17 MOKSLEIVIŲ INFORMATIKOS OLIMPIADA

3 ETAPO 2 DALIS

2006 m. balandžio 5–8 d.

Švenčionys

Sprendimai

Kontrolinis darbas (VII-IX klasėms)

1 **Užduotis**. *Kiek yra n-ženklių skaičių (be nulių priekyje), kurių skaitmenų sandauga yra nelyginis skaičius?*

Skaitmenų sandauga yra nelyginis skaičius tada ir tik tada, kai visi skaičiaus skiemenys yra nelyginiai, t.y. kiekvienas skaitmuo gali būti renkamas iš aibės { 1,3,5,7,9 }. Taigi, kiekvienas iš *n* skaitmenų gali būti parinktas 5 išvardintais variantais, tad ieškomų skaičių kiekis lygus 5^n.

Pavyzdžiui, jei *n*=4, tai tokių skaičių gali būti 5^4=625, jei *n*=10, tai ieškomų skaičių gali būti 5^{10}=9765625.

2 **Užduotis.** Kokiu didžiausiu laipsniu galima pakelti skaičių 10, kad gautume skaičių, kuris neviršija skaičiaus, gauto išsprendus pirmą užduotį?

Ieškomas atsakymas gaunamas iš pirmoje dalyje gauto skaičiaus skaitmenų skaičiaus atėmus vienetą. Pavyzdžiui, jei gautas atsakymas yra 625, tuomet šios dalies atsakymas lygus 2: 10^2=100≤625 ir 10^3>625. Jei gautas atsakymas lygus 9765625, tuomet šios dalies atsakymas lygus 6: 10^6=1 000 000≤9 765 625 ir 10^7=10 000 000>9 765 625.

3 Užduotis. Kokio didžiausio skaičiaus kvadratas neviršija skaičiaus, gauto išsprendus pirmą užduotį?

Kadangi pirmos užduoties sprendinys lygus 5^n, tai ieškomasis skaičius lygus $\sqrt{5^n} = 5^{\frac{n}{2}}$. Kadangi n – lyginis skaičius, tai 5 reikės kelti sveikuoju laipsniu.

4 Užduotis. Užrašykite pirmoje užduotyje gautą skaičių dvejetainėje sistemoje.

Skaičių verčiame į dvejetainę sistemą, t. y. daliname iš dviejų įsimenant liekanas ir jas užrašant atvirkščia tvarka:

Pavyzdžiui:

625 mod 2 = 1	625 div 2 = 312
312 mod 2 = 0	312 div 2 = 156
156 mod 2 = 0	156 div 2 = 78
78 mod 2 = 0	78 div 2 = 39
39 mod 2 = 1	39 div 2 = 19
19 mod 2 = 1	19 div 2 = 9
9 mod 2 = 1	9 div 2 = 4
4 mod 2 = 0	4 div 2 = 2
2 mod 2 = 0	2 div 2 = 1 (pirmasis skaitmuo)

Gavome: $625_{10}=1001110001_2$

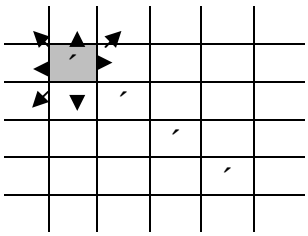
5 Užduotis. Kokį skaičių gautume, jeigu iš dvejetainio skaičiaus, gauto išsprendus 4-tą užduotį, pašalintume visus nulius ir gautą dvejetainį skaičių užrašytume dešimtaine sistema?

Pasinaudokime tapatybe : $\underbrace{1111\ldots11}_{k-vienetų}{}_2 = 2^k - 1$

Pavyzdžiui, pašalinę vienetus iš 1001110001_2 gauname $11111_2=2^5-1=32-1=31$

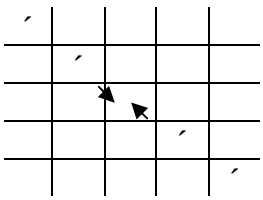
Kryžiukai – nuliukai (VII-IX klasėms)

Uždavinys yra paprastas: pagal padarytų ėjimų skaičių galime nuspręsti, kurio žaidėjo ėjimas (jei padarytas lyginis ėjimų skaičius – turės eiti žaidžiantysis nuliukais, jei nelyginis – kryžiukais). Tuomet galima išnagrinėti visus variantus, kadangi jų skaičius nėra didelis.



Pagal žaidimo taisykles, žaidėjo tikslas sudėti bent 5 simbolių ilgio grandinę. Kadangi galime atlikti tik vieną ėjimą, tai prasmingiausia ieškoti jau egzistuojančių pradėtų grandinių ir bandyti jas tęsti arba sujungti dvi.

Bendras uždavinio sprendimo algoritmas galėtų atrodyti taip:



1. Paeiliui nagrinėjame visus lentoje esančius savus ženklus (tuos, kuriais žaidžiame);
2. Aplink kiekvieną ženklą visomis kryptimis bandome atlikti ėjimą ir tikriname, ar laimime.

Sprendžiant uždavinį pateiktuoju algoritmu reikia saugotis ir neįtraukti tarp rezultatų to paties ėjimo kelis kartus (žiūrėti antrą pavyzdį).

