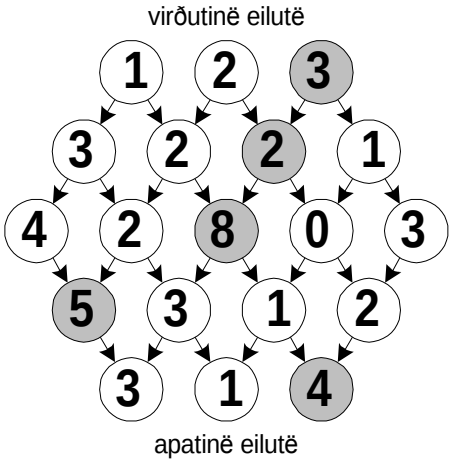
Šeštoji Baltijos šalių informatikos olimpiada

KORYS Jei nebūtų leidžiamas sukeitimas, tai būtų klasikinis dinaminio programavimo uždavinys. Korys sutalpinamas į dvimatę lentelę. Po to korys peržiūrimas iš viršaus į apačią (ir iš kairės į dešinę) ir kiekvienoje korio akutėje įrašytas skaičius pakeičiamas didžiausia akučių, vedančių iki tos korio akutės, suma. Ieškomoji suma būtų lygi didžiausiam apatinės eilutės skaičiui.



Korio kraštinė lygi 3.

Prieš modifikuodami šį sprendimą atkreipkime dėmesį, kad nėra akivaizdu kaip sutalpinti visas eilutes į atmintį. Todėl rašydami programą, darome paprasčiau: atmintyje laikome tik naujai perskaitytą ir buvusią korio eilutes. Mat ir aukščiau aprašytu atveju ir modifikuotame sprendime norint užpildyti kažkurią eilutę (t.y. kiekvienoje nagrinėjamos eilutės akutėje įrašytą skaičių pakeisti didžiausia akučių, vedančių iki tos akutės, suma), reikia žinoti tik nagrinėjamą ir prieš tai buvusią eilutę.

Modifikuojame turimą sprendimą.

Algoritme vartosime keturias lenteles:

eilutė – naujai perskaityta korio eilutė;

nekeista – korio eilutė; jos kiekviename langelyje įrašyta didžiausia kelią iki to langelio sudarančių akučių suma, jei *sukeitimas iki šiol nebuvo atliktas;*

sukeista_seniau – korio eilutė; jos kiekviename langelyje įrašyta didžiausia kelią iki to langelio sudarančių akučių suma, jei *sukeitimas būtinai buvo atliktas ankstesnėje (ne šioje) eilutėje;*

sukeista_dabar – korio eilutė; jos kiekviename langelyje įrašyta didžiausia kelią iki to langelio sudarančių akučių suma, jei *sukeitimas atliktas šioje eilutėje;*

Perskaitome pirmąją korio eilutę; Ją įsimename lentelėje *nekeista*.

nekeista[i] := *eilutė*[i];

Jei šioje eilutėje būtų atliekamas sukeitimas, tai visus jos elementus būtų galima pakeisti maksimalia reikšme. Tą ir padarome. Rezultatą įsimename lentelėje *sukeista_seniau*

sukeista_seniau[i] :=max(*eilutė*)

Perskaitome tolesnę eilutę. Galimi trys variantai.

a) *sukeitimas iki šiol nebuvo darytas ir nebus daromas šioje eilutėje;*

Tuomet remdamiesi lentelėmis *eilutė* bei *nekeista* atnaujiname eilutę *nekeista*, t.y.randame kelius su didžiausia suma nuo viršutinės eilutės iki perskaitytosios eilutės kiekvieno elemento.

b) *Sukeitimas jau buvo padarytas ankstesnėje eilutėje.*

Tuomet remdamiesi lentelėmis *eilutė* bei *sukeista_seniau* atnaujiname eilutę *sukeista_seniau*, t.y. randame ilgiausius kelius nuo viršutinės eilutės iki perskaitytosios eilutės kiekvieno elemento.

c) *Sukeitimas daromas šioje eilutėje.*

Randame didžiausią lentelės *eilutė* elementą. Jį pridedame prie visų lentelės *nekeista* elementų ir rezultatą priskiriame lentelei *sukeista_dabar*. Eilutės *nekeista* reikšmė nepasikeičia.

Taigi, perskaitę naują eilutę, perskaičiuojame lentelių *nekeista, sukeista_seniau, sukeista_dabar* reikšmes. Tolesniems skaičiavimams turės įtakos tik tai ar buvo atliktas sukeitimas, ar ne. Nebebus svarbu kurioje iš ankstesnių eilučių tai buvo padaryta. Todėl dar kartą atnaujiname eilutę *sukeista_seniau*:

sukeista_seniau[*i*] := max(*sukeista_seniau*[*i*], *sukeista_dabar*[i]), *i* kinta nuo 1 iki eilutės ilgio.

Aukščiau aprašyti veiksmai kartojami kol peržiūrimos visos eilutės. Tada belieka išrinkti didžiausią lentelių *nekeista* bei *sukeista_seniau* elementą. Tai ir bus ieškomoji suma.

Reikia nepamiršti, kad duotas korys, o ne pavyzdžiui, kvadratinė lentelė. T.y. reikia nepamiršti, kad keičiasi eilučių ilgiai, kad į kai kurias akutes iš aukščiau esančios eilutės ves tik vienas kelias, o ne du ar atvirkščiai. Kuriant programą labai nesunku atsižvelgti į šiuos faktorius. Todėl aprašydami algoritmą į tai nesigilinome.

Panagrinėkime sąlygoje pateiktą pavyzdį. Trumpumo dėlei imkime ne visą korį, o tik pirmąsias keturias jo eilutes ir jose pabandykime rasti kelią su didžiausia suma.

kelio su didžiausia suma ieškokime ne visame kory

Imkime sąlygos piešinyje pateiktą korį ir panagrinėkime kaip bus atliekamas algoritmas. Perskaitome pirmąją korio eilutę:

eilutė	1	2	3		

Ją įsimename lentelėje *nekeista*:

nekeista	1	2	3		

Randame didžiausią pirmosios eilutės elementą ir juo užpildome lentelę *sukeista_seniau.*

Sukeista_seniau	3	3	3		

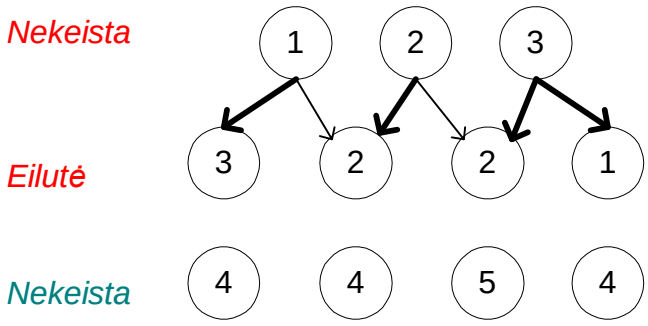
Perskaitome tolesnę eilutę:

eilutė	3	2	2	1	

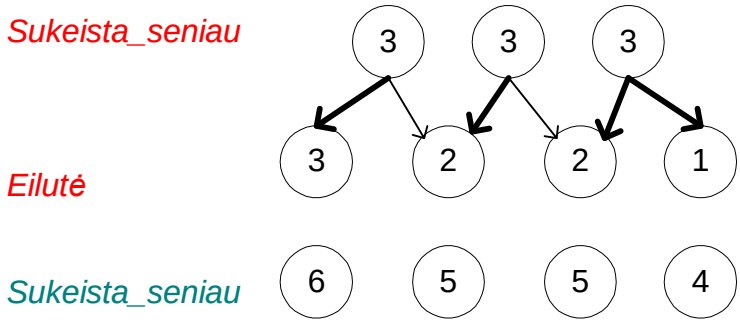
Žemiau pateikti piešiniai iliustruoja tolesnį algoritmo veikimą:

