremale. 2 punkti.
16. Nihe diagonaalis alla paremale. 2 punkti.
17. Nihet ei ole. 2 punkti.

Kokku 30 punkti.

3. Digitaalne pildiraam

Selle ülesande juurde asudes võib kohe tähele panna, et piltide järjestamiseks ei kulu mingil juhul rohkem kui $2 \cdot N$ ümbernimetamist: kindlasti võime kõigepealt ilma konflikte kartmata teha N ümbernimetamist $i \to A_i + N$ ($i \in 1 \ldots N$) ja seejärel veel N ümbernimetamist $j \to j - N$ ($j \in N+1 \ldots 2 \cdot N$). Paraku näeme järgnevalt, et see strateegia pole kunagi optimaalne ja sellel põhinev lahendus seega ühtki punkti ei teeniks.

Optimaalse strateegia leidmiseks peame natuke uurima ümberjärjestuste ehk permutatsioonide struktuuri. Vaatleme näiteks permutatsiooni $1 \to 6$, $2 \to 5$, $3 \to 1$, $4 \to 4$, $5 \to 7$, $6 \to 3$, $7 \to 2$. Esialgne element 1 on vaja viia elemendi 6 kohale, esialgne 6 elemendi 3 kohale, esialgne 3 omakorda elemendi 1 kohale. Samal moel jätkates saame, et uuritav permutatsioon koosneb kolmest tsüklist $1 \to 6 \to 3 \to 1$, $2 \to 5 \to 7 \to 2$, $4 \to 4$, mida sageli märgitakse lühemalt suluavaldisena $(163)(257)(4)$.

Osutub, et sellise lõikumatute tsüklite ühendina on esitatav igasugune lõpliku hulga elementide permutatsioon. Tõepoolest, vaatleme elementi $i = x_1$. Element x_1 on vaja viia kohale $A_i = x_2$, x_2 omakorda kohale $A_{A_i} = x_3$ jne. Kui hulk on lõplik, peame jadas $x_1 \to x_2 \to \ldots \to x_k$ varem või hiljem jõudma tagasi mingi juba selles jadas esinenud elemendi juurde. Olgu esimene kordus $x_k = x_j$. Kui $j > 1$, peaks kaks erinevat elementi (x_{k-1} ja x_{j-1}) olema uues järjestuses samal positsioonil, mis aga ei ole võimalik. Seega $j = 1$ ja olemegi tuvastanud tsükli $x_1 \to x_2 \to \ldots \to x_k = x_1$. Pärast selle elementide eemaldamist esialgsest hulgast peavad ülejäänud elemendid jälle moodustama permutatsiooni, millest saame eraldada järgmise tsükli jne, kuni järele on jäänud tühi hulk.

Iga tsükli $x_1 \to x_2 \to \ldots \to x_k = x_1$ saab ümber paigutada k ümbernimetamisega: $x_k \to N+1$, $x_{k-1} \to x_k$, $\ldots$, $x_1 \to x_2$, $N+1 \to x_1$. Muidugi pole erijuhul, kui tsükkel $x_1 \to x_1$ koosneb vaid ühest elemendist, vaja üldse midagi teha. Samas pole ka raske näha, et mistahes pikema tsükli korral on eeltoodud ühe täiendava ümbernimetamisega skeem optimaalne, sest kohe esimesel ümbernimetamisel ühtki elementi oma lõplikule kohale viia ei saa — igal pool on mõni teine element ees.

Eeltoodul põhinebki failis `dplah1.pas` toodud lahendus, mille ainus tehniline trikk on tsüklite korduva töötlemise vältimiseks juba loendatud elementide nullimine. Loomulikult võiks seda teha ka N-elemendilise tõeväärtusmuutujate massiivi või tsüklite esindajate meetodi (ingl *cycle leader algorithm*) abil.

Peaks olema ilmne, et ümbernimetamiste arv on seda suurem, mida rohkem mittetriviaalseid (st rohkem kui ühest elemendist koosnevaid) tsükleid permutatsioonis on. See arv on maksimaalne, kui kõik tsüklid (välja arvatud viimane tsükkel paaritu N korral) on kaheelemendilised, ja sel juhul on kokku vaja teha $\frac{N}{2} \cdot 3$ ümbernimetamist.