Kadangi ėjimų koordinatės kinta labai plačiuose rėžiuose, tai žaidimo situacijos nepavyks išsaugoti dvimačiame loginiame masyve, nes pritrūks atminties. Problemą galime išspręsti naudodami paprasčiausią vienmatį ėjimų koordinačių masyvą. Nors tokią struktūrą nėra patogu sparčiai analizuoti (norint sužinoti, ar langelyje yra padėtas koks nors ženklas, reikia patikrinti visus masyvo elementus), greičio problemų nekils, nes ėjimų gali būti daugiausiai 100.

Šuo Deilas ir avys (VIII-IX klasės)

Uždavinį galima bandyti spręsti tiesiogiai realizuojant algoritmą, aprašytą sąlygoje, tačiau tam reikėtų kiekviename žingsnyje laikyti visą eilutę atmintyje. Kadangi atminties turime tik 16MB, eilutė, ilgesnė negu 2^{22} skaičių (~4 mln. įrašų po 4 baitus), nebetilps, o sąlyga numato iki 2^{30} avių. Galima bandyti sugalvoti gudresnį algoritmą, taupantį vietą, tačiau daug paprasčiau uždavinį išspręsti pasinaudojus uždavinyje aprašyto skaičiavimo algoritmo simetriškumo savybe:

Jei *m*-oji avis gavo numerį *n*, tada *n*-oji avis gavo numerį *m*.

Iš šios savybės išplaukia, kad norint sužinoti, kokį numerį gavo *n*-oji avis, užtenka išsiaiškinti, kokia numerį *n* gavusios avies vieta. Tai padaryti daug paprasčiau, nes nebereikia laikyti atmintyje visos eilutės — kiekviename žingsnyje užtenka suskaičiuoti tik numerį *n* gavusios avies poziciją, kuri priklauso nuo vienintelio skaičiaus — avies, turėjusios numerį *n* praeitame žingsnyje, pozicijos.

Pavyzdyje matyti, kad visuose žingsniuose avies 7 numeriu pozicija sutampa su 7-osios avies numeriu. Pirmoje eilutėje 7-a avis turi 2 numerį, 2-a avis turi 7 numerį, antroje eilutėje 7-a avis turi 3 numerį, o 3-a avis turi 7 numerį ir t.t.

Ieškomą skaičių randa toks algoritmas:

rasti_numerį (*m, k*):

```
kairė = 1
dešinė = 2^k
pozicija = m
while kairė ≠ dešinė do:
        pozicija = dešinė − (pozicija − kairė)
        riba = (kairė + dešinė − 1) / 2
        if pozicija ≤ riba:
                then dešinė = riba
                else   kairė = riba + 1
return pozicija
```

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

8	*7*	6	5	4	3	2	1
5	6	*7*	8	1	2	3	4
6	5	8	*7*	2	1	4	3
6	5	8	*7*	2	1	4	3

Kiekvienoje algoritmo iteracijoje skirtumas *dešinė-kairė* sumažėja perpus, todėl algoritmo sudėtingumas logaritminis — $O(log_2 n)$.

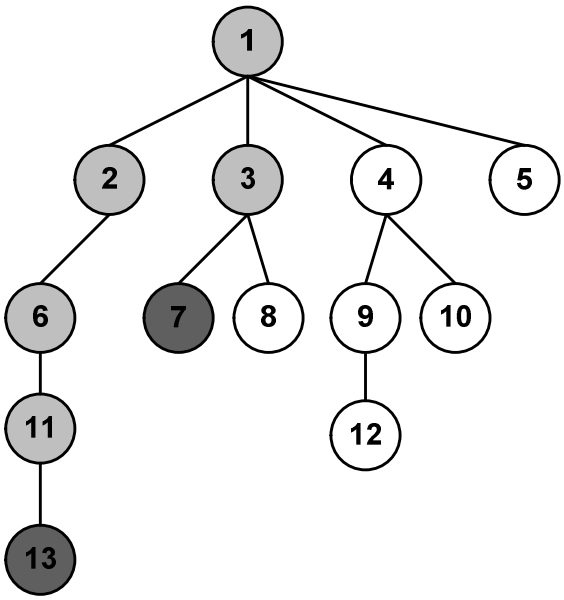
Burbuliukų girlianda (VII-XII klasėms)

1 Didžiausio girliandos aukščio radimas.

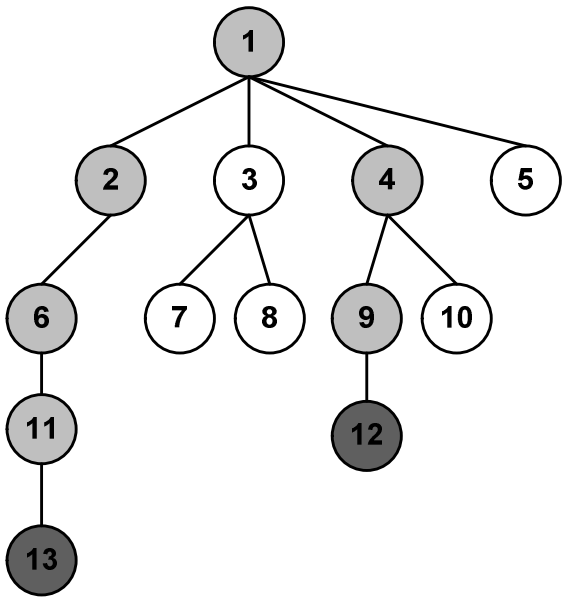
Uždavinį suvedame į ilgiausio kelio paiešką medyje:

- Pasirenkam bet kurį medžio lapą (lapas – medžio viršūnė, turinti tik vieną kaimyną) (L1).
- Surandame ilgiausią kelia iš lapo L1 (taikome bangos algoritmą). Lapą kuris yra toliausiai nutolęs nuo L1 pažymime L2.
- Surandame ilgiausią kelia iš lapo L2 (banga). Lapą kuris yra toliausiai nutolęs nuo L2 pažymime L3.
- Ilgiausias kelias medyje bus nuo L2 iki L3.

Žemiau pateikti paveikslai iliustruojantys šį



1 pav. Pasirinkome bet kurį lapą L1=7; Toliausiai nuo šio lapo nutolęs yra lapas L2=13



2 pav. Toliausiai nuo lapo L2=13 nutolęs lapas yra lapas L3=12; Didžiausias galimas girliandos aukštis lygus atstumui tarp L2 ir L3, t.y. 8.

algoritmą.

2 Girliandos kabinimas ant kito burbuliuko.

Šiai daliai nereikalingas joks specialus algoritmas ar metodas. Tiesiog į pirmą lygį įdedame duotąjį burbuliuką. Į antrą lygį dedame visus burbuliukus kurie yra sujungti su duotuoju burbuliuku (nepamirštame jų išrikiuoti numerių didėjimo tvarka). Pildant trečią lygį nereikia pamiršti, kad į jį nereikia dėti visų antro lygio burbuliukų kaimynų – vienas kaimynas jau yra pirmame lygyje. Tai taikoma ir tolimesniems lygiams.

Šiam uždaviniui patogu saugoti ne visus žemesniame lygyje esančius burbuliukus, o tik vieną, patį kairiausią (turintį mažiausią numerį). Kiekvienas burbuliukas taip pat turėtų nuorodą į dešiniau savęs esantį kaimyną (nuorodas tarp burbuliukų galima realizuoti rodyklės arba masyvo pagalba). Juk perkabinant girliandą ant kito burbuliuko mums perrikiuoti viso vaikų sąrašo nereikia: reikia į jį tik įterpti naują burbuliuką (burbuliuką iš aukštesnio lygio) ir po to vieną burbuliuką išimti (burbuliuką naujai tapusį tėvu). Šios operacijos sąraše atliekamos labai greitai..

Epidemijos modeliavimas (VII-XII klasės)

Uždavinį modeliuosime grafais. Paukščiai atitiks neorientuoto grafo viršūnes, o nuolatiniai kontaktai – grafo briaunas. Grafas gali būti nejungus, t.y., jame gali egzistuoti keletas jungių komponenčių, kuriuos toliau sprendimo aprašyme vadinsime paukščių šeimomis. Reikia atkreipti dėmesį į tai, kad šeimą gali sudaryti tik vienas paukštis.

Didžiausio galimo užsikrėtusių epidemijos metu skaičiaus radimas (VII-XII klasės)

Užkrėtus nors vieną paukštį iš šeimos, nuo šio paukščio užsikrės visa šeima. Tad užsikrėtusių paukščių bus daugiausia, jei užsikrės iš šeimos tik po vieną paukštį ir iš kuo gausesnių šeimų.

Taigi, bus taikomas godusis algoritmas. Reikia rasti visų šeimų dydžius (t.y. kiek jos yra narių) ir surikiuoti juos nedidėjančia tvarka. Ieškomas skaičius bus lygus pirmųjų K sekos narių sumai.

Sąlygoje pateikto pavyzdžio atveju būtų 15 šeimų: 13 šeimų turi po vieną paukštį, 1 šeima turi 2 paukščius ir dar viena šeima – 5 paukščius. Išrikiavę gautume: 5 2 1 1 1 1 1 1 1 1 1 1 1.

Žinoma, kad užsikrėtė 3 paukščiai. Tad didžiausias galimų užsikrėtusių paukščių skaičius lygus: 5+2+1=8.

Liko klausimas kaip suskaičiuoti kiek ir kokio dydžio yra paukščių šeimų (t. y. reikia rasti kiek yra jungių grafo komponenčių ir kiek kiekviena komponentė turi viršūnių). Tam galima panaudoti paiešką į gylį.

Mažiausio galimo užsikrėtusių epidemijos metu skaičiaus radimas (X–XII klasės)

Tarkime, šeimų dydžiai sudaro tokią nemažėjančią seką: $s_1, s_2, \ldots, s_Z$, kur Z yra šeimų skaičius. Nesunku parašyti algoritmą, paremtą dinaminio programavimo principais, kuris randa visas įmanomas sumų, kurių dėmenys paimti iš aibės $\{s_1, s_2, \ldots, s_Z\}$, reikšmes. Mažiausia iš tų reikšmių, kurios skaičius neviršija K ir yra ieškomas mažiausias galimas užsikrėtusių paukščių skaičius.