1 žingsnis

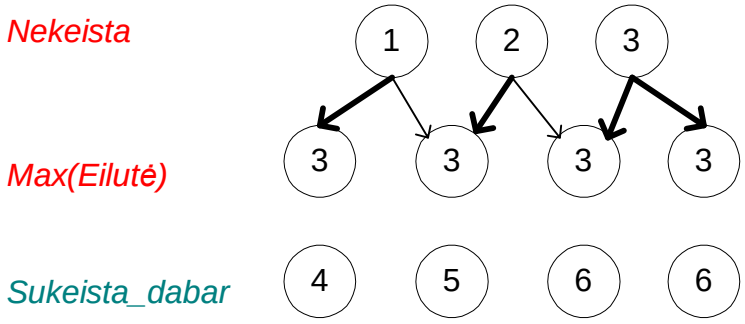
a)



b)



c)



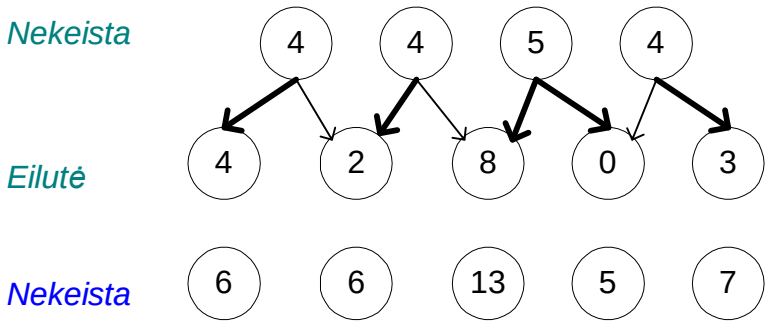
Atnaujiname eilutę *Sukeista_seniau*:

2 žingsnis

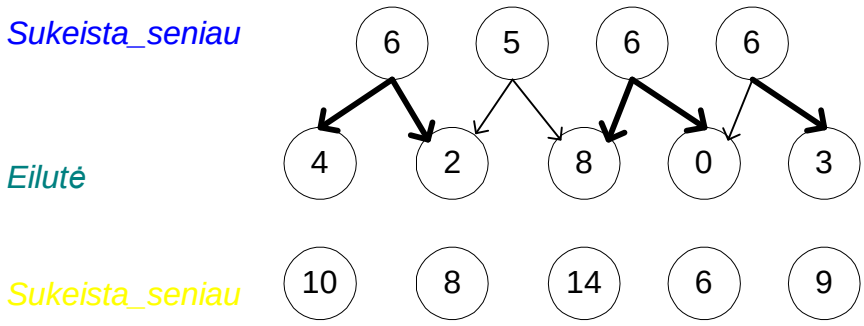
Perskaitome tolesnę eilutę:

eilutė	4	2	8	0	3

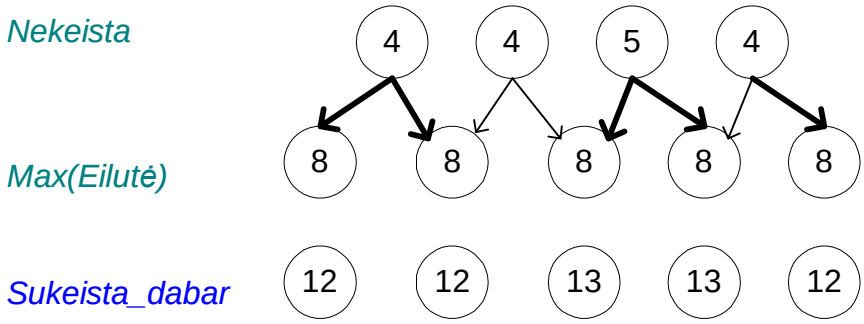
a)



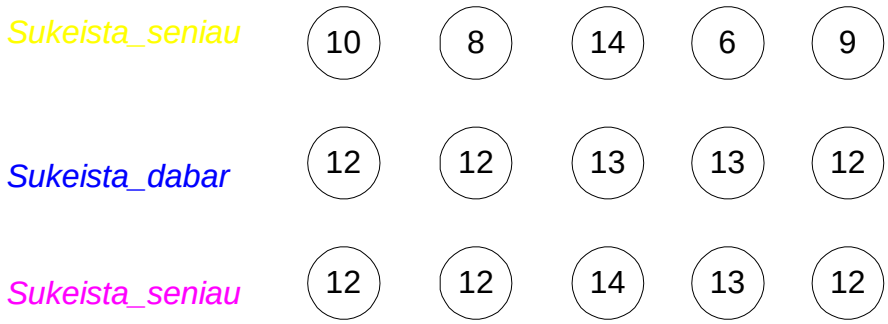
b)



c)



Atnaujiname eilutę *Sukeista_seniau*:

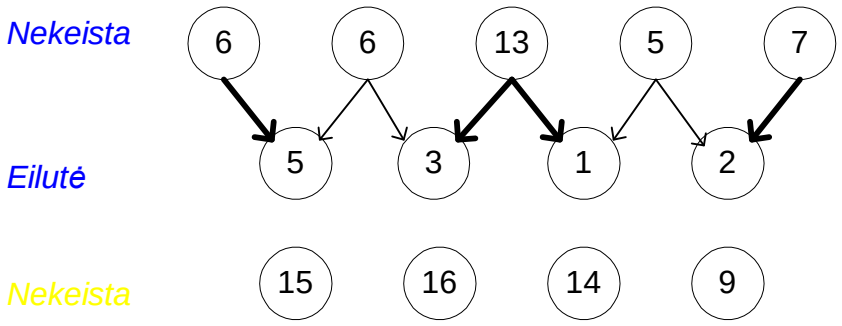


3 žingsnis

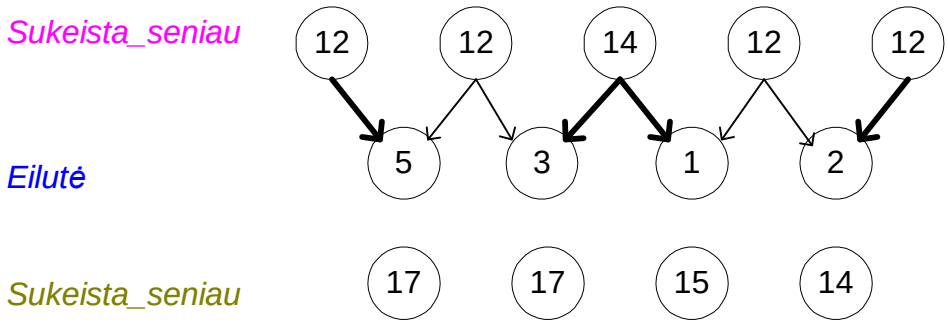
Perskaitome tolesnę eilutę:

eilutė	5	3	1	2	

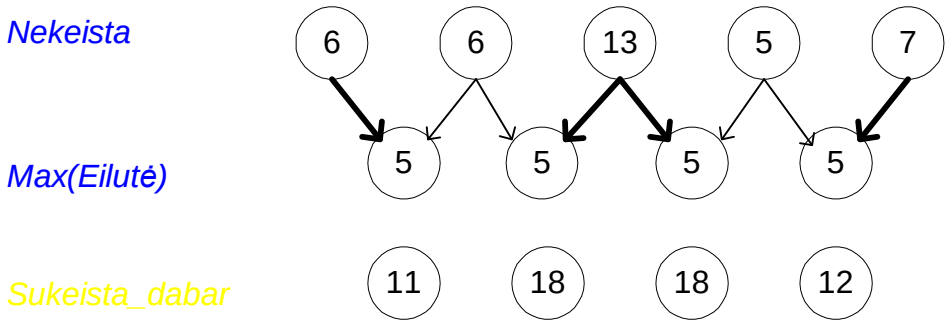
a)



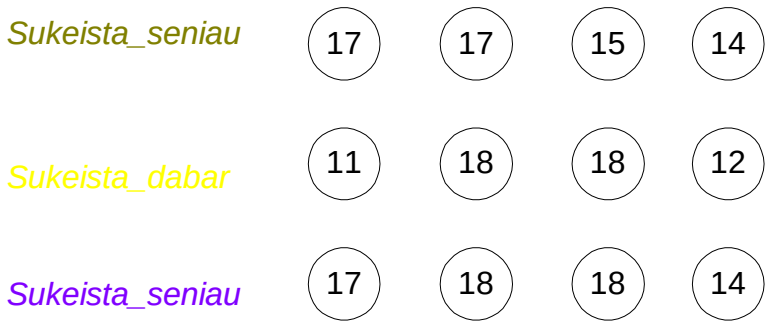
b)



c)



Atnaujiname eilutę *Sukeista_seniau*:



Išrenkame didžiausią eilučių *nekeista* bei *sukeista_seniau* elementą (18). Tai ir yra ieškoma kelio su didžiausia suma vertė. Atlikę dar porą žingsnių rastume ir didžiausią kelio, einančio per visą korį, akučių sumą.

