

## Turinys

<b>1. Aibės .....</b>	<b>7</b>
1.1. Aibės sąvoka .....	7
1.2. Poaibiai .....	11
1.3. Poaibių generavimo algoritmai .....	14
1.4. Grėjaus kodų taikymo pavyzdžiai .....	19
1.5. Veiksmai su aibėmis (aibių algebra) .....	23
1.6. Aibės galia ir jos apskaičiavimas .....	31
<b>2. Grafų teorija .....</b>	<b>34</b>
2.1. Įvadas .....	34
2.2. Pradinės sąvokos .....	35
2.3. Grafo jungiosios komponentės .....	43
2.4. Metrinės grafo charakteristikos .....	46
2.5. Veiksmai su grafais .....	48
2.6. Grafų izomorfizmas .....	53
2.7. Grafo vaizdavimo kompiuteryje būdai .....	57
2.8. Grafo viršunių peržiūros metodai .....	65
2.8.1. Paieška gilyn .....	66
2.8.1.1. Paieška gilyn, naudojant rekursiją .....	68
2.8.1.2. Paieškos gilyn su mažiausia atminties apimtimi organizavimas .....	68
2.8.1.3. Paieška gilyn, nenaudojant rekursijos .....	71
2.8.2. Paieška platyn .....	73
2.9. Trumpiausių kelių besvoriniame grafe ieškojimo uždavinys .....	76
2.10. Dvidalis grafas .....	79
2.11. Pagrindiniai grafų teorijos skaičiai .....	82
2.11.1. Ciklomatinis skaičius .....	82
2.11.2. Chromatinis skaičius .....	89
2.11.3. Nepriklausomumo skaičius .....	100
2.11.4. Dominavimo skaičius .....	107
2.12. Plokštjieji grafai .....	112
2.13. Medžiai .....	117
2.13.1. Dengiančiojo medžio apskaičiavimo uždavinys .....	119
2.13.2. Trumpiausio dengiančiojo medžio svoriniame grafe apskaičiavimo uždavinys .....	126
2.14. Optimalių kelių ieškojimas .....	141
2.14.1. Trumpiausio kelio radimo uždavinys .....	141
2.14.2. Didžiausios keliamosios galios (plačiausios siauros vietos) apskaičiavimo uždavinys .....	144

2.14.3.	Uždavinys apie stiprinimą .....	146
2.14.4.	Ilgiausio kelio uždavinys .....	149
2.14.5.	Trumpiausių kelių tarp visų viršūnių porų apskaičiavimas ...	156
2.15.	Maršrutai .....	160
2.15.1.	Oilerio maršrutai.....	160
2.15.2.	Hamiltono maršrutai .....	165
2.16.	Jungumas.....	175
2.16.1.	Dviryšiai grafai .....	178
2.17.	Srautai tinkluose ir giminingsi uždaviniai .....	187
2.17.1.	Pagrindinės sąvokos .....	187
2.17.2.	Maksimalaus srauto konstravimo algoritmas.....	191
2.17.3.	Maksimalaus suporavimo uždavinys dvidaliame grafe .....	207
2.18.	Grafo viršūnių laipsnių sekos.....	217
2.18.1.	Grafinės sekos .....	217
2.18.2.	Grafinės sekos kriterijai .....	219
2.18.3.	Grafinės sekos realizacijos algoritmas.....	219
2.18.4.	Maksimaliai jungi grafinės sekos realizacija .....	221
2.18.5.	Hamiltono grafo konstravimas pagal grafinę seką .....	223
3.	<b>Kombinatorika</b> .....	225
3.1.	Įvadas .....	225
3.2.	Bendrieji kombinatorikos dësniai .....	225
3.3.	Junginiai .....	228
3.3.1.	Gretiniai be pasikartojimų .....	228
3.3.2.	Gretiniai su pasikartojimais .....	228
3.3.3.	Keliniai be pasikartojimų .....	229
3.3.4.	Keliniai su pasikartojančiais elementais .....	229
3.3.5.	Deriniai be pasikartojimų .....	230
3.3.6.	Deriniai su pasikartojimais .....	231
3.3.7.	Derinių savybės .....	232
3.4.	Kombinatorinių objektų generavimo algoritmai .....	236
3.4.1.	Derinių generavimo algoritmai .....	237
3.4.2.	Keliniių generavimo algoritmai .....	248
3.4.3.	Aibės išskaidymas .....	255
3.4.4.	Sveikujų skaičių kompozicija ir išskaidymas .....	263
3.5.	Rekurentiniai sąryšiai .....	268
3.5.1.	Rekurentinio sąryšio sąvoka ir pavyzdžiai .....	268
3.5.2.	Rekurentinių sąrysių sprendimas .....	275
3.6.	Generuojančios funkcijos .....	286
3.7.	Rekurentiniai sąryšiai ir generuojančios funkcijos .....	290

<b>4. Matematinė logika .....</b>	295
4.1. Įvadas .....	295
4.2. Teiginių logika .....	295
4.3. Įrodinėjimo metodai .....	301
4.4. Predikatų logika .....	304
4.4.1. Predikatai .....	304
4.4.2. Kvantoriai .....	305
4.4.3. Operacijos su predikatais .....	307
<b>5. Bulio algebra .....</b>	309
5.1. Bulio algebra kaip algebrinė sistema .....	309
5.2. Bulio funkcijos .....	311
5.2.1. BF atvaizdavimas teisingumo lentelėmis .....	312
5.2.2. BF atvaizdavimas diagramomis .....	313
5.2.3. Analitinis BF užrašymo būdas .....	320
5.2.4. Grafinis BF atvaizdavimo būdas .....	322
5.2.5. Matricinis BF atvaizdavimo būdas .....	322
5.3. Bulio funkcijų minimizavimas .....	324
Literatūra.....	330

# 1. Aibės

## 1.1. Aibės sąvoka

Aptarsime aibų teorijos, kuri matematikoje vadinama naiviaja aibų teorija, pradmenis. Šioje teorijoje aibės sąvoka – tai fundamentali neapibrėžiama sąvoka. Aibų teorijos kūrėjas vokiečių matematikas G.Kantorius (1845 – 1918) sakė, kad aibė – tai objektų, kuriems būdingas tam tikras požymis, visuma. Pavyzdžiu, lotynų abécélės raidžių aibė, krepšinio komandos žaidėjų aibė, studentų akademinių grupės aibė, euklidinės plokštumos taškų aibė ir pan. Lotynų abécélės aibę sudaro raidės, krepšinio komandą sudaro jos žaidėjai, studentų akademinę grupę sudaro studentai, euklidinę plokštumą sudaro taškai. Apskritai, objektais, sudarantys aibę, vadinami aibės elementais. Kitaip tariant, aibė – tai elementų “maišelis”. Reikia pabrėžti, kad aibėje negali būti vienodų elementų, t.y. visi aibės elementai yra skirtiniai.

Aibės paprastai žymimos lotynų raidyno didžiosiomis raidėmis, pavyzdžiu,  $A$ ,  $B$ ,  $C$  ir pan. Matematikoje dažniausiai naudojamų aibų simbolika yra nusistovėjusi. Pavyzdžiu:

- $N$  – natūraliųjų skaičių aibė,
- $Z$  – sveikujų skaičių aibė,
- $Q$  – racionaliųjų skaičių aibė,
- $R$  – realiųjų skaičių aibė,
- $C$  – kompleksinių skaičių aibė,
- $\emptyset$  – tuščioji aibė.

### *Aibės nusakymo būdai*

Paprastai aibė nusakoma tokiais būdais:

- *išvardinimo*,
- *aprašymo*,
- *grafiniu*.

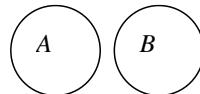
*Išvardinimo* būdu aibė aprašoma išvardinant visus jos elementus. Pavyzdžiu,  $A = \{a_1, a_2, a_3\}$ ,  $B = \{1, 3, 5, 7, \dots\}$ ,  $X = \{x_1, x_2, \dots, x_n\}$ . Daugtaškis skaitomas “ir taip toliau” arba “ir taip toliau iki”. Aibės elementai rašomi riestiniuose skliaustuose ir skiriami vienas nuo kito kableliais. Išvardinimo būdas yra naudojamas, kai aibės elementų skaičius yra nedidelis arba aiškus elementų dėsningumas.

Nusakant aibę ***aprašomuoju*** būdu, nurodomas predikatas<sup>1</sup>, ir aibę sudaro tie elementai, kuriems šis predikatas yra teisingas:  $A = \{a | p(a)\}$  arba  $X = \{x \in M | p(x)\}$ , čia aibę  $A$  sudaro elementai  $a$ , kuriems predikatas  $p(a)$  yra teisingas, o aibę  $X$  sudaro tie aibės  $M$  elementai, kuriems predikatas  $p(x)$  yra teisingas. Pavyzdžiui,

$$\begin{aligned} A &= \{a \in N | 6 \leq a \leq 10\}, \\ B &= \{x \in R | x^2 - 5x + 6 = 0\}, \\ C &= \{x | x \text{ yra IF-0/1 grupės pirmūnė}\}, \\ P &= \{p | p \text{ yra pirminis skaičius}\}. \end{aligned}$$

Aišku, kad aibę  $A$  sudarys natūralieji skaičiai 6, 7, 8, 9, 10; aibę  $B$  sudarys skaičiai 2 ir 3; aibę  $C$  sudarys IF-0/1 grupės pirmūnai, o aibę  $P$  sudarys natūralieji pirminiai skaičiai.

Vaizduojant aibę ***grafiniu*** būdu, naudojamos Veno diagramos. Jos paprastai naudojamos vaizdžiai parodyti santykius bei veiksmus su aibėmis. Veno diagrama – tai aibė plokštumos taškų, aprabotų uždarąja kreive (paprastai apskritimu). Pavyzdžiui,



### Simbolika

Aibė, neturinti nei vieno elemento, vadinama *tuščiaja aibe* ir, kaip buvo minėta aukščiau, žymima simboliu  $\emptyset$ .

Simboliu  $x \in A$  žymime, kad  $x$  yra aibės  $A$  elementas.

Simboliu  $x \notin A$  žymime, kad  $x$  nėra aibės  $A$  elementas.

Aibės  $A$  elementų skaičių žymime  $|A|$  arba  $n(A)$ .

Aibės elementų skaičius kitaip vadinamas *aibės galia*. Jei aibės elementų skaičius yra baigtinis, tai aibė vadinama baigtine, priešingu atveju – begaline. Pavyzdžiui, aibė  $P = \{x | 1 \leq x \leq 100 \text{ ir } p(x) = x \text{ yra pirminis skaičius}\}$  yra baigtinė aibė, o aibės  $N, Z, R, A = \{x \in N | p(x) = x \text{ yra lyginis skaičius}\}$  yra begalinės aibės.

---

<sup>1</sup> Predikatas – tai funkcija, kurios apibrėžimo sritis yra aibė  $M$  ir kuri kiekvienam aibės  $M$  elementui priskiria reikšmę “tiesa” arba “melas”. Kitaip tariant, tai funkcija, atvaizduojanti aibę  $M$  į aibę {tiesa, melas} ( $\{\text{true, false}\}$ ,  $\{1, 0\}$ ).

Kaip buvo minėta aukščiau, aibė negali turėti vienodų elementų. Užrašas  $\{a_1, a_2, \dots, a_{i-1}, a_i, a_i, a_{i+1}, \dots, a_n\}$  yra nekorekтиškas ir jį reikia pakeisti taip:  $\{a_1, a_2, \dots, a_{i-1}, a_i, a_{i+1}, \dots, a_n\}$ .

### **Santykiai**

Aibės  $A$  ir  $B$  yra lygios ir žymimos  $A = B$ , jei jos sudarytos iš tų pačių elementų. Priešingu atveju  $A \neq B$ .

Griežtai matematiškai aibų lygybę galime užrašyti taip:

$$A = B \leftrightarrow \forall x(x \in A \leftrightarrow x \in B),$$

čia simbolis “ $\leftrightarrow$ ” reiškia ekvivalenciją (“tada ir tik tada, kai”).

Jei tarp dviejų aibų  $A$  ir  $B$  nustatyta abipusiai vienareikšmė atitiktis – atvaizdis<sup>2</sup>, tai sakome, kad aibės  $A$  ir  $B$  yra ekvivalentūs. Žymime  $A \sim B$ .

Aibė  $A$  vadinama *skaičiaga*, kai  $A \sim N$ . Kitaip tariant,  $A$  yra skaičioji aibė, kai visus jos elementus galima sunumeruoti visais natūraliaisiais skaičiais taip, kad skirtinti elementai gautų skirtinges numerius. Aišku, kad skaičiosios aibės galia yra lygi natūraliųjų skaičių aibės galiai. Šios aibės yra begalinės.

Didesnę galiau nei natūraliųjų skaičių aibės galia turi aibės, ekvivalentūs atkarpos  $[0, 1]$  taškų aibei. Ši aibė yra neskaičioji ir jos galia vadinama *kontinuumu*.

### **Raselo paradoksas**

Kadangi aibės sąvoka yra neapibrėžiama, tai reikia vengti tokių aibų, kurių elementai nėra visiškai aiškūs. Pavyzdžiu, vargu ar protinė kalbėti apie idėjų aibę, apie vandens lašų stiklinėje aibę ir pan. Taip pat negalima kalbėti ir apie visų aibų aibę, kadangi tokios aibės nagrinėjimas gali duoti prieštaravimą. Šis pavyzdys žinomas kaip Raselo paradoksas. Dažniausiai pateikiama pavyzdžiai aibų, kurios nėra pačios savęs elementais: tai aibės  $N, Q, Z$  ir pan. Tačiau nagrinėjant visų aibų aibę, tai ji yra pati savęs elementas. Raselias siūlo nagrinėti tokią aibę  $M$ , kurios elementai yra aibės  $X$ , tenkinančios sąlygą  $X \notin X$ . Kyla klausimas, ar  $M \in M$ ? Galimas vienas iš dviejų atvejų: arba  $M \notin M$  arba  $M \in M$ . Tarkime,  $M \notin M$ . Tada, pagal aibės  $M$  apibrėžimą, darome išvadą, kad  $M \in M$ . Tačiau šiuo atveju  $M \notin M$  ir t.t.

Kitas šio paradokso variantas yra barzdaskučio (Gonseto) paradoksas. Barzdaskutys prie savo dirbtuvės durų pakabino tokį užrašą: “čia skutami visi tie, kas patys nesiskuta”. Kyla klausimas, ar pats barzdaskutys gali save skusti? Galimas vienas iš dviejų atvejų: arba gali, arba negali. Jei barzdaskutys negali

---

<sup>2</sup> Matematikoje labai svarbus yra aibės atvaizdis į aibę, kai vieną aibės  $A$  elementą  $a \in A$  atitinka vieną aibės  $B$  elementą  $b \in B$  ir atvirkščiai. Toks atvaizdis vadinamas abipusiai vienareikšmiu  $A$  atvaizdžiu į  $B$ , arba bijekcija.

skustis, t.y. jis pats nesiskuta, tai pagal skelbimą jis gali skustis. Tačiau, jam pradėjus skustis, šis veiksmas prieštarautų skelbimui. Vadinasi, jis negali skustis. Tačiau šiuo atveju pagal skelbimą jis gali skustis ir t.t. Gauname prieštaravimą arba padėti be išeities.

### **Pavyzdys [PS85]**

Tegul turime aibes:

$$B = \{0,1\},$$

$$D = \{0,1,0\},$$

$$A = \{0,1,B\},$$

$$G = \{\emptyset, \{0\}, \{1\}, \{0,1\}\},$$

$$C = \{y \mid y = B \vee y \in B\}.$$

Kurios iš šių aibų yra lygios?

Pirmiausia pastebime, kad aibės  $B$  ir  $D$  yra lygios. Tiesiog aibė  $D$  užrašyta nekorektiškai, jos elementas 0 pakartotas du kartus. Aibė  $A$  turi tris skirtinges elementus: 0, 1 ir aibę  $B$ , o aibė  $G$  – keturis skirtinges elementus: tuščiąją aibę ir aibes  $\{0\}$ ,  $\{1\}$  ir  $\{0, 1\}$ . Todėl  $A \neq G$ . O kaip aibė  $C$ ? Pastebime, kad aibė  $B$  yra aibės  $C$  elementas, o 0 ir 1 priklauso aibei  $C$  kaip aibės  $B$  elementai. Taigi,  $C = \{0,1,B\} = A$ .

### **Uždaviniai [PS85, KM77]**

1. Išvardykite šių aibų elementus:

a)  $A = \{n \in Z \mid -4 < n < 5\},$

b)  $B = \{x \in R \mid x^3 - x^2 - 2x = 0\}.$

Ats: a)  $A = \{-3, -2, -1, 0, 1, 2, 3, 4\}$ , b)  $b = \{-1, 0, 2\}$ .

2. Nustatykite, ar teisingi šie santykiai:

a)  $1 \in \{1,2\}$ , b)  $3 \notin \{0,1\}$ , c)  $3 \notin \emptyset$ , d)  $\emptyset \in \emptyset$ , e)  $\emptyset \in \{\emptyset\}$ ,

f)  $\{0,1\} \in \{\{\{0,1\}, 2\}, \{1,2\}, 0, 1\}$ ,  $\{1,2\} \notin \{\{1,2\}, 1, 2\}$ , g)  $\{1,2\} \in \{\{1,2\}\}$ .

3. Pateikite tokį aibų  $A$ ,  $B$ ,  $C$ ,  $D$  pavyzdžius, kad  $A \in B$ ,  $B \in D$ ,  $D \in C$ ,  $A \notin D$ ,  $B \notin C$ .

4. Patvirtinkite arba paneikite žemiau pateiktus teiginius. Bet kurioms aibėms  $A$ ,  $B$  ir  $C$ :

a) jei  $A \notin B$  ir  $B \notin C$ , tai  $A \in C$ ,

b) jei  $A \neq B$  ir  $B \neq C$ , tai  $A \neq C$ ,

c) jei  $A \in B$  ir  $B \neq C$ , tai  $A \notin C$ ,

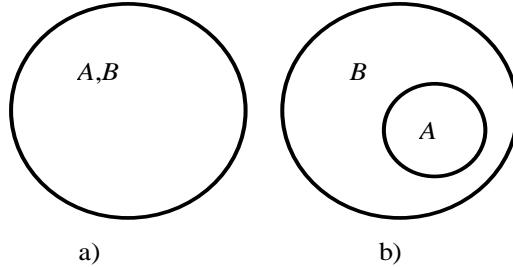
d) jei  $A \in B$  ir  $B = C$ , tai  $A \in C$ ,

e) jei  $A \notin B$  ir  $B \notin C$ , tai  $A \notin C$ .

## 1.2. Poaibiai

*Apibrėžimas.* Aibė  $A$  yra aibės  $B$  poaibis (žymime  $A \subseteq B$ ), jei kiekvienas aibės  $A$  elementas yra ir aibės  $B$  elementas.

Šį apibrėžimą iliustruoja Veno diagrammos, parodytos 1.2.1 paveiksle.

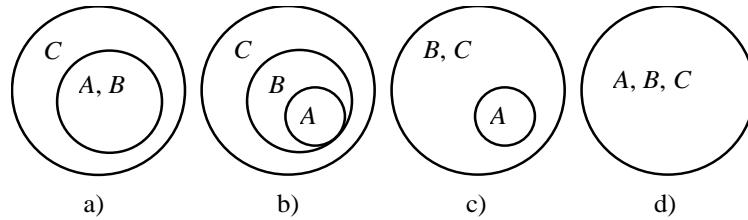


**1.2.1 pav.** Poaibių Veno diagrammos

Jei  $A \subseteq B$  ir  $A \neq B$ , tai  $A \subset B$ . Kitaip tariant, jei kiekvienas aibės  $A$  elementas yra ir aibės  $B$  elementas, tačiau egzistuoja bent vienas aibės  $B$  elementas, kuris nėra aibės  $A$  elementas, tai aibė  $A$  yra griežtas aibės  $B$  poaibis. Šiuo atveju rašome  $A \subset B$ . (žr. 1.2.1 pav. b) atveji). Tokie poaibiai vadinami *tikriniais poaibiais*. Aibės  $B$  poaibiai: tuščioji aibė  $\emptyset$  ir aibė  $B$ , vadinami aibės  $B$  *netikriniais poaibiais*.

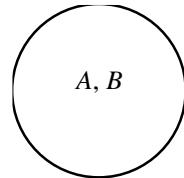
### Poaibių savybės

1.  $\emptyset \subseteq M$ , čia  $M$  – bet kuri aibė. Kitaip tariant, tuščioji aibė yra kiekvienos aibės poaibis.
2. Jei  $A \subseteq B$  ir  $B \subseteq C$ , tai  $A \subseteq C$ . Šią savybę iliustruoja Veno diagrammos, pavaizduotos 1.2.2 paveiksle.



**1.2.2 pav.** Veno diagrammos, iliustruojančios 2-ają savybę

3. Jei  $A \subset B$  ir  $B \subset C$ , tai  $A \subset C$ . Šią savybę iliustruoja 1.2.2 paveikslo b) atvejo Veno diagrama.
4. Jei  $A \subseteq B$  ir  $B \subseteq A$ , tai  $A = B$ . (Žr. 1.2.3 pav.)



**1.2.3 pav.** Veno diagrama, iliustruojanti 4-tąjų savybę

5. Jei  $A \subset B$  ir  $B \subseteq C$ , tai  $A \subset C$ . Šią savybę iliustruoja 1.2.2 paveikslo b) ir c) atvejai.

### Pavyzdys [PS85]

Tegul turime aibes:

$$A = \{x \in Z \mid -20 \leq x \leq 20\},$$

$$B = \{n \in N \mid n^2 \leq 49\},$$

$$C = \{A, B, -3, 4, 6\},$$

$$D = \{-3, 4, 6, 7\}.$$

Tada teisingi šie teiginiai:

$$B \subseteq A,$$

$$C \not\subseteq A, \text{ kadangi } A \in C, \text{ bet } A \notin A,$$

$$D \subseteq A,$$

$$C \not\subseteq B,$$

$$D \not\subseteq B, \text{ kadangi } -3 \in D, \text{ bet } -3 \notin B,$$

$$D \not\subseteq C, \text{ kadangi } 7 \in D, \text{ bet } 7 \notin C.$$

### Realiujų skaičių poaibiai

Tarkime, kad  $a$  ir  $b$  yra realieji skaičiai ir  $a < b$ . Tada žemiu išvardinti realiujų skaičių aibės poaibiai turi vardus.

**Intervalas** (atkarpa, atvirasis intervalas). Žymime

$$(a, b) = \{x \in R \mid a < x < b\}.$$

**Segmentas** (uždarasis intervalas). Žymime  $[a, b] = \{x \in R \mid a \leq x \leq b\}$ .

**Pusiau atvirieji intervalai:**

$$(a, b] = \{x \in R \mid a < x \leq b\},$$

$$[a, b) = \{x \in R \mid a \leq x < b\}.$$

### Aibės A visų galimų poaibių aibė

Aibės  $A$  visų galimų poaibių aibė žymima  $P(A)$ . Kitaip tariant  $P(A) = \{X \mid X \subseteq A\}$ . Pavyzdžiu, jei  $A = \{1, 2, 3\}$ , tai

$$P(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}.$$

**Teorema.** Jei  $|A| = n$ , tai  $|P(A)| = 2^n$ .

**Irodymas.** Tegul  $A = \{a_1, a_2, \dots, a_n\}$ . Kiekvienam aibės  $A$  poaibiui  $B$  priskirkime skaičių (kodą)  $S_B = \alpha_1, \alpha_2, \dots, \alpha_n$ ,  $\alpha_i \in \{0, 1\}$ , kuris nusakomas taip:  $\alpha_i = 1$ , jei  $a_i \in B$  ir  $\alpha_i = 0$ , - priešingu atveju. Aišku, kad kiekvienam aibės  $A$  poaibiui atitinka vienas kodas ir atvirkščiai. Tada  $P(A)$  elementams atitiks visi  $n$ -skilčiai dvejetainiai skaičiai nuo 0 0 ... 0 iki 1 1 ... 1. Tokių skaičių yra  $2^n$ . Pavyzdžiu, aibės  $A = \{a_1, a_2, a_3\}$  visi galimi poaibiai bus nusakomi kodais.

Kodas	Poaibis
000	$\emptyset$
001	$\{a_3\}$
010	$\{a_2\}$
011	$\{a_2, a_3\}$
100	$\{a_1\}$
101	$\{a_1, a_3\}$
110	$\{a_1, a_2\}$
111	$\{a_1, a_2, a_3\}$

### Uždaviniai [PS85]

1. Išvardykite visus šių aibų poaibius:

- a)  $\emptyset$ ,
- b)  $\{1\}$ ,
- c)  $\{1, 2\}$ ,
- d)  $\{1, 2, 3\}$ ,
- e)  $\{\emptyset, 1\}$ ,
- f)  $\{\emptyset, 1, \{1\}\}$ .

Ats: a)  $\emptyset$ , b)  $\emptyset$ ,  $\{1\}$ , c)  $\emptyset$ ,  $\{1\}$ ,  $\{2\}$ ,  $\{1, 2\}$ , d)  $\emptyset$ ,  $\{1\}$ ,  $\{2\}$ ,  $\{3\}$ ,  $\{1, 2\}$ ,  $\{1, 3\}$ ,  $\{2, 3\}$ ,  $\{1, 2, 3\}$ , e)  $\emptyset$ ,  $\{\emptyset\}$ ,  $\{1\}$ ,  $\{\emptyset, 1\}$ , f)  $\emptyset$ ,  $\{\emptyset\}$ ,  $\{1\}$ ,  $\{\{1\}\}$ ,  $\{\emptyset, 1\}$ ,  $\{\emptyset, \{1\}\}$ ,  $\{1, \{1\}\}$ ,  $\{\emptyset, 1, \{1\}\}$ .

2. Duota aibė  $A = \{1, 2, 3, 4\}$ . Pasinaudojė teoremos įrodymu:

- a) sukonstruokite aibės  $A$  poaibius, atitinkančius dvejetainių skaičių kodus 0101, 1111, 1001, 1000, 0100, 0110, 0111, 0001,

- b) nustatykite dvejetainių skaičių kodus, atitinkančius poaibius  
 $\{3\}, \{4, 3\}, \{1, 3\}, \emptyset, \{4, 2, 3\}, \{1, 4, 3\}, \{1, 2, 3\}, \{1, 2, 3, 4\}$ .  
 Ats: a)  $\{2, 4\}, \{1, 2, 3, 4\}, \{1, 4\}, \{1\}, \{2\}, \{2, 3\}, \{2, 3, 4\}, \{4\}$ , b) 0010,  
 0011, 1010, 0000, 0111, 1011, 1110, 1111.

### 1.3. Poaibių generavimo algoritmai

Ivairiuose taikomuosiuose uždaviniuose tenka vienokia ar kitokia tvarka generuoti aibės poaibius. Aptarkime aibės  $A$ , turinčios  $n$  elementų, poaibių generavimo algoritmus.

#### Bendra poaibių generavimo algoritmo struktūra

Poaibių, kaip ir kitų kombinatorinių objektų, generavimo algoritmo struktūra yra tokia.

```
begin
    generuoti pradinj poaibj;
    while "generavimo sąlyga" do
        begin
            nagrinėti (spausdinti) poaibj;
            generuoti poaibj, gretimą išnagrinėtam (atspausdintam) poaibiui;
        end;
    end;
```

Priklasomai nuo pasirinktos poaibių generavimo tvarkos, šiame algoritme turi būti konkretizuoti sakiniai: "generuoti pradinj poaibj", "generavimo sąlyga", "poaibis, gretimas išnagrinėtam poaibiui".

Žemiau aptarsime du poaibių generavimo algoritmus: poaibių generavimo leksikografine didėjimo tvarka algoritmą ir poaibių generavimo taip, kad bet kokie du gretimi poaibiai skirtuši tik vienu elementu, algoritmą. Kadangi kiekvieną poaibj galime nusakyti dvejetainiu kodu, tai antru būdu sugeneruoti poaibiai vadinami Grejaus kodais.

#### Poaibių generavimas leksikografine didėjimo tvarka

##### Uždavinio formulavimas

Duota:  $n$  – aibės elementų skaičius,  
 $a[1.. n]$  – aibės  $A$  elementai.

Rasti: generuoti aibės  $A$  poaibius leksikografine didėjimo tvarka.

Kaip buvo minėta aukščiau, kiekvieną aibės  $A$  poaibj galima nusakyti dvejetainiu kodu  $b_1, b_2, \dots, b_n$ ,  $b_i \in \{0, 1\}$ . Jei kodo elementas  $b_i = 1$ , tai elementas  $a_i$  priklauso poaibiui. Tuo būdu poaibj nusakys masyvas  $b[0.. n]$ .

Aptarkime kodų leksikografinę tvarką. Iš kodą galima žiūrėti kaip į vektorių, todėl, aptardami leksikografinę tvarką, naudosime žodį vektorius.