Testid

1. $N = 1$. Minimaalne test. 0 ümbernimetamist. 4 punkti.
2. $N = 2$. Kaks pilti omavahel vahetada. 3 ümbernimetamist. 4 punkti.
3. $N = 4$. Kõik pildid tsükliliselt nihkes. 5 ümbernimetamist. 4 punkti.
4. $N = 6$. Kolm vahetuses paari. 9 ümbernimetamist. 4 punkti.
5. $N = 8$. Kaks õige numbriga pilti ja kolm vahetuses paari. 9 ümbernimetamist. 4 punkti.
6. $N = 10$. Kõik pildid õigete numbritega. 0 ümbernimetamist. 4 punkti.
7. $N = 100$. Juhuslik test. Tsüklid pikkustega 1, 5, 43 ja 51 elementi. 102 ümbernimetamist. 4 punkti.
8. $N = 200$. Juhuslik test. Tsüklid pikkustega 1, 4, 4, 20, 22, 26 ja 123 elementi. 205 ümbernimetamist. 4 punkti.
9. $N = 500$. Juhuslik test. Tsüklid pikkustega 1, 1, 21, 24 ja 453 elementi. 501 ümbernimetamist. 4 punkti.
10. $N = 1000$. Juhuslik test. Tsüklid pikkustega 1, 39, 92 ja 868 elementi. 1002 ümbernimetamist. 4 punkti.

Kokku 40 punkti.

4. Logifailid

Seda ülesannet lahendama asudes paneme esmalt tähele, et faili `access.log.0` võib alati nimetada ümber failiks `access.log.1`. Tõepoolest, kui faili `access.log.1` ei ole, on see ilmselt optimaalne. Kui fail `access.log.1` on olemas, aga mõni teine `access.log.`i puudub, ei või me siiski faili `access.log.0` kohe failiks `access.log.`i nimetada. Ülesande tekstis toodud järjestuse säilimise nõude täitmiseks peaksime ikkagi ka faili `access.log.1` ümber nimetama. Kui me faili `access.log.1` ümbernimetamisest niikuinii ei pääse, võime faili `access.log.0` pärast seda ilma midagi kaotamata igal juhul nimetada ümber failiks `access.log.1`.

Analoogiline arutelu kehtib lisaks failile `access.log.0` ka kõigi teiste failide ümbernimetamisel. Seega on ülesande lahendamiseks vaja leida vähim i, mille korral faili `access.log.`i arhiivis ei ole ja nimetada kõik sellest väiksemate numbritega failid ümber ühe võrra suurema numbriga.

Üks viis vaba koha leidmiseks on arhiiv iga võimaliku i olemasolu tuvastamiseks läbi vaadata, nagu on tehtud failis `lflah0.pas` toodud lahenduses, mis kulutab selleks halvimal juhul $O(M^2)$ operatsiooni.

Teine võimalus on arhiiv ära sorteerida ja seejärel tuvastada vähim vaba number arhiivi ühekordse läbivaatusega. Sellise lahenduse efektiivsus sõltub kasutatavast sorteerimismeetodist. Failides `lflah1.pas` ja `lflah1.c` toodud lahendused kasutavad arhiivi sorteerimiseks loendamismeetodit (ingl *counter sort*), mille keerukus on $O(N)$.

Tasub tähele panna, et eeltoodud arutelu ei kehti juhul, kui meil on vaja optimeerida failide ümbernimetamist pikemas perspektiivis kui ainult ühe uue faili lisamiseks. Täpsem strateegia sõltub sellest, kui paljude uute failide lisamisega on vaja arvestada.

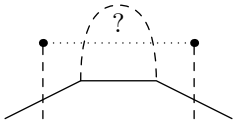
Testid

1. $N = 100$, $M = 0$. Vastus: 1 ümbernimetamine. Lahendus pole ühene. 2 punkti.
2. $N = 20$, $M = 19$. Vastus: 11 ümbernimetamist. 2 punkti.
3. $N = 8$, $M = 7$. Vastus: 1 ümbernimetamine. 2 punkti.
4. $N = 20$, $M = 20$. Vastus: 20 ümbernimetamist. 2 punkti.
5. $N = 4000$, $M = 2501$. Vastus: 1051 ümbernimetamist. Lahendus pole ühene. 3 punkti.
6. $N = 10\,000$, $M = 9999$. Vastus: 1 ümbernimetamine. 3 punkti.
7. $N = 9500$, $M = 9500$. Vastus: 9500 ümbernimetamist. 3 punkti.
8. $N = 10\,000$, $M = 9701$. Vastus: 9001 ümbernimetamist. Lahendus pole ühene. 3 punkti.

Kokku 20 punkti.