LAIKO JUOSTOS Pirmiausia sudaroma $n \times n$ dydžio kvadratinė lentelė *len*. Lentelės elemento *len*[*pran, juosta*] reikšmė lygi gavėjo juostos laikui, kada turėjo ateiti pranešimas, kurio numeris *pran*, jei tas pranešimas buvo išsiųstas iš laiko juostos *juosta*.

Pavyzdžiui, antrasis pranešimas iš vietinės laiko juostos buvo išsiųstas 02:50 val. Jeigu jis buvo išsiųstas iš pirmosios laiko juostos, tai gavėją turėjo pasiekti 03:50 val, jei iš antrosios – 04:50 val ir pan. T.y. *len*[2, 1] = 03:50; *len*[2, 2] = 04:50 ir t.t.

Imkime konkretų pavyzdį. Sakykime, buvo gauti penki pranešimai. Jie buvo išsiųsti žemiau esančioje lentelėje pateiktais laiko momentais. Bus patogiau, jei pranešimus sunumeruosime jų atėjimo tvarka.

Pranešimo nr.	*Vietinis išsiuntimo laikas*
1	00:17
2	02:50
3	04:00
4	02:01
5	00:02

Užpildome lentelę *len*.

Len	0 juosta	1 juosta	2 juosta	3 juosta	4 juosta
1 pranešimas	00:17	01:17	02:17	03:17	04:17
2 pranešimas	02:50	03:50	04:50	05:50	06:50
3 pranešimas	04:00	05:00	06:00	07:00	08:00
4 pranešimas	02:01	03:01	04:01	05:01	06:01
5 pranešimas	00:02	01:02	02:02	03:02	04:02

Kadangi para turi tik 5 valandas, tai pranešimai galėjo ateiti nuo 0:00 val iki 4:59 val. Pilkai pažymime negalimus atvejus:

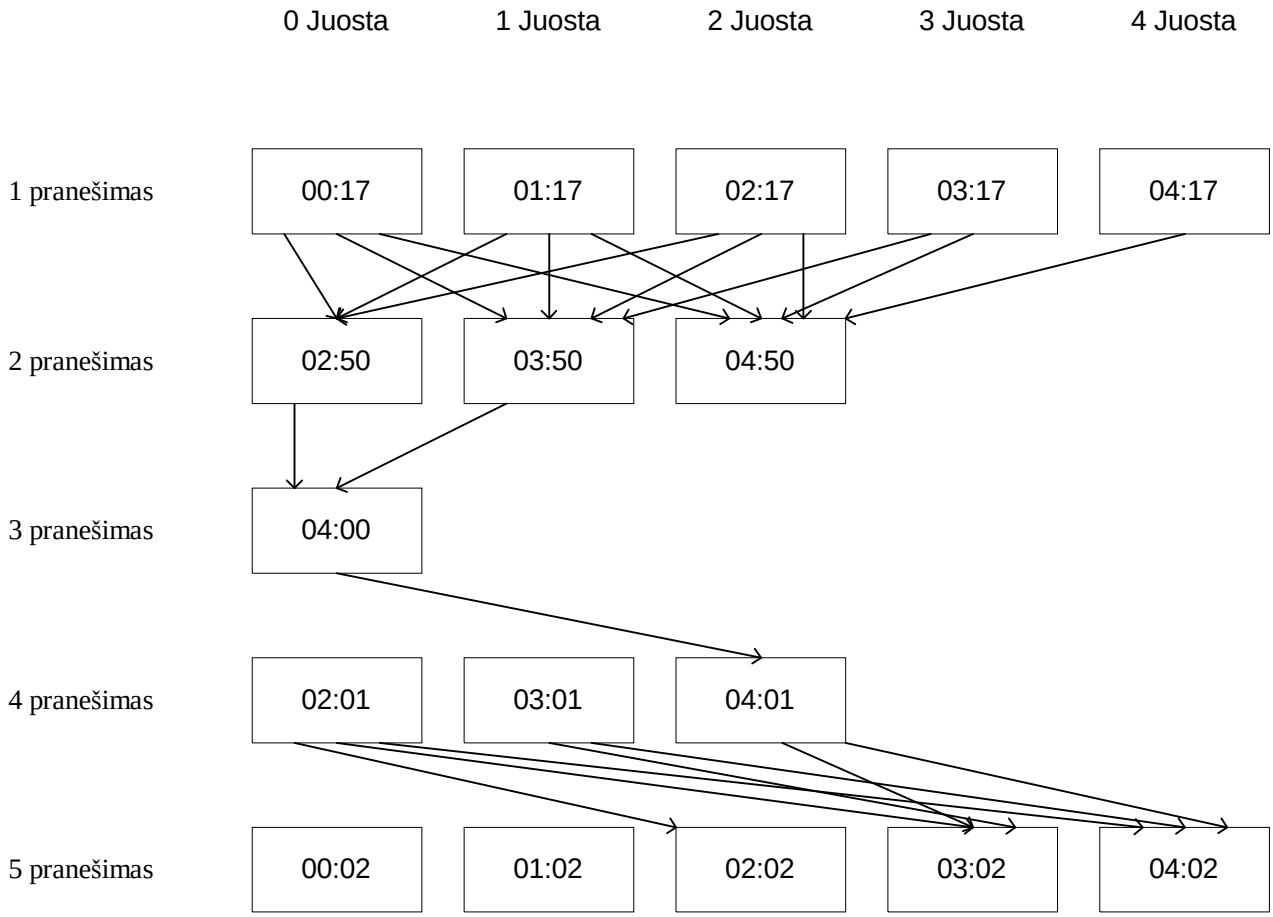
.

len	0 juosta	1 juosta	2 juosta	3 juosta	4 juosta
1 pranešimas	00:17	01:17	02:17	03:17	04:17

2 pranešimas	02:50	03:50	04:50	05:50	06:50
3 pranešimas	04:00	05:00	06:00	07:00	08:00
4 pranešimas	02:01	03:01	04:01	05:01	06:01
5 pranešimas	00:02	01:02	02:02	03:02	04:02

Remdamiesi šia lentele sudarome grafą. Kiekvienas lentelės elementas yra grafo viršūnė. Iš viršūnes A į viršūnę B eis lankas, jei pranešimai A ir B atėjo vienas po kito (pirmiau A, po to – B).