Vektorius  $\mathbf{x}$  yra leksikografiškai didesnis už  $\mathbf{0}$ , (žymime  $\mathbf{x} \succ \mathbf{0}$ ), jei jo pirmoji nenulinė komponentė yra teigiamą.

Vektorius  $\mathbf{x} \succ \mathbf{y}$ , jei  $\mathbf{x} - \mathbf{y} \succ \mathbf{0}$ .

Iš vektorių leksikografinės tvarkos išplaukia, kad, jei į kodą žiūrėsime kaip į dvejetainį skaičių, tai kodai sudarys visus  $n$  skilčių dvejetainius skaičius, išdėstytais didėjimo tvarka. Pavyzdžiui, jei  $n = 4$ , tai kodai, išdėstyti leksikografine didėjimo tvarka, atrodys taip: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111.

Dabar galime konkretizuoti aukščiau išvardintus sakinius.

*Pradinio kodo generavimas.*  $b_i = 0, i = \overline{0, n}$ .

*Gretimo kodo generavimas.* Gretimą poaibį nusako kodas – dvejetainis skaičius, vienetu didesnis nei išnagrinėtasis. Vadinas, masyve  $b$  ieškome pirmo iš dešinės elemento, kuris lygus nuliui. Tarkime, kad  $b_i$  yra toks elementas.

Tada  $b_i := 1$ , o  $b_j := 0, j = \overline{i+1, n}$ .

*Generavimo sąlyga.* Aišku, kad generavimą baigsime, kai masyvo  $b$  pirmojo iš dešinės elemento, lygaus nuliui, indeksas bus lygus 0, t.y. kai  $b_0 := 1$ , o  $b_j := 0, j = \overline{1, n}$ .

Žemiau pateikiamas poaibių generavimo leksikografine didėjimo tvarka algoritmas.

```

begin
    {Pradinio poaibio generavimas}
    for i:= 0 to n do b[i]:= 0;
    t:= true; {generavimo sąlyga}
    while t do
        begin
            {Spausdinti poaibį}
            for i:= 1 to n do
                if b[i] = 1 then spausdinti a[i];
            {Gretimo poaibio generavimas}
            i:=n;
            while (i ≥ 1) and (b[i] = 1) do
                begin
                    b[i]:= 0;
                    i:= i - 1;
                end;
            if i=0 then t :=false {generavimo pabaiga}

```

```

        else  $b[i] := 1;$ 
    end;
end;

```

### **Gréjaus kodo generavimo algoritmai**

Aptarsime Gréjaus kodą, t.y. poaibį, kai bet kokie du gretimi poaibiai skiriasi tik vienu elementu, generavimo algoritmus. Kadangi Gréjaus kodai plačiai taikomi kodavimo teorijoje, tai 1.4 paragrafe trumpai aptarsime kodavimo teorijos elementus ir Gréjaus kodų taikymą kodavimo uždaviniuose.

**Pirmasis algoritmas.** Šis algoritmas remiasi aukščiau išnagrinėtu poaibį generavimo leksikografine tvarka algoritmu. Kitais tariant, nuosekliai generuojant dvejetainius skaičius, jie gali būti perskaičiuoti į Gréjaus kodus.

Tarkime, kad  $b_1 b_2 \dots b_n$  yra poaibio, sugeneruoto pagal aukščiau pateiktą algoritmą, kodas. Tada šis kodas perskaičiuojamas į Gréjaus kodą  $c_1 c_2 \dots c_n$  pagal formulę:

$$\begin{aligned} c_1 &:= b_1; \\ c_i &:= b_{i-1} \oplus b_i, i = \overline{2, n}, \end{aligned} \tag{1.3.1}$$

čia simbolis  $\oplus$  žymi operaciją "suma moduli 2".

Pavyzdžiuui, jei  $n = 3$ , tai perskaičiavimą iš dvejetainio kodo į Gréjaus kodą iliustruoja 1.3.1 lentelė.

lentelė. Perskaičiavimas iš dvejetainio kodo į Gréjaus kodą

Dvejetainis kodas	Gréjaus kodas
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

(1.3.1) formulės teisingumą nesunku pagrįsti taip.

Aišku, kad du iš eilės einantys dvejetainiai skaičiai turės pavidalą

$$\begin{aligned} * * \dots * 0 &1 1 \dots 1, \\ * * \dots * 1 &0 0 \dots 0, \end{aligned}$$

čia "\*" žymi simbolį 1 arba 0. Tarkime, kad pirmajame skaičiuje "0" yra i-tojoje skiltyje, o dešiniau jos yra vienetukai. Aišku, kad "\*" pažymėtos skilčių vertės abejuose skaičiuose sutampa.

Pritaikius šiemis skaičiams (1.3.1) formulę, pirmojo ir antrojo skaičiaus  $i$ -tosios skiltys bus priešingos: jei pirmojo skaičiaus  $i$ -toji skiltis bus 0, tai antrojo – 1 ir atvirkščiai. Abiejų skaičių  $(i+1)$ -oji skiltis bus lygi 1, o abiejų skaičių  $(i+2), (i+3), \dots, n$  skiltys bus lygios 0. Aišku, kad abiejų skaičių skilčių nuo 1 iki  $(i-1)$  vertės taip pat sutampa. Tuo būdu (1.3.1) formulė dvetainius skaičius perskaičiuoja į Grėjaus kodus.

**Pastaba.** Pertvarkyta (1.3.1) formulė įgalina perskaičiuoti Grėjaus kodus į dvejetainius skaičius. Tarkime, kad  $c_1 c_2 \dots c_n$  yra Grėjaus kodas, o  $b_1 b_2 \dots b_n$  – jam atitinkantis dvejetainis skaičius. Tada iš (1.3.1) formulės gausime

$$\begin{aligned} b_1 &:= c_1; \\ b_i &:= c_i \oplus b_{i-1}, i = \overline{2, n}. \end{aligned} \tag{1.3.2}$$

Iš pateikto nagrinėjimo aišku, kad aukščiau pateiktas poaibiu generavimo algoritmas tiks Grėjaus kodų generavimui, jei prieš komentara “Spausdinti poaibį” pagal (1.3.1) formulę apskaičiuosime masyvo  $c[1..n]$  elementus.

### Antrasis algoritmas

**Metodo idėja.** Jei aibė  $A$  turi vieną elementą, tai Grėjaus kodai bus 0 ir 1. Grėjaus kodus aibei, turinčiai du elementus, generuosime remdamiesi Grėjaus kodais aibei, turinčiai vieną elementą. Turėdami Grėjaus kodus aibei, turinčiai  $(n-1)$  elementų, generuosime Grėjaus kodus aibei, turinčiai  $n$  elementų, tokiu būdu:

- 1) prie aibės, turinčios  $(n-1)$  elementą, Grėjaus kodų iš dešinės prirašome “0”,
- 2) po to prie aibės iš  $(n-1)$  elementų Grėjaus kodų, rašomų atvirkšcia tvarka, dešinės pusės prirašome “1”.

Tuo būdu gausime:

- jei  $n = 1$ , tai Grėjaus kodai yra 0, 1;
- jei  $n = 2$ , tai Grėjaus kodai yra 00, 10, 11, 01;
- jei  $n = 3$ , tai Grėjaus kodai yra 000, 100, 110, 010, 011, 111, 101, 001 ir t.t.

**Pastaba.** Pabraukti simboliai yra Grėjaus kodai aibei iš  $(n-1)$  elemento.

Kaip matome, metodo idėja yra labai paprasta, tačiau ji neatitinka bendros poaibių generavimo algoritmų schemas: norint generuoti Grėjaus kodus aibei iš  $n$  elementų, reikia turėti  $(n-1)$  elementų aibės visus Grėjaus kodus. Subtili šio metodo realizacija, tenkinanti bendrą poaibių generavimo algoritmo schemą, pateikta literatūroje [Lip88]. Čia aptarsime šią realizaciją.

**Funkcija  $Q(i)$ ,  $i \in N$ .**  $Q(i)$  – tai toks didžiausias dvejeto laipsnis, kad  $2^{Q(i)}$  yra  $i$  daliklis, t.y.  $i \bmod 2^{Q(i)} = 0$ .

Pavyzdžiu,  $Q(3) = 0$ ;  $Q(4) = 2$ ;  $Q(5) = 0$ ;  $Q(6) = 1$ ;  $Q(8) = 3$ ;  $Q(12) = 2$ .

**Funkcijos  $Q(i)$  savybė.** Nesunku pastebeti, kad  $Q(2^k + m) = Q(2^k - m)$ , čia  $0 \leq m \leq 2^k - 1$ .

Pavyzdžiu, kai  $k = 2$ , turėsime

$$Q(4+0) = Q(4-0) = 2,$$

$$Q(4+1) = Q(4-1) = 0,$$

$$Q(4+2) = Q(4-2) = 1,$$

$$Q(4+3) = Q(4-3) = 0.$$

Remdamiesi funkcija  $Q(i)$  sudarysime Gręjaus kodų generavimo algoritmą pagal aukščiau aprašytą metodą.

Įėjimo duomenys tokie pat, kaip ir algoritmo, generuojančio poaibius leksikografine didėjimo tvarka, t.y.

$n$  – aibės elementų skaičius;

$a[1.. n]$  – aibės  $A$  elementai.

Poaibius (Gręjaus kodus) talpinsime į masyvą  $b[1.. n]$ . Kintamasis  $i$  žymi Gręjaus kodo eilės numerį.

```

begin
    {Pradinio poaibio generavimas}
    for i := 1 to n do b[i] := 0;
    i := 0; {i – Gręjaus kodo eilės numeris}
    t := true; {t – poaibiu generavimo sąlyga}
    while t do
        begin
            for k := 1 to n do
                if b[k] = 1 then nagrinéti (spausdinti)  $a_k$ ;
                {Generuoti poaibį (kodą), gretimą išnagrinétam}
                i := i + 1;
                p := 1 {Po apskaičiavimo p reikšmė bus lygi  $1 + Q(i)$ }
                j := i;
                while j mod 2 = 0 then
                    begin
                        p := p+1;
                        j := j div 2;
                    end;
                { $p = 1 + Q(i)$ }
                if p <= n then {generavimo sąlyga galioja, nes  $i \leq 2^n - 1$ }
                    b[p] := 1 - b[p] {invertuojame kodo p-ajq skiltį}
                else { $i = 2^n$  ir generavimo pabaiga}
        
```

```

    t := false;
end;

```

1.3.2 lentelėje parodytas algoritmo veikimas, kai  $n = 3$

### 1.3.2 lentelė. Grējaus kodų generavimas, kai $n = 3$

$i$	$b_1$	$b_2$	$b_3$	$p := 1 + Q(i)$
0	0	0	0	
1	1	0	0	1
2	1	1	0	2
3	0	1	0	1
4	0	1	1	3
5	1	1	1	1
6	1	0	1	2
7	0	0	1	1
8				4 { $p > 3$ pabaiga}

1.3.2 lentelėje laužtiniais skliaustais išryškinti Grējaus kodai aibei iš vieno elemento, o riestiniai skliaustai – Grējaus kodai aibei iš 2-jų elementų. Šie išryškinimai vaizdžiai parodo, kad dėl aukščiau aptartos funkcijos  $Q(i)$  savybės, generuojant Grējaus kodus aibei iš  $n$  elementų, kai  $i \leq 2^{n-1} - 1$ , generuojami Grējaus kodai iš  $(n - 1)$  elementų. Šiemis kodams pozicija  $b_n$  yra lygi nuliui. Kai  $i = 2^{n-1} - 1$ ,  $b_n$  įgauna reikšmę 1 ir toliau, didinant  $i$  reikšmę, atvirkščia tvarka generuojami Grējaus kodai aibei iš  $(n - 1)$  elemento. Kitaip tariant, pateiktas algoritmas tikrai realizuoja aukščiau aptartą Grējaus kodų generavimo metodą.

**Pastaba.** Pateiktas algoritmas generuos Grējaus kodus nuo bet kokio (nebūtinai nulinio) pradinio kodo.

Trečiasis Grējaus kodo generavimo algoritmas išnagrinėtas 3.4.1 paragrale.

## 1.4. Grējaus kodų taikymo pavyzdžiai

Kaip minėjome, Grējaus kodai dažniausiai naudojami sprendžiant kodavimo uždavinius, todėl pirmiausia aptarsime kodavimo teorijos elementus.

### Kodavimo teorijos elementai

Kodavimas – tai procesas, kurio metu pradinė informacija keičiamā kokiui nors jos atitikmeniu – kodu. Priežastys, dėl kurių yra reikalingas kodavimas, gana įvairios. Pavyzdžiui, žmogus savo kasdienėje veikloje

kiekybiniam įverčiui (skaičiams) išreikšti paprastai naudojasi dešimtainė skaičiavimo sistema. Absoliuti dauguma pasaulyje šiuo metu naudojamų kompiuterių dėl savo konstruktyvinių ypatybių naudoja dvejetainę skaičiavimo sistemą. Be to, išaugus kompiuterių galimybėms, jie labai plačiai naudojami ne tik skaitinės, bet ir tekstinės informacijos apdorojimui. Tuo pačiu neišvengiamai atsiranda būtinybė dešimtainius skaičius bei tekstinę informaciją koduoti vienetukų ir nulių sekomis, nes tokis informacijos pateikimo būdas yra labiausiai priimtinės kompiuteriuose dėl jų vidinės sandaros. Kitas svarbus kodavimo atvejis – informacijos perdavimas. Informacijos perdavimo terpė praktiškai niekada nebūna ideali, ir perduodamą informaciją paprastai daugiau ar mažiau veikia triukšmai. Kaip priemonė šio žalingo triukšmų poveikio neutralizavimui arba bent jau sumažinimui naudojami specialūs kodavimo būdai, kurių esmę sudaro klaidas aptinkančių ar net jas ištaisančių kodų panaudojimas.

1.3.3 lentelėje pateiktii įvairūs dešimtainių skaičių kodavimo būdai: dvejetainis kodavimas, BCD kodavimas (nuo “Binary Coded Decimal” – dvejetainiu būdu užkoduotas dešimtainis skaičius), Grėjaus kodas, kodas su pertekliumi “3”, romeniškas skaičiaus kodavimas.

Trumpai aptarsime kiekvieną iš šių kodų.

Dvejetainis skaičių kodas – pats natūraliausias kompiuterio sandaros požiūriu kodas, naudojamas kompiuteriuose atliekant aritmetinius skaičiavimus.

BCD kodas – tarpinis kodas tarp vaizdavimo kompiuteryje ir žmogui įprasto vaizdavimo. Kiekvienas dešimtainis skaitmuo užkoduotas bitų ketveriuke (kaip kad įprasta kompiuteryje), tačiau patys skaičiai visgi dešimtainiai (kaip įprasta žmogui) ir patys veiksmai su tokiais skaičiais skiriiasi nuo dvejetainių skaičių aritmetikos.

Šešioliktainis kodas taip pat buvo pradėtas naudoti kaip tarpinis kodas grandyje “žmogus / kompiuteris”. Jei poroje “dvejetainis kodas / BCD” pagrindas yra dešimtainis skaičius, kurio kiekvienam dešimtainiui skaitmeniui užkoduoti naudojama keturių bitų seka (nuo 0000 iki 1001), tai poroje “dvejetainis skaičius / šešioliktainis skaičius” pagrindas yra keturių dvejetainių skaitmenų dvejetainis skaičius, kurio kodavimui naudojamas vienas iš 16-kos šešioliktainės sistemos skaitmenų nuo 0 iki F. Šešioliktainis kodas kaip tarpinis kartais naudojamas projektuojant kompiuterių blokus.

### 1.3.3 lentelė. Dešimtainių skaičių kodavimo būdai

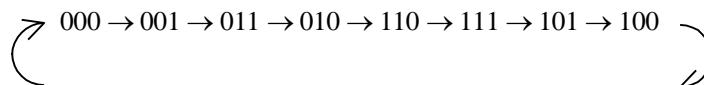
Dešimt-ainis sk.	Dvejet-ainis kodas	BCD kodas	Kodas "16"	Kodas su pertekl. "3"	Grėjaus kodas	Romēn-iškas kodas
0	0000	0000	0	0011	0000	-
1	0001	0001	1	0100	0001	I
2	0010	0010	2	0101	0011	II
3	0011	0011	3	0110	0010	III
4	0100	0100	4	0111	0110	IV
5	0101	0101	5	1000	0111	V
6	0110	0110	6	1001	0101	VI
7	0111	0111	7	1010	0100	VII
8	1000	1000	8	1011	1100	VIII
9	1001	1001	9	1100	1101	IX
10	1010	10000	A	10011	1111	X
11	1011	10001	B	10100	1110	XI
12	1100	10010	C	10101	1010	XII

Kodas su pertekliumi "3" kartais naudojamas kompiuteryje aritmetinių operacijų realizacijos efektyvumui padidinti.

Grėjaus kodo taikymas, skirtingai nuo aukščiau aprašytų, dominuoja ne kompiuterio aritmetinių operacijų realizavimui, bet tokiose srityse kaip kompiuterių įrangos projektavimas, taip pat kombinatorinių uždavininių sprendimo algoritmų sudarymas. Detaliau panagrinėsime šiuos taikymus.

*Apibrėžimas.* Hemingo atstumu tarp dviejų kodo kombinacijų vadinamas skaičius pozicijų, kuriose viena kombinacija skiriasi nuo kitos.

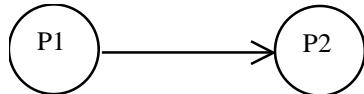
Akivaizdu, kad Grėjaus kodas turi tą savybę, kad Hemingo atstumas tarp bet kurių gretimų Grėjaus kodo kombinacijų yra lygus 1. Dar viena Grėjaus kodo savybė yra tai, kad jis yra ciklinis kodas. Pavyzdžiui, jei  $n = 3$ , tai



Tai, kad Hemingo atstumas tarp bet kurių dviejų gretimų Grėjaus kodo kombinacijų yra lygus 1, leidžia efektyviai panaudoti Grėjaus kodą spręsti įvairius kombinatorikos uždavinius.

### **Baigtinio automato padėčių kodavimas Grėjaus kodo kombinacijomis**

Sakykime, kad automato perėjimų grafe yra fragmentas (žiūr. 1.4.1 pav.).



**1.4.1 pav.** Automato perėjimo grafo fragmentas

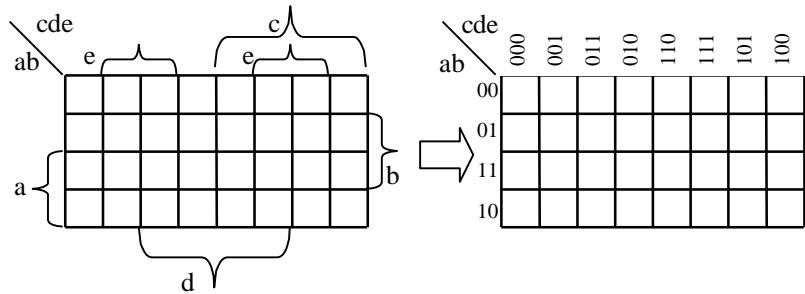
čia P1 ir P2 – automato būviai. Tegu šie du būviai yra užkoduoti kodais 001 ir 100 atitinkamai. Tegu šias tris skiltis atitinka trys būvių registro trigeriai T1, T2 ir T3. Perėjimas S1 → S2 reiškia, kad mes turėsime pversti trigerį T1 iš “0” į “1”, o trigerį T3 – iš “1” į “0”. Paprastai dėl skirtinį elektroninių kelių trigerių perjungimo shemos grandinėse perjungimo signalo vėlinimai šiek tiek skiriasi. Tuo pačiu, pereinant iš P1 į P2 automatas paklius visų pirma į kažkokį tarpinį būvį  $T^*$  (pvz. 101 – T1 jau persijungęs, T3 – dar ne). Tai reiškia, kad iš šio būvio ( $T^* \Leftrightarrow 101$ ) automatas gali nepereiti į T2 (kas būtų reikalinga pagal automato funkcionavimo algoritmą), bet į kažkokią kitą padėtį. Šis ypatumas, susijęs su automato techninė realizacija (nevienodais įvairių šakų elektrinės schemas vėlinimais) yra vadinamas “automato lenktynėmis”.

Akivaizdu, kad šio ydingo reiškinio galima išvengti, jei du gretimi automato būviai, tarp kurių yra perėjimas, būtų užkoduoti Grėjaus kodo pagalba.

### **Bulio funkcijų (BF) Karko diagramos konstravimas**

Bulio funkcijų vaizdavimas Karko diagramų pagalba visų pirma yra naudingas BF minimizavimui. Du nagrinėjamos minimizuojamos BF mintermus galima apjungti į vieną termą, jei tuos mintermus atitinkantys vienetukai yra gretimuose Karko diagramos langeliuose. Karko diagramai sukonstruoti galima panaudoti Grėjaus kodo kombinacijas.

Kadangi Hemingo atstumas tarp bet kurių dviejų Grėjaus kodų yra lygus 1, tai atitinkamai bet kuriuos du gretimus BF Karko diagramos langelius bus galima apjungti (žiūr. 1.4.2 pav.).



1.4.2 pav. Karko diagramma

## 1.5. Veiksmai su aibėmis (aibių algebra)

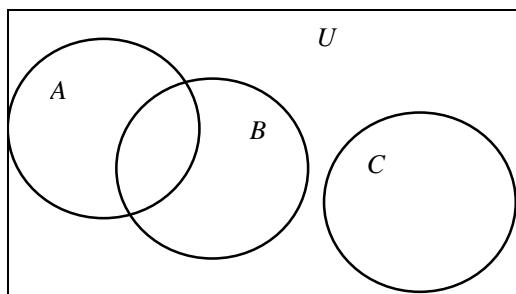
Aptarsime veiksmus su aibėmis.

Pirmiausia apibrėžiame **universaliosios aibės** sąvoką.

Tarkime, kad visos konkrečiu atveju nagrinėjamos aibės yra kažkuriros aibės  $U$  poaibiai. Tada aibė  $U$  vadinama universalija aibe.

Pavyzdžiu, jei nagrinėjame realiųjų skaičių poaibius, tai jų atžvilgiu  $U = \mathbb{R}$ . Jei nagrinėjame skaičiavimo uždavinį, sprendžiamą kompiuteriu, rezultatus, tai kompiuterio įsimenamų realiųjų skaičių aibė yra  $U$ .

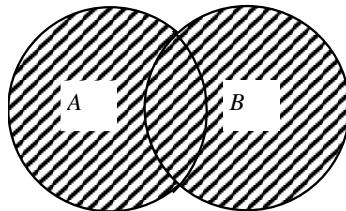
Universalioji aibė Veno diagrama paprastai vaizduojama stačiakampiu (žr. 1.5.1 pav.)



1.5.1 pav. Universaliosios aibės Veno diagramma

**Aibių  $A$  ir  $B$  sąjunga.** Aibių  $A$  ir  $B$  sąjunga yra aibė  $C$ , kurią sudaro elementai, priklausantys aibei  $A$  arba aibei  $B$ . Žymime

$$C = A \cup B = \{c \mid c \in A \vee c \in B\} \text{ (žr. 1.5.2 pav.).}$$

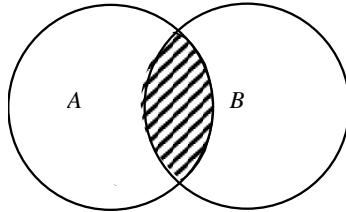


**1.5.2 pav.** Aibų  $A$  ir  $B$  sajungos Veno diagrama

**Aibų  $A$  ir  $B$  sankirta.** Aibų  $A$  ir  $B$  sankirta yra aibė  $C$ , kurią sudaro elementai, priklausantys ir aibei  $A$  ir aibei  $B$ . Žymime

$$C = A \cap B = \{c \mid c \in A \wedge c \in B\} \text{ (žr. 1.5.3 pav.)}.$$

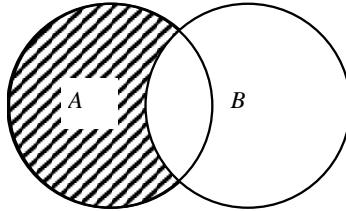
Jei  $A \cap B = \emptyset$ , tai sakome, kad aibės  $A$  ir  $B$  nesusikertančios.



**1.5.3 pav.** Aibų  $A$  ir  $B$  sankirtos Veno diagrama

**Aibų  $A$  ir  $B$  skirtumas.** Aibų  $A$  ir  $B$  skirtumas yra aibė  $C$ , kurią sudaro aibės  $A$  elementai, nepriklausantys aibei  $B$ . Žymime

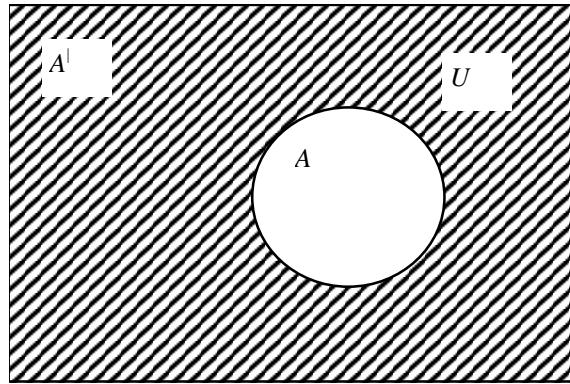
$$C = A - B \text{ arba } A \setminus B = \{c \mid c \in A \wedge c \notin B\} \text{ (žr. 1.5.4 pav.)}.$$



**1.5.4 pav.** Aibų  $A$  ir  $B$  skirtumo Veno diagrama

**Aibės  $A$  papildinys.** Aibės  $A$  papildinys yra aibė, papildanti aibę  $A$  iki universaliosios aibės. Žymime

$$A' = U - A \text{ arba } U \setminus A \text{ (žr. 1.5.5 pav.)}.$$



**1.5.5 pav.** Aibės  $A$  papildinio Veno diagrama

**Pavyzdžiai [PS85]**

1. Duotoms aibėms  $U = N$ ,  $A = \{1, 3, 5, 7, \dots\}$ ,  $P = \{p \in N \mid p \text{ yra pirminis}\}$  ir  $T = \{3, 6, 9, 12, \dots\}$  rasti  $A'$ ,  $P'$ ,  $A \cap P$ ,  $A - P$ ,  $P - A$ ,  $A' \cap T$  ir  $A \cup T'$ .

Aibė  $A$  susideda iš sveikujų teigiamų nelyginių skaičių, todėl jos papildinys  $A'$  susideda iš sveikujų teigiamų lyginių skaičių:  $A' = \{2, 4, 6, 8, \dots\}$ .

Aibė  $P$  yra pirminių skaičių aibė, todėl  $P' = \{1, 4, 6, 8, 9, 10, \dots\}$ .

Aibė  $A \cap P$  yra nelyginių pirminių skaičių aibė:

$$A \cap P = \{3, 5, 7, 11, 13, \dots\} = P - \{2\}.$$

Sveikujų teigiamų nelyginių skaičių, kurie nėra pirminiai, aibė yra  $A - P = \{1, 9, 15, 21, 25, 27, \dots\}$ .

Aibė  $P - A$  susideda iš pirminių lyginių skaičių, vadinasi  $P - A = \{2\}$ .

Aibė  $T$  yra kartotinių trims teigiamų skaičių aibė, todėl  $A' \cap T$  susideda iš tų aibės  $T$  elementų, kurie yra lyginiai:  $A' \cap T = \{6, 12, 18, 24, \dots\}$ .

Pagaliau,  $A \cup T'$  apima tuos sveikuosius teigiamus skaičius, kurie yra arba nelyginių, arba nekartotinių trims:

$$A \cup T' = \{1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 13, \dots\}.$$

Pastebėsime, kad  $A \cup T' = \{6, 12, 18, 24, \dots\}' = (A' \cap T)'$ .

2. Duotos aibės  $U = \{0, 1, 2, 3, 4, 5, 6\}$ ,  $A = \{0, 2, 4, 6\}$ ,  $B = \{2, 3, 5\}$  ir  $C = \{0, 1, 5, 6\}$ . Rasti ir palyginti tokias aibų poras:

- a)  $A \cap (B \cup C)$  ir  $(A \cap B) \cup (A \cap C)$ ,
- b)  $A \cup (B \cap C)$  ir  $(A \cup B) \cap (A \cup C)$ ,
- c)  $(A \cup B)'$  ir  $A' \cap B'$ ,
- d)  $(A \cap B)'$  ir  $A' \cup B'$ .

a) atveju randame, kad:

$$A \cap (B \cup C) = A \cap \{0, 1, 2, 3, 5, 6\} = \{0, 2, 6\},$$

$$(A \cap B) \cup (A \cap C) = \{2\} \cup \{0, 6\} = \{0, 2, 6\}.$$

Vadinasi,  $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ .

Analogiškai išsprendę b) atvejį, rasime, kad

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C) = \{0, 2, 4, 5, 6\}.$$

c) atveju gauname:

$$(A \cup B)' = \{0, 2, 3, 4, 5, 6\}' = \{1\},$$

$$A' \cap B' = \{1, 3, 5\} \cap \{0, 1, 4, 6\} = \{1\}.$$

Šiuo atveju aibės taip pat yra lygios.