Pavyzdys

Tarkime pradiniai duomenis yra tokie kaip žemiau pateiktoje lentelėje.

```
7 5
6
1 2
1 3
4 5
4 6
```

Šiuo atvejų yra trys šeimos, t.y. Z=3: {1, 2, 3}, {4, 5, 6} ir {7}. Paukščių skaičius šeimose yra *3, 3, 1.* Kadangi $K > Z$, todėl blogiausiu atveju užsikrės visi paukščiai. Visos galimos sumos iš skaičių *1,3,3* yra aibė *{ 1,3,4,6,7}*.

Matome, kad tarp tų reikšmių artimiausia K reikšmei (iš viršaus) yra 6. Taigi, geriausiu atveju gali užsikrėsti *6* paukščiai.

Kauliuko kelionė (X-XII klasės)

Paprastesnis uždavinys

Kadangi uždavinio sprendimui svarbūs keli principai, iš pradžių paaiškinsime vieną iš jų – paiešką platyn, išspręsdami paprastesnį uždavinį:

Žaidimo lenta – stačiakampis languotas popieriaus lapas, kurį sudaro balti ir juodi langeliai. Kauliuką iš vieno langelio galime perkelti į vieną iš gretimų keturių, paversdami jį ant vieno iš keturių šonų – tai vadinama ėjimu. Žaidimo tikslas – atliekant kuo mažiau ėjimų perkelti kauliuką iš langelio (x_a, y_a) į langelį (x_b, y_b). Kauliuką statyti galime tik baltuose langeliuose. Užduotis: suskaičiuoti, per kiek mažiausiai ėjimų pakanka kauliukui perkelti.

Uždaviniui spręsti naudosime paieškos platyn principą. Trumpiausius atstumus (reikalingų ėjimų pasiekti šiam langeliui skaičių) iki kiekvieno langelio saugosime dvimačiame masyve ATSTUMAS. Norėdami išvengti tikrinimo, ar keldami kauliuką „nepabėgsime“ iš masyvo, lentelę apibrėšime juodais langeliais.

Pradžioje atstumą iki kiekvieno balto langelio pažymime begalybe (programuojant - pakankamai dideliu skaičiumi), o iki juodų langelių – minus vienetais (arba bet kokiu kitu sutartiniu neigiamu skaičiumi).

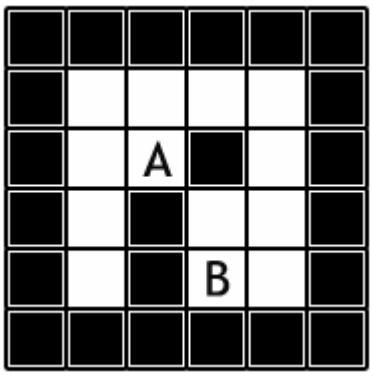
Toliau aprašomas algoritmas naudoja duomenų struktūrą eilę, palaikančią tris operacijas: įtraukti elementą į eilės galą, išimti elementą iš eilės pradžios ir patikrinti, ar eilė netuščia. Eilę paprasta realizuoti masyvu.

Mažiausią reikalingų ėjimų skaičių apskaičiuoja šis algoritmas:

Pažymime ATSTUMAS[x_a][y_a] = 0;
Langelio (x_a, y_a) koordinates įtraukiame į eilės galą;
Kol eilė netuščia, kartojame:
  Iš eilės pradžios paimame langelio koordinates (x, y);
  Nagrinėjame kiekvieną iš keturių gretimų langelių ($x`$, $y`$):
    Jei ATSTUMAS[$x`$][$y`$] > ATSTUMAS[x][y] + 1:
      Pažymime ATSTUMAS[$x`$][$y`$] = ATSTUMAS[x][y] + 1;
      Langelį ($x`$, $y`$) įtraukiame į eilės galą;
Atspausdiname ATSTUMAS[x_b][y_b];

Eilėje pirmiausia atsidurs pradinis langelis (x_a, y_a), tuomet visi langeliai pasiekiami iš pradinio langelio vienu ėjimu, tada visi pasiekiami dviem ėjimais ir t.t. Kiekvienas pasiekiamas langelis į eilę bus įtraukiamas lygiai vieną kartą. Taigi baigus vykdyti algoritmą, masyve ATSTUMAS bus surašyti trumpiausi atstumai iki kiekvieno pasiekiamo langelio.

Žemiau pateikti paveikslai iliustruoja žaidimo lentą bei masyvą ATSTUMAS prieš ir po algoritmo vykdymo.



Žaidimo lenta.

-1	-1	-1	-1	-1	-1
-1	∞	∞	∞	∞	-1
-1	∞	0	-1	∞	-1
-1	∞	-1	∞	∞	-1
-1	∞	-1	∞	∞	-1
-1	-1	-1	-1	-1	-1

Atstumų masyvas prieš vykdant algoritmą.

-1	-1	-1	-1	-1	-1
-1	2	1	2	3	-1
-1	1	0	-1	4	-1
-1	2	-1	6	5	-1
-1	3	-1	7	6	-1
-1	-1	-1	-1	-1	-1

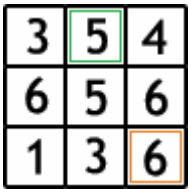
Atstumų masyvas įvykdžius algoritmą.