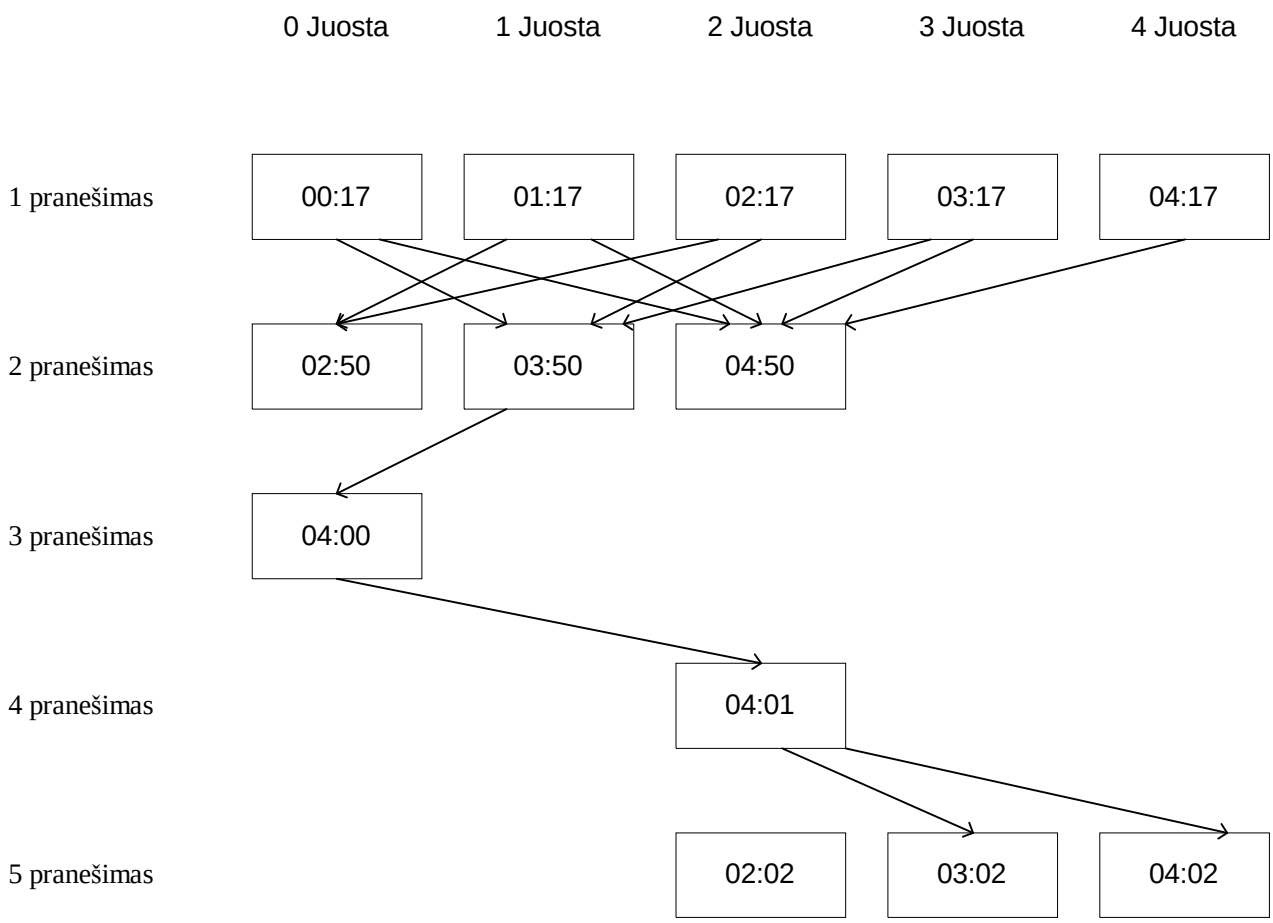
Aukščiau aprašytai lentelei gautume tokį grafą:



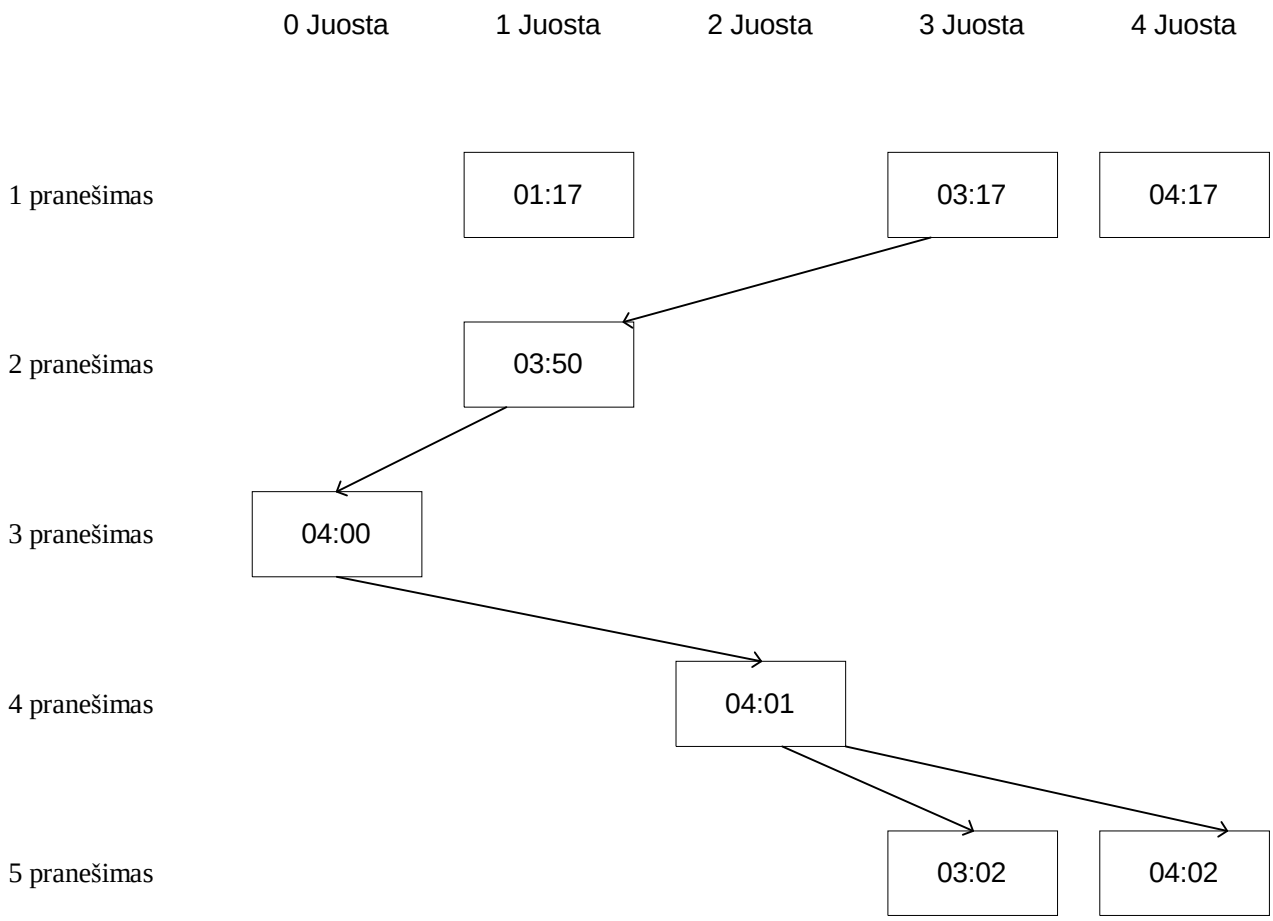
Uždavinio sprendimas suvedamas į kelio paiešką šiame grafe. Reikia rasti bet kurį kelią grafe, kuris prasidėtų bet kuria viršūne atitinkančia pirmąjį pranešimą ir baigtųsi bet kuria viršūne atitinkančią paskutinįjį (penktąjį) pranešimą. Visos šiame kelyje esančios viršūnės turi atitikti skirtingas laiko juostas.

Prieš pradedant paiešką, grafą galima supaprastinti. Pašaliname visas viršūnes iš kurių neišeina joks lankas (nebent jos būtų kelio pabaigoje) ir į kurias neateina joks lankas (nebent jos būtų kelio pradžioje). Taip pat pašaliname visas briaunas, jungiančias dvi tai pačiai laiko juostai