Ir, analogiškai, d) atveju gauname

$$(A \cap B)' = A' \cup B' = \{0, 1, 3, 4, 5, 6\}.$$

### **Uždavinys [PS85]**

Duotoms aibėms  $U = \{0, 1, 2, 3, 4, 5, 6\}$ ,  $A = \{0, 2, 4, 6\}$  ir  $B = \{2, 3, 5\}$  rasti  $A \cap B$ ,  $A \cup B$ ,  $A - B$ ,  $B - A$ ,  $A'$  ir  $B'$ .

Ats:  $A \cap B = \{2\}$ ,  $A \cup B = \{0, 2, 3, 4, 5, 6\}$ ,  $A - B = \{0, 4, 6\}$ ,  $B - A = \{3, 5\}$ ,  $A' = \{1, 3, 5\}$ ,  $B' = \{0, 1, 4, 6\}$ .

**Aibų A ir B simetrinė skirtumas arba A ir B suma moduliui 2.** Aibų A ir B simetrinė skirtumas yra aibė C, kurią sudaro elementai, priklausantys arba tik aibei A, arba tik aibei B. Žymime  $C = A \Delta B$  arba  $A \oplus B$ .

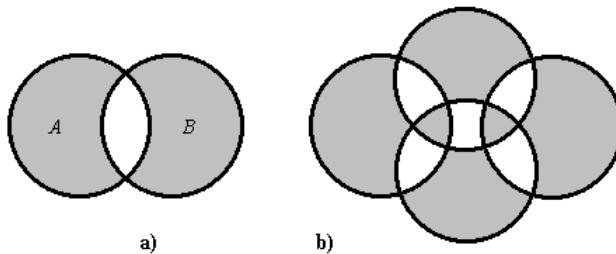
$$C = (A \setminus B) \cup (B \setminus A) = (A \cup B) \setminus (A \cap B) \quad (\text{žr. 1.5.6 a) pav.}).$$

Aibų simetrinio skirtumo arba sumos moduliui 2 operacija apibendrinama daugiau nei dviem aibėm.

Aibė B yra aibų  $A_1, A_2, \dots, A_n$  suma moduliui 2 (simetriniu skirtumu), jei

$$B = \bigoplus_{i=1}^n A_i = \{a \in B \mid a \text{ priklauso nelyginiam skaičiui aibų } A_i\} \quad (\text{žr. 1.5.6 b) pav.}).$$

1.5.6 b) pav.



**1.5.6 pav.** Aibų A ir B simetrinio skirtumo Veno diagrama

### ***Uždavinys***

Tarkime, kad  $X$  ir  $Y$  yra aibės. Kokie bus  $X$  ir  $Y$  turiniai po žemiau pateiktų veiksmų atlikimo?

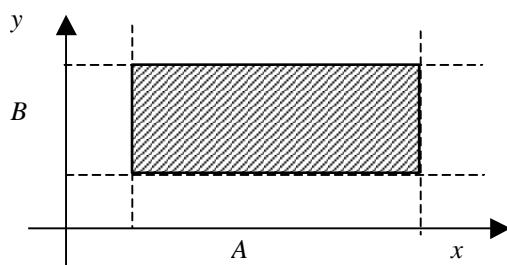
- 1)  $X := X \Delta Y$ ;
- 2)  $Y := (Y \setminus X) \cup (X \setminus Y)$ ;
- 3)  $X := (Y \setminus X) \cup (X \setminus Y)$ .

Ats: Aibų  $X$  ir  $Y$  turiniai bus sukeisti vietomis.

**Aibų  $A$  ir  $B$  Dekarto sandauga.** Aibų  $A$  ir  $B$  Dekarto sandauga yra aibė  $C$ , kurią sudaro visos galimos poros  $(a, b)$ , kai pirmasis poros elementas  $a \in A$ , o antrasis poros elementas  $b \in B$ . Žymime

$$C = A \times B = \{(a, b) \mid a \in A \wedge b \in B\}.$$

Vaizdžiai šis veiksmas parodytas 1.5.7 paveiksle.



**1.5.7 pav.** Aibų  $A$  ir  $B$  Dekarto sandauga

### ***Pavyzdys***

Duotos aibės  $A = \{0, 1, 2\}$  ir  $B = \{1, 3\}$ .

Rasti  $A \times B$  ir  $B \times A$ .

$$A \times B = \{(0, 1), (0, 3), (1, 1), (1, 3), (2, 1), (2, 3)\},$$

$$B \times A = \{(1, 0), (1, 1), (1, 2), (3, 0), (3, 1), (3, 2)\}.$$

Reikia pastebėti, kad  $A \times B \neq B \times A$ .

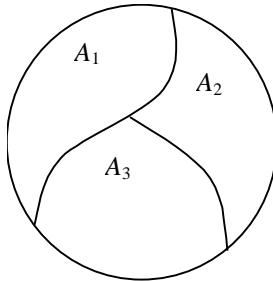
Dekarto sandauga gali būti apibendrinta ir daugiau nei dviem aibėm. Tarkime,  $A_1, A_2, \dots, A_n$  yra aibės. Tada šių aibų Dekarto sandauga yra aibė  $A$ , kurią sudaro visi galimi vektoriai  $(a_1, a_2, \dots, a_n)$ , čia  $a_i \in A_i, i = \overline{1, n}$ . Žymime

$$\begin{aligned} A &= A_1 \times A_2 \times \dots \times A_n = \prod_{i=1}^n A_i = \\ &= \{(a_1, a_2, \dots, a_n) \mid a_1 \in A_1 \wedge a_2 \in A_2 \wedge \dots \wedge a_n \in A_n\}. \end{aligned}$$

Dekarto sandauga yra patogi priemonė aprašyti erdves. Pavyzdžiu  $R^2$  (čia  $R$  – relijų skaičių aibė) žymi plokštumą,  $R^3$  – trimatę erdvę,  $R^n$  –  $n$ -matę erdvę.

### **Pavyzdys**

Duota  $A_1 = A_2 = A_3 = \{0, 1\}$ . Rasti  $A_1 \times A_2 \times A_3$ .  
 Sprendinys:  $A_1 \times A_2 \times A_3 = \{(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)\}$   
**Aibės  $A$  išskaidymas.** Aibės  $A_1, A_2, \dots, A_n$ , tenkinančios žemiau išvardintas savybes  
 1)  $A_1 \cup A_2 \cup \dots \cup A_n = A$ ,  
 2)  $A_i \cap A_j = \emptyset$ ,  $i \neq j$ ,  
 vadinamos aibės  $A$  išskaidymu (žr. 1.5.8 pav.).



**1.5.8 pav.** Aibės  $A$  išskaidymas, kai  $n = 3$ .

### **Sajungos ir sankirtos operacijų apibendrinimas**

Aukščiau sajunga  $A \cup B$  ir sankirta  $A \cap B$  buvo apibrėžtos dviems aibėms  $A$  ir  $B$ . Šias sąvokas praplėsime bet kokiam aibių kiekiui  $n$ . Tarkim, kad aibės  $A_i, i = \overline{1, n}$  yra universalios aibės  $U$  poaibiai.

**Aibių sajunga.** Aibių  $A_1, A_2, \dots, A_n$  sajunga susideda iš elementų, priklausančių bent vienai aibei  $A_i, i = \overline{1, n}$ , ir žymima  $\bigcup_{i=1}^n A_i$ :

$$\bigcup_{i=1}^n A_i = A_1 \cup A_2 \cup \dots \cup A_n = \{x \mid x \in A_1 \vee x \in A_2 \vee \dots \vee x \in A_n\}.$$

**Aibių sankirta.** Aibių  $A_1, A_2, \dots, A_n$  sankirta susideda iš elementų, priklausančių visoms aibėms  $A_i, i = \overline{1, n}$ , ir žymima  $\bigcap_{i=1}^n A_i$ :

$$\bigcap_{i=1}^n A_i = A_1 \cap A_2 \cap A_3 \dots \cap A_n = \{x \mid x \in A_1 \wedge x \in A_2 \wedge \dots \wedge x \in A_n\}.$$

**Teorema.** Duotos aibės  $A_1, A_2, \dots, A_n$ . Tuomet

- 1) jei  $A_1 \subseteq A_2 \subseteq A_3 \subseteq \dots$ , tai  $\bigcap_{n=1}^{\infty} A_n = A_1$ ,
- 2) jei  $A_1 \supseteq A_2 \supseteq A_3 \supseteq \dots$ , tai  $\bigcup_{i=1}^{\infty} A_i = A_1$ .

### Aibių algebro tapatybės

Tarkime, kad aibės  $A, B$ , ir  $C$  yra universaliosios aibės  $U$  poaibiai. Tada sajungos, sankirtos ir papildymo operacijos tenkina žemiau išvardintas tapatybes (dėsnius):

1. Komutatyvumo dėsnį:
  - a)  $A \cup B = B \cup A$ ,
  - b)  $A \cap B = B \cap A$ .
2. Asociatyvumo dėsnį:
  - a)  $A \cup (B \cup C) = (A \cup B) \cup C$ ,
  - b)  $A \cap (B \cap C) = (A \cap B) \cap C$ .
3. Distributyvumo dėsnį:
  - a)  $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ ,
  - b)  $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ .
4. De Morgano dėsniai
  - a)  $(A \cap B)' = A' \cup B'$ ,
  - b)  $(A \cup B)' = A' \cap B'$ .

**Teorema** (Išplėstinis De Morgano dėsnis). Tegu  $A = \{A_i \mid i \in I, I \neq \emptyset\}$ .

Tuomet

$$\left( \bigcup_{i \in I} A_i \right)' = \bigcap_{i \in I} A_i',$$

$$\left( \bigcap_{i \in I} A_i \right)' = \bigcup_{i \in I} A_i'$$

5.

- a)  $(A')' = A$ ,
- b)  $A \cup A' = U$ ,
- c)  $A \cap A' = \emptyset$ .

6. Apibendrintuosius De Morgano dėsnius:

- a)  $A \setminus (B \cap C) = (A \setminus B) \cup (A \setminus C)$ ,
- b)  $A \setminus (B \cup C) = (A \setminus B) \cap (A \setminus C)$ .

Čia nepateikiame šių dėsniių formalų įrodyti. Juos nesunku patikrinti naudojant Veno diagramas.

### ***Uždaviniai***

1. Duotos aibės:  $U = Z$ ,  $A = \{..., -4, -2, 0, 2, 4, ...\}$ ,

$B = \{..., -6, -3, 0, 3, 6, ...\}$ ,  $C = \{..., -8, -4, 0, 4, 8, ...\}$ . Rasti:

- a)  $A \cap B$ ,
- b)  $A - C$ ,
- c)  $C - A$ ,
- d)  $(A \cup B) \cap C$ .

Ats: a)  $\{..., -12, -6, 0, 6, 12, ...\}$ , b)  $\{..., -6, -2, 2, 6, ...\}$ , c)  $\emptyset$ , d)  $C$ .

2. Tegul  $A = \{x, y\}$ ,  $B = \{0, 1\}$  ir  $C = \{-1, 0, 1\}$ . Rasti šias aibės:

- a)  $A \times B$ ,
- b)  $B \times C$ ,
- c)  $A \times B \times C$ ,
- d)  $(A \times B) \times C$ .

Ats: a)  $\{(x, 0), (x, 1), (y, 0), (y, 1)\}$ , b)  $\{(0, -1), (0, 0), (0, 1), (1, -1), (1, 0), (1, 1)\}$ ,  
c)  $\{(x, 0, -1), (x, 0, 0), (x, 0, 1), (x, 1, -1), (x, 1, 0), (x, 1, 1), (y, 0, -1), (y, 0, 0), (y, 0, 1),$   
 $(y, 1, -1), (y, 1, 0), (y, 1, 1)\}$ , d)  $\{(x, 0, -1), (x, 0, 0), (x, 0, 1), (x, 1, -1), (x, 1, 0),$   
 $(x, 1, 1), (y, 0, -1), (y, 0, 0), (y, 0, 1), (y, 1, -1), (y, 1, 0), (y, 1, 1)\}$ .

3. Tegul  $A$  ir  $B$  yra universalios aibės  $U$  netušti poaibiai.

- a) Kokiomis sąlygomis esant  $A \times B = B \times A$ ?
- b) Kokiomis sąlygomis esant aibė  $(A \times B) \cap (B \times A)$  yra tuščioji?

Ats: a)  $A = B$ , b)  $A \cap B = \emptyset$ .

## 1.6. Aibės galia ir jos apskaičiavimas

**Aibės galia.** Baigtinės aibės  $A$  galia vadinamas jos elementų skaičius  $n(A)$ . Kitaip aibės galia žymima  $|A|$ .

Tuščioji aibė  $\emptyset$  yra vienintelė aibė, kurios elementų skaičius lygus nuliui.

**Nesusikertančios aibės.** Dvi aibės  $A$  ir  $B$  vadinamos nesusikertančiomis, jeigu  $A \cap B = \emptyset$ . Apibendrinus, jei  $A_1, A_2, \dots, A_n$  yra aibės ir  $A_i \cap A_j = \emptyset$  visiems  $i$  ir  $j$ ,  $1 \leq i < j \leq n$ , tai sakome, kad šios aibės yra poromis nesusikertančios. Taip pat sakome, kad  $\{A_1, A_2, \dots, A_n\}$  yra poromis nesusikertančių aibių rinkinys.

### Pavyzdys

Tegul turime aibes  $A = \{-1, 2, 3, 5, 8\}$ ,  $B = \{1, 3, 5, 7\}$ ,  $C = \{-1, 2, 4, 9\}$ ,  $D = \{-2, 0, 6\}$  ir  $E = \{-3, 8\}$ . Tada aibės  $A$  ir  $D$  yra nesusikertančios, o  $\{B, C, D, E\}$  yra poromis nesusikertančių aibių rinkinys.

Pastebėsime, kad aibės  $A$  ir  $B$  šiame pavyzdyje nėra nesusikertančios, bet  $A - B = \{-1, 2, 8\}$  ir  $A \cap B = \{3, 5\}$  yra nesusikertančios ir kad  $A = (A - B) \cup (A \cap B)$ .

**Adityvumo dėsnis.** Jei aibės  $A$  ir  $B$  yra nesusikertančios, tai

$$n(A \cup B) = n(A) + n(B).$$

Apibendrinus, jei  $A_1, A_2, \dots, A_k$  yra poromis nesusikertančios aibės, tai

$$n(A_1 \cup A_2 \cup \dots \cup A_k) = n(A_1) + n(A_2) + \dots + n(A_k).$$

Taip pat pastebėsime, kad

$$A \cap \emptyset = \emptyset$$

ir

$$A \cap U = A$$

bet kuriam universaliosios aibės  $U$  poaibiui  $A$ .

**1 teorema.** Bet kuriems universaliosios aibės  $U$  poaibiams  $A$  ir  $B$  aibės  $A - B$ ,  $B - A$  ir  $A \cap B$  yra poromis nesusikertančios ir

$$A \cup B = (A - B) \cup (B - A) \cup (A \cap B).$$

**2 teorema.** Bet kuriems universaliosios aibės  $U$  poaibiams  $A$  ir  $B$

$$n(A \cup B) = n(A) + n(B) - n(A \cap B).$$

**Irodymas.** Iš formulės  $n(A) = n(A - B) + n(A \cap B)$  gauname, kad

$$n(A - B) = n(A) - n(A \cap B).$$

Analogiškai,

$$n(B - A) = n(B) - n(A \cap B).$$

Pasinaudojė šiomis formulėmis ir aukščiau pateikta teorema, gauname, kad

$$\begin{aligned} n(A \cup B) &= n(A - B) + n(B - A) + n(A \cap B) = \\ &= (n(A) - n(A \cap B)) + (n(B) - n(A \cap B)) + n(A \cap B) = \\ &= n(A) + n(B) - n(A \cap B). \end{aligned}$$

### **Pavyzdys**

Gaminys gali turėti dvię tipų defektus: 1-ojo ir 2-ojo. Buvo patikrinta 100 gaminiių ir nustatyta, kad penkiolika gaminiių turėjo 1-ojo tipo defektą, dešimt – 2-ojo tipo defektą, o septyni – abiejų tipų defektus. Kiek gaminiių buvo brokuoti iš viso?

Tegul  $A$  yra aibė gaminiių, turinčių 1-ojo tipo defektą,  $B$  – aibė gaminiių, turinčių 2-ojo tipo defektą. Duota, kad  $n(A) = 15$ ,  $n(B) = 10$  ir  $n(A \cap B) = 7$ . Mums reikia suskaičiuoti  $n(A \cup B)$ , t.y. gaminiių, turinčių abiejų tipų defektus, skaičių. Pasinaudojė teorema, gauname:

$$\begin{aligned} n(A \cup B) &= n(A) + n(B) - n(A \cap B) = \\ &= 15 + 10 - 7 = 18. \end{aligned}$$

Taigi, brokuotų gaminiių skaičius – 18.

**3 teorema.** Tegul  $A$ ,  $B$  ir  $C$  yra baigtiniai universalios aibės  $U$  poaibiai.

Tada

$$\begin{aligned} n(A \cup B \cup C) &= n(A) + n(B) + n(C) - n(A \cap B) - n(A \cap C) - \\ &\quad - n(B \cap C) + n(A \cap B \cap C). \end{aligned}$$

Pasinaudojė aukščiau pateikta teorema, gaume:

$$\begin{aligned} n(A \cup B \cup C) &= n(A \cup (B \cup C)) = \\ &= n(A) + n(B \cup C) - n(A \cap (B \cup C)) = \\ &= n(A) + (n(B) + n(C) - n(B \cap C)) - \\ &\quad - n((A \cap B) \cup (A \cap C)) = \\ &= n(A) + n(B) + n(C) - n(B \cap C) - \\ &\quad - (n(A \cap B) + n(A \cap C) - n((A \cap B) \cap (A \cap C))) = \\ &= n(A) + n(B) + n(C) - n(A \cap B) - n(A \cap C) - \\ &\quad - n(B \cap C) + n(A \cap B \cap C). \end{aligned}$$

### **Pavyzdys [PS85]**

Draudimo kompanija skirsto klijentus pagal amžių, lytį ir šeimyninę padėtį. Ištyrus 500 apdraustujų duomenis, nustatyta, kad:  
350 – susituokę,

240 – vyresni negu 25 metų,  
 230 – susituokę vyrai,  
 110 – susituokę, vyresni negu 25 metų,  
 100 – vyrai, vyresni negu 25 metų,  
 40 – susituokę vyrai, vyresni negu 25 metų,  
 10 – vienišos moterys, vyresnės negu 25 metų.  
 Rasti apsidraudusiu vyrų skaičių.

Tegul  $A$  – apdraustujų vyrų aibė,  $B$  – susituokusių apdraustujų aibė,  $C$  – vyresnių negu 25 metai apdraustujų aibė. Duota, kad  $n(A' \cap B' \cap C') = 10$ , todėl

$$\begin{aligned} n(A \cup B \cup C) &= n[(A' \cap B' \cap C')'] = n(U) - n(A' \cap B' \cap C') = \\ &= 500 - 10 = 490. \end{aligned}$$

Iš 3-iosios teoremos turime:

$$\begin{aligned} n(A \cup B \cup C) &= n(A) + n(B) + n(C) - n(A \cap B) - n(A \cap C) - \\ &\quad - n(B \cap C) + n(A \cap B \cap C). \end{aligned}$$

Todėl

$$\begin{aligned} 490 &= n(A) + 350 + 240 - 230 - 100 - 110 + 40, \\ n(A) &= 300. \end{aligned}$$

### ***Uždaviniai***

1. Buvo patikrinta 100 gaminiai. Dešimt iš jų turėjo  $A$  tipo defektą, aštuoni –  $B$  tipo defektą ir 85 gaminiai buvo be defektų. Keli gaminiai iš 100 turėjo abiejų tipų defektus?
2. Informatikos fakultete mokosi 225 baigamojo kurso studentai. 100 studentų pasirinko modulį CS260, 75 – modulį CS360, 50 – modulį CS450, septyni studentai pasirinko abu modulius – CS260 ir CS360, keturi – CS260 ir CS450, trisdešimt vienas – CS360 ir CS450 ir vienas studentas pasirinko visus tris modulius. Kiek studentų nepasirinko nė vieno iš šių trijų modulių?

Ats: 1) 3, 2) 41.

## **2. Grafų teorija**

### **2.1. Įvadas**

Pirmasis grafų teorijos uždavinys, nagrinėtas 1736 m., buvo uždavinys apie Karaliaučiaus (Kionigsbergo) tiltus. L. Oileris ne tik sėkmingai išsprendė šį uždavinį, bet ir suformulavo būtiną ir pakankamas sąlygas, kurias tenkinant grafas turi specialų maršrutą, kuris dabar vadinamas Oilerio ciklu (grandine). Tačiau maždaug 100 metų laikotarpyje šio uždavinio sprendinys buvo vienintelis grafų teorijos rezultatas. Vėliau, 19 amžiaus viduryje, inžinierius elektrikas Kirchhofas išvystė medžių teoriją ir ją taikė elektrinėms grandinėms nagrinėti. Maždaug tuo pačiu laikotarpiu matematikas A. Keli (A. Cayley) trijų tipų medžius naudojo, norėdamas apskaičiuoti organinių junginių izomerų skaičių.

Žymus impulsas vystyti grafų teoriją buvo 1852 m. A. De Morgano iškelta keturių spalvų hipotezė. Ši hipotezė grafų teorijos vystymesi suvaidino analogišką vaidmenį, kaip didžioji Ferma teorema skaičių teorijoje. Keturių spalvų hipotezė buvo įrodyta 1976 m. (žr. K. Appel, W. Haken, Every planar map is four colorable.- Bulletin of the American Mathematical society, vol.82, №5, September, 1976, 711-712p.p.). Įdomi detalė yra ta, kad kai kurie grafų dažymo variantai buvo pilnai perrinkti ir tam sunaudota 1200 kompiuterinio laiko valandų.

Visi sutaria, kad grafų teorijos, kaip savarankiškos matematikos šakos, gimimas yra 1936 m., kai matematikas D. Kionigas išleido monografiją "Baiginių ir begalinį grafų teoriją" (žr. König D. Theorie der Endlichen und Unendlichen Graphen, Leipzig, 1936.). Jis pirmasis vietoje įvairiuose moksluose naudojamų skirtinguų schemų pavadinimų: sociogramos (psichologija), simpleksai (topologija), grandinės (fizika), diagramos (ekonomika), ryšių tinklai, vandentiekio tinklai, geneologiniai medžiai ir t.t., pasiūlė naudoti vieną terminą "grafas". Šis faktas rodo, kad grafai yra įvairiausios fizinės prigimties reiškinį matematinių modeliai. Tai ir lemia grafų teorijos, kaip savarankiškos matematikos šakos, audringą vystymąsi ir plačias taikymo galimybes.

Tuo būdu, grafų teorija, prasidėjusi nuo įdomių uždavinių ir galvosūkių nagrinėjimo (uždaviniai apie šachmatų valdovę (žirgus), 15-kos mergaičių uždavinys, sargybinių uždavinys, penkių valdovių uždavinys ir kt.) šiuo metu tampa pagrindine diskrečiosios matematikos kalba, leidžiančia taikyti diskrečiosios matematikos metodus įvairose mokslo ir technikos srityse.

## 2.2. Pradinės sąvokos

Kaip buvo minėta, pirmasis grafo terminą įvedė matematikas D. Kionigas 1936 m.

**Grafo apibrėžimas.** Formaliai grafą apibrėžti galima dviem būdais.

- Pirmas būdas.** (K. Beržas (C. Berge, Theory des Graphes et ses applications. Paris, 1958. Yra vertimas į rusų kalbą К. Берж. Теория графов и ее применения, И.Л.,Москва, 1962)).

Sakoma, kad grafas žinomas, jeigu:

- duota netuščia aibė  $V(V \neq \emptyset)$ ,
- duotas aibės  $V$  atvaizdis  $\Gamma$  į aibę  $V$ .

Grafas žymimas simboliu  $G=(V, \Gamma)$ .

- Antras būdas.** Tarkime  $V$  - netuščioji aibė, o aibė  $E$  yra aibės  $V$  visų galimų dvielemenčių poaibių aibė, t.y.  $E=\{\{x,y\}:x,y \in V \wedge x \neq y\}$ . Tada grafas yra pora  $(V, U)$ , čia  $U \subseteq E$ . Žymime  $G=(V, U)$ .

Aibė  $V$  vadinama grafo **viršunių aibe**. Aibės  $V$  elementų skaičius yra lygus grafo viršunių skaičiui ir dažnai vadinamas grafo **eile**. Tolesniame nagrinėjime laikysime, kad  $|V|=n$ . Aibė  $U$  neorientuotojo grafo atveju vadinama grafo **briaunų aibe**, o orientuotojo grafo (orgrafo) atveju – **lankų aibe**. Tuo būdu viršunių  $v_1$  ir  $v_2$  pora  $(v_1, v_2)$  sudaro arba briauną (žr. 2.2.1 pav. a) arba lanką (žr. 2.2.1 pav. b)) Vadinasi lankas yra sutvarkytoji viršunių pora (sutvarkytasis dvielementis poaibis), tuo būdu lankų aibė yra  $V \times V$  poaibis, t.y.  $E=V \times V$ . Tolesniame nagrinėjime laikysime, kad  $|U|=m$ , o apie grafą  $G$ , turintį  $n$  viršunių ir  $m$  briaunų (lankų) sakysime, kad  $G$  yra  $(n, m)$ -grafas.



2.2.1 pav. Grafo briauna ir lankas

Briauną  $(v_1, v_2)$  ribojančios viršūnės  $v_1$  ir  $v_2$  vadinamos briaunos galais, o lanko atveju, viršūnė  $v_1$  yra lanko pradžia, o  $v_2$  – lanko pabaiga (galas), arba  $v_1$  ir  $v_2$  yra lanko kraštinių viršūnės.

Briaunos galų viršūnės vadinamos **gretimomis viršūnėmis**.

Dvi **briaunas** yra **gretimos**, jei jos turi bendrą galą. Orientuotojo grafo viršūnės yra gretimos, jei jos yra kraštinių lanko viršūnės, o lankai yra gretimi, jei jie turi bendrą kraštine višūnę.

**Pastaba.** Užrašant grafą pav. gretumo matrica, gretumo struktūra ar lankų ir jų adresų masyvais (žr. 2.7. paragrafą), orientuotojo grafo gretimų

viršunių savoką patogiau apibrėžti taip: lanko  $(v_1, v_2)$  atveju viršūnė  $v_2$  yra gretima viršūnei  $v_1$ .

Jei turime briauną (lanką)  $(v_1, v_2)$ , tai sakome, kad **viršūnė**  $v_1(v_2)$  **incidentiška briaunai (lankui)**  $(v_1, v_2)$  ir atvirkščiai.

**Pastaba.** Apskritai, žodis **gretimas** naudojamas tarp tos pačios rūšies objektų, o - **incidentiškas** tarp skirtingos rūšies objektų.

Grafą patogu vaizduoti paveikslėliais, susidedančiais iš taškų ir linijų. Plokštumos taškai vaizduoja grafo viršunes. Viršūnės plokštumoje išdėstomos laisvai: svarbu ne viršunių taškų koordinatės, bet santykis tarp viršunių. Dvi viršūnės  $v_1$  ir  $v_2$  jungiamos briauna arba lanku, jei jos yra gretimos.

Jei grafas turi tik **briaunas**, tai jis vadinas **neorientuotuoju** grafu. (žr. 2.2.2 a), b), e), g), h), i), j) pav.).

Jei grafas turi tik **lankus**, tai jis vadinas **orientuotuoju** grafu (**orgrafu**). (žr. 2.2.2 c) pav.).

Jei grafas turi ir briaunų ir lankų, tai jis vadinas **mišriuoju** grafu. (žr. 2.2.2 k) pav.).

Jei grafas turi bent vieną viršunių porą, kuri jungiama keliomis briaunomis, tai grafas vadinas **multigrafu**. (žr. 2.2.2 g) ir h) pav.).

Jei kiekviena grafo viršūnė jungiama briaunomis su visomis likusiomis viršūnėmis, tai toks grafas yra **pilnasis** grafas. Pilnasis  $n$  viršūninis grafas žymimas  $K_n$ . (žr. 2.2.2 d) ir e) pav.).

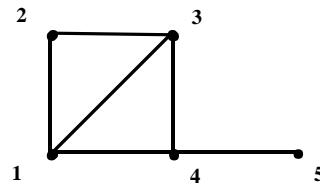
Grafas, kurio briaunų (lankų) aibė yra tuščioji aibė, vadinas tuščiuoju grafu. Tuščiasis  $n$  viršūninis grafas žymimas  $O_n$ . (žr. 2.2.2 f) pav.).