Kauliuko kelionė

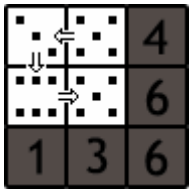
Uždavinio „Kauliuko kelionė“ sprendimas yra panašus į ką tik aptartąjį, tačiau svarbu atsižvelgti į kelis dalykus:

1) reikia laikytis pagrindinės žaidimo taisyklės – kauliukas visąlaik turi atvirsti tokiu akučių skaičiumi, koks yra įrašytas langelyje

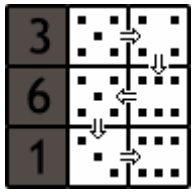
2) į tą patį langelį kauliuką galima atristi keliais skirtingais būdais (t.y. kauliukas gali atsidurti langelyje skirtingai pasuktas), o nuo to priklauso, kokius tolimesnius ėjimus galėsime atlikti. (žr. pav. apačioje)



Žaidimo lenta.



Pirmą ėjimą atlikę kairėn, pasiekiame vidurinį langelį, bet toliau negalime atlikti jokio ėjimo.



Pirmą ėjimą atlikę dešinėn, pasiekiame vidurinį langelį ir laimime žaidimą.

Taigi langelio koordinatės nevisiškai apibūdina kauliuko poziciją – reikia saugoti, kaip pasuktas kauliukas stovi langelyje. Vienas iš būdų (nors ne pats efektyviausias, bet paprastas) saugoti kauliuko pasisukimą – įsiminti, koks akučių skaičius yra priekyje (nepainioti su viršumi). Tuomet kauliuko poziciją vienareikšmiškai apibūdina trejetas (x, y, p), kur (x, y) yra kauliuko koordinatės lentoje, o p – akučių skaičius *priekyje* (skaičius nuo 1 iki 6).

Masyve ATSTUMAS saugosime trumpiausią atstumą iki kiekvienos galimos kauliuko *pozicijos* (taigi naudosime trimatį masyvą).

Žinodami kauliuko poziciją (x, y, p), galime vienareikšmiškai apskaičiuoti, kaip atsiverstų kauliukas, jei atliktume kokį nors ėjimą (viršun, apačion, kairėn arba dešinėn). Toliau pateikiamame algoritme šį darbą atlieka funkcija PARITUS[1]: jai perduodama dabartinė kauliuko pozicija (x, y, p) ir ėjimo numeris k (pavyzdžiui, skaičius nuo 1 iki 4), o grąžinama reikšmė - naujos koordinatės (x, y), akučių skaičius priekyje p, bei akučių skaičius viršuje v (šio skaičiaus mums reikės patikrinimui, ar bandomasis ėjimas neprieštarauja pagrindinei žaidimo taisyklei).

[1] Kaip paprastai realizuoti funkciją PARITUS, žiūrėkite uždavinį sprendžiančioje programoje.

Tuomet uždavinį sprendžia šis algoritmas:

(Suskaičiuojami trumpiausi atstumai iki kiekvienos pozicijos)

Visiems galimiems kauliuko pasukimams pradiniame langelyje p_a:
  Pažymime ATSTUMAS[x_a][y_a][p_a] = 0;
  Poziciją (x_a, y_a, p_a) įtraukiame į eilės galą;
  Kol eilė netuščia, kartojame:
    Iš eilės pradžios paimame poziciją (x, y, p);
    Kiekvienam iš keturių galimų ėjimų k:
      (x`, $y$`, p`, v) = PARITUS(x, y, p, k);
      Jei v = SKAIČIUS[x`][$y$`] (*t.y. ėjimas legalus*):
        Jei ATSTUMAS[x`][$y$`][p`] > ATSTUMAS[x][y][p] + 1:
          Pažymime ATSTUMAS[x`][$y$`][p`] = ATSTUMAS[x][y][p] + 1;
          Langelį (x`, $y$`, p`) įtraukiame į eilės galą;

(Apskaičiavus trumpiausius atstumus iki kiekvienos pozicijos,
reikia išrinkti artimiausią poziciją, kurios koordinatės (x_b, y_b))

  Pažymime ATSAKYMAS = ∞;
  Visiems galimiems kauliuko pasukimams galiniame langelyje p_b:
    Jei ATSAKYMAS > ATSTUMAS[x_b][y_b][p_b]:
      Pažymime ATSAKYMAS = ATSTUMAS[x_b][y_b][p_b];
Atspausdiname ATSAKYMAS;

Efektyvumas

Su vienu eilės elementu atliekamų veiksmų skaičius yra O(1) (konstantinis, nepriklausantis nuo pradinių duomenų dydžio). Kadangi kiekvienas lentos langelis į eilę įtraukiamas tikrai ne daugiau negu keturis kartus, o langelių yra $M{\cdot}N$, tai algoritmo sudėtingumas yra O($M{\cdot}N$), visiškai pakankamas uždaviniui išspręsti su visais galimais pradiniais duomenimis.

Karo akademija (X-XII klasėms)

Jeigu Karo akademijos uždavinio pradinius duomenis – žvaigždžių ir joms priskirtų pavadinimų porų aibę – surašysime į N×N dydžio lentelę, kurios i-ojoje eilutėje ir j-ajame stulpelyje įrašysime, kiek kartų pradiniuose duomenyse kartojasi pora (*i*, *j*), iš pavyzdyje pateiktų duomenų gausime lapo viršuje dešinėje pavaizduotą lentelę.