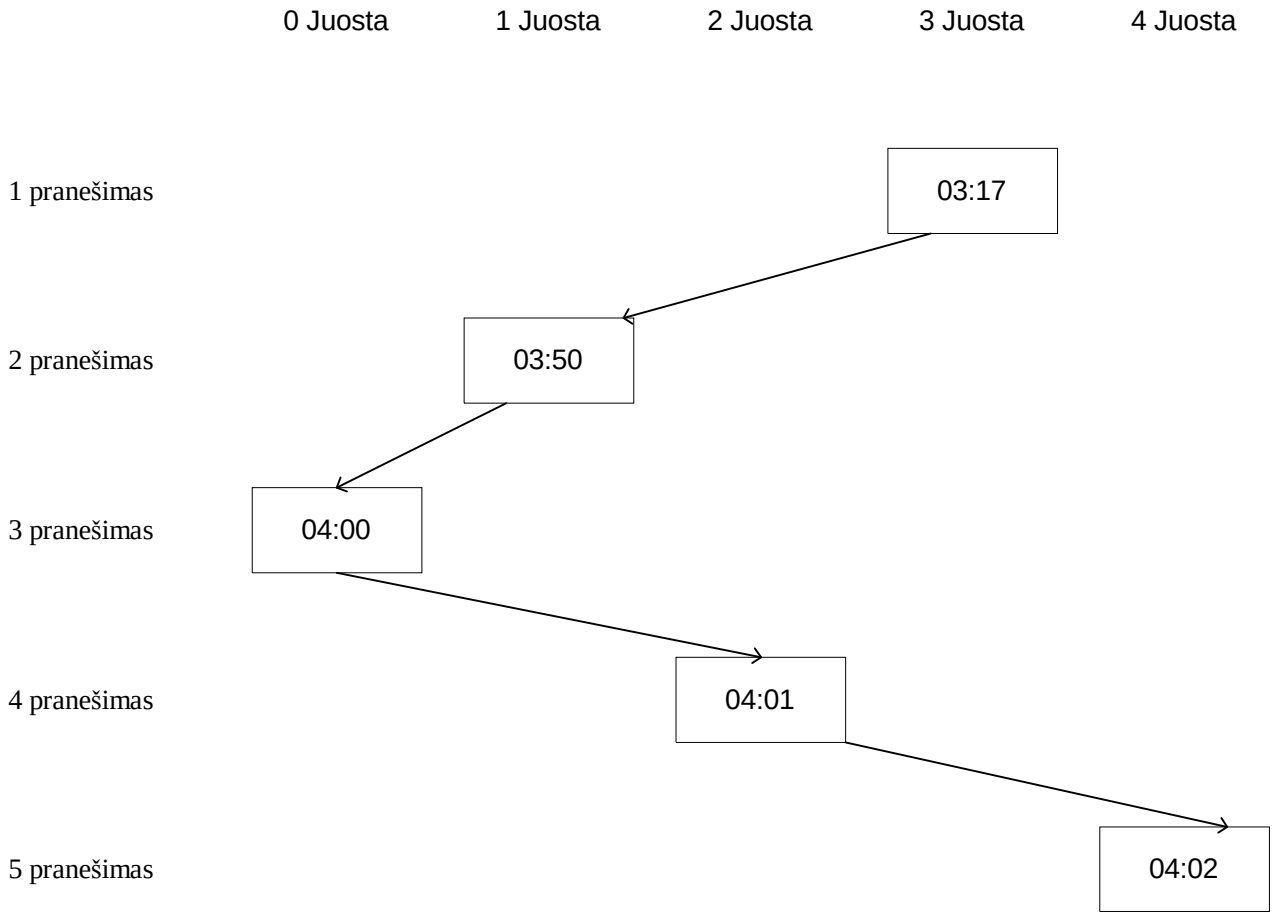
priklausančias viršūnes (žinome, kad visi pranešimai atėjo iš skirtingų laiko juostų). Gauname tokį grafą:



Matome, kad pašalinus atsirado naujų viršūnių į kurias niekas neveda. O taip pat iš kurių niekas neišeina. Be to, tapo aišku, kad trečiasis pranešimas tikrai atėjo iš nulinės laiko juostos, o ketvirtasis iš antrosios. Taigi, galime pašalinti visas kitas grafo viršūnes, atitinkančias nulinę ir antrąją laiko juostas.

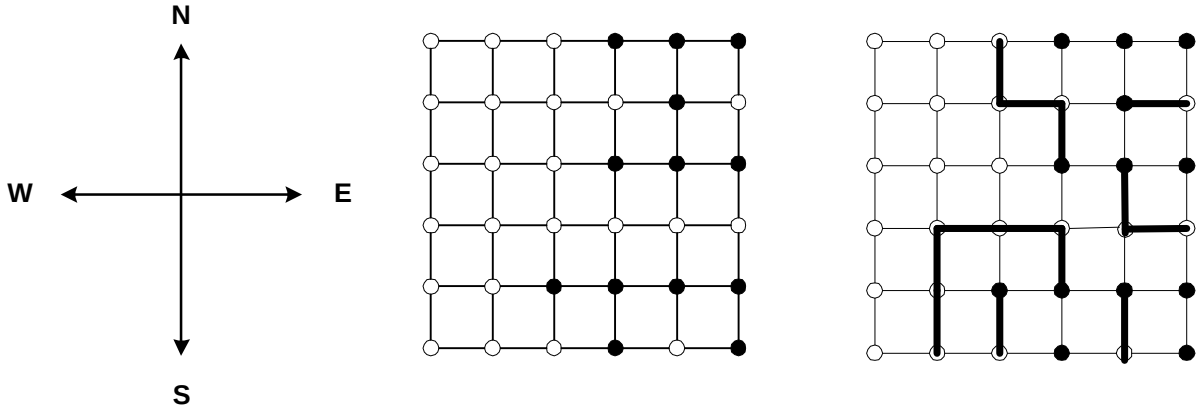


Dar kartą atlikę aukščiau aprašytus veiksmus gauname grafą, kuriame yra vienintelis kelias:



Bendru atveju pašalinus nereikalingas viršūnes bei briaunas, gausite grafą, kuriame bus daugiau nei vienas kelias iš pradinės viršūnės į galinę. Tada reikės perrinkinėti įvairius variantus kol bus gautas sprendinys. Sąlygoje įvestas ribojimas, kad egzistuoja vienintelis sprendinys labai sumažina perrenkamų atvejų skaičių.

ELEKTRONINĖ PLOKŠTELĖ



Tinklelio su kontaktais pavyzdys(centre) ir to pavyzdžio sprendimas (dešinėje)

Šį uždavinį galime suvesti į *Maksimalaus srauto* uždavinį. Priminsime kas tai yra. Turime orientuotą grafą, kurį sudaro viršūnių ir briaunų aibė. Grafas yra svorinis, t. y. kiekviena briauna (*u*, *v*) turi neneigiamą svorį *sv*(*u*, *v*) > 0. Tą svorį vadinkime briaunos *pralaidumu*. Dvi grafo viršūnės išsiskiria iš kitų viršūnių: tai pradinė viršūnė *p* ir galinė viršūnė *g*. Susitarkime, kad grafas pasižymi tokia savybe: iš pradinės viršūnės *p* galima pasiekti bet kurią kitą viršūnę ir iš bet kurios viršūnės galime pasiekti galinę viršūnę *g*.

Toks grafas gali modeliuoti ne tik kelių žemėlapį. Tai gali būti ir elektros grandinė ar namo šildymo sistema. Pavyzdžiui, į namą ateina storas vamzdis su karštu vandeniu ir išsišakoja į storesnius ir plonesnius vamzdžius. Tie vamzdžiai išvedžiojami po namą, tai sujungiami, tai vėl atskiriami įvairiuose aukštuose, kol ataušęs vanduo surenkamas kitame vamzdyje. Žinomi visų vamzdžių pralaidumai. Koks didžiausias kiekis vandens gali ateiti į storąjį vamzdį.

Srautai s yra apibrėžti visoms grafo briaunoms. Jie pasižymi šiomis savybėmis:

1) *s*(*u*, *v*) ≤ *sv*(*u*, *v*). Tai reiškia, kad briauna praleidžiami srautai negali viršyti briaunos pralaidumo. Pavyzdžiui, jei siauroje gatvelėje tiesiamas kelias ir jame tilps tik dvi juostos, negalima leisti mašinoms važiuoti trimis juostomis.

2) *s*(*u*, *v*) = -*s*(*v*, *u*). Į viršūnę ateinantys srautai žymimi teigiamu skaičiumi, iš jos išeinantys – neigiamu.

3) Kiekvienai viršūnei, išskyrus *p* ir *g*, galioja taisyklė: *Į ją ateinančių srautų suma lygi iš jos išeinančių srautų sumai*. Jei taisyklės nebūtų laikomasi galėtų susidaryti nekorektiška situacija. Pavyzdžiui, į sankryžą mašinos atvažiuoja penkiomis eismo juostomis, išvažiuoja tik viena. Arba iš pastato išeina daugiau žmonių nei įėjo. Matyt, kažkas lipo per langą....

Srautu grafe vadinsime visų iš pradinės viršūnės *p* išeinančių srautų sumą.