Grafas, kurio viršunių aibę galima išskaidyti į du poaibius  $A$  ir  $B$  taip, kad kiekvienos briaunos galai priklausytų skirtiniems poaibiams, vadinas dvidaliu grafu. (žr. 2.2.2 i) ir j) pav.). Simboliu  $K_{S,T}$  žymime pilnajį dvidalių grafą, t.y. grafą, kai kiekviena aibės  $A$  viršūnė jungiama briauna su visomis aibės  $B$  viršūnėmis. (žr. 2.2.2 j) pav.).

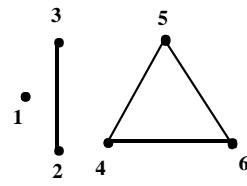
**Viršūnės laipsnis.** Viršūnės  $v$  laipsnis, tai skaičius viršunių gretimų viršūnei  $v$ . Viršūnės laipsnį žymėsime  $\rho(v)$  arba  $d(v)$ .

Aibė viršunių, gretimų viršūnei  $v$ , žymima  $N(v)$  ir vadina viršūnės aplinka. Aišku, kad  $\rho(v)=|N(v)|$ .

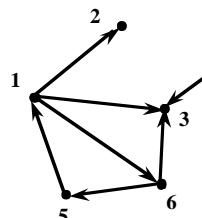
Orientuotojo grafo atveju skirtimi viršūnės **jėjimo ir išėjimo puslaipsniai**. **Jėjimo puslaipsnis**, tai skaičius lankų, įeinančių į viršūnę, o **išėjimo puslaipsnis**, tai skaičius lankų, išeinančių iš viršūnės.



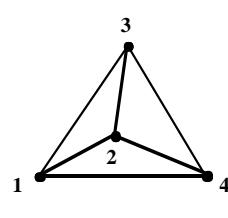
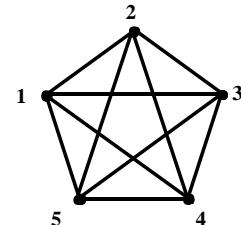
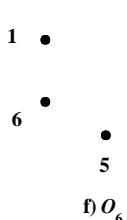
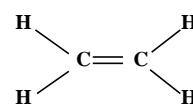
a)



b)



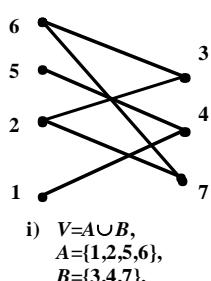
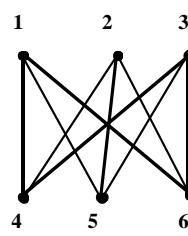
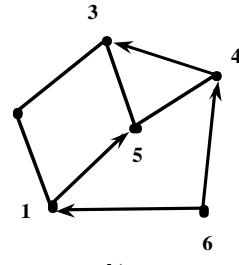
c)

d)  $K_4$ e)  $K_5$ f)  $O_6$ 

g) etilenas



k) acitilenas

i)  $V=A \cup B$ ,  
 $A=\{1,2,5,6\}$ ,  
 $B=\{3,4,7\}$ .j)  $K_{3,3}$ 

k)

**2.2.2 pav.** Grafų pavyzdžiai

Seka  $\rho(v_1), \rho(v_2), \dots, \rho(v_n)$  vadinama **grafo viršūnių laipsnių** seka. Yra paprastas ryšys tarp grafo briaunų skaičiaus  $m$  ir jo viršūnių laipsnių:  $m = \frac{1}{2} \sum_{v \in V} \rho(v)$ , t.y. grafo briaunų skaičius yra lygus visų jo viršūnių laipsnių sumos pusei. Iš čia išplaukia išvada, kad nelyginio laipsnio viršūnių skaičius yra lyginis.

Pilnojo grafo visų viršūnių laipsniai yra lygūs ir lygūs ( $n-1$ ). Vadinasi, pilnojo grafo briaunų skaičius  $m = \frac{n(n-1)}{2}$ .

Kiekvienos viršūnės laipsnis tenkina nelygybę  $0 \leq \rho(v) \leq n-1$ .

Jei grafo visų viršūnių laipsniai yra lygūs, tai grafas vadinamas **reguliariuoju grafu**.

Aptarsime **grandinės** ir **ciklo** bei **kelio** ir **kontūro** sąvokas. Pirmosios dvi sąvokos siejamos su neorientuotaisiais grafais, o kitos dvi – su orientuotaisiais grafais.

Seka briaunų  $(v_1, v_2), (v_2, v_3), (v_3, v_4), \dots, (v_{k-1}, v_k)$  t.y. gretimų briaunų seka, vadinama **grandine**. Grandinę patogu apibrėžti viršūnių, per kurias ji eina, seka. Pavyzdžiui,  $\mu = (v_1, v_2, v_3, v_4, \dots, v_{k-1}, v_k)$ . Jei grandinės pirmoji ir paskutinė viršūnės sutampa, tai tokia grandinė vadinama **ciklu**. Grandinė (ciklas) vadinama **paprastaja grandine (paprastuoju ciklu)**, jei visos ją sudarančios briaunos yra skirtingos.

Grandinė ciklas vadinama **elementarija**, jei ji eina per skirtingas viršūnes.

Neorientuotojo grafo grandinės (ciklo) sąvokoms yra analogiškos orografo **kelio** ir **kontūro** sąvokos. Apibrėžiant kelią (kontūrą) žodį “briauna” reikia keisti žodžiu “lankas”. Tolesniame dėstyme, kur tai nesukels painiavos, mes dažniau naudosime kelio ir ciklo sąvokas, kalbëdami tiek apie neorientuotuosius, tiek apie orientuotuosius grafas.

Grandinės (ciklo) ilgis, tai skaičius briaunų, sudarančių grandinę (ciklą). Grandinės ilgi žymësime  $l(\mu)$  arba  $l(v_1, v_k)$ , čia  $v_1$  ir  $v_k$  atitinkamai pradinė ir galinė grandinės viršūnės.

Tolesniame dėstyme naudosime sąvoką: “*is viršūnės a nukeliauti į viršūnę b*”. Tai reiškia, kad egzistuoja bent viena grandinė (kelias), jungianti tas viršūnes.

Aptarkime **dalinio grafo** ir **pografinio** sąvokas.

Grafo  $G=(V, U)$  dalinis grafas, tai grafas, turintis tą pačią viršūnių aibę ir dalį pradinio grafo briaunų (lankų). Kalbant tiksliau,  $H=(V, U_H)$  yra grafo  $G=(V, U)$  dalinis grafas, jei  $U_H \subseteq U$ .

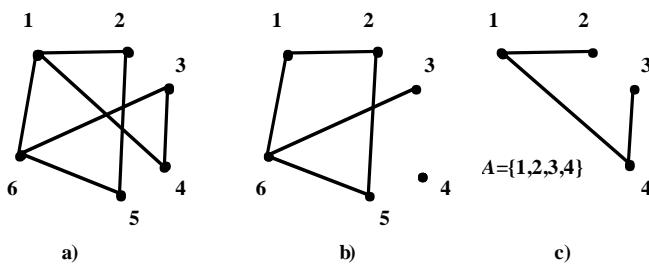
Tarkime  $A$  yra grafo  $G$  viršūnių aibės  $V$  poaibis. Tada grafas  $P=(A, B)$  yra **pografinis**, kurį indukuoja viršūnių aibė  $A$  (**indukuotas pografis**), jei jo

viršunių aibė sutampa su aibe  $A$ , o briaunų (lankų) aibę  $B$  sudaro tos grafo  $G$  briaunos (lankai), kurių abu galai priklauso aibei  $A$ . Indukuotojo pografio dalinis grafas vadinamas **pografiu**.

Tolesniame dėstyme naudosime savoką pografis, suprastami (jei tai nebus atskirai paaiškinta), kad tai indukuotasis pografis.

Pavyzdžiuui, jei Lietuvos kelių žemėlapis yra grafas  $G$ , tai pagrindinių kelių žemėlapis yra dalinis grafas, o Žemaitijos kelių žemėlapis yra pografis.

Paveiksle 2.2.3 a) pav. pavaizduotas grafas  $G$ , paveiksle 2.2.3 b) pav. jo dalinis grafas, o paveiksle 2.2.3 c) pav. pografis, kurį indukuoja viršunių aibė  $A=\{1,2,3,4\}$ .



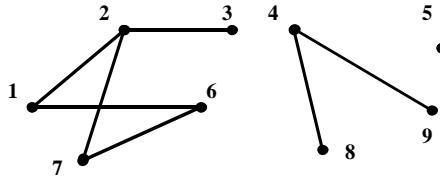
**2.2.3 pav.** Grafas, dalinis grafas ir pografis

Aptarkime neorientuotojo grafo **jungiosios komponentės** savoką.

**Pastaba.** Kadangi tolesniame nagrinėjime daugiausia kalbėsime apie neorientuotuosius grafus, tai naudodamai žodį "grafas", (jei nebus pabrėžta atskirai), suprasime kaip neorientuotasis grafas.

Grafo  $G=(V,U)$  **jungioji komponentė**, tai pografis, kurį indukuoja aibė  $A$ , sudaryta iš bet kurios grafo  $G$  viršūnės  $v$  ir visų tų viršunių, į kurias galima nukeliauti iš viršūnės  $v$ , t.y.  $A=\{v\}\cup\{\text{viršūnės, į kurias galima nukeliauti iš viršūnės } v\}$ .

Pavyzdžiuui, 2.2.4 pav. pavaizduotas grafas, turintis tris jungiamias komponentes. Pirmają komponentę sudaro pografis, kurį indukuoja viršunių aibė  $\{1,2,3,6,7\}$ , antrają – pografis, kurį indukuoja viršunių aibė  $\{4,8,9\}$  ir trečiąją – pografis, indukuotas viršunių aibės  $\{5\}$ .



**2.2.4 pav.** Jungiosios komponentės

*Orientuotojo* grafo atveju naudojamos sąvokos: *stipriai jungus* grafas (*stiprusis* grafas), *vienakrytiškai jungus* grafas ir *silpnai jungus* grafas (*silpnasis* grafas).

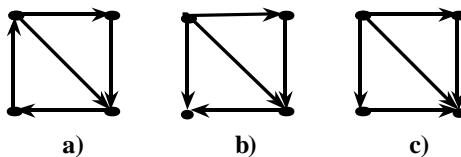
Jei orientuotame grafe yra kelias, jungiantis viršūnę  $x$  su viršūne  $y$ , tai sakome, kad viršūnė  $y$  yra pasiekiamoji viršūnė iš viršūnės  $x$ .

*Orientuotasis* grafas  $G$  yra *stipriai jungus*, jeigu bet kokios dvi viršūnės  $x$  ir  $y$  yra pasiekiamos viena iš kitos. Kitaip tariant, iš bet kurios viršūnės  $x$  galime nukeliauti į bet kurią viršūnę  $y$  ir atvirkščiai.

*Orientuotasis* grafas yra *vienakrytiškai jungus*, jeigu bet kokiai porai viršūnių  $x$  ir  $y$ , jos yra pasiekiamos bent viena kryptimi, t.y. arba  $y$  pasiekiamā iš viršūnės  $x$ , arba  $x$  pasiekiamā iš viršūnės  $y$ .

*Orientuotasis* grafas yra *silpnai jungus*, jei yra jungus neorientuotasis grafas gautas iš orientuotojo, pakeitus lankus briaunomis.

2.2.5 a), b) ir c) pav. atitinkamai pavaizduotas stipriai jungus grafas, vienakrytiškai jungus grafas ir silpnai jungus grafas.



**2.2.5 pav.** Orientuotojo grafo jungumas

Plačiau apie jungumą žr. 2.16 paragrade.

### *Uždaviniai*

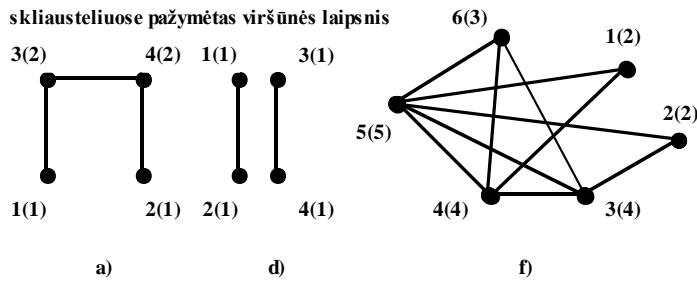
1. Nupiešti, jei galima grafus, (ne multigrafus ir be kilpų), pagal duotas jų

viršūnių laipsnių sekas. Jei nupiešti negalima, paaiškinti, kodėl.

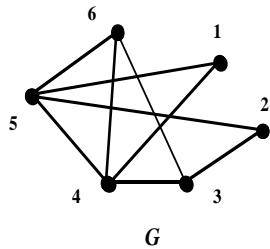
- |              |              |                  |
|--------------|--------------|------------------|
| a) (1,1,2,2) | c) (4,3,2,1) | e) (3,5,5,2,2,1) |
| b) (2,2,1,2) | d) (1,1,1,1) | f) (2,2,4,4,5,3) |

a t s a k y m a s

- b) nupiešti negalima, nes čia  $\sum_{i=1}^n \rho(i) = \sum_{i=1}^4 \rho(i) = 2 + 2 + 2 + 1 = 7$  - nelyginis skaičius;
- c) nupiešti negalima, nes čia  $\rho(i) > n - 1$ , t.y.  $n=4$ , o  $\rho(4) = 4$ ;
- e)  $\rho(2)=\rho(3)=5$  reiškia, kad šiame 6 viršunių grafe turėtų būti dvi viršūnės gretimos visoms grafo viršūnėms, o  $\rho(6)=1$  reiškia, kad viena viršūnė turėtų būti gretima tik vienai viršūnei. Akivaizdu, tame pačiam grafe tai neįmanoma.



2. Duotas grafas  $G$ :

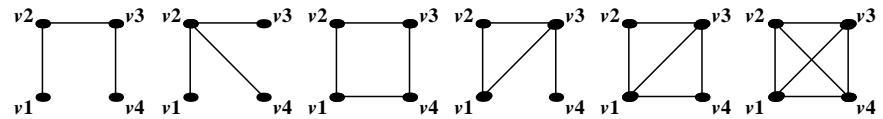


B. Nustatyti, ar duotasis grafas turi:

- d) dvidalį indukuotąjį pografi  $K_{2,2}$ ; jei taip, pavaizduoti jį;
- e) elementariųjų paprastųjų ilgio 3,4,5 ir 6 ciklų; jei taip, užrašyti šiuos ciklus viršūnių, per kurias jie eina, sekomis;

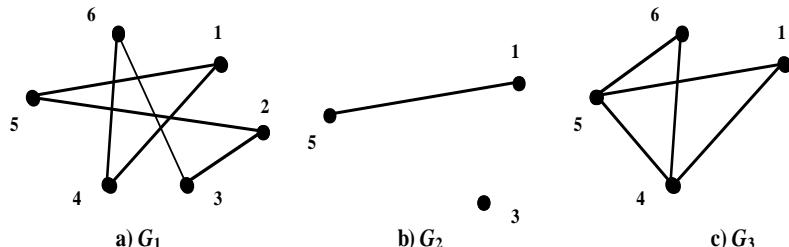
- A. Nupiešti duotajam grafui  $G$ :
- dalinių reguliarųjų grafų  $G_1$ , kurio visų viršūnių laipsniai  $\rho_i=2$ ,
  - pografi  $G_2$ , kurį indukuoja viršūnių aibė  $\{1,3,5\}$ ; pasakyti, kiek jungiuju komponentečių jis turi;
  - pografi  $G_3$ , kurį indukuoja viršūnių aibė  $\{1,4,5,6\}$ ; pasakyti, kiek ir kokių briaunų trūksta, kad galėtume jį pavadinti pilnuoju grafu  $K_4$ ;

f) pavaizduotas indukuotuosius pografius:

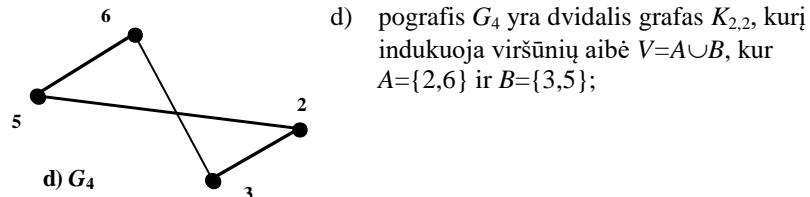


Jei taip, tai kokios viršūnių aibės juos indukuoja.

a t s a k y m a s

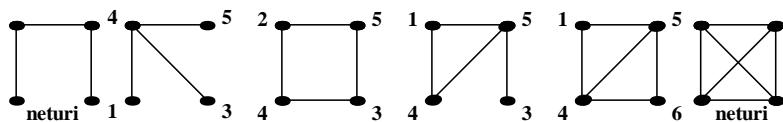


- b) pografas  $G_2$  turi dvi jungiamias komponentes, viršūnių aibė  $\{1,5\}$  priklauso pirmajai jungiajai komponentei, viršūnių aibė  $\{3\}$  – antrajai;
- c) pografas  $G_3$  būtų pilnuoju grafu  $K_4$ , jei jis turėtų dar vieną briauną  $(1,6)$ ;



- d) pografas  $G_4$  yra dvidalis grafas  $K_{2,2}$ , kurį indukuoja viršūnių aibė  $V=A \cup B$ , kur  $A=\{2,6\}$  ir  $B=\{3,5\}$ ;
  - e) duotasis grafas turi tokius elementariuosius paprastuosius ciklus:
- | <u>ilgio 3</u> | <u>ilgio 4</u> | <u>ilgio 5</u> | <u>ilgio 6</u> |
|----------------|----------------|----------------|----------------|
| 1,4,5,1        | 2,3,4,5,2      | 1,4,3,2,5,1    | 1,4,6,3,2,5,1  |
| 3,4,6,3        | 2,3,6,5,2      | 2,3,6,4,5,2    |                |
| 4,5,6,4        | 3,4,5,6,3      |                |                |

f)



### 2.3. Grafo jungiosios komponentės

#### Grafo jungiųjų komponenčių apskaičiavimas

Grafo jungiosios komponentės apibrėžimas yra konstruktyvus, t.y. iš jo išplaukia jungiųjų komponenčių apskaičiavimo algoritmas.

Suformuluokime uždavinį.

Duotas grafas  $G=(V,U)$ , t.y. žinome

- $n$  – grafo viršūnių skaičių,
- $m$  – grafo briaunų skaičių,
- grafo viršūnių aibę  $V$  ir briaunų aibę  $U$ .

**Pastaba.** Konkrečius grafo nusakymo būdus nagrinėsime vėliau.

Rasti grafo jungiasias komponentes t.y. rasti:

- $p$  – jungiųjų komponenčių skaičių,
- masyvą  $S[1..n]$ ; jei  $S_i=k$ , tai reiškia, kad viršūnė  $i$  priklauso  $k$ -tajai jungiajai komponentei.

Pavyzdžiuui, grafui, pavaizduotam 2.2.4 pav.

$$p = 3, \quad S : \begin{array}{cccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ & \hline 1 & 1 & 1 & 2 & 3 & 1 & 1 & 2 & 2 \end{array}$$

#### Pirmasis grafo jungiųjų komponenčių apskaičiavimo algoritmas

Grafo jungiųjų komponenčių apskaičiavimo algoritmą galime užrašyti taip:

```

begin
  p:=0; for i:=1 to n do S[i]:=0;
  {Šie operatoriai reiškia, kad pradžioje jungiųjų komponenčių
  skaičius lygus nuliui. Kadangi visi S[i]=0 , i =  $\overline{1, n}$  , tai reiškia, kad
  visos viršūnės nepriklauso jokiai jungiajai komponentei}
  for i:=1 to n do
    if S[i]=0 then {radome viršūnę nepriklausančią jokiai
                     jungiajai komponentei}
    begin
      p:=p+1; {didiname jungiųjų komponenčių skaičių}
      S[i]:=p;
      Apskaičiuojame aibę  $A$  viršūnių, kurias galima
      nukeliauti iš viršūnės  $i$ . {Tokios aibės
      apskaičiavimo metodai bus nagrinėjami vėliau (žr.
      "Paieška platyn", "Paieška gilyn")}.
      Tarkime, kad
       $A=\{v_1, v_2, \dots, v_t\}$ , tada  $S[v_j]:=p$ ,  $j=1, t$  ;
    end;
  end.

```

### Antrasis grafo jungiųjų komponenčių apskaičiavimo algoritmas

Aptarkime dar vieną jungiųjų komponenčių apskaičiavimo metodą, kuris paprastai reikalauja daugiau veiksmų, nei anksčiau pateiktas algoritmas.

**Metodo idėja.** Pradžioje tarkime, kad grafas yra tuščiasis, t.y. kiekviena jo viršūnė priklauso skirtingai jungiajai komponentei. Dabar nuosekliai, viena po kitos įveskime grafo briaunas. Aišku, kad jei įvedamų grafo briaunų galai priklauso tai pačiai jungiajai komponentei, tai šios briaunos įvedimas nekeičia grafo jungiųjų komponenčių skaičiaus, tačiau, jei įvedamos grafo briaunos galai priklauso skirtingoms jungiosioms komponentėms, tai ši briauna vienetu sumažina grafo jungiųjų komponenčių skaičių. Šiuo atveju grafo jungiųjų komponenčių skaičių sumažiname vienetu ir perskaiciuojame jungiųjų komponenčių masyvo  $S$  komponentes: jei įvedame briauną  $(a,b)$  ir  $S[a] \neq S[b]$ , tai visus masyvo  $S$  elementus, lygius elementui  $S[b]$ , keičiame elementu  $S[a]$ .

Aišku, kad po paskutinės grafo briaunos įvedimo, masyvo  $S$  elementai, atitinkantys grafo viršūnėms, priklausančioms tai pačiai jungiajai komponentei, bus lygūs.

Pavyzdžiu, 2.2.4 pav. pavaizduotam grafui gausime.

$$\text{Pradžioje} \quad p = 9, \quad S : \begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array}$$

Tarkime, kad grafo briaunas įvesime tokia tvarka:  $(2,3), (7,6), (1,2), (2,7), (1,6), (4,9), (4,8)$ .

Įvedus briauną  $(2,3)$ , gausime:

$$p = 8, \quad S : \begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 2 & 4 & 5 & 6 & 7 & 8 & 9 \end{array}$$

Įvedus briauną  $(7,6)$ , gausime:

$$p = 7, \quad S : \begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 2 & 4 & 5 & 7 & 7 & 8 & 9 \end{array}$$

Įvedus briauną  $(1,2)$ , gausime:

$$p = 6, \quad S : \begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 1 & 1 & 4 & 5 & 7 & 7 & 8 & 9 \end{array}$$

Įvedus briauną  $(2,7)$ , gausime:

$$p = 5, \quad S : \begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 1 & 1 & 4 & 5 & 1 & 1 & 8 & 9 \end{array}$$

Įvedus briauną  $(1,6)$ ,  $p$  ir  $S$  turiniai nepasikeis, nes  $S_1 = S_6$ .

Įvedus briauną  $(4,9)$ , gausime:

$$p = 4, \quad S : \begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 1 & 1 & 4 & 5 & 1 & 1 & 8 & 4 \end{array}$$

Pagaliau, įvedus briauną (4,8), gausime:

$$p = 3, \quad S : \begin{array}{cccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \hline & 1 & 1 & 1 & 4 & 5 & 1 & 1 & 4 & 4 \end{array}$$

Gautas rezultatas rodo, kad grafo jungiųjų komponenčių skaičius yra lygus 3, o viršūnės, priklausančios tai pačiai jungajai komponentei, masyve  $S$  pažymėtos tuo pačiu skaičiumi. Tam, kad masyvo  $S$  elementai, žymintys jungiąsias komponentes, būtų natūralieji skaičiai nuo 1 iki  $p$ , masyvo  $S$  elementus pernumeruosime.

Po pernumeravimo gausime:

$$p = 3, \quad S : \begin{array}{cccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \hline & 1 & 1 & 1 & 2 & 3 & 1 & 1 & 2 & 2 \end{array}$$

Dabar, jei  $S_i=k$ , tai, kaip ir anksčiau, reiškia, kad  $i$ -toji viršūnė priklauso  $k$ -tajai jungajai komponentei.

Aptartąjį metodą realizuoja žemiau pateiktas algoritmas.

Tarkime, kad

$n$  – grafo viršūnių skaičius,

$m$  – grafo briaunų skaičius,

$B[1..2,1..m]$  – grafo briaunų matrica;  $(b_{1j}, b_{2j})$ ,  $j = \overline{1, m}$ , -  $j$ -oji grafo briauna,

$p$  – jungiųjų komponenčių skaičius,

$S[1..n]$  – jungiųjų komponenčių masyvas; jei  $S[i]=k$ , tai  $i$ -toji viršūnė priklauso  $k$ -tajai jungajai komponentei.

```

begin
  p:=n;
  for i:=1 to n do S[i]:=i;
  for j:=1 to m do
    begin
      k:=b[1,j]; l:=b[2,j];
      if S[k]<>S[l] then
        begin
          p:=p-1;
          u:=S[k]; v:=S[l];
          for i:=1 to n do
            if S[i]=v then S[i]:=u
        end;
    end;
end;
```

```

{Masyvo S pernumeravimas}
t:=0; for i:=1 to n do temp[i]:=0;
for i:=1 to n do
    if temp[i]=0 then
        begin
            t:=t+1;
            k:=S[i];
            for l:=i to n do
                if S[l]=k then
                    begin
                        temp[l]:=1; S[l]:=t
                    end;
            end;
        end.
end.

```

## 2.4. Metrinės grafo charakteristikos

Nagrinėkime grafą  $G=(V,U)$ , turintį  $n$  viršūnių ir  $m$  briaunų.

**Atstumas** tarp viršūnių  $x$  ir  $y$  yra trumpiausios grandinės, jungiančios tas viršunes, ilgis. Atstumą žymėsime simboliu  $d(x,y)$ .

Aišku, kad taip apibrėžtas atstumas, tenkina tokias metrikos aksiomas:

- 1)  $d(x, y) \geq 0$ ,
- 2)  $d(x, y) = 0$ , tada ir tikta tada, kai  $x = y$ ,
- 3)  $d(x, y) = d(y, x)$ ,
- 4)  $d(x, z) + d(z, y) \geq d(x, y)$  (trikampio nelygybė).

Įvesta atstumo sąvoka leidžia apibrėžti ***k-taji grafo laipsnį***. Tarkime  $G$  – jungusis grafas, o  $k$ - natūralusis skaičius. Tada  $k$ - tasis grafo laipsnis  $G^k$  yra grafas, kurio viršūnių aibė sutampa su grafo  $G$  viršūnių aibe, viršūnės  $u$  ir  $v$  ( $u \neq v$ ) jungiamos briauna, jei  $d(u,v) \leq k$ . Aišku, jei  $k \geq |V|-1$ , tai  $G^k$  – pilnasis grafas.

**Viršūnės  $v$  ekscentricitetas.** Viršūnės  $v$  ekscentricitetas, tai dydis, apskaičiuojamas pagal formulę

$$e(v) = \max_{u \in V} d(v, u),$$

t.y. ilgiausios grandinės nuo viršūnės  $v$  iki likusių grafo viršūnių ilgis.

**Grafo spindulys** – tai skaičius, apibrėžiamas formule

$$r(G) = \min_{v \in V} e(v),$$

t.y. skaičius, lygus mažiausiam viršūnių ekscentricitetui.

**Grafo skersmuo** – tai skaičius, kurį nusako formulė  $d(G) = \max_{v \in V} e(v)$ , t.y. skaičius, lygus didžiausiam viršunių ekscentricitetui.

Skersmens ilgio grandinė, jungianti bet kurias dvi periferines grafo viršunes, vadinama **skersmens grandine**. Viršūnės, kurių ekscentricitetas lygus spinduliu, vadinamos **centru** viršūnėmis. Aibė, kurią sudaro centro viršūnės, vadinama **grafo centru**. Viršūnės, kurių ekscentricitetas lygus skersmeniui, vadinamos **periferinėmis** viršūnėmis.

2.4.1 pav. parodytas besvorinio grafo viršunių ekscentricitetų, grafo skersmens ir spindulio radimas,

Analogiškos sąvokos apibrėžiamos ir **svoriniams grafui**, t.y. grafui, kurio kiekvienai briaunai priskirtas skaičius, vadinamas **briaunos svoriu**. Briaunos svoris gali turėti įvairias prasmes; reikštį briaunos ilgi (atstumą), keliamają galią, pralaidumą ir pan. Kalbant apie svorinio grafo metrines charakteristikas, briaunos svorį traktuosime kaip atstumą tarp gretimų viršunių.

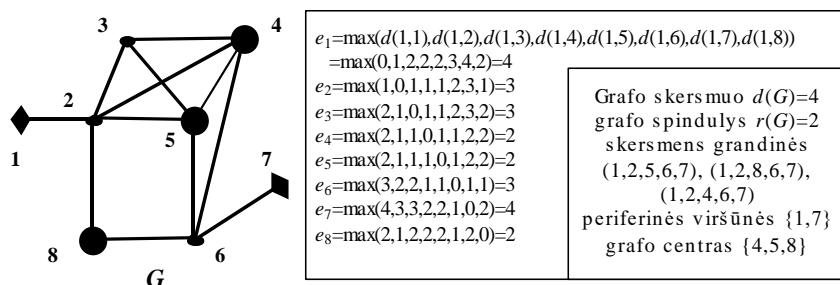
**Grandinės**, jungiančios **svorinio grafo** viršunes  $x$  ir  $y$  ilgis, tai šią grandinę sudarančių briaunų svorių suma.