5. WiMAX

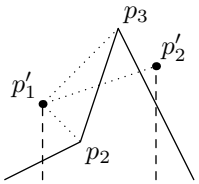
Selle ülesande lahendamisel tasub esimese asjana panna tähele, et otsenähtavus kahe antenni vahel puudub parajasti siis, kui sisendfailis leidub profiilipunkt, mille X-koordinaat on nende antennide X-koordinaatide vahel ja kõrgus suurem kui neid antenne ühendava sirglõigu kõrgus sellel X-koordinaadil. Nagu allolevalt jooniselt näha, ei saaks me sellist tingimust kasutada, kui poleks eeldust, et profiil on antud punktide vahel lineaarne.



Üks võimalus ongi vaadelda kõiki antennipaare ja kontrollida iga paari jaoks vahetult, kas mõni nende vahel asuvatest profiilipunktidest on liiga kõrgel. Antenne ühendava sirglõigu kõrguse otsene väljaarvutamine profiilipunkti kõrgusega võrdlemiseks võib aga piirjuhtudel ümardamisvigade tõttu valesid tulemusi anda. Parem on punktide kõrguste võrdlemiseks kasutada vektorkorrutist. Näiteks alloleval joonisel toodud olukorras oleks

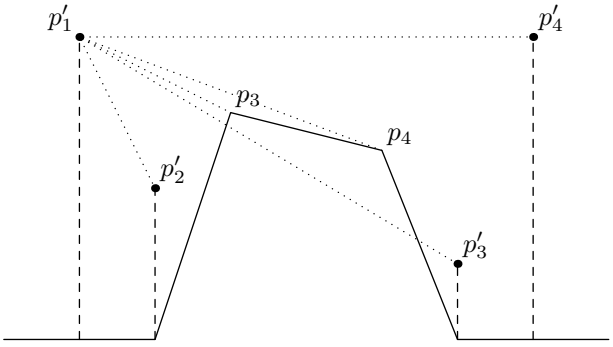
$$\vec{p'_1p'_2} \times \vec{p'_1p_2} = (x'_2 - x'_1)(y_2 - y'_1) - (x_2 - x'_1)(y'_2 - y'_1)$$

negatiivne, andes märku, et p_2 on madalamal kui p'_1 ja p'_2 ühendav sirglõik; $\vec{p'_1p'_2} \times \vec{p'_1p_3}$ oleks aga positiivne. Kui punktide koordinaadid on täisarvulised, on seda ka neist moodustatud vektorite korrutised ja ümardamisvigu pole vaja karta (küll aga tuleks mõelda ületäitumise ohule).



Sellel ideel põhinebki failis `wmlah0.pas` toodud lahendus, mis aga jääb suurimas testis ajahätta, sest vaatab läbi $O(M^2)$ antennipaari ja iga antennipaari kohta $O(N)$ profiilipunkti, tehes kokku $O(M^2N) \approx 10^9$ operatsiooni.

Efektiivsema lahenduse saamiseks paneme tähele, et kuigi alloleval joonisel antennist p'_1 paremale jääv profiilipunkt p_3 võib teoreetiliselt mõjutada p'_1 nähtavust kõigist neist antennidest, mis jäävad omakorda punktist p_3 paremale, on ta tegelikult ebaoluline, sest p_4 paistab p'_1 juures olevale vaatlejale temast kõrgem. Sellest saamegi idee vaadata iga antenni jaoks läbi kõik temast paremale jäävad objektid nende X-koordinaatide kasvamise järjekorras ja võrrelda iga antenni ainult "kõrgeima" seni nähtud profiilipunktiga.

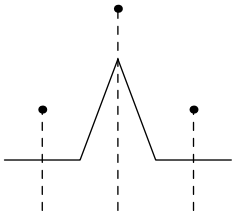


Seega tuleb meil iga antenni jaoks teha $O(M + N)$ operatsiooni, mis annab lahenduse kogukeerukuseks $O(M(M + N)) \approx 10^6$ operatsiooni.

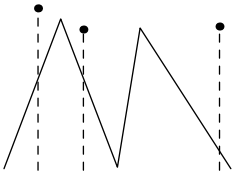
Failis `wmlah1.pas` toodud lahendus koondab kõik sisendfailis antud punktid ilmutatud kujul ühte jadasse. Teine (mitte oluliselt keerulisem) võimalus oleks töödelda profiilipunktide ja antennide jadasid paralleelselt.

Testid

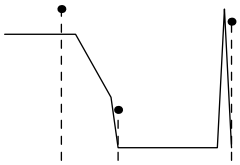
1. $N = 5$, $M = 3$. Väike lihtne test. 3 punkti.