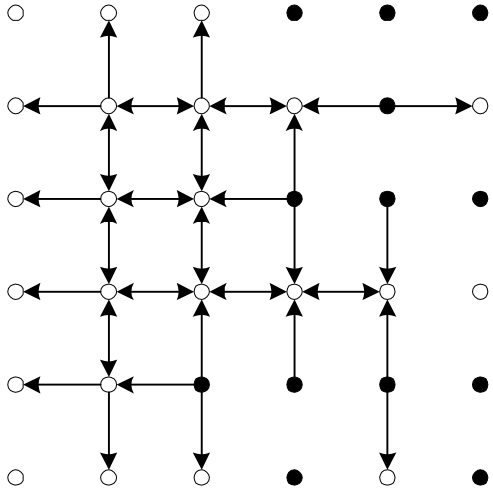
Grįškime prie mūsų uždavinio ir jį suveskime į Maksimalaus srauto uždavinį. Turima elektroninė plokštelė iš tiesų yra orientuotas grafas. Visi mazgai tampa grafo viršūnėmis. Jei tinklelio linija iš mazgo (kontakto) *u* į mazgą *v* galima tiesti grandinę, tuomet ši linija yra grafo lankas vedantis iš *u* į *v*.

Sudarant grafą būtina nepamiršti, kad:

1) lankas gali tik išeiti iš grafo viršūnės su kontaktu, bet negali į ją ateiti;

2) lankas negali išeiti iš krašte esančios viršūnės;

Sąlygoje pateiktą pavyzdį atitiktų toks grafas:

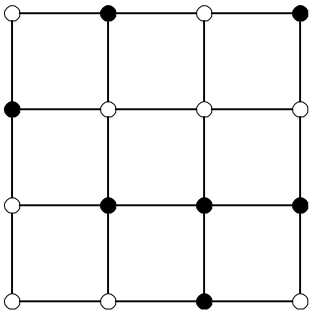


Matome, kad ne kiekviena tinklelio kraštinė tampa grafo lanku.

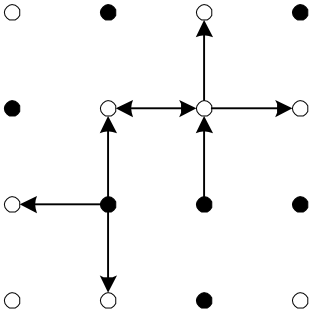
Visos grandinės prasideda kontaktuose ir baigiasi kraštiniuose mazguose. Papildomai sukuriame vieną pradinę ir vieną galinę viršūnes. Iš pradinės viršūnės nuvedame lankus į visas viršūnes su kontaktais, o iš visų krašte esančių viršūnių nuvedame lankus į galinę viršūnę. Gauname maksimalaus srauto uždavinį: maksimalus srautas, kurį galima praleisti iš pradinės viršūnės į galinę lygus maksimaliam grandinių, jungiančių kontaktus su kraštiniais mazgais, skaičiui. Grafo lankų bei viršūnių pralaidumai lygūs vienetui.

Pavyzdyje pateiktą grafą atitiktų gan sudėtingas paveikslas, todėl imkime paprastesnį 4×4 dydžio tinklelį:

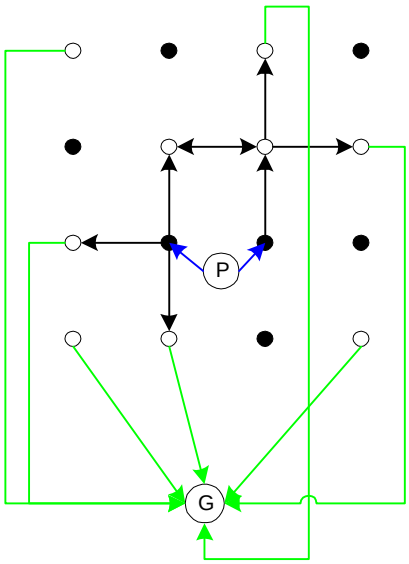
a) tinklelis:



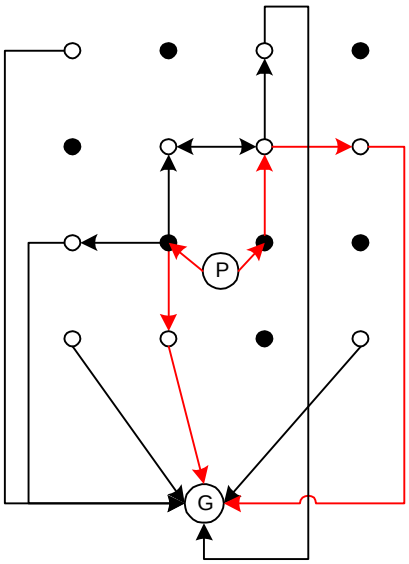
b) tinklelį atitinkantis grafas



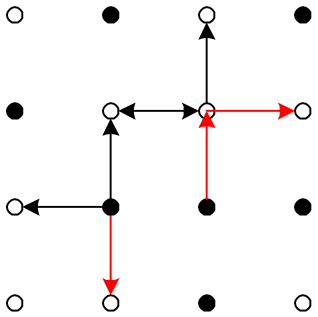
c) tas pats grafas suvestas į maksimalaus srauto uždavinį; Pradinė viršūnė pažymėta raide P, galinė – raide G, papildomi lankai – kitokiomis (??? Šiame paveiksle – kitos spalvos) linijomis



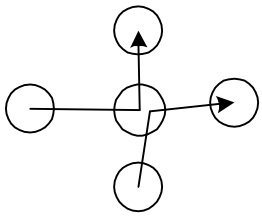
d) Grafe rastas galimas maksimalus srautas; Paveiksle srautas pažymėtas kitokia linija (?? Raudonos spalvos).



e) Galimas sprendinys (maksimalus elektrinių grandinių skaičius) gautas remiantis rastuoju maksimaliu srautu:



Šis uždavinys nuo įprastinio *Maksimalaus srauto* uždavinio skiriasi tuo, kad ribojami ne tik briaunų, bet ir mazgų pralaidumai. Mazgų pralaidumai turi būti ribojami, nes vienu mazgu gali būti tiesiama tik viena grandinė.

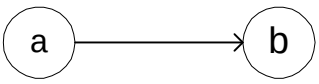


Įprastame maksimalaus srauto uždavinyje toks srauto tekėjimas galimas, nes bakų (viršūnių) talpos neribojamos. Tuo tarpu šiame uždavinyje to negali būti, nes draudžiama tiesti kelias elektrines grandines per tą patį mazgą.

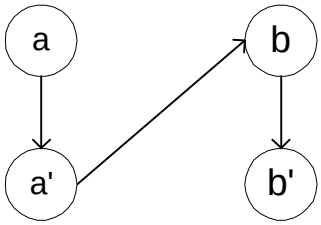
Suvesime šį uždavinį į įprastinį Maksimalaus srauto uždavinį, kai mazgų pralaidumai yra neriboti. Tam sukuriame pradinio tinklelio kopiją ir vieną tinklelį „uždedame“ ant kito tinklelio. Tuomet iš pradinės viršūnės vedame lankus į viršutinio tinklelio mazgus su kontaktais, o su galiniu mazgu jungiame visus apatinio tinklelio kraštinius mazgus.

Pirmiausia, nuvedami lankai iš visų kraštuose esančių kontaktų į juos atitinkančius apatinio tinklelio mazgus.

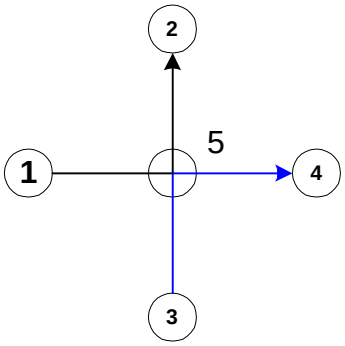
Viršutinio tinklelio mazgas gali praleisti srautą į tiesiai po juo esantį apatinio tinklelio mazgą. O pastarasis mazgas gali praleisti srautą tik į virš jo esančio viršutinio tinklelio mazgo kaimyninius mazgus (šiaurinį, pietinį, rytinį, vakarinį). Jei iš mazgo *a* turėjo eiti lankas į mazgą *b*, tai dabar lankas į mazgą *b* eis iš mazgą *a* atitinkančio apatinio tinklelio mazgo *a*’.