Atstumas tarp svorinio grafo viršunių  $x$  ir  $y$  (žymėsime  $d(x,y)$ ) yra trumpiausios grandinės, jungiančios šias viršunes, ilgis.

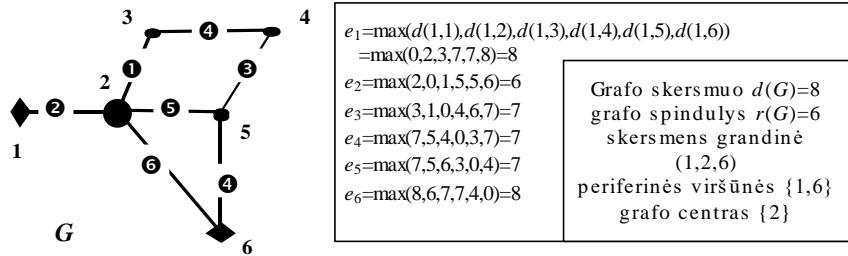
Aišku, kad taip apibrėžus atstumą, jam negalioja 4-toji metrikos aksioma – trikampio nelygybė. Kitos metrinės svorinio grafo sąvokos sutampa su anksčiau apibrėžtomis besvorinio grafo metrinėmis sąvokomis.

2.4.2 pav. parodyta svorinio grafo viršunių ekscentricitetų, jo skersmens, spindulio, centro ir periferinių viršunių radimas.

Grafo metrinės charakteristikų apskaičiavimo uždavinys turi praktinę prasmę. Tarkime, kad grafo viršūnės vaizduoja gyvenvietes, o briaunos - šias gyvenvietes jungiančius kelius. Briaunų svoriai reiškia šių kelių ilgius. Kur reikėtų statyti mokyklą, bažnyčią, ligoninę ir kt., kad atstumas nuo visų gyvenviečių iki šių objektų būtų trumpiausias. Aišku, kad juos reikėtų statyti centro viršūnėse.



2.4.1 pav. Besvorinio grafo metrinės charakteristikos



2.4.2 pav. Svorinio grafo metrinės charakteristikos

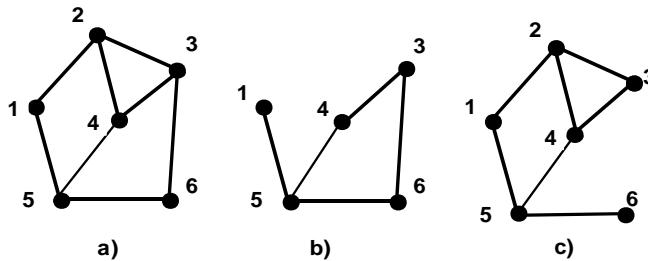
## 2.5. Veiksmai su grafais

Aptarkime veiksmus su grafais.

**Viršūnės šalinimas.** Duotas grafas  $G=(V,U)$ . Pašalinti viršūnę  $x$ , tai iš grafo pašalinti šią viršūnę drauge su jai incidentinėmis briaunomis. Gaunamas grafas  $H=(V_1, U_1)$ , čia  $V_1=V\setminus\{x\}$ , o  $U_1=U\setminus\{\text{briaunos, incidentiškos viršūnei } x\}$ .

**Briaunos ( $v_1, v_2$ ) šalinimas.** Iš grafo  $G=(V,U)$  šalinama briauna  $(v_1, v_2)$ . Gaunamas grafas, turintis tą pačią viršūnių aibę  $V$  ir briaunų aibę  $U^*=U\setminus\{(v_1, v_2)\}$ .

2.5.1 a) pav. pavaizduotas grafas  $G$ ; 2.5.1 b) pav. pavaizduotas grafas, gautas iš grafo  $G$  pašalinus 2-ąją viršūnę; 2.5.1 c) pav. pavaizduotas grafas, gautas iš grafo  $G$  pašalinus (3,6) briauną.



2.5.1 pav. Grafo viršūnės ir briaunos šalinimas

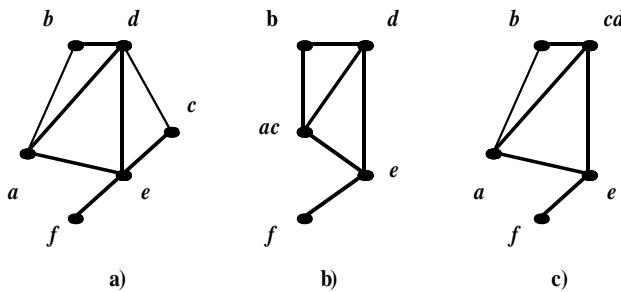
**Viršūnių sutapatinimas.** Tarkime, kad  $v_1$  ir  $v_2$  grafo  $G=(V,U)$  viršūnės. Viršūnių  $v_1$  ir  $v_2$  sutapatinimas atliekamas taip:

- 1) iš grafo  $G$  pašalinamos viršūnės  $v_1$  ir  $v_2$ ;
- 2) įvedama nauja viršūnė  $v$ ;

- 3) viršūnė  $v$  jungiama briaunomis su tomis viršūnėmis, kurios buvo gretimos arba viršūnei  $v_1$ , arba viršūnei  $v_2$ , t.y.

$$N(v)=N(v_1)\cup N(v_2).$$

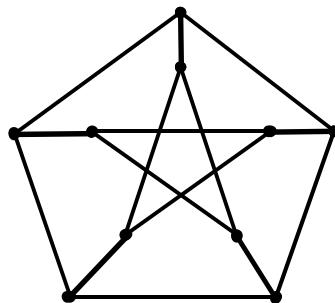
**Briaunos sutraukimas.** Tarkime  $(v_1, v_2)$  yra grafo  $G=(V, U)$  briauna. Tada briaunos  $(v_1, v_2)$  sutraukimas, tai gretimų viršūnių  $v_1$  ir  $v_2$  sutapatinimas. 2.5.2 a), b) ir c) pav. atitinkamai pavaizduotas grafas  $G=(V, U)$ , šio grafo viršūnių  $a$  ir  $c$  sutapatinimas ir šio grafo briaunos  $(c, d)$  sutraukimas.



**2.5.2 pav.** Grafo viršūnių sutapatinimas ir briaunos sutraukimas

Sakoma, kad grafas  $G$  sutraukiamas į grafą  $H$ , jei egzistuoja tokia briaunų seka, kurias nuosekliai sutraukiant iš grafo  $G$  gaunamas grafas  $H$ .

2.5.3 pav. parodytas Peterseno grafas, kurį galima sutraukti į penkių viršūnių pilnaji grafą, t.y. į grafą  $K_5$ . Briaunos, kurias reikia sutraukti, pažymėtos storiau.



**2.5.3 pav.** Peterseno grafas

Aišku, kad bet koks netuščiasis jungasis grafas sutraukiamas į  $K_2$ . Tačiau ne kiekvienas netuščiasis grafas sutraukiamas į  $K_3$ . Pavyzdžiui, grafas pavaizduotas 2.5.4 pav. (t.y. grandinė), nesutraukiamas į  $K_3$ .



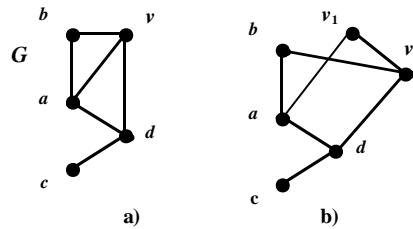
**2.5.4 pav.** Grafas, kurio negalima sutraukti iš  $K_3$

Tuo būdu natūraliai kyla klausymas, - iš kokį didžiausios eilės pilnajį grafą galima sutraukti duotąjį grafą  $G$ . Tokio pilnojo grafo eilė žymima  $\eta(G)$  ir vadinama Hadvigerio skaičiumi. Pavyzdžiui, Peterseno grafo Hadvigerio skaičius yra 5. Hadvigerio skaičius yra susijęs su keturių spalvų hipoteze.

**Viršūnės išskaidymo operacija.** Operacija, dualinė briaunos sutraukimui, yra viršūnės išskaidymo operacija. Tarkime  $v$  yra viena iš grafo  $G$  viršūnių ir  $N(v)=A\cup B$ ,  $A\cap B=\emptyset$ . Tada viršūnės išskaidymo operacija atliekama taip:

- 1) iš grafo  $G$  pašalinama viršūnė  $v$ ;
- 2) įvedamos dvi naujos viršūnės  $v_1$  ir  $v_2$  ir jas jungiančioji briauna;
- 3) viršūnė  $v_1$  jungiama briaunomis su aibės  $A$  viršūnėmis, o  $v_2$  - su aibės  $B$  viršūnėmis.

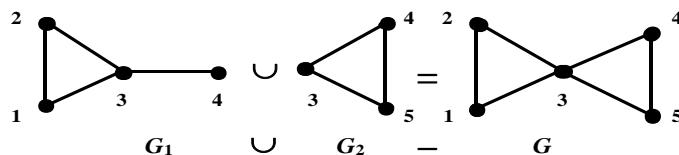
Paveiksle 2.5.5 pav. parodytas grafo  $G$  viršūnės  $v$  išskaidymas iš viršūnes  $v_1$  ir  $v_2$ . Čia  $N(v)=\{a,b,d\}$ ,  $A=\{a\}$  ir  $B=\{b,d\}$ .



**2.5.5 pav.** Grafo  $G$  viršūnės  $v$  išskaidymas iš  $v_1$  ir  $v_2$

**Grafų sąjunga.** Tai viena iš svarbiausių grafų operacijų. Tarkime, kad duoti grafai  $G_1=(V_1, U_1)$  ir  $G_2=(V_2, U_2)$ . Tada grafas  $G=(V, U)$  yra šių grafų sąjunga (žymime  $G=G_1 \cup G_2$ ), jei  $V=V_1 \cup V_2$ , o  $U=U_1 \cup U_2$ . Jei  $V_1 \cap V_2 = \emptyset$ , tai grafų  $G_1$  ir  $G_2$  sąjunga vadinama **disjunktyvine sąjunga**.

2.5.6 pav. Pavaizduota grafų  $G_1$  ir  $G_2$  sąjunga.



**2.5.6 pav.** Grafų  $G_1$  ir  $G_2$  sąjunga

Ši operacija apibendrinama imant daugiau nei du grafus.

$$G(V, U) = \bigcup_{i=1}^n G_i(V_i, U_i), \quad V = \bigcup_{i=1}^n V_i, \quad U = \bigcup_{i=1}^n U_i.$$

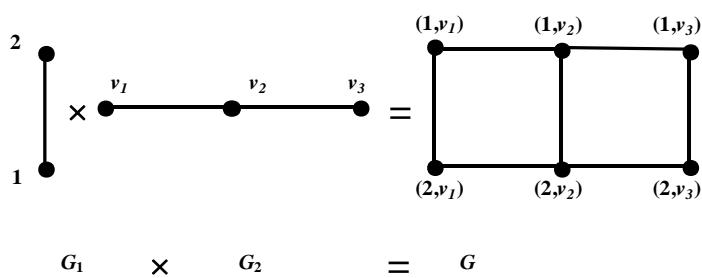
**Grafų sandauga.** Grafų  $G_1=(V_1, U_1)$  ir  $G_2=(V_2, U_2)$  sandaugos grafas  $G=(V, U)$  (žymime  $G=G_1 \times G_2$ ) apibrėžiamas taip:

- 1)  $V=V_1 \times V_2$  - aibiu Dekarto sandauga,
- 2) Viršūnė  $(a, b)$  jungiama su viršūne  $(c, d)$ , jeigu:

  - a)  $a=c$  ir  $(b, d) \in U_2$  arba
  - b)  $b=d$  ir  $(a, c) \in U_1$ .

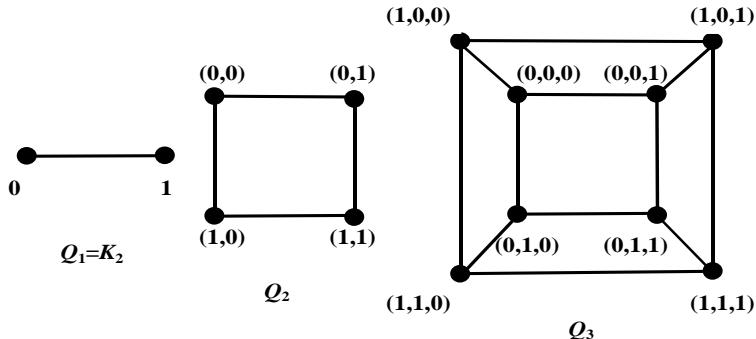
Aišku, kad grafo  $G$  viršūnių skaičius yra lygus  $|V_1| \cdot |V_2|$ , t.y.  $|V|=|V_1| \cdot |V_2|$ , o  $|U|=|V_1| \cdot |U_2| + |V_2| \cdot |U_1|$ .

2.5.7 pav. parodyta grafų  $G_1$  ir  $G_2$  sandauga.



2.5.7 pav. Grafų  $G_1$  ir  $G_2$  sandauga

Naudojant sandaugos operaciją, apibrėžiama svarbi grafų klasė –  **$n$ -mačiai kubai**, kurie žymimi simboliu  $Q_n$ . Šie kubai apibrėžiami rekurentine formulė:  $Q_1=K_2$ ;  $Q_n=K_2 \times Q_{n-1}$ ,  $n>1$ . 2.5.8 pav. parodyti kubai  $Q_1$ ,  $Q_2$  ir  $Q_3$ .

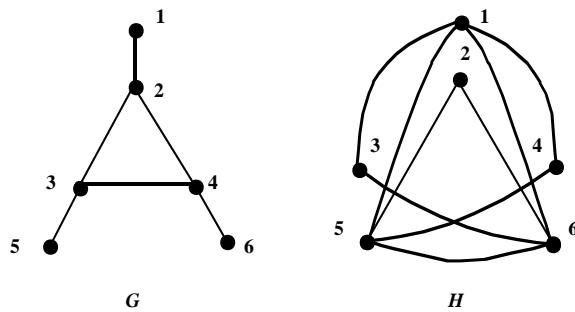


2.5.8 pav.  $n$ -mačių ( $n=1,2,3$ ) kubų pavyzdžiai

Aišku, kad  $Q_n$  viršūnių skaičius lygus  $2^n$ , nes kiekviena viršūnė yra vienmatis vektorius susidedantis iš nulių ir vienetų (dvejetainis  $n$  skilčių skaičius). Dvi viršūnės yra gretimos, jei joms atitinkantys vektoriai skiriasi tik viena komponente. Kadangi kiekviena viršūnė yra incidentiška  $n$  briaunų, tai kiekvienos viršūnės laipsnis lygus  $n$  ir bendras briaunų skaičius

$$m = \frac{1}{2} n \cdot 2^n = n \cdot 2^{n-1}.$$

**Papildomasis grafas.** Grafas  $H=(V, U_H)$  yra grafo  $G=(V, U_G)$  papildomasis grafas, jei  $G \cup H = K_n$ , čia  $n=|V|$ , tai yra grafas, papildantis grafą  $G$  iki pilnojo grafo. Tuo būdu papildomasis grafas  $H$  turi tą pačią viršūnių aibę, kaip ir grafas  $G$ ; grafe  $H$  viršūnės  $v_1$  ir  $v_2$  jungiamos briauna, jei jei šios viršūnės grafe  $G$  nėra gretimos. 2.5.9 pav. pavaizduotas grafas  $G$  ir jo papildomasis grafas.

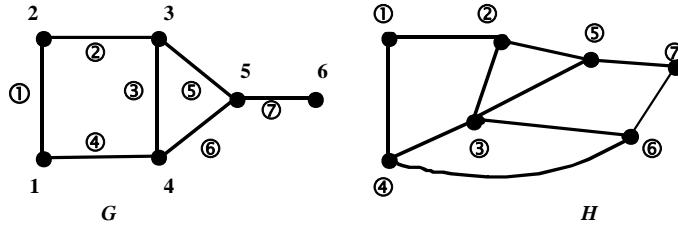


2.5.9 pav. Grafas  $G$  ir papildomasis grafas  $H$

**Briauninis grafas.** Grafo  $G=(V, U)$  briauninis grafas  $H=(A, B)$  apibrėžiamas taip:

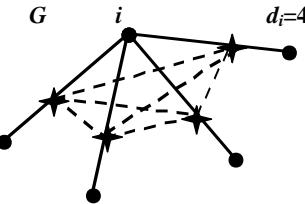
- 1) viršūnių aibės  $A$  elementų skaičius yra lygus grafo  $G$  briaunų skaičiui, t.y. kiekviena grafo viršūnė vaizduoja (atitinka) grafo  $G$  briauną;
- 2) viršūnės  $a_1 \in A$  ir  $a_2 \in A$  jungiamos briauna, jeigu toms viršūnėms atitinkančios grafo briaunos yra gretimos.

2.5.10 pav. pavaizduotas grafas  $G$  ir jį atitinkantis briauninis grafas  $H$ .



**2.5.10 pav.** Grafas  $G$  ir jo briauninis grafas  $H$

Nesunku įsitikinti, kad jei  $d_1, d_2, \dots, d_n$  yra  $(n,m)$  – grafo  $G$  viršūnių laipsnių seka, tai jo briauninis grafas  $H$  yra  $(m,l)$  – grafas, čia  $l = \frac{1}{2} \sum_{i=1}^n d_i^2 - m$ . Iš tikro, kiekviena  $i$ -oji grafo  $G$  viršūnė generuoja  $C_{d_i}^2 = \frac{d_i(d_i-1)}{2} = \frac{d_i^2 - d_i}{2}$  grafo  $H$  briaunu, (žr. 2.5.11 pav.) todėl  $l = \frac{1}{2} \sum_{i=1}^n d_i^2 - \frac{1}{2} \sum_{i=1}^n d_i = \frac{1}{2} \sum_{i=1}^n d_i^2 - m$ .



**2.5.11 pav.** Briauninio grafo briaunos

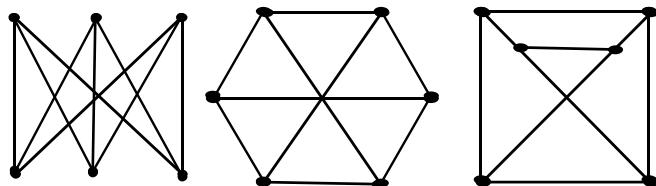
Briauninis grafas yra atskirasis **sankirtos grafo** atvejas. Tarkime, kad  $S \neq \emptyset$ , o  $F = \{S_1, S_2, \dots, S_n\}$  jo poaibių aibė. Tada sankirtos grafo  $S = (V, U)$  viršūnių skaičius yra lygus aibės  $F$  elementų skaičiui, t.y. kiekviena viršūnė  $v_i$  atitinka poaibį  $S_i$ ,  $i = \overline{1, n}$ . Viršūnės  $v_i$  ir  $v_j$  jungiamos briauna, jei  $S_i \cap S_j \neq \emptyset$ .

## 2.6. Grafų izomorfizmas

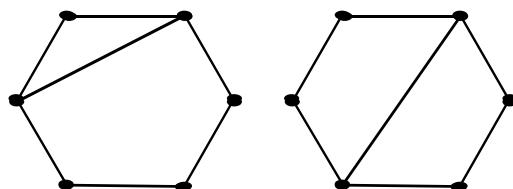
**Apibrėžimas.** Grafai  $G = (VG, UG)$  ir  $H = (VH, UH)$  yra izomorfiniai , jei:

- 1)  $|VG| = |VH|$ ,
- 2)  $|UG| = |UH|$  ir
- 3) yra toks aibės  $VG$  atvaizdis (bijekcija)  $\varphi$  į aibę  $VH$ , kad, jei  $v_1$  ir  $v_2$  yra gretimos grafo  $G$  viršūnės, tai  $\varphi(v_1)$  ir  $\varphi(v_2)$  yra gretimos grafo  $H$  viršūnės.

Jei grafai yra izomorfiniai, tai rašome  $G \cong H$ . Pavyzdžiu, 2.6.1 pav. pavaizduoti trys izomorfiniai grafai, 2.6.2 pav. du neizomorfiniai grafai.



**2.6.1 pav.** Izomorfiniai grafai

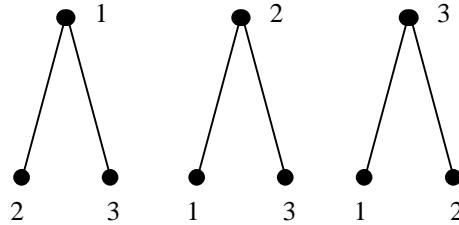


**2.6.2 pav.** Neizomorfiniai grafai

Aišku, kad grafų izomorfizmas yra ekvivalentiškumas, t.y. izomorfizmas visų grafų aibę išskaido į klasės taip, kad vienos klasės grafai tarpusavyje yra izomorfiniai.

Todėl natūralu visus vienos klasės grafus sutapatinti, t.y. visus juos galima pavaizduoti vienu ir tuo pačiu piešiniu. Norėdami pabrėžti, kad nagrinėjami grafai skiriasi izomorfizmo tikslumu, sakome **abstraktusis grafas**. Šis grafas žymi visą izomorfinių grafų klasę.

**Žymėtieji grafai.** Daugelyje situacijų reikia skirti izomorfinius grafus. Tam tikslui kiekvienai grafo viršūnei priskiriama žymė, pavyzdžiu, aibės  $\{1,2,3,\dots,n\}$  skaičius. Toks grafas vadinamas **žymėtuoju grafu**. Du žymėtieji grafai, turintys tą patį viršūnių skaičių, yra lygūs, jei jų briaunų (lankų) aibės yra lygios ir skirtini, jei jų briaunų (lankų) aibės yra nelygios. Pavyzdžiu, 2.6.3 pav. pavaizduoti trys skirtini žymėtieji grafai.



2.6.3 pav. Skirtingi žymėtieji grafai

Iki šiol mes nagrinėjome žymėtuosius grafus. Todėl ir tolesniame nagrinėjime sakydami žodį grafas suprasime, kad tai žymėtasis grafas, o norėdami kalbėti apie izomorfinius grafus, tai atskirai pabréšime.

Koks yra  $n$  viršunių žymėtujų ir izomorfinių grafų skaičius?

**Žymėtujų grafų skaičius.** Didžiausias  $n$  viršunių grafo briaunų skaičius yra lygus  $S = \frac{n(n-1)}{2} = C_n^2$ . Tada grafų, turinčių  $k$  briaunų, skaičius yra lygus  $C_S^k$ . Vadinasi, visų  $n$  viršunių žymėtujų grafų skaičius

$$g_n = \sum_{k=0}^S C_S^k = 2^S = 2^{C_n^2} = 2^{\frac{n(n-1)}{2}}.$$

**Izomorfinių grafų skaičius.** Yra žinoma Pojos formulė:  $g_n$  asymptotiškai lygu  $2^{C_n^2} / n!$ . Žodžiai “asymptotiškai lygu” reiškia, kad

$$\lim_{n \rightarrow \infty} \frac{2^{C_n^2} / n!}{g_n} = 1.$$

Iš pirmo žvilgsnio atrodo, kad žymėtujų  $n$  viršunių grafų yra  $n!$  daugiau negu abstrakčių  $n$  viršunių grafų. Šis faktas atrodo intuityviai aiškus, nes egzistuoja lygiai  $n!$  būdų, kaip priskirti žymes viršūnėms, kurių skaičius yra  $n$ . Tačiau šio teiginio klaidingumą rodo tai, kad ne iš kiekvieno abstraktaus grafo gauname  $n!$  žymėtujų grafų. Pavyzdžiu, visi žymėtieji tuštieji grafai yra lygūs; paprastoji trijų viršunių grandinė duoda tris, o ne šešis žymėtuosius grafus (žr. 2.6.3 pav.). Vienok, daugumoje atvejų, iš abstraktaus  $n$  viršunių grafo gauname  $n!$  žymėtujų grafų.

**Grafų izomorfizmo nustatymo uždavinys.** Dažnai labai svarbu nustatyti faktą, ar du žymėtieji grafai yra izomorfiniai. Tai NP pilnas uždavinys [GD82], neturintis efektyvaus sprendimo algoritmo.

Aišku, kad du žymėtieji grafai yra izomorfiniai, jei galima vieno grafo viršunes pernumeruoti taip, kad abiejų grafų briaunų aibės sutaptų. Šis

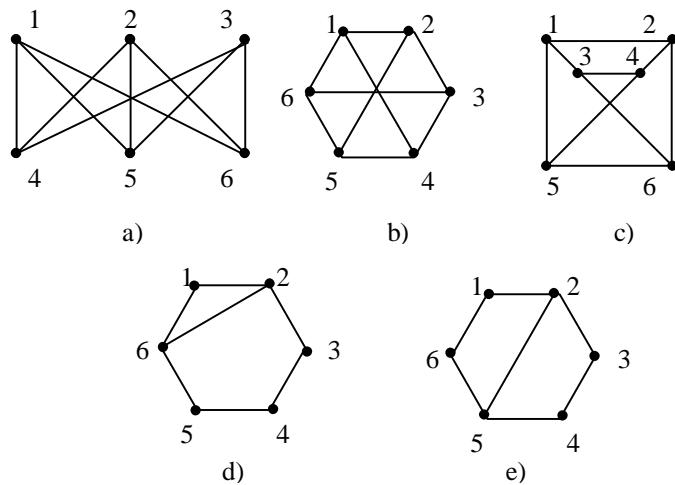
pernumeravimas ir apibrėžia aukščiau minėtą bijekciją. Iš to išplaukia tokios būtinos izomorfizmo sąlygos:

- izomorfinių grafų viršunių laipsnių, išrikiuotų mažėjimo (didėjimo) tvarka, sekos sutampa;
- izomorfinių grafų gretimumo matricos yra panašios, t.y. jų tikrinės reikšmės yra lygios (žr. 2.7 paragrafą).

Tačiau reikia pabrėžti, kad šios sąlygos yra būtinos, bet nepakankamos.

**Pastaba.** Dvieju grafų izomorfizmo nustatymo algoritmas, besiremiantis daliniu visų  $n!$  variantų perrinkimu, detaliai išdėstytas literatūroje [RND80, 396 – 402 psl.].

**Pavyzdys.** Panagrinėkime 2.6.1 pav. ir 2.6.2 pav. pavaizduotus grafus. Pirmiausia jų viršūnėms priskirsiame žymes – numerius (žr. 2.6.4 pav.).



#### 2.6.4 pav. Žymetujų grafų izomorfizmas

Nesunku pastebėti, kad iš b) grafo gausime a) grafą, b) grafo viršūnes pernumeravę taip:

$$\begin{aligned} &1 \ 2 \ 3 \ 4 \ 5 \ 6 \text{ a),} \\ &1 \ 5 \ 3 \ 4 \ 2 \ 6 \text{ b),} \end{aligned}$$

t.y. b) grafo 2-osios viršūnės numerį pakeisime 5-uoju numeriu ir 5-osios viršūnės numerį pakeisime 2-uoju numeriu.

Bijekcija, pevedanti c) grafą į a) grafą, yra

$$\begin{aligned} &1 \ 2 \ 3 \ 4 \ 5 \ 6 \text{ a),} \\ &1 \ 4 \ 6 \ 2 \ 5 \ 3 \text{ c).} \end{aligned}$$

Nors grafų d) ir e) viršūnių laipsnių, išrikiuotų mažėjimo tvarka, sekos yra tos pačios, tačiau jų gretimumo matricos nėra panašios, t.y. šių gretimumo matricų tikrinės reikšmės yra nelygios. Grafo d) gretimumo matricos tikrinės reikšmės yra 2.43828 ,0.61803, -0.82025, -1.61803, -1.75660, 1.13856, o grafo e) gretimumo matricos tikrinės reikšmės yra: 0.41421, -0.41421, -1.00000, 2.41421, -2.41421, 1.00000.

## 2.7. Grafo vaizdavimo kompiuteryje būdai

Aišku, kad žmogui patogiausias ir suprantamiausias grafų vaizdavimo būdas, kai grafai vaizduojami plokštumoje taškų (viršūnių) ir juos jungiančių linijų (briaunų) pavidalu, yra visiškai nepriimtinas, jei mes norime spręsti grafų teorijos uždavinius kompiuteriu. Vienokios ar kitokios duomenų struktūros pasirinkimas turi esminę įtaką algoritmu efektyvumui. Todėl čia aptarsime ir įvertinsime įvairias grafo vaizdavimo kompiuteryje duomenų struktūras.