0	2	1
1	1	0
1	2	0

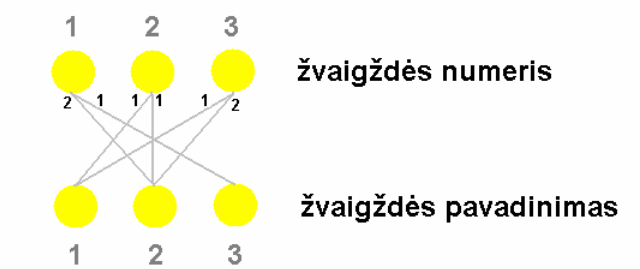
Uždavinį performuluojame taip: lentelės kiekvienoje eilutėje ir kiekviename stulpelyje reikia išrinkti lygiai po vieną langelį taip, kad išrinktuose langeliuose įrašytų skaičių suma būtų maksimali. Tai yra *tiesinis priskyrimo uždavinys.*

Klasikinis *optimalaus priskyrimo uždavinys* skamba maždaug taip: N darbininkų reikia atlikti N užduočių, duota N×N dydžio lentelė, kurios kiekvienoje ląstelėje įrašyta, kiek „kainuos“ jei *m*-ąją užduotį atliks *n*-asis darbininkas (darbininkų įkainiai skiriasi).

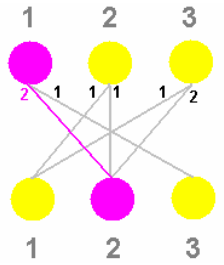
Pateiksime du šio uždavinio sprendimo būdus: maksimalaus suporavimo ir Vengriškąjį algoritmą.

Maksimalus suporavimas svoriniame dvidaliame grafe

Galima suformuoti svorinį dvidalį grafą iš N+N viršūnių:

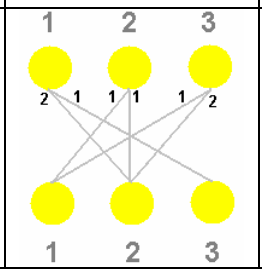
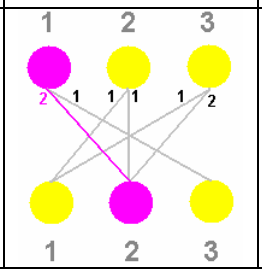
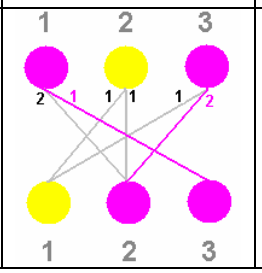
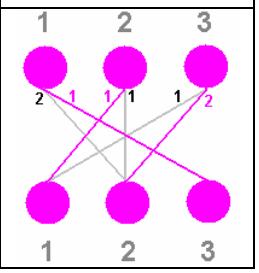


Šiame grafe reikia rasti maksimalaus svorio suporavimą. Pradžioje laikykime, kad nei vienam žvaigždės numeriui nėra priskirta nei vieno žvaigždės pavadinimo. Tarkime, kad pirmam žvaigždės numeriui priskyrėme antrą pavadinimą, šiame priskyrime dalyvaujančias grafo viršūnes ir briaunas pažymime:



Tolimesnei paieškai apibrėšime „didinančio“ kelio sąvoką. „Didinančiu“ keliu svoriniame dvidaliame grafe vadinsime kelią, kuris prasideda nuo vieno iš žvaigždžių pavadinimo, baigiasi ties bet kuria dar nepažymėta grafo viršūne, o nepažymėtų kelią sudarančių briaunų svorių suma yra didesnė už pažymėtų briaunų sumą. Be to, griežtai kas antra šio kelio briauna yra pažymėta, o likusios - nepažymėtos. Tokio kelio svoriu vadinsime jį sudarančių nepažymėtų ir pažymėtų briaunų svorių skirtumą. Pavyzdyje kelias 3-1-2-3 yra „didinantis“, jo svoris – 1. Maksimalaus svorio suporavimą galima rasti kartojant šiuos žingsnius:

- ieškome mažiausiai briaunų turinčio maksimalaus svorio „didinančio“ kelio;
- jeigu tokį randame, jį sudarančias nepažymėtas briaunas sukeičiame su pažymėtomis, priešingu atveju, radome maksimalaus svorio suporavimą.

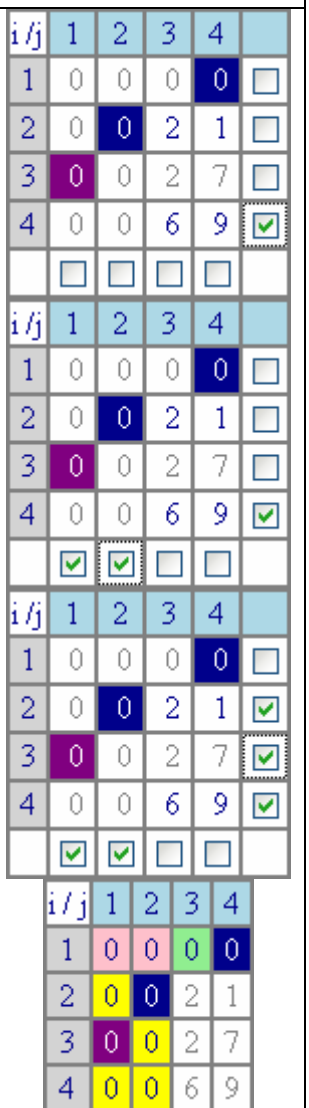
Didinantis kelias (svoris)	**-**	**2-1 (2)**	**3-1-2-3 (1)**	**1-2 (1)**
Dvidalis grafas				