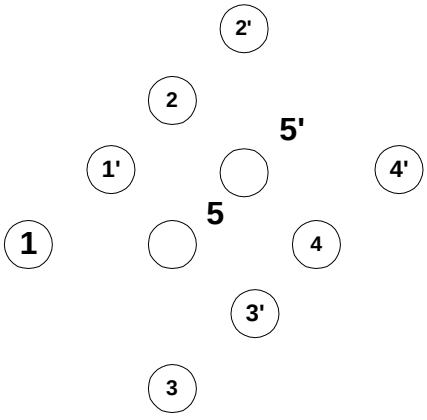
Aukščiau esantis lankas keičiamas žemiau pavaizduotais trimis lankais



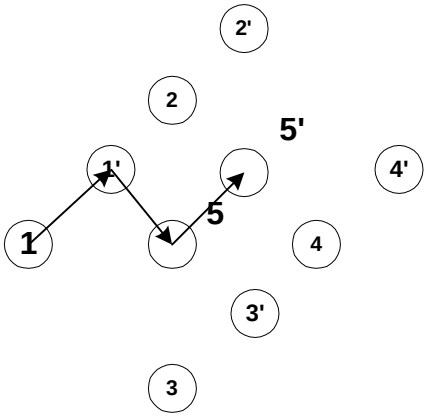
Panagrinėkime pavyzdį. Imkime tinklelio dalį.



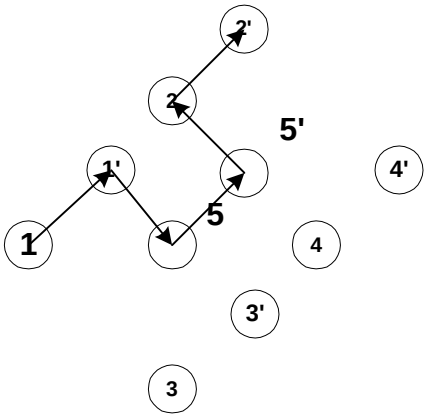
Grandinė tiesiama iš pirmojo mazgo į antrą ir iš antrojo į ketvirtą. Abu kartus grandinė tiesiama per penktą mazgą. To negali būti: per vieną mazgą gali eiti tik viena grandinė. Padarome šios tinklelio dalies kopiją ir vieną tinklelio dalį uždedame ant kitos. Iš viršutinio tinklelio mazgų nutiesiame lankus į atitinkamus apatinio tinklelio mazgus:



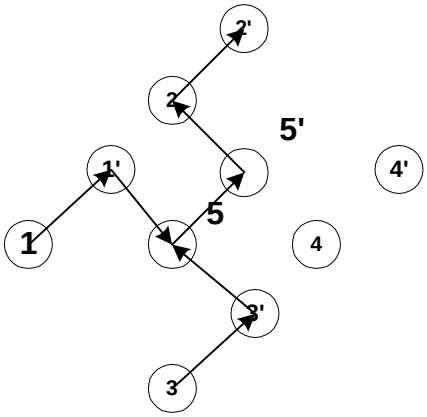
Imame po vieną lanką ir jį keičiame trimis lankais kaip aprašyta aukščiau. Pradėkime nuo lanko vedančio iš pirmojo mazgo į penktąjį:



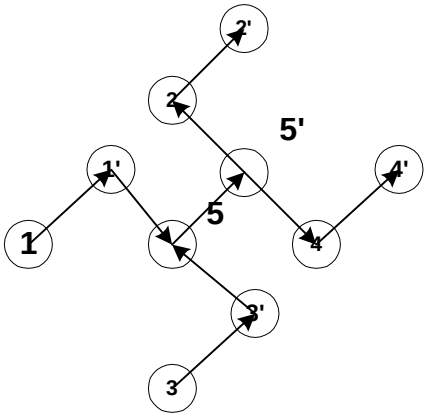
Einame prie antro lanko vedančio iš penktojo mazgo į antrąjį:



Iš trečiojo mazgo eina lankas į penktąjį:



Na ir paskutinysis lankas vedantis iš penktojo į ketvirtąjį:



Matome, kad iš penktojo mazgo viršutiniame tinklelyje į penktąjį mazgą apatiniame tinklelyje veda tik vienas lankas. Taigi, šioje tinklelio dalyje nebegalima nutiesti dviejų grandinių einančių per penktąjį mazgą.

Belieka pritaikyti algoritmą, randantį maksimalų srautą grafe. Deja, iš karto kyla keblumų su atmintimi. Jei tinklelyje yra $n \times n$ mazgų, tai grafe bus $2n^2+2$ viršūnės. Kadangi maksimali n reikšmė lygi 15, tai daugiausia grafe gali būti $2 \times 15^2+2 = 452$ viršūnės. Tai reiškia, kad negalėsime grafo vaizduoti įprastu būdu.

```
const MAX_N = 15;
      MAX_M = MAX_N*MAX_N*2 + 2; { maksimalus viršūnių skaičius grafe }
type grafas1 = array [1..MAX_N, 1..MAX_N] of integer;
var pral1: grafas;
```

Tokiam vaizdavimo būdui prireiktų $452 \times 452 \times 2 \approx 409$K atminties, tuo tarpu kai Turbo Paskalis duomenims leidžia naudoti tik 64K. Pasinaudosime uždavinio specifika. Bet kurios briaunos pralaidumas gali būti lygus arba 0 arba 1, o bet kuria briauna tekantis srautas: -1, 0, arba 1.

Kadangi briaunų pralaidumai gali įgyti tik dvi reikšmes, jiems saugoti galime aprašyti tokią duomenų struktūrą:

```
type  aibė = array [1..2] of set of 0..255;
      grafas2 = array [1..MAX_M] of aibė;
var pral2 = grafas2;
```

Maksimalaus srauto algoritmą galite rasti Lietuvos moksleivių informatikos olimpiadų internetiniame puslapyje prie aštuntosios nacionalinės informatikos olimpiados uždavinių sprendimų.

LIPDUKAI Sprendžiant šį uždavinį reikia mokėti modeliuoti veiksmus su dideliais skaičiais. Plačiau apie tai galite pasiskaityti 30–ame „Informatikos“ numeryje, J. Bulotaitės ir G. Grigo straipsnyje „Dideli skaičiai“.

Pirmas į galvą šovęs sprendimas – taikyti dalijimo pusiau metodą. T.y. pasirinkti mažiausią modelių skaičių *a*, kuriam lipdukų užteks ir pakankamai didelį modelių skaičių *b*, kuriam tikrai neužteks lipdukų. Turimą intervalą dalinti pusiau ir tikrinti, ar užteks lipdukų sunumeruoti $(a+b)/2$ modeliams. Jei užteks imti dešiniąją intervalo pusę, jei užteks – kairiąją. Veiksmus kartoti kol intervalas sumažės iki vieno skaičiaus.

Deja, šis sprendimas yra visiškai neteisingas. Galime surasti tokį skaičių *x*, kad užteks lipdukų sunumeruoti *x* bei *x*+2 modeliams, tačiau negalėsime sunumeruoti *x*+1 modelio.

Šio uždavinio autoriai pasiūlė tokį sprendimą.

Imkime pirmuosius 9 modelius. Jei galime juos sunumeruoti, tikriname, ar galime sunumeruoti ir visus modelius nuo 10 iki 99 naudodami tik *pirmųjų devynių modelių lipdukus*. Jei galime, tuomet tikriname, ar galime sunumeruoti ir visus modelius nuo 100 iki 999 *naudodami tik pirmųjų 99 modelių lipdukus*. Jei galime vėl tikriname, ar galime sunumeruoti visus modelius nuo 1000 iki 9999 *naudodami tik pirmųjų* 999 *modelių lipdukus* ir t.t.

Sakykime, negalime sunumeruoti modelių nuo 1000 iki 9999. Tuomet tikriname, ar galime sunumeruoti modelius nuo 1000 iki 1999, *naudodami tik pirmųjų 999 modelių lipdukus*. Jei negalime, tuomet tikriname, ar galime sunumeruoti modelius nuo 1000 iki 1099, jei galime, tada bandome numeruoti modelius nuo 2000 iki 2999 ir t.t.

Jei kažkurio modelio sunumeruoti nebegalime naudodami tik anksčiau atplėštas dėžutes, teks paimti ir to modelio dėžutės lipdukus.

Veiksmus kartojame kol nepavyksta sunumeruoti kažkurio modelio naudojant net ir jo paties dėžutėje esančius lipdukus.