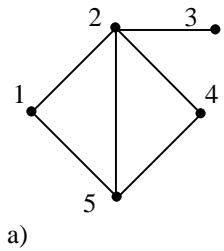
Aptarsime  $(n, m)$ -grafo  $G = (V, U)$  vaizdavimo būdus (duomenų struktūras), kurias vertinsime remdamiesi tokiais kriterijais: 1) vaizdavimui reikalingos informacijos apimtis, 2) galimybė padaryti klaidą, 3) kaip sužinoti viršunes, gretimas pasirinktai viršūnei, t.y. gretimų viršūnių išrinkimas.

**Gretimumo matrica.** Grafo  $G = (V, U)$  gretimumo matrica yra kvadratinė  $n$ -osios eilės matrica  $S = [s_{ij}]$ ,  $i = \overline{1, n}$ ,  $j = \overline{1, n}$ , kurios elementas  $s_{ij}$  apibrėžiamas taip

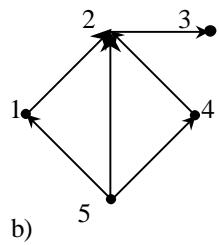
$$s_{ij} = \begin{cases} 1, & \text{jei viršūnės } i \text{ ir } j \text{ yra gretimos,} \\ 0, & \text{priešingu atveju.} \end{cases}$$

Neorientuotojo grafo gretimumo matrica yra simetrinė, o orientuotojo – nesimetrinė. Gretimumo matricos  $i$ -tojoje eilutėje vienetukų skaičius yra lygus  $i$ -tosios viršūnės laipsniui neorientuotojo grafo atveju ir išėjimo puslaipsniui – orientuotojo grafo atveju. Pavyzdžiui, 2.7.1 paveiksle pavaizduoti grafai ir juos atitinkančios gretimumo matricos.

Kap buvo minėta aukščiau, iš grafų izomorfizmo apibrėžimo išplaukia išvada, kad izomorfinių grafų gretimumo matricos gaunamos viena iš kitos nuosekliai sukeičiant vietomis eilutes ir stulpelius bijekcijoje nurodyta tvarka. Pavyzdžiui, 2.6.4 b) pav. grafo gretimumo matrica yra gaunama iš 2.6.4 a) pav. grafo gretimumo matricos nuosekliai sukeičiant 2-ają ir 5-ają eilutes bei 2-ąjį ir 5-ąjį stulpelius.



$$S = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 1 & 1 & 1 \\ 3 & 0 & 1 & 0 & 0 & 0 \\ 4 & 0 & 1 & 0 & 0 & 1 \\ 5 & 1 & 1 & 0 & 1 & 0 \end{array}$$



$$S = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 1 & 0 & 0 & 0 \\ 5 & 1 & 1 & 0 & 1 & 0 \end{array}$$

### 2.7.1 pav. Grafų gretimumo matricos

$$S_a = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 2 & 0 & 0 & 0 & 1 & 1 & 1 \\ 3 & 0 & 0 & 0 & 1 & 1 & 1 \\ 4 & 1 & 1 & 1 & 0 & 0 & 0 \\ 5 & 1 & 1 & 1 & 0 & 0 & 0 \\ 6 & 1 & 1 & 1 & 0 & 0 & 0 \end{array}$$

Sukeitus 2-ajq ir 5-ajq eilutes, gausime

$$\begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 2 & 1 & 1 & 1 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 1 & 1 & 1 \\ 4 & 1 & 1 & 1 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 1 & 1 & 1 \\ 6 & 1 & 1 & 1 & 0 & 0 & 0 \end{array}.$$

Sukeitus 2-ajį ir 5-ajį stulpelius, gausime 2.6.4 b) grafo gretimumo matricą

$$S_b = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 2 & 1 & 0 & 1 & 0 & 1 & 0 \\ 3 & 0 & 1 & 0 & 1 & 0 & 1 \\ 4 & 1 & 0 & 1 & 0 & 1 & 0 \\ 5 & 0 & 1 & 0 & 0 & 0 & 1 \\ 6 & 1 & 0 & 1 & 0 & 1 & 0 \end{array}.$$

Kadangi bijekciją  $\varphi$  nusako perstatymų matrica

$$P = [p_{ij}], i = \overline{1, n}, j = \overline{1, n}, p_{ij} = \begin{cases} 1, & \text{jei } i = \varphi(j), \\ 0, & \text{priešingu atveju,} \end{cases}$$

tai tokį eilučių ir stulpelių sukeitimą vietomis gausime matricą  $S_a$  padauginus iš matricos  $P$ , t.y.

$$S_b = P \cdot S_a \cdot P,$$

čia

$$P = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 1 & 0 \\ 3 & 0 & 0 & 1 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 1 & 0 & 0 \\ 5 & 0 & 1 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}.$$

Kadangi  $P^{-1} = P$ , tai  $S_b$  ir  $S_a$  yra panašios, o panašių matricų tikrinių reikšmių aibės yra lygios.

Įvertinkime grafo vaizdavimą gretimumo matrica pagal aukščiau minėtus kriterijus.

**Informacijos apimtis.** Gretimumo matrica turi  $n^2$  elementų ir paprastai ji yra reta, t.y. vienetu kū skaičius žymiai mažesnis nei nulių skaičius.

**Galimybė padaryti klaidą**, užrašant matricą  $S$ , yra labai didelė, esant didesniams viršūnių skaičiui.

Viršūnės, gretimos viršūnei  $k$ , randamos taip:  
for  $j := 1$  to  $n$  do  
if  $s[k, j] = 1$  then "j-oji viršūnė gretima viršūnei  $k$ ".

Pagal šiuos kriterijus gretimumo matrica turi daugiau teorinę nei praktinę reikšmę.

**Incidencijų matrica.**  $(n, m)$  – grafo  $G = (V, U)$  incidencijų matrica yra stačiakampė  $(n \times m)$  formato matrica  $A = [a_{ij}]$ ,  $i = \overline{1, n}$ ,  $j = \overline{1, m}$ , ir jos elementas  $a_{ij}$  neorientuotojo grafo atveju apibrėžiamas taip:

$$a_{ij} = \begin{cases} 1, & \text{jei } i - \text{oji viršūnė incidentiška } j - \text{ajai briaunai,} \\ 0, & \text{priešingu atveju,} \end{cases}$$

o orientuotojo grafo atveju –

$$a_{ij} = \begin{cases} 1, & \text{jei } i - \text{oji viršūnė yra } j - \text{ojo lanko pradžia,} \\ -1, & \text{jei } i - \text{oji viršūnė yra } j - \text{ojo lanko galas,} \\ 0, & \text{jei } i - \text{oji viršūnė neincidentiška } j - \text{ajam lankui.} \end{cases}$$

Pavyzdžiu, grafams, pavaizduotiems 2.7.1 paveiksle, incidencijų matricos, jei briaunos (lankai) sunumeruoti tokia tvarka (1, 2), (1, 5), (2, 5), (2, 4), (2, 3), (4, 5), yra:

$$A = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 1 & 1 & 0 \\ 3 & 0 & 0 & 0 & 0 & 1 & 0 \end{array} \quad \text{grafui 2.7.1 a) ir}$$

$$\begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 2 & -1 & 0 & -1 & -1 & 1 & 0 \\ 3 & 0 & 0 & 0 & 0 & -1 & 0 \end{array}$$

$$A = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 2 & -1 & 0 & -1 & -1 & 1 & 0 \\ 3 & 0 & 0 & 0 & 0 & -1 & 0 \\ 4 & 0 & 0 & 0 & 1 & 0 & -1 \\ 5 & 0 & 1 & 1 & 0 & 0 & 1 \end{array} \quad \text{grafui 2.7.1 b).}$$

**Informacijos apimtis.** Kaip ir gretimumo matricos atveju, incidencijų matrica turi  $n \cdot m$  elementų ir yra reta.

**Galimybė padaryti klaidą** yra didelė prie didesnių  $n$  ir  $m$  reikšmių.

**Viršūnės, gretimos viršūnei  $k$ ,** neorientuotojo grafo atveju randamos taip:

*for  $j := 1$  to  $m$  do*

*if  $a[k, j] = 1$  then for  $i := 1$  to  $n$  do*

*if ( $a[i, j] = 1$ ) and ( $i <> k$ ) then “viršūnė  $i$  yra gretima viršūnei  $k$ ”;*

**Pastaba.** Orientuotojo grafo atveju sąlyga “ $a[i, j] = 1$ ” turi būti pakeista sąlyga “ $a[i, j] = -1$ ”.

Pagal šiuos kriterijus incidencijų matrica yra dar blogesnis grafo vaizdavimo būdas nei gretumumo matrica ir turi daugiau teorinę nei praktinę reikšmę.

**Briaunų (lankų) matrica.**  $(2 \times m)$  formato matrica  $B$  vadinama briaunų (lankų) matrica, jei  $(b_{1j}, b_{2j})$ ,  $j = \overline{1, m}$  yra  $j$ -oji grafo briauna (lankas).

Orientuotojo grafo atveju  $b_{1j}$  žymi  $j$ -ojo lanko pradžią, o  $b_{2j} - j$ -ojo lanko pabaigą.

Pavyzdžiui, 2.7.1 a) pav. pavaizduoto neorientuotojo grafo briaunų matrica yra

$$B = \begin{pmatrix} 1 & 1 & 2 & 2 & 2 & 4 \\ 2 & 5 & 5 & 4 & 3 & 5 \end{pmatrix},$$

o 2.7.1 b) pav. pavaizduoto orientuotojo grafo lankų matrica yra

$$B = \begin{pmatrix} 1 & 5 & 2 & 5 & 5 & 4 \\ 2 & 1 & 3 & 2 & 4 & 2 \end{pmatrix}.$$

**Informacijos apimtis.** Informacijos apimtis – minimali, matricos elementų skaičius yra  $2m$ .

**Galimybė padaryti klaidą** nėra didelė, nes grafo briaunų (lankų) išvardinimas yra natūralus ir priimtinas žmogui. Be to, briaunų išdėstymo tvarka matricoje  $B$  yra laisva.

**Viršūnės, gretimos viršūnei  $k$ ,** neorientuotojo grafo atveju randamos taip:

*for*  $j := 1$  to  $m$  do

*begin*

*if*  $b[1, j] = k$  *then* “viršūnė  $b[2, j]$  gretima viršūnei  $k$ ”;

*if*  $b[2, j] = k$  *then* “viršūnė  $b[1, j]$  gretima viršūnei  $k$ ”;

*end;*

o orientuotojo grafo atveju –

*for*  $j := 1$  to  $m$  do

*if*  $b[1, j] = k$  *then* “viršūnė  $b[2, j]$  gretima viršūnei  $k$ ”;

Kaip matyti, briaunų (lankų) matrica yra patogus ir kompaktiškas grafo užrašas, tačiau gretimų viršūnių radimas – sudėtingas.

**Gretumumo struktūra.** Gretumumo struktūra – tai viršūnėms gretimų viršūnių aibė (viršūnių aplinką) šeima.

2.7.1 a) grafo gretumumo struktūra yra:

1: {2, 5};

2: {1, 3, 4, 5};

3: {2};

4: {2, 5};

5: {1, 2, 4};

2.7.1 b) grafo gretumumo struktūra yra:

1: {2};

2: {3};

3:  $\emptyset$ ;

4: {2};

5: {1, 2, 4};

Gretumumo struktūrą kompiuteryje galima vaizduoti įvairiomis duomenų struktūromis. Pavyzdžiu, galima vaizduoti  $(n \times \max_{v \in V} d(v))$  formato matrica, čia  $d(v)$  –  $n$ -tosios viršūnės laipsnis. Tada matricos  $k$ -osios eilutės nenuliniai elementai yra viršūnei  $k$  gretimos viršūnės.

Gretumumo struktūrą galima vaizduoti ir vienryšių sąrašu šeima.

Kiekvienas vienryšis sąrašas nusakys gretimų viršūnių aibę.

Gretumumo struktūros *informacijos apimtis* yra minimali, o *klaidos tikimybė* užrašant struktūrą – nedidelė.

Tarkime, kad gretumumo struktūra užrašyta matrica  $T = [t_{ij}]$ ,  $i = \overline{1, n}$ ,  $j = \overline{1, \max_{v \in V} d(v)}$  (informacijos apimties požiūriu matrica nėra optimalus gretumumo struktūros užrašymo būdas). Tada **viršūnei  $k$  gretimos viršūnės gaunamos taip:**

*for*  $j := 1$  *to*  $\max_{v \in V} d(v)$  *do*

*if*  $t[k, j] \neq 0$  *then* “ $t[k, j]$  viršūnė gretima viršūnei  $k$ ”;

**Pastaba.** Tolesniame dėstyme, formaliai užrašant algoritmus, naudosime gretumumo struktūrą. Kaip minėjome aukščiau, simboliu  $N(v)$  žymėsime viršūnės  $v$  gretimų viršūnių aibę. Todėl, norėdami pasakyti, kad “**nagrinėjame viršūnes, gretimas viršūnei  $v$** ”, rašysime:

*for*  $u \in N(v)$  *do* “nagrinėti viršūnę  $u$ ”.

**Nuoseklaus peržiūrėjimo masyvas.** Tai masyvas, turintis  $n + 2m$  elementų neorientuotojo grafo atveju ir  $n + m$  – orientuotojo grafo atveju, ir kuris sudaromas taip: iš eilės, pradedant pirmaja viršūne ir baigiant paskutiniaja, kiekvienai viršūnei rašomas viršūnės numeris su minuso ženklu, o po jo rašomas tai viršūnei gretimos viršūnės. Jei ši masyva pažymėsime simboliu  $P$ , tai 2.7.1 pav. grafams gausime:

$P : -1, 2, 5, -2, 1, 3, 4, 5, -3, 2, -4, 2, 5, -5, 1, 2, 4;$

$P : -1, 2, -2, 3, -3, -4, 2, -5, 1, 2, 4 .$

Norėdami rasti viršunes, **gretimas viršūnei  $k$** , turėsime masyve  $P$  rasti elementą, lygū  $-k$ , tada po jo einantys teigiami masyvo elementai bus viršūnės, gretimos viršūnei  $k$ . Formaliai ši veiksmą galima užrašyti taip:

```
i := 1;
while p[i] ≠ -k do i := i + 1;
l := i + 1;
while p [l] > 0 do begin "viršūnė p [l] yra gretima viršūnei k";
    l = l+1
end;
```

Vertinant šį grafo vaizdavimo būdą pagal aukščiau minėtus kriterijus, galima pasakyti, kad **informacija kompaktiška, suklydimo galimybė** nėra didelė, tačiau ilgas **gretimų viršūnių išrinkimas**. Todėl žymiai efektyvesnis yra žemiau pateiktas grafo užrašas, kurį vadinsime tiesioginių nuorodų masyvais arba briaunų (lankų) ir jų adresų masyvais.

**Briaunu (lankų) ir jų adresų masyvai (Tiesioginių nuorodų masyvai).** Briaunu (lankų) masyvas  $L$  turi  $2m$  elementų (neorientuotiesiems grafams) ir  $m$  elementų (orientuotiesiems grafams). Jis sudaromas taip: nuosekliai pradedant pirmaja viršūne ir baigiant paskutiniaja, iš eilės kiekvienai viršūnei surašomos jai gretimos viršūnės, t.y. masyvas  $L$  gaunamas iš nuoseklaus peržiūrėjimo masyvo  $P$  pašalinus neigiamus elementus.

2.7.1 a) pav. grafui masyvas  $L$  bus tokis:

$$\begin{array}{cccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ \hline L: & 2, & 5, & 1, & 3, & 4, & 5, & 2, & 2, & 5, & 1, & 2, & 4; \end{array}$$

o 2.7.1 b) pav. grafui –

$$\begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ \hline L: & 2, & 3, & 2, & 1, & 2, & 4. \end{array}$$

Skaičiai virš elementų rodo jų vietą (adresą) masyve  $L$ .

Turint tik masyvą  $L$  sužinoti viršunes, gretimas viršūnei  $k$ , yra neįmanoma. Todėl įvedamas antras – viršūnių adresų masyvas  $lst$ , turintis  $n + 1$  elementą. Šis masyvas sudaromas taip:

$lst[1] := 0;$

$lst[i + 1] := lst[i] + d[i], i = \overline{1, n},$

čia  $d[i]$  –  $i$ -osios viršūnės laipsnis. Aišku, kad  $lst[k]$  parodo, kiek reikia praleisti masyvo  $L$  elementų, kad rastume viršūnes, gretimas viršūnei  $k$ . Vadinas, viršūnės, gretimos viršūnei  $k$ , masyve  $L$  yra išsidėstę pradedant adresu  $lst[k] + 1$  ir baigiant adresu  $lst[k + 1]$ .

2.7.1 a) pav. grafui masyvas  $lst$  bus:

$$\begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ \hline lst: & 0, & 2, & 6, & 7, & 9, & 12, \end{array}$$

o 2.7.1 b) pav. grafui –

	1	2	3	4	5	6
<i>lst</i> :	0,	1,	2,	2,	3,	6.

Tuo būdu viršūnės, *gretimos viršūnei k*, randamos taip:  
*for i := lst[k] + 1 to lst[k + 1] do*

```

begin
    u := L[i];
    viršūnė u gretima viršūnei k;
end;
```

Šis grafo užrašymo būdas yra *kompaktiškas*, o *gretimų viršūnių išrinkimas yra efektyvus*, tačiau, rašant masyvus *L* ir *lst*, *lengva padaryti klaidą*. Todėl paprastai informacija apie grafo įvedama briaunų masyvu, o masyvai *L* ir *lst* apskaičiuojami programiškai.

Žemiau aptarsime algoritmą, pervedantį grafo užrašą iš briaunų matricos į tiesioginės nuorodos masyvus.

**Procedūra, pervedanti neorientuotojo grafo užrašą iš briaunų matricos į masyvus L ir lst**

```

procedure BLlst (n, m : integer; b : matr; var L, lst : mas);
{ Procedūra BLlst pvereda neorientuotojo grafo briaunu matrica į tiesioginių nuorodų masyvus L ir lst.
```

Formalūs parametrai:

```

n – grafo viršūnių skaičius,
m – grafo briaunų (lankų) skaičius,
b – grafo briaunų matrica;
(b [1, j], b [2, j]) – j-toji grafo briauna;
L, lst – grafa nusakantys tiesioginių nuorodų masyvai.
```

Vidiniai kintamieji:

```

d[1..n] – viršūnių laipsnių masyvas;
d[i] – i-osios viršūnės laipsnis.
fst[1..n] – adresų masyvas;
pradžioje fst [i] = lst [i] + 1, i = 1, 2, ..., n. fst[i] – tai adresas masyve L,
kur turi būti talpinamas numeris pirmos viršūnės, gretimos viršūnei i. }
var i, j, k : integer;
fst, d : mas;
begin
{ Viršūnių laipsnių apskaičiavimas. }
for i := 1 to n do d [i] := 0;
for i := 1 to 2 do { *}
    for j := 1 to m do
```

```

begin
    k := b [i, j];
    d [k] := d [k] + 1;
end;
{ Masyvo lst formavimas }
lst [1] := 0;
for i: = 1 to n do lst [i + 1] := lst [i] + d [i];
{ Masyvo L formavimas }
for i: = 1 to n do fst [i]:=lst [i] + 1;
for j := 1 to m do
begin
    k := b[1 ,j];
    L [fst [k]] := b [2, j];
    fst [k] :=fst [k] + 1;
    k := b [2,j]; { **}
    L [fst [k]] := b [1, j]; { **}
    fst [k] :=fst [k] + 1; { **}
end;
end;

```

Norėdami pervesti orientuotojo grafo užrašą iš lankų matricos į masyvus  $L$  ir  $lst$ , aukščiau pateiktą procedūrą reikėtų truputį modifikuoti – pašalinti žvaigždute pažymėtus operatorius, t.y.:

- apskaičiuojant viršinių laipsnius, reikia pašalinti žvaigždute (\*) pažymėtą operatorių;
- formuojant masyvą  $L$ , reikia pašalinti dviejomis žvaigždutėmis (\*\*) pažymėtus operatorius.

Kokį iš išvardytų grafo užrašų naudoti? Atsakymas į šį klausimą priklauso nuo sprendžiamo uždavinio. Jei uždavinio sprendimo metodas reikalauja žinoti gretimas viršunes, tai labai patogūs  $L$  ir  $lst$  masyvai. Jei – reikalauja nagrinėti briaunas, patogu grafą vaizduoti briaunų matrica. Todėl tolesniame dėstyme, formaliai užrašant algoritmus, dažnai naudosime gretimumo struktūrą, o detaliems algoritmams naudosime arba briaunų matricą, arba masyvus  $L$  ir  $lst$ .

## 2.8. Grafo viršunių peržiūros metodai

Daugelio grafių teorijos uždaviniių sprendimo algoritmų pagrindą sudaro sisteminga grafo viršunių peržiūra, t.y. toks grafo viršunių apėjimas, kad kiekviena viršunė nagrinėjama vienintelį kartą. Todėl labai svarbus uždavinyse yra rasti gerus grafo viršunių peržiūros metodus. *Apskritai kalbant, viršunių peržiūros metodas yra “geras”, jei:*

- nagrinėjamo uždavinio sprendimo algoritmas lengvai įsikomponuoja į peržiūros metodą;
- kiekviena grafo briauna analizuojama ne daugiau kaip vieną kartą (arba, kas iš esmės nekeičia situacijos, briaunos nagrinėjimo skaičius apribotas konstanta).

Pagrindiniai grafo viršunių peržiūros metodai, tenkinantys pateiktus reikalavimus, yra *paieškos gilyn metodas* ir *paieškos platyn metodas*.

### 2.8.1. Paieška gilyn

Paieškos gilyn (rus. poisk v glubinu; angl. Depth first search) metodą pirmasis 1972 m. pasiūlė R.Tarjanas<sup>1</sup>. Metodo idėja yra labai paprasta. Pradžioje visos grafo viršūnės yra **naujos (neaplankytos)**. Tarkime, kad paieška pradedama iš viršūnės  $v_0$ . Viršūnė  $v_0$  tampa **nenuja**, ir išrenkame viršūnę  $u$ , kuri yra gretima viršūnei  $v_0$ . Jei viršūnė  $u$  yra **nauja**, peržiūros procesą tęsiame iš viršūnės  $u$ .

Tarkime, kad esame viršūnėje  $v$ .

Jei yra **nauja** dar neaplankyta viršūnė  $u$ , gretima viršūnei  $v$ , tai nagrinėjame viršūnę  $u$  (ji tampa **nenuja**) ir paiešką tęsiame iš viršūnės  $u$ .

Jei nėra nei vienos **naujos** viršūnės, gretimos viršūnei  $v$ , tai sakome, kad viršūnė  $v$  **išsemta**; grįžtame į viršūnę, iš kurios patekome į viršūnę  $v$ , ir paiešką tęsiame iš šios viršūnės.

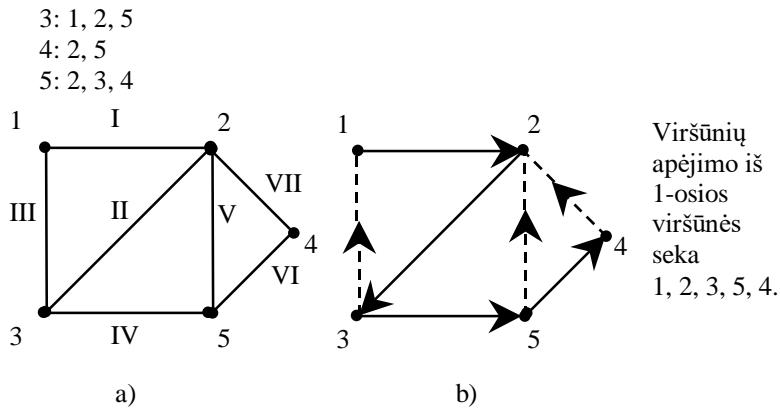
Paiešką baigiamo, kai pradinė paieškos viršūnė  $v_0$  tampa **išsemta** viršūne.

**Pavyzdys.** Panagrinėkime, kaip bus peržiūrėtos 2.8.1 a) pav. pavaizduoto grafo viršūnės, jei paieška pradedama iš viršūnės  $v_0 = 1$ , o bet kokiai viršūnei gretimos viršūnės gretumumo struktūroje išdėstyotos jų numeriu didėjimo tvarka:

- 1: 2, 3
- 2: 1, 3, 4, 5

---

<sup>1</sup> Tarjan R. A. Depth first search and linear graph algorithms. SIAM J. Comput., 1972, 1, .p.p. 146 – 160.



**2.8.1 pav.** Paieška gilyn iš 1-osios viršūnės

Paieškos gilyn metu pirmiausia iš 1-osios viršūnės einame į 2-ają viršūnę. Kadangi 2-oji viršūnė yra nauja, tai toliau peržiūrą tęsiame iš 2-osios viršūnės. (Pirmoji ir antroji viršūnės tampa aplankytomis). Iš 2-osios viršūnės pirmiausia einame į 1-ają viršūnę. Kadangi pirmoji viršūnė nenaauja (aplankyta), tai iš 2-osios viršūnės bandome eiti į kita jai gretimą 3-iajį viršūnę. Trečioji viršūnė nauja, todėl ją aplankome ir peržiūrą tęsiame iš 3-osios viršūnės. Iš 3-iosios viršūnės bandome eiti į pirmąjį jai gretimą viršūnę, – į 1-ają viršūnę. Ši viršūnė nenaauja. Tada bandome eiti į kitą 3-ajai viršūnei gretimą 2-ają viršūnę. Ši viršūnė taip pat aplankyta, todėl bandome eiti į paskutiniąjį 3-iajai viršūnei gretimą 5-ają viršūnę. Ši viršūnė yra nauja, todėl peržiūrą dabar tęsime iš 5-osios viršūnės. Iš 5-osios viršūnės, po bandymų keliauti į 2-ają ir 3-iajį viršūnes, ateisime į 4-ają viršūnę. Kadangi 4-osios viršūnės visos gretimos viršūnės nėra naujos, tai 4-oji viršūnė išsemta. Paiešką bandome tęsti iš 5-osios viršūnės, nes į 4-ają viršūnę atėjome iš 5-osios viršūnės. 5-ajai viršūnei visos gretimos viršūnės yra nenaaujos, todėl 5-oji viršūnė yra išsemta ir paiešką bandome tęsti iš 3-iosios viršūnės. 3-ioji viršūnė taip pat išsemta, todėl paiešką tęsime iš 2-osios viršūnės, nes į 3-iajį viršūnę atėjome iš 2-osios viršūnės. 2-ajai viršūnei dar nenagrinėta gretima yra 5-toji viršūnė. Tačiau ši viršūnė nenaauja, todėl 2-oji viršūnė tampa išsemta ir paiešką tęsiame iš 1-osios viršūnės. Iš 1-osios viršūnės bandome eiti į 3-iajį, tačiau ji jau aplankyta. Tuo būdu ir 1-oji viršūnė tampa išsemta, o tai yra paieškos gilyn algoritmo pabaiga.

**Pastaba.** 2.8.1 a) pav. romėniškais skaitmenimis pažymėti briaunų apėjimo eilės numeriai.

2.8.1 b) pav. ištisinėmis linijomis pavaizduotos briaunos (su nurodyta apėjimo kryptimi), kurios veda į naujas viršūnes, o punktyrais pažymėtos briaunos, kurios paieškos gilyn metu vedė į aplankytas (nenaujas) viršūnes.

Panagrinėkime, kaip programiškai organizuoti paieškos gilyn metodą. Aptarsime tris paieškos gilyn organizavimo metodus: 1) paieškos gilyn, naudojant rekursiją, būdą, 2) paieškos gilyn su mažiausia atminties apimtimi organizavimo būdą ir 3) paieškos gilyn, nenaudojant rekursijos, būdą.

### 2.8.1.1. Paieška gilyn, naudojant rekursiją

Tarkime,  $(n, m)$ -grafas –  $G = (V, U)$  nusakytas gretimumo struktūra:  $N(v)$  – aibė viršūnių, gretimų viršūnei  $v$ ,  $v \in V$ . Kaip minėjome aukščiau, sakinį “**nagrinėti viršūnes, gretimas viršūnei  $v$** ”, formaliai užrašysime: *for  $u \in N(v)$  do nagrinėti  $u$ ;*

Apibrėžkime masyvą **naujas** [1.. $n$ ], čia **naujas** [ $i$ ] = *true*, jei viršūnė  $i$  **nauja (aplankyta)** ir **naujas** [ $i$ ] = *false*, jei viršūnė  $i$  **nenauja (aplankyta)**.