Vengriškas algoritmas

Vengriškas algoritmas - tai optimalaus priskyrimo uždavinį sprendžiantis algoritmas, kurį, apibendrinęs dviejų vengrų matematikų - Denes König ir Jenő Egerváry – darbus, sukūrė ir 1955 metais publikavo Harold Kuhn.

Uždavinys taikant Vengriškąjį algoritmą sprendžiamas tokiu būdu. Tarkime, ieškome minimalios „kainos“:

Žingsnis	*Atliekami veiksmai*	*Pavyzdys*
0	Imama pradinė lentelė	
1	Kiekvienoje lentelės eilutėje surandame minimalią reikšmę ir ją atimame iš kiekvieno tos eilutės elemento. Taip kiekvienoje lentelės eilutėje gauname bent po vieną nulį.	
2	Lentelėje pažymime tuos nulius, kurie yra vieninteliai savo eilutėje arba stulpelyje. Pažymėdami nulį, kuris yra vienintelis savo stulpelyje, uždengiame lentelės eilutę, kuriai jis priklauso. O pažymėdami nulį, kuris yra vienintelis savo eilutėje, uždengiame lentelės stulpelį, kuriam jis priklauso. Pavyzdyje pirmoje eilutėje ir ketvirtame stulpelyje esantis nulis yra vienintelis savo stulpelyje, nes šiame stulpelyje nėra nei vieno neuždengto nulio. Pažymėję šį nulį uždengiame eilutę, kuriai jis priklauso. Daugiau žymėjimui tinkamų nulių lentelėje nėra. Toliau žymėdami ir tikrindami, ar nuliai yra vieninteliai savo eilutėje ar stulpelyje, uždengtus nulius (esančių uždengtoje eilutėje ar stulpelyje) ignoruojame. Pažymėdami nulį atliekame atitinkamos užduoties priskyrimą darbininkui. Mūsų tikslas - atlikti lygiai N priskyrimų. Pereiname į 3 žingsnį.	
3	Toliau nagrinėjame, kokioje būsenoje yra lentelė. Galimi trys variantai: Ø atlikome N priskyrimų, taigi baigiame darbą; Ø lentelėje yra neuždengtų nulių, pereiname į 4 žingsnį; Ø visi nuliai lentelėje yra uždengti, pereiname į 5 žingsnį.	

4	Pažymime bet kurį neuždengtą nulį. Kartu uždengiame jo eilutę ir stulpelį. Pereiname į 2 žingsnį.	
5	Priklausomai nuo to, ar vykdėme 4 žingsnį, pereiname į: Ø 5.1 žingsnį, jei vykdėme 4 žingsnį; Ø 5.2 žingsnį, jei nevykdėme 4 žingsnio.	
5.1	Ieškome minimalaus skaičiaus linijų (eilučių arba stulpelių), kuriomis galima uždengti visus nulius lentelėje, tokiu būdu: Ø pažymime eilutes, kuriose nėra nei vieno pažymėto nulio; Ø pažymime stulpelius, kurie turi nulių pažymėtose eilutėse; Ø pažymime eilutes, kuriose yra pažymėtų nulių iš pažymėtų stulpelių. Paskutinius du žingsnius kartojame tol, kol atliekame nors vieną pakeitimą žymėjimuose. Tada uždengiame pažymėtuosius lentelės stulpelius ir nepažymėtąsias lentelės eilutes. Šioje vietoje pamirštame, kad jau vykdėme 4 algoritmo žingsnį ir pereiname į 5.2 žingsnį.	
5.2	Surandame minimalią neuždengtą reikšmę lentelėje ir ją atimame iš visų neuždengtų lentelės reikšmių bei pridedame prie visų lentelės reikšmių, ties kuriomis kertasi eilutės ir stulpelius dengiančios linijos. Atidengiame visas lentelės eilutes ir stulpelius. Pereiname į 2 žingsnį.	

Miesto tvarkymas (X-XII klasės)

Pasiūlyti rikiavimai nusako santykį NEANKSČIAU tarp darbų porų.

NEANKSČIAU(B, A) reiškia, kad darbas B negali būti atliekamas ankstesnę dieną nei darbas A.

Iš pradinių duomenų žinome, kad jei darbas B kažkokiame pasiūlytame rikiavime paminėtas vėliau nei darbas A, tai NEANKSČIAU(B, A) yra tiesa. Tačiau šis santykis yra tranzityvus, t.y. jei NEANKSČIAU(C, B) ir NEANKSČIAU(B, A), tai NEANKSČIAU(C, A). Tuo remiantis turime apskaičiuoti santykį NEANKSČIAU tarp visų darbų porų.

Tai galima atlikti naudojant tranzityvaus uždarinio radimo algoritmą.