Liko neatsakytas klausimas – kaip apskaičiuoti reikalingų bei turimų lipdukų skaičių. Turimų lipdukų skaičių rasti nesunku. Sakykime, kiekvienoje dėžutėje yra k_i lipdukų su skaitmeniu *i*. Naudosime lipdukus iš *n* dėžučių. Akivaizdu, kad su skaitmeniu *i* turėsime $n \times k_i$ lipdukų.

Kiek sunkiau apskaičiuoti lipdukų, reikalingų sunumeruoti *n* modelių, skaičių. Paimkime konkretų pavyzdį. Suskaičiuokime kiek lipdukų reikia sunumeruoti pirmiesiems 999 modeliams. Pradžioje laikykime, kad modeliai numeruojami nuo 000 iki 999. Iš viso bus 1000 modelių po 3 lipdukus. Viso – 40000 lipdukų su skaitmenimis. Visų skaitmenų bus po lygiai. Taigi, kiekvienas skaitmuo nuo 1 iki 9 bus panaudotas 300 kartų. Kiek kebliau su nuliu. Mat sąlygoje nurodyta, kad skaičiaus priekyje nuliai nerašomi. Triženklio skaičiaus priekyje nulis bus užrašytas 100 kartų (nuo 000 iki 099). Atmetus šį pirmąjį nulį dviženklių skaičių priekyje nulis bus užrašytas 10 kartų (nuo 00

iki 09). Atmetus ir šį nulį vienženklių skaičių pradžioje nulis bus užrašytas 1 kartą (skaičiuje 0). Viso reikia atmesti 100 +10+1 = 111 nulių.

Gavome, kad norint sunumeruoti 999 modelius reikės po 300 lipdukų kiekvieno skaitmens nuo 1 iki 9 ir 189 (300–111=189) lipdukų su skaičiumi 0. Analogiškai skaičiuojame reikalingų lipdukų skaičių ir kitais atvejais.

Remiantis šiais samprotavimais jau nesunku parašyti algoritmą, kuris apskaičiuotų reikalingų lipdukų skaičių.

REIŠKINIAI SU DALYBA Tai pats lengviausias olimpiados uždavinys. Kai kurie dalyviai sakėsi ilgai sėdėję prie šio uždavinio ieškodami paslėptų kabliukų, netikėdami, kad gali būti toks paprastas uždavinys...

Duotąjį reiškinį perrašome tokiu pavidalu:

$$\frac{x_1}{x_2 \quad x_3 \quad \cdots \quad x_k}$$

Kaip besudėtume skliaustus, x_1 visada bus trupmenos skaitiklyje, x_2 – visada vardiklyje. Visi likę nariai gali būti ir skaitiklyje ir vardiklyje. Juos perkeliame į skaitiklį:

$$x_1/(x_2/x_3/\cdots/x_k) = \frac{x_1 \quad x_3 \quad \cdots \quad x_k}{x_2}$$

Belieka patikrinti, ar skaitiklyje esančius skaičius galima suprastinti iš vardiklio. Tam nebūtina įsiminti visų reiškinį sudarančių skaičių. Perskaitomas eilinis reiškinio narys. Randamas šio nario bei nesuprastintos vardiklio dalies bendras didžiausias daliklis ir likęs vardiklis padalijamas iš šio daliklio. Toliau perskaitomas dar vienas reiškinio narys ir veiksmai kartojami kol vardiklis tampa lygus 1 arba kol perskaitomi visi reiškinį sudarantys skaičiai.

Ieškant bendrojo didžiausio daliklio svarbu vartoti dalybą (kaip tai daroma Euklido algoritme), o ne atimtį. Pastarasis didžiausio daliklio paieškos algoritmas yra žymiai lėtesnis.

Pavyzdžiui, duotas toks reiškinys:

$$4/64/5/24/3/2/18/28/19/15 = 4/(64/5/24/3/2/18/28/19/15) = \frac{4\times5\times24\times3\times2\times18\times28\times19}{64}$$

Imame pirmąjį bei antrąjį reiškinio narius (4 ir 64). Pirmasis narys bus skaitiklyje, antrasis – vardiklyje. Reikia suprastinti trupmeną $\frac{4}{64}$. Bendrasis didžiausias 4 ir 64 dalikis yra 4. Vardiklis

sumažėja iki 16: $\frac{4}{64}=\frac{1}{16}$. Imame trečiąjį narį (5). Trupmenos $\frac{5}{16}$ suprastinti negalime. Imame ketvirtąjį narį (24). Šį kartą vardiklis sumažėja iki 2: $\frac{24}{16}=\frac{3}{2}$. Imame penktąjį narį (3). Trupmenos $\frac{3}{2}$ suprastinti negalime. Paėmus šeštąjį narį (2) trupmenos vardiklis sumažėja iki 1. ***Atsakymas*:** Reiškinį **galima** taip pertvarkyti, kad gautume sveikąjį skaičių.

MUTEKSAI Nesusidūrusiems su lygiagrečiu programavimu gali būti sunkoka suprasti kaip muteksais sinchronizuojamas konvejerių veikimas. Panagrinėkime paprasčiausią atvejį. Skirtingi konvejeriai gali naudoti tuos pačius kintamuosius bei keisti jų reikšmes. Vienu metu to paties kintamojo reikšmę gali keisti tik vienas konvejeris. Jei tą galėtų daryti keli konvejeriai vienu metu, operacijos rezultatas būtų nenuspėjamas. Taip pat jei vienas konvejeris keičia kintamojo reikšmę, kiti konvejeriai tuo metu negali tos reikšmės skaityti.

Pavyzdžiui, reikia perskaityti du skaičius ir rasti jų sumą. Tai gali padaryti du konvejeriai. Prieš pradedant konvejeriams vykdyti savo komandas, kintamojo *suma* reikšmė prilyginama nuliui: *suma* := 0. Muteksai X, Y, ir SUMA kontroliuos kintamųjų *x, y,* ir *suma* panaudojimą.

1 konvejeris	*2 konvejeris*
`LOCK X`	`LOCK Y`
`read (x)`	`read (y)`
`LOCK SUMA`	`LOCK SUMA`
`suma := suma + x`	`suma := suma + y`
`UNLOCK SUMA`	`UNLOCK SUMA`
`UNLOCK X`	`UNLOCK y`

Grįžkime prie paties uždavinio. Jo autoriai siūlo vartoti grįžimo metodą. T.y. generuoti visus galimus variantus kol bus rasta amžino užrakinimo situacija arba perrinkti visi atvejai. Kiekviename rekursijos žingsnyje reiktų žinoti kiekvieno mutekso būseną (*užrakintas, atrakintas*) bei kuri kiekvieno konvejerio komanda yra vykdoma.

Daugiausia gali būti 5 konvejeriai, kurių kiekvieną sudaro ne daugiau 10 komandų. Tad gali būti $11^5 \approx 161000$ skirtingų programos būsenų (nepamirškite situacijos, kai konvejeris baigė darbą).

Rekursiškai perrenkant bus žymiai daugiau variantų, nes programos būsenos kartosis. Todėl būtų efektyviau įsiminti tas būsenas, kurios jau buvo sutiktos ir antrą kartą sutikus tą pačią būseną

jos nebenagrinėti. Tą pačią programos būseną visuomet atitiks tos pačios muteksų būsenos, net jei ta programos būsena būtų gauta skirtingais keliais. Akivaizdu, kad visų programos būsenų į masyvą lengvai sutalpinti nepavyks. Tai užimtų $161000 \times 5 = 805000$baitų $\approx$ 790K, kai Turbo Paskalis leidžia tik 64K.

Tačiau būsenas galime sunumeruoti iš eilės savo pasirinkta tvarka. Tuomet nebereikės įsiminti pačių būsenų, o tik apie kiekvieną būseną pasižymėti, ar ji jau išnagrinėta, ar ne. Tam nusakyti užteks vieno bito. Reikalingas atminties kiekis sumažėja iki $161000/8 \approx 20000$baitų $\approx$ 20K.

Žinant programos būseną bei visų muteksų būsenas nesunku nustatyti, ar tai nėra amžinojo užrakinimo situacija: tereikia patikrinti, ar tikrai negalime įvykdyti nė vienos komandos.