Tarkime, kad gretimumo struktūra ir masyvas **naujas** yra globalieji masyvai. Tada paieškos gilyn iš viršūnės  $v$  procedūra bus užrašoma taip:

```
procedure gylis ( $v$ );
begin
    “nagrinėti viršūnę  $v$ ”; naujas [ $v$ ] := false;
    for  $u \in N(v)$  do
        if naujas[ $u$ ] then gylis ( $u$ );
    end;
```

Paieška gilyn grafe, kuris gali būti ir nejungusis, bus vykdoma taip:

```
begin
    for  $v \in V$  do naujas [ $v$ ] := true; {inicializacija}
    for  $v \in V$  do
        if naujas[ $v$ ] then gylis ( $v$ );
    end;
```

### 2.8.1.2. Paieškos gilyn su mažiausia atminties apimtimi organizavimas

Ši algoritmą aptarkime, laikydami, kad grafas  $G$  užrašytas masyvais  $L$  ir  $lst$  (žr. 2.7 paragrafą). Kitaip tariant, šios procedūros įejimo parametrai yra:

$n$  – grafo viršūnių skaičius,  
 $m$  – grafo briaunų (lankų) skaičius,  
 $L$  [1..  $2m$ ] (neorientuotiesiems grafams) – briaunų masyvas,  
 $(L$  [1..  $m$ ] (orientuotiesiems grafams) – lankų masyvas),

$lst[1..n+1]$  – briaunų (lankų) adresų masyvas,

$v$  – paieškos gilyn viršūnės numeris.

Reikia organizuoti paiešką gilyn iš viršūnės  $v$ .

**Vidiniai kintamieji ir darbo masyvai.** Aptarkime vidinius kintamuosius ir darbo masyvus, kurie naudojami organizuojant paiešką gilyn.

**Masyvas  $fst[1..n]$ ;  $fst[i]$**  – reiškia adresą masyve  $L$  dar nenagrindėtos viršūnės, gretimos viršūnei  $i$ ,  $i = \overline{1, n}$ ; tiksliau briauna (jei tokia yra)  $(i, L[fst[i]])$  yra paieškos gilyn metu dar nenagrindėta briauna, incidentiška viršūnei  $i$ . Pradinės masyvo  $fst$  elementų reikšmės yra  $fst[i] := lst[i] + 1$ ,  $i = \overline{1, n}$ .

**Masyvas  $prec[1..n]$ .** Šio masyvo elementai naudojami keliems tikslams:

$$prec[i] = \begin{cases} k, & \text{jei paieškos gilyn metu } i \text{ viršūnė } i \text{ atėjome iš viršūnės } k; \\ i, & \text{jei } i - \text{oji viršūnė yra pradinė paieškos viršūnė}; \\ 0, & \text{jei viršūnė } i \text{ nauja.} \end{cases}$$

Pradžioje  $prec[i] := 0$ ,  $i = \overline{1, n}$  (visos grafo viršūnės yra naujos).

**Kintamasis  $k$**  žymi viršūnę, iš kurios tesiame paiešką. Pradžioje  $k := v$  ;  
Loginis kintamasis  $p$ :

$$p = \begin{cases} true, & \text{jei paieškos gilyn metu einame "pirmyn"}; \\ false, & \text{jei paieškos gilyn metu einame "atgal".} \end{cases}$$

Loginis kintamasis  $t$ :

$$t = \begin{cases} true, & \text{jei paieška gilyn baigtą}; \\ false, & \text{jei paieška gilyn nebaigtą.} \end{cases}$$

Tada paieškos gilyn procedūra gali būti užrašyta taip.

```
const c = 500;
type
    mas = array [1..c] of integer;
    matr = array [1..2, 1..c] of integer;
procedure gylis1 (v, n, m : integer; L, lst : mas);
{ Procedūra gylis1 organizuoja paiešką gilyn iš viršūnės v, kai grafas
  nusakytas L ir lst masyvais.
Formalūs parametrai:
```

$v$  – pradinė paieškos viršūnė,

$n$  – grafo viršūnių skaičius,

$m$  – grafo briaunų (lankų) skaičius,

$L, lst$  – grafą nusakantys tiesioginių nuorodų masyvai. }

```
var i, k, u : integer;
```

```

 $t, p : boolean;$ 
 $fst, prec : mas;$ 
begin
{ Inicializacija }
for  $i := 1$  to  $n$  do begin
     $fst [i] := lst [i] + 1;$ 
     $prec [i] := 0;$ 
end;
 $k := v;$ 
if  $fst [k] <= lst [k + 1]$  then {yra nenagrinėtų briaunų, incidentiškų viršūnei  $k$ }
begin
     $t := false;$ 
     $p := true;$ 
     $prec [k] := k;$  {  $k$  – pradinė paieškos viršūnė }
    { Nagrinėti viršūnę  $k$  }
    writeln( $k$ );
end
else {viršūnė  $k$  yra arba izoliuota viršūnė, arba neturi išeinančių
lankų (orientuotojo grafo atveju); paieškos pabaiga }
 $t := true;$ 
while not  $t$  do { paieška nebaigta }
begin
{ Pirmyn }
while  $p$  do
begin
     $u := L [fst [k]];$ 
    if  $prec [u] = 0$  then {virsunė  $u$  nauja}
    begin { Nagrinėti viršūnę  $u$  }
        writeln( $u$ );
         $prec [u] := k;$  { $j$  viršūnę  $u$  atėjome iš viršūnės  $k$ }
        if  $fst [u] <= lst [u + 1]$  then {viršūnė  $u$  neišsemta}
             $k := u$ 
            else {viršūnė  $u$  išsemta}
             $p := false;$ 
        end
    else
         $p := false;$  {viršūnė  $u$  nenauja}
    end;
{Atgal}
while not  $p$  and not  $t$  do
begin

```

```

{ Imama nauja, dar nenagrinēta briauna, incidentiška viršūnei k }
fst [k] := fst [k] + 1;
if fst[k] <= lst [k + 1] then { tokia briauna egzistuoja }
    p := true
    else {viršūnė k išsemta}
    if prec [k] = k then { pradinė paieškos viršūnė išsemta;
        paieškos pabaiga }
        t := true
        else { grīžome į viršūnę, iš kurios buvome atėję į
            viršūnę k } k := prec [k];
    end;
end;
end;

```

#### 2.8.1.3. Paieška gilyn, nenaudojant rekursijos

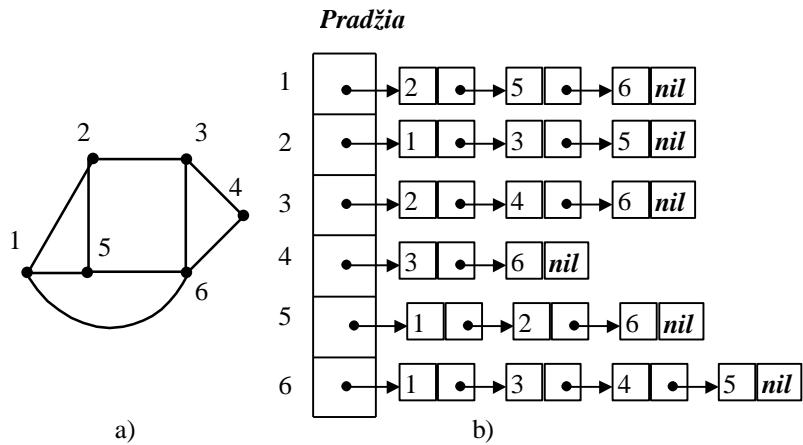
Tarkime,  $(n,m)$ -grafas –  $G = (V, U)$  nusakytas gretimumo struktūra  $N(v)$ ,  $v \in V$ , čia  $N(v)$  – viršūnei  $v$  gretimų viršūnių aibė.

Aibes  $N(v)$  vaizduosime *vienryšiai sąrašais*. Kiekvienas  $v$ -ojo sąrašo elementas yra įrašas  $r$ , nusakantis viršūnę  $r.viršūnė$  ir rodyklę  $r.pėdsakas$ , rodančią, kur yra patalpintas kitas aibės  $N(v)$  elementas.

Jei  $r.pėdsakas = nil$ , tai reiškia, kad  $r.viršūnė$  yra paskutinysis  $v$ -ojo sąrašo elementas (paskutinysis aibės  $N(v)$  elementas).

Kiekvieno sąrašo pradinio elemento adresas saugomas masyve (lentelėje) **Pradžia**. Tiksliau, **Pradžia** [ $v$ ] yra rodyklė (adresas), nurodantis kur yra patalpintas pradinis  $v$ -ojo sąrašo elementas.

**Pavyzdys.** 2.8.2 pav. pavaizduotas grafas ir jo gretimumo struktūra, kaip vienryšių sąrašų aibė.



**2.8.2 pav.** Grafas a) ir jo gretimumo struktūra b) – vienryšių sąrašų masyvas

Aprašysime paieškos gilyn organizavimo, nenaudojant rekursijos, procedūrą, kai grafo gretimumo struktūra nusakoma vienryšių sąrašų masyvu. Tam tikslui įveskime steko sąvoką.

*Stekas* – tai tiesinis sąrašas, kuriame naujų elementų patalpinimas bei elementų šalinimas atliekamas viename sąrašo gale.

Dažnai stekas vadinamas vardu *LIFO*, nuo angliskų žodžių: *Last In First Out* (paskutinis į sąrašą, pirmas iš jo).

Steką patogu realizuoti vienmačiu masyvu, pavyzdžiui,  $E[0..n]$ . Jis charakterizuojamas vienu parametru *top* – steko viršūnė.

*Stekas tuščias*  $\{E := \emptyset\}$

*top* := 0;

*Elemento patalpinimas į steką*  $\{E \leftarrow \text{naujas elementas}\}$

*top* := *top* + 1;

*if top <= n then E [top] := naujas elementas*

*else "persipildymas";*

*Elemento pašalinimas iš stoko*  $\{y \leftarrow E\}$

*if top = 0 then "stekas tuščias"*

*else begin y := E [top]; top := top - 1; end;*

*Viršutinio stoko elemento nuskaitymas, neperskaičiuojant stoko viršūnės*  $\{u := \text{top}(E)\}$

*if top = 0 then "stekas tuščias"*

*else u := E [top];*

Paieškos gilyn organizavimas, nenaudojant rekursijos

Imkime rekursyvinę procedūrą **gylis(v)** (žr. 8.2.1.1 paragrafą) ir rekursiją pašalinkime standartiniu būdu – naudodami steką. Kiekviena **aplankyta** viršūnė talpinama į steką STEK ir šalinama iš jo po jos išsėmimo.

*procedure gylis2(v); [Lip88]*

*{Paiėška gilyn grafe iš viršūnės v. Nerekursyvinė procedūros gylis versija (žr. 2.8.1.1 paragrafą).*

*Tarkime, kad paieškos proceso pradžioje  $P[v] = \text{rodyklė (ląstelės adresas)} iš pirmajų N(v) elementų kiekvienai viršūnei v \in V$ .*

*Masyvai P ir Naujas – globalieji.}*

```

begin
    STEK := Ø; STEK ⊜ v; nagrinēti v;
    Naujas [v] := false;
    while STEK ≠ Ø do
        begin
            t := top(STEK); {t – viršutinis steko elementas}
            {Sąraše N(t) rasti pirmą naują viršūnę}
            if P[t] = nil then b:= false
                else b := not Naujas[P[t]↑.viršūnė];
            while b do
                begin
                    P[t] := P[t]↑.pėdsakas;
                    if P[t] = nil then b:= false
                        else b := not Naujas[P[t]↑.viršūnė];
                end;
            if P[t] ≠ nil then {Radome naują viršūnę}
                begin
                    t := P[t]↑.viršūnė;
                    STEK ⊜ t;
                    nagrinēti t;
                    Naujas[t] := false;
                end
                else {viršūnė t išsemta}
                {Viršūnė t šalinti iš steko, t.y. šalinti viršutinį steko elementą.}
                t ⊜ STEK;
            end; {while}
        end; {gylis2}
    
```

### 2.8.2. Paieška platyn

Dabar aptarsime grafo viršūnių peržiūros iš viršūnės  $v_0$  paieškos platyn metodą (rus. poisk v širinu; angl. Breadth first search).

Kaip ir paieškos gilyn metode, pradžioje visos grafo viršūnės yra **naujos** (*neaplankytos, neperžiūrētos*). Tarkime, kad paiešką pradedame iš viršūnės  $v_0$ . Nagrinėjame viršūnę  $v_0$ ; ji tampa nenauja (aplankyta). Toliau nagrinėjamos ir tampa nenaujomis visos viršūnės, gretimos viršūnei  $v_0$ , t.y. nagrinėjamos viršūnės, kurios nuo viršūnės  $v_0$  nutolę atstumu, lygiu 1. Po to nagrinėjamos ir tampa **nenaujomis** visos **naujos** viršūnės, gretimos prieš tai nagrinėtomis viršūnėmis, t.y. nagrinėjamos visos **naujos** viršūnės, kurios nuo viršūnės  $v_0$  nutolę atstumu, lygiu 2. Apskritai,  $k$ -ajame žingsnyje nagrinėjamos ir tampa **nenaujomis** visos **naujos** viršūnės, gretimos ( $k-1$ )-ajame žingsnyje nagrinėtomis viršūnėmis, t.y. viršūnės, kurios nuo viršūnės  $v_0$  nutolę atstumu, lygiu  $k$ . Paieška platyn baigiamā, kai visos grafo viršūnės tampa **nenaujomis**, t.y. kai peržiūrimos visos viršūnės.

**Paieškos platyn organizavimas.** Paiešką platyn patogu organizuoti naudojant *eilę*.

Priminsime, kad *eilė* – tai tiesinis sąrašas, kuriamė naujas elementas talpinamas viename sąrašo gale, o elementas šalinamas (aptarnaujamas) kitame sąrašo gale. Todėl sąrašą dar vadina *FIFO* vardu, nuo žodžių *first in first out* (pirmas į eilę, pirmas iš eilės).

Yra įvairių eilės organizavimo būdų: naudojant užciklintą vienmatį masyvą, dinaminį atminties paskirstymą ir kt. Čia aptarsime eilės organizavimą panaudojant paprasčiausią duomenų struktūrą – užciklintą vienmatį masyvą: *Eilė* [1..  $n$ ];

Eilė charakterizuojama dvimi parametrais:  $r$  ir  $f$ , –  $r$  žymi eilės galą, o  $f$  – eilės pradžią (žr. 2.8.3 pav.), tiksliau,  $f$  rodo į prieš pirmajį eilės elementą esančią ląstelę.

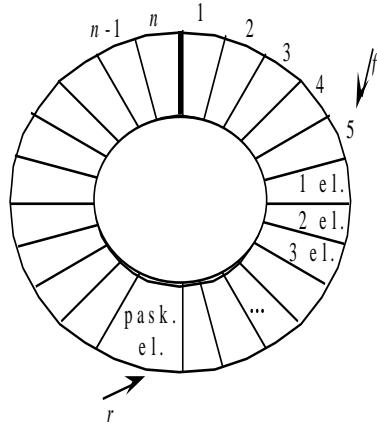
Aptarsime operacijas su eile. Prie operacijos riestiniuose skliaustuose rašysime simbolinį šios operacijos žymėjimą.

**Eilei priskiriama tuščia aibė** { *Eilė* :=  $\emptyset$  }

$r := 1; f := 1;$

**Ar Eilė tuščia?** { *Eilė* =  $\emptyset$  }

*if r = f then {Eilė tuščia} ...;*



**2.8.3 pav.** Eilė kaip viematis užciklintas masyvas

**Naujo elemento patalpinimas į užciklintą eilę** { $Eilė \Leftarrow naujas\ elementas$ }

```

if  $r = n$  then  $r := 1$ 
else  $r := r + 1$ ;
if  $r = f$  then "eilės persipildymas"
else  $Eilė[r] := naujas\ elementas$ ;
```

**Elemento šalinimas iš eilės** { $u \Leftarrow Eilė$ }

```

if  $r = f$  then "eilė tuščia"
else
begin
if  $f = n$  then  $f := 1$ 
else  $f := f + 1$ ;
 $u := Eilė[f]$ ;
end;
```

Tarkime, kad paieškos platyn procedūroje  $(n,m)$ -grafas  $G$  nusakytas **gretimumo struktūra**:  $N(v)$  – aibė viršūnių, gretimų viršūnei  $v$ . Be to, visos grafo viršūnės yra **naujos**. Tą žymi masyvas **naujas** [1..  $n$ ], čia **naujas** [ $v$ ] = *true*, jei viršūnė  $v$  – nauja ir **naujas** [ $v$ ] = *false* – priešingu atveju. Laikykime, kad **gretimumo struktūra** ir masyvas **naujas** yra globalieji kintamieji. Tada paieška **platyn** iš viršūnės  $v$  užrašoma taip:

```

procedure plotis ( $v$ );
begin
```

```

Eile :=  $\emptyset$ ;
Eile  $\Leftarrow v$ ;
naujas [v] := false;
while Eile  $\neq \emptyset$  do
begin
    p  $\Leftarrow$  Eile;
    aplankytu (nagrinėti) p;
    for u  $\in N(p)$  do
        if naujas [u] then
            begin
                Eile  $\Leftarrow u$ ;
                naujas [u] := false;
            end;
        end;
    end;

```

Aptarti paieškos gilyn ir paieškos platyn metodai plačiai taikomi sprendžiant įvairius grafų teorijos uždavinius.

Kaip buvo minėta aukščiau, apskaičiuojant grafo jungiamas komponentes, vienas iš uždavinių yra rasti aibę viršūnių, į kurias galima nukeliauti iš pasirinktosios viršūnės, neprilausančios jau apskaičiuotoms jungiosioms komponentėms. Aišku, kad “*orakulas*”, nurodantis tokią viršūnių aibę, yra arba “*paiešką gilyn*” arba “*paieška platyn*”.

Dabar bei tolesniame dėstyme aptarsime paieškos gilyn ir paieškos platyn metodų taikymą, sprendžiant grafų teorijos uždavinius.

## 2.9. Trumpiausių kelių besvoriniame grafe ieškojimo uždavinys

Aptarkime uždavinio: “besvoriniame grafe rasti trumpiausius kelius nuo viršūnės  $s$  iki likusių viršūnių” sprendimą.

Duota:  $n$  – grafo viršūnių skaičius,

$m$  – grafo briaunų skaičius,

grafas, nusakytas masyvais  $L$  [1.. 2 $m$ ] (neorientuotojo grafo atveju)

arba  $L$  [1..  $m$ ] (orientuotojo grafo atveju) ir  $lst$  [1..  $n+1$ ],

$s$  – viršūnė, nuo kurios norime rasti trumpiausius kelius.

Rasti:  $d$  [1.. $n$ ], čia  $d_i = d(s, i)$ , t.y. ilgis trumpiausio kelio nuo viršūnės  $s$  iki viršūnės  $i$ ,

$prec$  [1..  $n$ ], čia

$$prec[i] = \begin{cases} k, & \text{jei kelas } i \text{ viršūnė } i \text{ veda iš viršūnės } k, \\ i, & \text{jei } i \text{ - pradinė kelio viršūnė.} \end{cases}$$

Tuo būdu masyvo  $d$  elementai nusakys trumpiausių kelių ilgius, o masyvo  $prec$  elementai nurodys, per kurias viršūnes šie keliai eina. Šio uždavinio sprendimui panaudosime paiešką platyn, kadangi paieškos platyn  $k$ -tojo žingsnio metu yra nagrinėjamos (aplankomos) viršūnės, nutolusios nuo pradinės paieškos viršūnės atstumu  $k$ . Įvertinant tai, kad atstumas tarp viršūnių yra trumpiausio kelio, jungiančio šias viršūnes, ilgis, nesunku suvokti, kad trumpiausių kelių nuo viršūnės  $s$  iki likusių besvorinio grafo viršūnių uždavinio sprendimo algoritmas natūraliai įsilieja į paieškos platyn algoritmą.

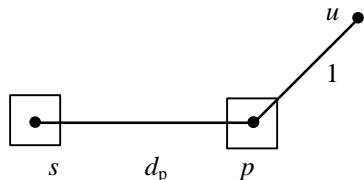
Pradžioje visos grafo viršūnės **naujos**. Tą žymi masyvo  $naujas$  elementas:

$$naujas[i] = \begin{cases} 1, & \text{jei viršūnė } i \text{ nauja,} \\ 0, & \text{priešingu atveju.} \end{cases}$$

Atstumų masyvo  $d$  pradinės reikšmės yra "begalybė". Kadangi besvoriniame grafe nėra nei vieno kelio, kuris būtų ilgesnis už briaunų skaičių  $m$  plius 1, tai pradžioje visi  $d_i = m+1$ ,  $i = \overline{1, n}$ . Vadinas, jei grafas yra nejungasis, tai, įvykdžius žemiau pateiktą procedūrą, masyvo  $d$  elementai, atitinkantys grafo viršūnes, nepriklausančias viršūnės  $s$  jungajai komponentei, bus lygūs  $m+1$ .

Pradžioje visi masyvo  $prec$  elementai yra lygūs nuliui. Taip pat aišku, kad  $d[s] = 0$ , o  $prec[s] = s$ .

Tarkime, kad paieškos platyn  $k$ -ajame žingsnyje nagrinėjame naują viršūnę  $u$ , gretimą  $(k-1)$ -ojo žingsnio viršūnei  $p$  (žr. 2.9.1 pav.).



**2.9.1 pav.** Trumpiausias kelas nuo viršūnės  $s$  iki viršūnės  $u$

Tada, kaip matyti iš 2.9.1 pav.,  $d[u] = d[p] + 1$ , o  $prec[u] = p$ .

Aišku, kad pagal šias formules apskaičiuosime visus masyvo  $d$  ir  $prec$  elementus, atitinkančius paieškos platyn  $k$ -ojo žingsnio naujoms viršūnėms.

Žemiau pateikta trumpiausių kelių besvoriniame grafe apskaičiavimo procedūra, kai eilė organizuojama užciklintu vienmačiu masyvu.

```

const c=500;
type
    mas = array [1..c] of integer;
procedure kelias (s, n, m : integer; L, lst : mas; var d, prec : mas);
{ Procedūra kelias apskaičiuoja trumpiausius kelius nuo viršunės s iki likusių
grafo viršunių besvoriniame grafe, kai grafas nusakytas L ir lst masyvais.
Formalūs parametrai:
    s – pradinė kelio viršunė,
    n – grafo viršunių skaičius,
    m – grafo briaunų (lankų) skaičius,
    L, lst – grafą nusakantys tiesioginių nuorodų masyvai;
    d [1..n] – trumpiausių kelių masyvas;
    d [v] = d(s,v);
    prec [1..n] – trumpiausių kelių medis;
    jei prec [v] = k, tai trumpiausias kelias į viršunę v ateina iš viršunės k. }
var r,f,i,p,u,c : integer;
    naujas ,eile : mas;
begin
    c := n + 1;
    for i := 1 to n do
        begin
            naujas[i] := 1;
            d[i] := m + 1;
            prec [i] := 0;
        end;
    { Eile = Ø }
    r := 1; f := 1;
    { Eile ← s }
    if r = c then r := 1 else r := r + 1;
    if r = f then begin
        writeln( 'eilės persipildymas' );
        exit;
    end
    else eile [r] := s ;
    naujas [s] := 0;
    d [s] := 0;
    prec [s] := s;
    { while eilė netuščia do }
    while r <> f do
        begin

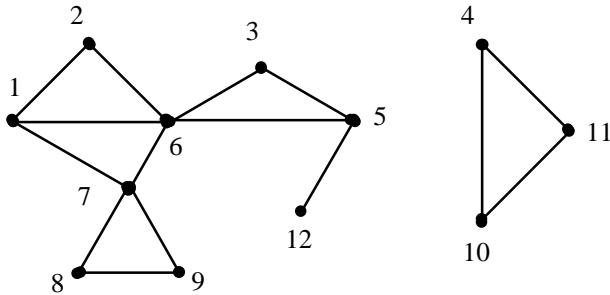
```

```

{  $p \leftarrow eile$  }
  if  $r = f$  then begin
    writeln('eilė tuščia');
    exit;
  end
  else begin
    if  $f = c$  then  $f := 1$ 
    else  $f := f + 1$ ;
     $p := eile[f]$ ;
  end;
for  $i := lst[p] + 1$  to  $lst[p + 1]$  do
begin
   $u := L[i]$ ;
  if  $naujas[u] = 1$  then
  begin
     $d[u] := d[p] + 1$ ;
     $prec[u] := p$ ;
     $naujas[u] := 0$ ;
    {  $eilė \leftarrow u$  }
    if  $r = c$  then  $r := 1$  else  $r := r + 1$ ;
    if  $r = f$  then begin
      writeln ('eilės persipildymas');
      exit;
    end
    else  $eile[r] := u$ ;
  end;
end;
end;
end;

```

**Pavyzdys.** Panagrinėkime trumpiausius kelius nuo 1-osios viršūnės iki likusių grafo, pavaizduoto 2.9.2 pav., viršūnių.



### 2.9.2 pav. Grafo trumpiausi kelai

Po procedūros *kelias* įvykdymo, kai  $s=1$ , gausime tokius  $d$  ir  $prec$  masyvus:

$$d : \begin{array}{cccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ \hline 0 & 1 & 2 & 16 & 2 & 1 & 1 & 2 & 2 & 16 & 16 & 3, \end{array}$$

$$prec : \begin{array}{cccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ \hline 1 & 1 & 6 & 0 & 6 & 1 & 1 & 7 & 7 & 0 & 0 & 5. \end{array}$$

2.9.2 pav. grafas turi 12 viršūnių, 15 briaunų ir nėra jungusis, todėl  $d_4 = d_{10} = d_{11} = m+1 = 16$ . Tai ir reiškia, kad iš 1-osios viršūnės nėra nei vieno kelio, vedančio į šias viršunes.

Elementas  $d_{12} = 3$  reiškia, kad  $d(1,12) = 3$  ir kelias, kaip rodo masyvas  $prec$ , eina per viršunes 12, 5, 6, 1.

**Pastaba.** Procedūra *kelias* leidžia apskaičiuoti visas besvorinio grafo metrines charakteristikas (žr. 2.4 paragrafą).

## 2.10. Dvidalis grafas

Kaip buvo minėta aukščiau,  $(n,m)$ -grafas –  $G = (V,U)$  yra dvidalis, jei jo viršūnių aibę  $V$  galima išskaidyti į du poaibius  $A$  ir  $B$  taip, kad visų grafo  $G$  briaunų galai priklausytų skirtingiemis poaibiams. Todėl dvidalį grafą dažnai žymime  $G = (A,B,U)$ . Čia aptarsime du dvidalio grafo uždavinius:

- dvidalio grafo kriterijaus uždavinį,
- dalinio dvidalio grafo konstravimo uždavinį.

**Dvidalio grafo kriterijaus uždavinys.** Tarkime, kad duotas grafas  $G = (V,U)$ . Nustatyti, ar šis grafas yra dvidalis.

Šį uždavinį 1936 metais išsprendė D.Kionigas.

**Teorema.** (Kionigo teorema). Būtina ir pakankama sąlyga, kad grafas  $G = (V, U)$  būtų dvidalis yra ta, kad jis neturėtų nelyginio ilgio ciklų.

Pateiksime šios teoremos įrodymą, nes šis įrodymas yra konstruktyvus, t.y. iš jo išplaikia algoritmas, leidžiąs nustatyti, ar grafas yra dvidalis.

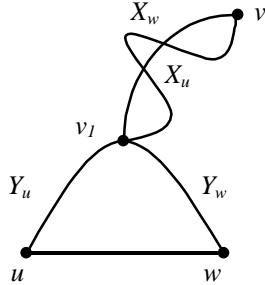
**Būtinumas.** Tarkime,  $G$  – dvidalis grafas, o  $\mu$  – vienas iš jo ciklų, kurio ilgis yra  $k$ . Apeikime visas šio ciklo briaunas jų surašymo tvarka, pradedant viršūne  $v$ . Po  $k$  žingsnių grįsime į viršūnę  $v$ . Kadangi dvidalio grafo kiekvienos briaunos galai priklauso skirtingoms dalims(aibėms), tai  $k$  yra lyginis skaičius.

**Pakankamumas.** Nesiaurinant uždavinio, galime laikyti, kad grafas  $G$  yra jungasis, nes dvidalių grafų sąjunga yra dvidalis grafas.

Tarkime, kad  $(n, m)$ -grafas neturi nelyginio ilgio ciklų, o  $v$  – bet kuri grafo viršūnė. Viršūnių aibę  $V$  į dvi dalis  $A$  ir  $B$  išskaidysime taip: grafo viršūnes  $u$ , kurioms  $d(v, u)$  yra lyginis skaičius, talpinsime į aibę  $A$ , o viršūnes, kurioms  $d(v, u)$  yra nelyginis skaičius – į aibę  $B$ .

Belieka įrodyti, kad šių aibų indukuoti pografiai yra tuštieji.

Tarkime priešingai, kad egzistuoja dvi gretimos viršūnės  $u$  ir  $w$ , kurios priklauso vienai iš aibų. Aišku, kad nei viena iš šių viršūnių nesutaps su viršūne  $v \in A$ , nes visos viršūnėi  $v$  gretimos viršūnės yra aibėje  $B$ .