Algoritmo idėja yra tokia. Jei NEANKSČIAU(A_1, A_m) todėl, kad NEANKSČIAU(A_1, A_2), NEANKSČIAU(A_2, A_3), ..., NEANKSČIAU(A_{m-1}, A_m), tai sakysime, kad NEANKSČIAU(A_1, A_m) apskaičiuotas pasinaudojant tarpiniais darbais A_2 ,A_3,..., A_{m-1}.

Tegu $NEANKSČIAU_K$, bus nepilnas santykis NEANKSČIAU, apskaičiuotas tik toms darbų poroms, kurioms tai galima padaryti pasinaudojant tik tarpiniais darbais 1,2,...,K. Nesunku matyti, kad turėdami $NEANKSČIAU_K$ nesunkiai galime apskaičiuoti $NEANKSČIAU_{K+1}$:

$NEANKSČIAU_{K+1}$ (B,A) tada ir tiktai tada, kai $NEANKSČIAU_K$(B, K+1) ir $NEANKSČIAU_K$(K+1, A).

Taigi, iteruodami K nuo 1 iki N, apskaičiuosime santykį NEANKSČIAU tarp visų darbų porų. Pastebėsime, kad pradiniam santykio NEANKSČIAU įvertinimui (t.y. $NEANKSČIAU_0$) iš pageidaujamų rikiavimų užtenka paimti tik gretimų darbų poras – visos kitos priklausomybės bus apskaičiuotos.

Jei NEANKSČIAU(B, A) ir NEANKSČIAU(A, B), tai A ir B privalo būti atliekami tą pačią dieną; sakysime, kad tokie darbai yra *lygūs*. Visi darbai gali būti suskirstyti į nesikertančias grupes, kai kiekvieną grupę sudaro visi tarpusavyje lygūs darbai. Tai galima atlikti taip: imame bet kurį darbą ir priskiriame į naują grupę šį darbą ir visus

darbus, lygius jam; tą patį kartojame darbams, dar nepriskirtiems jokiai grupei.

Bet kurioms dviems grupėms, visi vienos grupės darbai gali būti atliekami prieš visus kitos grupės darbus. Iš tikrųjų, jei visų grupės $\{A_1,\ldots,A_n\}$ darbų negalėtume atlikti prieš visus grupės $\{B_1,\ldots,B_m\}$ darbus arba atvirkščiai, tai reikštų, kad egzistuoja tokie *i,j,k,l*, kad NEANKSČIAU(A_i, B_j) ir NEANKSČIAU(B_k, A_l). Tačiau tada, įvertinus, kad NEANKSČIAU(B_j, B_k), NEANKSČIAU(B_k, A_l) ir NEANKSČIAU(A_l, A_i), gautume, kad NEANKSČIAU(B_j, A_i). Kadangi taip pat turime, kad NEANKSČIAU(A_i, B_j), tai reikštų, kad B_j ir A_i lygus ir turėtų priklausyti vienai grupei – gautume prieštarą.

Taigi tarp darbų grupių yra santykis GRUPĖ_NEANKSČIAU, nusakomas taip: dviem darbų grupėms G ir H, GRUPĖ_NEANKSČIAU(G,H) tada ir tiktai tada, kai kažkoks grupės G darbas negali eiti anksčiau nei kažkoks grupės H darbas. Santykis GRUPĖ_NEANKSČIAU yra tranzityvus bei antisimetrinis (t.y., jei GRUPĖ_NEANKSČIAU(G,H) ir GRUPĖ_NEANKSČIAU(H,G), tai G ir H yra ta pati grupė). Nesunku parodyti, kad dėl šių savybių santykis GRUPĖ_NEANKSČIAU neturi ciklų ir grupes galima išrikiuoti taip, kad šio santykio būtų laikomasi. (Įrodymą paliekame kaip savarankišką pratimą).

Darbų negalima atlikti per daugiau dienų nei yra grupių. Iš kitos pusės, grupes galima išrikiuoti taip, kad kiekvieną dieną būtų atliekam tik vienos grupės darbai. Taigi ieškomas dienų skaičius lygus grupių skaičiui. Pačias grupes paprasčiausia išrikiuoti pasinaudojant bet kuriuo pageidaujamu rikiavimu, kadangi grupių tvarka negali jam prieštarauti. Jei pageidaujamas darbų rikiavimas yra $r_1,r_2,\ldots,r_N$, tai pirma grupė turi būti grupė, kuriai priklauso darbas r_1; visus šios grupės darbus (t.y darbus lygius r_1) pašalinam iš rikiavimo. Tada, jei r_k yra pirmas likęs darbas, tai antroji grupė yra grupė, kuriai priklauso r_k. Visus šios grupės darbus taip pat pašalinam ir tesiam procesą, kol bus surikiuotos visos grupės.

Pateikto algoritmo efektyvumo įvertinimas. Duomenų įvedimas ir darbų porų įsiminimas reikalauja $O(N^2 + R\cdot N)$ veiksmų, tranzityvaus uždarinio radimas $O(N^3)$, grupių suformavimas ir išvedimas $O(N^2)$. Taigi bendras algoritmo atliekamų veiksmų sakičius yra $O(N^3 + R\cdot N)$.