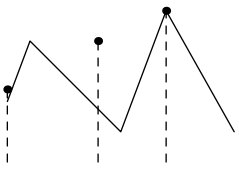
2. $N = 5$, $M = 3$. Piirjuhud: esimest tippu riivatakse, teist läbitakse väga napilt. 3 punkti.



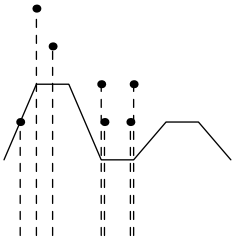
3. $N = 7$, $M = 3$. Piirjuht: lauget nõlva riivatakse. 3 punkti.



4. $N = 5$, $M = 3$. Piirjuht: antenn täpselt maapinnal (tipus). 3 punkti.



5. $N = 8$, $M = 7$. Piirjuht: antenn täpselt maapinnal (nõlval). 3 punkti.



6. $N = 50$, $M = 50$. Juhuslik test. 3 punkti.
7. $N = 100$, $M = 100$. Juhuslik test. 3 punkti.
8. $N = 200$, $M = 200$. Juhuslik test. 3 punkti.
9. $N = 500$, $M = 500$. Juhuslik test. 3 punkti.
10. $N = 1000$, $M = 1000$. Juhuslik test. Kõik näevad kõiki. Naiivne lahendus jääb ajahätta. 3 punkti.

Kokku 30 punkti.

6. Lehelugemine

Üldiselt ei ole selle ülesande lahendamiseks teada paremat algoritmi kui ajalehtede kõigi võimalike järjestuste läbivaatus.

Keeltes, kus antud elementide hulga ümberjärjestuste (permutatsioonide) genereerimine on põhivarustuses olemas, ongi kõige lihtsam seda kasutada ja iga võimaliku ümberjärjestuse jaoks arvutada lehtede selles järjekorras lugemisele kuluv koguaeg. Nii ongi tehtud failis `lllah1.cpp` toodud lahenduses (mis kasutab ära asjaolu, et kui lehes pole ühtki uut uudist, siis selle lugemine ülesande püstituse kohaselt aega ei võta ja väljastab alati kõigi N ajalehe numbrid) ja failis `lllah1.py` toodud lahenduses (mis väljastab ainult nende ajalehtede numbrid, kust midagi loetakse ka).

Teine võimalus on programmeerida variantide läbivaatus tagurdusmeetodi abil. See on eelmisest märksa efektiivsem, sest variantide järk-järguline koostamine võimaldab sageli juba üsna varakult märgata, et parajasti käsil olevast osalisest variandist ei ole mingil juhul võimalik saada senisest parimast tulemusest paremat, ja jätta seega koostamata kõik selle osalise variandi võimalikud lõpetused. Selline lahendus on toodud failis `lllah2.pas`.

Testid

1. Näide ülesande teksti esialgsest versioonist. 0,5 punkti.
2. Minimaalne test: 1 ajaleht, 1 lehekülg, 1 sündmus. 0,5 punkti.
3. 1 ajaleht, 50 lehekülge, 1 sündmus. 0,5 punkti.
4. 1 ajaleht, 50 lehekülge, 1 sündmus. 0,5 punkti.
5. 1 ajaleht, 50 lehekülge, 50 sündmust. 0,5 punkti.
6. 1 ajaleht, 50 lehekülge, 25 sündmust. 0,5 punkti.

7.–11. Nagu 2.–6., kuid lisatud üks tühi ajaleht. 0,5 punkti testi kohta.

12.–16. Nagu 2.–6., kuid lisatud üks mittetühi ajaleht. 0,5 punkti testi kohta.

17.–20. Variatsioon 13.–16.-st, kus on kolm ajalehte. 0,5 punkti testi kohta.

21.–25. $4, 5, 6, 7, 8$ ajalehte, 5 leheülge, $N \cdot M$ sündmust. 2 punkti testi kohta.

26. 8 ajalehte, 50 lehekülge, $N \cdot M$ sündmust. 2 punkti.
27. 8 ajalehte, 50 lehekülge, 1 sündmus. 2 punkti.
28. 8 ajalehte, 50 lehekülge, 1 sündmus. 2 punkti.
29. 8 ajalehte, 40 lehekülge, 40 sündmust, juhuslik jaotus. 2 punkti.
30. 8 ajalehte, 40 lehekülge, 120 sündmust, juhuslik jaotus. 2 punkti.

31.–50. Juhuslikud testid: $1 \ldots 8$ ajalehte, $1 \ldots 50$ lehekülge, $1 \ldots N \cdot M$ sündmust, juhuslik jaotus. 1 punkt testi kohta.

Kokku 50 punkti.