**2.10.1 pav.** Kionigo teoremos iliustracija

Tarkime, kad  $U$  – trumpiausia  $(u, v)$ -grandinė, o  $W$  – trumpiausia  $(w, v)$ -grandinė. Tarkime, kad šių grandinių paskutinė (skaičiuojant nuo viršūnės  $v$ ) bendra viršūnė yra  $v_1$  (žr. 2.10.1 pav.). Simboliais  $X_u$  ir  $Y_u$  atitinkamai pažymėkime grandinės  $U$  subgrandines  $(v, v_1)$  ir  $(v_1, u)$ , o simboliais  $X_w$  ir  $Y_w$  – grandinės  $W$  subgrandines  $(v, v_1)$  ir  $(v_1, w)$ . Aišku, kad subgrandinių  $X_w$  ir  $X_u$  ilgiai yra lygių. Vadinasi,  $Y_w$  ir  $Y_u$  ilgiai turi tą patį lyginumo charakterį.

Tačiau, tada, sujungus subgrandines  $Y_u$ ,  $Y_w$  ir briauną  $(u, v)$  gausime elementarųjį nelyginio ilgio ciklą.

**Išvada.** Grafas yra dvidalis tada ir tik tai tada, kai jis neturi elementariųjų nelyginio ilgio ciklų.

Kionigo teoremos įrodymas nusako metodą, leidžiantį nustatyti, ar grafas yra dvidalis. Šis metodas susideda iš dviejų etapų.

1. Pasirenkama bet kuri grafo viršūnė  $s$ , ir 2.9 paragrafe aprašytois procedūros **kelias** pagalba randame atstumus nuo viršūnės  $s$  iki visų likusių viršūnių. Tada viršūnės, kurioms šis atstumas yra lyginis skaičius, talpinamos į aibę  $A$ , o likusios – į aibę  $B$ .
2. Tikriname kiekvieną grafo briauną  $(u, v)$ , ar jos galai priklauso skirtinoms aibėms.

Jei visoms grafo briaunoms  $(u, v)$  viršūnės  $u$  ir  $v$  priklauso skirtinoms aibėms ( $A$  ir  $B$ ), tai grafas yra dvidalis, t.y.  $G = (A, B, U)$ .

Jei egzistuoja bent viena briauna  $(u, v)$ , kurios abi viršūnės  $u$  ir  $v$  priklauso arba aibei  $A$ , arba aibei  $B$ , tai grafas  $G$  nėra dvidalis.

Realizuoti 2-ąjį etapą galima įvairiais būdais. Labai patogus šio etapo realizavimo būdas būtų tokis.

Apibrėžkime masyvą  $ab[1..n]$ , čia  $ab[i] = 0$ , jei viršūnė  $i$  priklauso aibei  $A$  (jei procedūros **kelias** atstumų masyvo  $d[1..n]$  elementas  $d[i]$  yra lyginis skaičius), ir  $ab[i] = 1$ , jei viršūnė  $i$  priklauso aibei  $B$  ( $d[i]$  – nelyginis skaičius).

Tada briaunos  $(u, v)$  galai priklausys skirtinoms aibėms, jei  $ab[u] \neq ab[v]$ ; priešingu atveju viršūnės  $u$  ir  $v$  priklausys vienai aibei.

**Pastaba.** Ar briaunos  $(u, v)$  galai priklauso skirtinoms aibėms, galima nustatyti tiesiogiai iš procedūros **kelias** masyvo  $d[1..n]$ :

*if (( $d[n] + d[v]$ ) mod 2) = 0 then "briaunos galai priklauso tai pačiai aibei, t.y. grafas nėra dvidalis".*

**Dalinio dvidalio grafo konstravimo uždavinys.** Duotas grafas  $G(V, U)$ .

Rasti šio grafo dalinių grafų, atmetant dalį pradinio grafo briaunu taip, kad atmetamų briaunu skaičius būtų nedidesnis nei  $\lfloor m/2 \rfloor$ , čia  $m = |U|$ .

Toks dalinis grafas konstruojamas taip:

$A := \emptyset$  ;  $B := \emptyset$  ;

$A := A \cup \{s \mid s - \text{bet kuri grafo viršūnė}\}$  ;

for ( $v \in V$ ) and ( $v \neq s$ ) do

*if*  $|A \cap N(v)| < |B \cap N(v)|$  *then*  $A := A \cup \{v\}$

*else*  $B := B \cup \{v\}$ ;

čia  $N(v)$  – viršūnės  $v$  aplinka, t.y. viršūnei  $v$  gretimų viršūnių aibė.

Tada  $|A \cap N(v)|$  parodo atmetamų briaunų skaičių, jei viršūnė  $v$  talpiname į aibę  $A$ , o  $|B \cap N(v)|$  – atmetamų briaunų skaičių, jei viršūnė  $v$  talpintume į aibę  $B$ . Vadinasi, skaidant aibę  $V$  į dvi aibes  $A$  ir  $B$ , kiekvieną kartą viršūnę  $v$  talpinsime į tą aibę, kad atmetamų briaunų skaičius būtų mažiausias ir neviršytų  $\lfloor d(v)/2 \rfloor$ , čia  $d(v)$  –  $v$ -osios viršūnės laipsnis.

Aišku, kad taip konstruojant dalinį dvidalį grafą, atmetamų briaunų skaičius bus nedidesnis nei  $\lfloor m/2 \rfloor$ .

## 2.11. Pagrindiniai grafų teorijos skaičiai

Pagrindiniai grafų teorijos skaičiai yra:

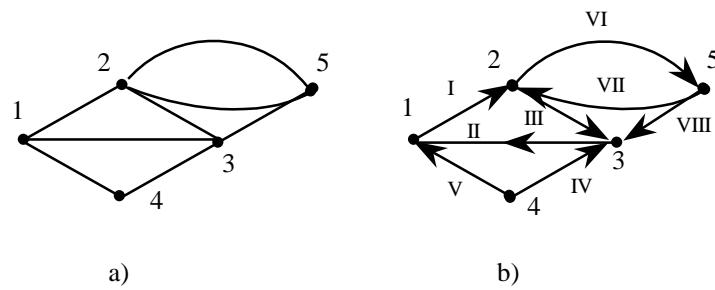
- 1) **ciklomatinis** skaičius,
- 2) **chromatinis** (spalvinis) skaičius,
- 3) **nepriklausomumo** (vidinio stabilumo) skaičius,
- 4) **dominavimo** (išorinio stabilumo) skaičius.

### 2.11.1. Ciklomatinis skaičius

Nagrinėsime neorientuotuosius grafus, kurie gali būti ir multigrafai. Grafo  $G = (V, U)$  **ciklomatinis skaičius**  $\nu(G)$  apibrėžiamas formule

$$\nu(G) = m - n + p,$$

čia  $m$  – briaunų skaičius,  $n$  – viršūnių skaičius, o  $p$  – jungiujų komponenčių skaičius. Pavyzdžiui, grafui, pavaizduotam 2.11.1 pav., ciklomatinis skaičius  $\nu(G) = 8 - 5 + 1 = 4$ .



**2.11.1 pav.** Grafo ciklomatinis skaičius

Kiekvienam grafo ciklui galima priskirti  $m$ -matij vektorių tokiu būdu:

1. sunumeruokime briaunas ir kiekvienai briaunai laisvai suteikime orientaciją (žr. 2.11.1 b) pav.);
2. jei apeinant ciklą,  $k$ -toji briauna  $(u, v)$  rodyklės kryptimi praeinama  $s$  kartu, o pries rodyklę –  $t$  kartu, tai šiam ciklui atitinkančio vektoriaus  $k$ -oji komponentė lygi  $s - t$ .

Pavyzdžiu, 2.11.1 b) pav. ciklą  $\mu_1 = (1,2,5,3,1)$  atitiks vektorius

$$c_1 = \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \end{array} \quad \text{o} \quad \text{ciklą} \quad \mu_2 = (1,3,4,1) \quad - \quad \text{vektorius}$$

$$c_2 = \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 0 & -1 & 0 & -1 & 1 & 0 & 0 & 0 \end{array}.$$

Nepriklausomiems vektoriams atitinkantys ciklai yra nepriklausomi.

**Teorema.** Multigrafo  $G$  ciklomatinis skaičius lygus didžiausiam nepriklausomų ciklų skaičiui.

#### Išvados.

- Grafas  $G$  neturi ciklų tada ir tik tada, kai  $v(G) = 0$ .
- Grafas  $G$  turi vienintelį ciklų tada ir tik tai tada, kai  $v(G) = 1$ .

Pavyzdžiu, 2.11.1 a) grafui nepriklausomų ciklų aibė (bazė) yra ciklai:

$$\mu_1 = (1,2,3,1), \mu_2 = (1,3,4,1), \mu_3 = (2,5,2), \mu_4 = (2,5,3,2).$$

Aišku, kad nepriklausomų ciklų rinkinys yra nevienintelis. Pavyzdžiu, tam pačiam 2.11.1 a) grafui nepriklausomų ciklų bazę sudaro ir ciklai  $\mu_1 = (1,2,5,3,1)$ ,  $\mu_2 = (2,5,3,2)$ ,  $\mu_3 = (2,5,2)$ ,  $\mu_4 = (1,3,4,1)$ . Šie ciklai yra tiesiskai nepriklausomi, nes kiekvienas iš jų turi bent po vieną unikalą briauną, nepriklausančią likusiems ciklams. Ciklas  $\mu_1$  turi unikalą briauną (1,2),  $\mu_2$  – (3,2),  $\mu_3$  – (5,2) ir  $\mu_4$  – (3,4) arba (4,1).

**Nepriklausomų ciklų apskaičiavimo uždavinys.** Duotas jungusis neorientuotasis  $(n,m)$ -grafas (ne multigrafas)  $G = (V, U)$ . Rasti nepriklausomų ciklų bazę.

Įveskime **atvirkštinės briaunos** sąvoką. Tarkime, kad  $H = (V, U_H)$  yra grafo  $G$  dalinis grafas, neturintis ciklų. Toks dalinis grafas vadinamas dengiančiu medžiu. (Apie medžius bus kalbama 2.13 paragafe). Dengiantis medis turi  $(n-1)$  briauną ir neturi ciklų.

**Apibrėžimas.** Grafo  $G$  briauna  $(u, v)$  yra **atvirkštinė briauna (styga)**, jei ji neprikouso dengiančio medžio  $H$  briaunų aibei.

Kadangi  $|U_H| = n - 1$ , tai atvirkštinių briaunų skaičius yra lygus  $m - n + 1$ , t.y. kiekviena atvirkštinė briauna  $e = (u, v)$  drauge su dengiančio medžio briaunomis, priklausančiomis grandinei, jungiančiai viršūnę  $u$  su viršūne  $v$ , sudaro vieną nepriklausomą grafo  $G$  ciklą. **Šį ciklą žymėsime  $C_e$ .**

Vadinasi, kiekvienas grafo  $G$  dengiantis medis apibrėž nepriklausomų ciklų aibę. Be to, neizomorfinių medžių nepriklausomų ciklų aibės bus skirtingos.

**Pastaba.** Taip apibrėžus nepriklausomus ciklus, visus kitus grafo ciklus galima išreikšti per nepriklausomų ciklų simetrinį skirtumą (sumą moduliu 2).

**Apibrėžimas.** Grafo  $G = (V, U)$  briaunų aibė  $C$  vadinama **pseudociklu**, jei grafo  $(V, C)$  visų viršūnių laipsniai yra lyginiai.

**Tuščia aibė ir bet koks grafo ciklas yra pseudociklas.**

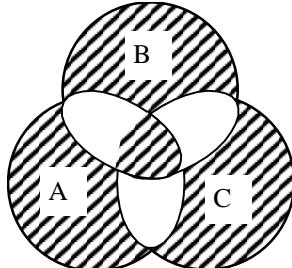
Priminsime aibių **simetrinio skirtumo** operaciją:

$$A \oplus B = (A \cup B) \setminus (A \cap B).$$

Šią operaciją taikysime didesniams aibių  $A_1, A_2, \dots, A_k$  skaičiui:

$$\bigoplus_{i=1}^k A_i = A_1 \oplus A_2 \oplus \dots \oplus A_k.$$

Nesunku įsitikinti, kad aibių  $A_1, A_2, \dots, A_k$  simetrinį skirtumą, nepriklausomai nuo skliaustų, nurodančių veiksmų atlikimo tvarką, sudėjimo sudarys tik tie elementai, kurie priklauso nelyginiam skaičiui poaibių  $A_i$  (žr. 2.11.2 pav.)



**2.11.2 pav.** Aibių  $A, B$  ir  $C$  simetrinis skirtumas  $A \oplus B \oplus C$

**Lema.** Bet koks skaičiaus pseudociklų simetrinis skirtumas yra pseudociklas.

**Irodymas.** Aišku, kad pakanka panagrinėti dviejų pseudociklų  $C_1$  ir  $C_2$  atvejį.

Simboliais  $S_1(v)$ ,  $S_2(v)$  ir  $S(v)$  pažymėkime viršūnei  $v$  incidentiškų briaunų skaičių atitinkamai pseudocikluose  $C_1$ ,  $C_2$  ir  $C = C_1 \oplus C_2$ . Aišku, kad aibę  $S_1(v)$  ir  $S_2(v)$  elementų skaičius yra lyginis. Tada

$|S(v)| = |S_1(v) \oplus S_2(v)| = |S_1(v)| + |S_2(v)| - 2|S_1(v) \cap S_2(v)|$   
taip pat yra lyginis skaičius, ir  $C$  yra pseudociklas.

**Teorema.** Tarkime,  $G = (V, U)$  – junginis neorientuotasis grafas, o  $(V, T)$  – jį dengiantis medis. Tada bet koks grafo  $G$  ciklas  $C$  vienareikšmiškai išreiškiamas per nepriklausomus ciklus  $C_e$  taip:

$$C = \bigoplus_{e \in C \setminus T} C_e.$$

**Irodymas.** Simetrinio skirtumas  $\bigoplus_{e \in C \setminus T} C_e$ , remiantis lema, yra pseudociklas, susidedantis iš atvirkštinų briaunų (stygų) aibės  $C \setminus T$  ir kažkokiu medžio  $T$  briaunu. Tai išplaukia iš to faktro, kad kiekviena atvirkštinė briauna  $e \in C \setminus T$  priklauso tik vienam nepriklausomam ciklui  $C_e$ .

Aibė  $C \oplus \bigoplus_{e \in C \setminus T} C_e$ , remiantis lema, taip pat nusakys pseudociklą, kurį gali sudaryti tik medžio  $T$  briaunos. (Pseudociklo  $C$  atvirkštinės briaunos  $e$  priklauso ir pseudociklui  $\bigoplus_{e \in C \setminus T} C_e$ , todėl jos nepriklausys aibei  $C \oplus \bigoplus_{e \in C \setminus T} C_e$ ). Kadangi medis  $T$  neturi ciklų, tai šis pseudociklas yra tuščioji aibė. Vadinasi,

$C = \bigoplus_{e \in C \setminus T} C_e$ . šios formulės vienareikšmiškumas išplaukia iš to, kad ciklas  $C_e$  yra vienintelis nepriklausomas ciklas, turintis atvirkštinę briauną  $e$ .

Iš šio nagrinėjimo išplaukia, kad grafo nepriklausomus ciklus patogu ieškoti, naudojant paieškos gilyn metodą, aptartą 2.8.2 paragrafe.

Kaip paieškos gilyn metodu formaliai nustatyti **atvirkštinę briauną (stygą)?**

Tam tikslui reikia žinoti viršūnių aplankymo eilės numerius. Įveskime masyvą  $nr[1..n]$ , čia  $nr[i]$  yra  $i$ -osios viršūnės aplankymo eilės numeris.

Tarkime, kad paieškos gilyn metu iš viršūnės  $u$  atejome į viršūnę  $v$ . Briauna  $(u, v)$  bus atvirkštinė briauna (styga) ir išsauks nepriklausomą ciklą, jei bus teisinga sąlyga: “**viršūnė  $v$  nenauja**” and “**į viršūnę  $u$  buvome ateję ne iš viršūnės  $v$** ” and “ **$nr[v] < nr[u]$ , t.y. viršūnė  $v$  buvo aplankyta anksčiau nei viršūnė  $u$** ”.

2.8.1.2 paragrafe aprašytam paieškos gilyn algoritme ši sąlyga bus užrašoma taip: (***prec*[*v*] ≠ 0) and (***prec*[*u*] ≠ *v*) and (***nr*[*v*] < *nr*[*u*])**).****

Žinant šią atvirkštinės briaunos sąlygą nesunku modifikuoti 2.8.1.2 paragrafe aprašytą paieškos gilyn metodą taip, kad jis apskaičiuotų nepriklausomus cilus. Žemiau ir pateikta ši nepriklausomų ciklų apskaičiavimo procedūra.

```

const c = 500;
type
    mas = array [1..c] of integer;
procedure ciklai(v, n, m : integer; L, lst : mas);
{ Procedūra ciklai, naudodama paiešką gilyn iš viršūnės v, kai grafas
nusakytas L ir lst masyvais, apskaičiuoja grafo nepriklausomus ciklus.
Formalūs parametrai:
    v – pradinė paieškos viršūnė,
    n – grafo viršūnių skaičius,
    m – grafo briaunų (lankų) skaičius,
    L, lst – grafą nusakantys tiesioginių nuorodų masyvai. }
var i, k, u : integer;
    sk, j : integer;
    t, p : boolean;
    fst, prec : mas;
    nr : mas;
begin
    { Inicjalizacija }
    for i:=1 to n do begin
        fst[i]:=lst[i] + 1;
        prec[i]:=0;
        end;
        k := v;
        if fst[k] <= lst[k+1] then {yra nenagrinetų briaunų, incidentiškų viršūnei k}
            begin
                t := false;
                p := true;
                prec[k]:=k; { k – pradinė paieškos viršūnė }
                { Nagrinėti viršūnę k }
                nr[k]:=1;
                sk := 1; { Aplankytų viršūnių skaičius }
            end

```

```

else {viršūnė k yra arba izoliuota viršūnė, arba neturi išeinančių
lankų (orientuotojo grafo atveju); paieškos pabaiga }
t := true;
while not t do { paieška nebaigta}
begin
{ Pirmyn }
while p do
begin
u := L [fst [k]];
if prec [u] = 0 then {viršūnė u nauja}
begin
{ Nagrinėti viršūnę u }
sk := sk + 1;
nr [u] := sk;
prec [u] := k;{ i viršūnę u atėjome iš viršūnės k }
if fst [u] <= lst [u + 1] then { viršūnė u neišsemta }
k:=u
else {viršūnė u išsemta} p:=false;
end
else
begin
p := false; { viršūnė u nenauja}
if (prec [k] <> u) and (nr [k] > nr [u]) then
{ Briauna (k, u) yra atvirkštinė briauna. Spausdinti ciklą }
begin
writeln;
write (u : 3);
write (k : 3);
j := k;
while prec [j] <> u do
begin
j := prec [j];
write (j : 3);
end;
write(u : 3);
writeln;
end;
end;
end;
{ Atgal }
while not p and not t do

```

```

begin
    {Imama nauja, dar nenagrinėta briauna, incidentiška viršūnei k}
    fst [k] := fst [k] + 1;
    if fst [k] <= lst[k + 1] then { tokia briauna egzistuoja }
        p := true
        else { viršūnė k išsemta }
        if prec [k] = k then { pradinė paieškos viršūnė
            išsemta; paieškos pabaiga } t := true
            else { grįžome į viršūnę, iš kurios buvome atėję
                į viršūnę k } k := prec [k];
    end;
end;
end;

```

Aptarkime dar vieną nepriklausomų ciklų apskaičiavimo, naudojant rekursiją, procedūrą.

Šią procedūrą patogu organizuoti naudojant steką (žr. 2.8.1.3 paragrafą).

Tarkime, kad grafas  $G = (V, U)$  nusakytas gretimumo struktūra, t.y.  $N(v)$  –aibė viršūnių, gretimų viršūnei  $v$ .

```

const c = 500;
type
    mas = array [0..c] of integer;
    matr = array [1..2, 1..c] of integer;
procedure ncikl (v : integer);
{ Grafo G jungiosios komponentės, turinčios savyje viršūnę v, nepriklausomų
ciklų apskaičiavimas.
Kintamieji num, top, stec, nr, L, lst – globalieji }
var u, i, top1 : integer;
begin
    top := top +1;
    stek [top] := v;
    num := num + 1;
    nr [v] := num;
    for i := lst [v] + 1 to lst [v + 1] do
        begin
            u := L [i];
            if nr [u] = 0 then ncikl (u)
            else
                if (nr [v] > nr [u]) and (u <> stek [top - 1]) then

```

```

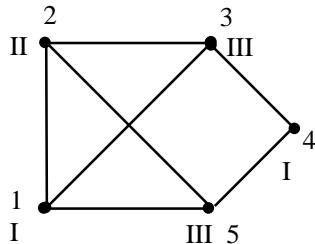
{ Briauna (v,u) – atvirkštinė briauna. Išsiminti ciklą, einantį per
viršūnes: u, stek [top], stek [top - 1], ..., stek [c], čia stek [c] = u.
}
begin
    writeln;
    write (u : 3);
    top1 := top;
    while stek [top1] <> u do
        begin
            write (stek[top1]:3);
            top1:=top1-1;
        end;
        write (u : 3);
        writeln;
    end;
end;
top := top - 1; { Išsemta viršūnė v šalinama iš steko. }
end; { ncikl }

```

### 2.11.2. Chromatinis skaičius

$(n,m)$ -grafo  $G = (V, U)$  **chromatinis skaičius** yra mažiausias skaičius spalvų, kurioms grafo viršūnes galima nudažyti taip, kad bet kokios dvi gretimos viršūnės būtų nudažytos skirtine spalva. Šį skaičių žymėsime  $\gamma(G)$ .

Pavyzdžiu, 2.11.3 pav. grafui chromatinis skaičius yra 3. Aišku, kad šio grafo negalima nudažyti su mažesniu spalvų skaičiumi, nes jis turi ilgį 3 ciklų.



**2.11.3 pav.** Grafo chromatinis skaičius

Aišku, kad  $K_n$  grafo chromatinis skaičius lygus  $n$ , o bet koks dvidalis grafas  $G = (A, B, U)$  yra bichromatusis: aibės  $A$  viršūnės dažomos pirmaja spalva, o aibės  $B$  viršūnės – antraja spalva.

Perfrazavus Kionigo teoremą, galime teigti, kad grafas yra bichromatusis tada ir tikta tada, kai jis neturi nelyginio ilgio ciklų.

Chromatinio skaičiaus apskaičiavimo uždavinys yra diskrečiojo optimizavimo uždavinys. Formaliai jį galima užrašyti taip.

$\min z = p$ , čia  $p$  – spalvų skaičius,  
esant apribojimams:

- kiekviena viršūnė turi būti dažoma viena spalva,
- bet kokios gretimos viršūnės turi būti nudažytos skirtinė spalva.

Šiemis apribojimams užrašyti įvesime kintamuosius

$$x_{ij} = \begin{cases} 1, & \text{jei } i\text{-toji viršūnė dažoma } j\text{-ają spalva,} \\ 0, & \text{priešingu atveju,} \end{cases}$$

$$i = \overline{1, n}, \quad j = \overline{1, p}.$$

Tada a) apribojmas bus užrašomas taip:

$$\sum_{j=1}^p x_{ij} = 1, \quad i = \overline{1, n},$$

o b) apribojimas –

$$\sum_{i=1}^n x_{ij} \cdot a_{ik} \leq 1, \quad j = \overline{1, p}, \quad i = \overline{1, m},$$

čia  $a_{ik}$  – incidencijų matricos (žr. 2.7 paragrafą) elementas.

Pavyzdžiu, 2.11.3 pav. grafui optimalus šio uždavinio sprendinys yra:

$$X = \begin{matrix} & \text{I} & \text{II} & \text{III} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \left[ \begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{array} \right] \end{matrix}.$$

Kaip buvo minėta aukščiau, šis uždavinys yra NP pilnasis [GD82] ir neturi efektyvių sprendimo algoritmų. todėl jis sprendžiamas naudojant įvairias euristikas.

Dažniausiai naudojami grafių dažymo algoritmai remiasi tokiomis euristikomis:

- pirma spalva, po to viršūnė,

- pirma viršūnė, po to spalva.
- Detaliu aptarkime šiuos euristinius algoritmus.

#### **Euristinis algoritmas, paremtas principu “pirma spalva, o po to viršūnė”**

Šios euristikos idėja yra labai paprasta. Pagal kokį nors požymį sudaroma grafo viršūnių seka. Po to imama nauja spalva ir šia spalva iš eilės dažomos, jeigu galima, sekos viršūnės. Taip elgiamės iki nudažome visas grafo viršūnes.

Tuo būdu šis algoritmas susideda iš dviejų etapų.

1. Grafo viršūnės išrikiuojamos jų laipsnių mažėjimo tvarka  
 $v_1, v_2, \dots, v_n$ , čia  $v_i \in V$  ir  $d(v_{i+1}) \leq d(v_i)$ ,  $i = \overline{1, n-1}$ .
2.  $p := 0$ ; {spalvų skaičius}

while “yra nenudažytų viršūnių” do

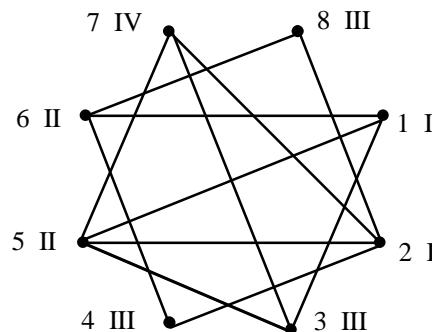
*begin*

$p := p + 1$ ;

*Pradedant pirmaja nenudažyta sekos viršūne,  $p$  spalva nuosekliai viena po kitos dažomos, jei galima, nenudažytos sekos viršūnės.*  
*end;*

**Pavyzdys.** Panagrinėkime grafo, pavaizduoto 2.11.4 pav., dažymą, laikantis principo “pirma spalva, o po to viršūnė”.

2.11.4 pav. pavaizduotam grafui seka viršūnių, išrikuotų jų laipsnių mažėjimo tvarka, yra: 2, 5, 1, 3, 6, 7, 4, 8.



**2.11.4 pav.** Grafo dažymas laikantis principo “pirma spalva, o po to viršūnė”

Pirmaja spalva dažome 2-ąją viršūnę, po to pirmaja spalva galime dažyti tik 1-ąją viršūnę. Gausime:

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

I            I

Imame antrają spalvą ir dažome pirmają nenudažytą sekos viršūnę – 5-ąją, po to antraja spalva galime dažyti 6-ąją viršūnę. Gausime:

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

I   II   I      II

Trečiajai spalva pirmiausia dažome 3-ąją viršūnę, po to – 4-ąją viršūnę ir, galiausiai, 8-ąją viršūnę. Gausime:

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

I   II   I   III   II      III   III

Kadangi dar yra nenudažytų viršūnių, tai imame IV-ąją spalvą ir dažome 7-ąją viršūnę, t.y. gausime:

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

I   II   I   III   II   IV   III   III

Gautas sprendinys nėra optimalus. Ši grafa galima nudažyti trimis spalvomis (žr. euristiką “pirma viršūnė, o po to spalva”).

Žemiau pateikta grafo dažymo, naudojant euristiką “pirma spalva, o po to viršūnė” procedūra.

```

const c = 500;
type
    mas = array [1..c] of integer;
    matr = array [1..2, 1..c] of integer;
procedure grfdaz1 (n, m : integer; b : matr; var p: integer; var d : mas);
{ Procedūra grfdaz1 dažo grafo viršunes, remdamasi euristika “pirma spalva,
o po to viršūnė”.
Formalūs parametrai:
    n – grafo viršūnių skaičius,
    m – grafo briaunų (lankų) skaičius,
    b – grafo briaunų matrica ,
    p – spalvų skaičius,
    d [1..n] – viršūnių spalvų masyvas;
    jei d [i] = k, tai reiškia, kad i-oji viršūnė dažoma k-qaja spalva. }
    var i, k, z, sv, u,j ,x : integer;
    t : boolean;
    L, lst, v, s : mas;
begin
    BLst(n,m,b,L,lst);
{ Pradinių reikšmių suteikimas darbo masyvams ir kintamiesiems }
```

```

for i := 1 to n do
begin
    v [i] := i; { Masyve v iš eilės surašomi viršūnių numeriai }
    s [i] := lst [i + 1] - lst [i]; {s [i] – i-tosios viršnės laipsnis }
    d[i] := 0;
end;
{ Viršūnes masyve v išrikuojame jų laipsnių mažėjimo tvarka }
for k := 1 to n - 1 do
for i := 1 to n - k do
if s [i] < s [i + 1] then { Keičiame vietomis s [i] su s [i + 1] ir v [i] su v [i + 1] }
begin
    z := s [i]; s [i] := s [i + 1]; s [i + 1] := z;
    z := v [i]; v [i] := v [i + 1]; v [i + 1] := z;
end;
p := 0; { p – spalvų skaičius }
sv := 0; { sv - nudažytų viršūnių skaičius }
while sv < n do
begin
    p := p + 1;
    for i:=1 to n do
begin
    u := v [i];
    if d [u] = 0 then { Viršūnė u – nenudažyta.
    Ar ją galima dažyti p-qja spalva? }
begin
    { Ar viršūnių, gretimų viršūnei u, tarpe yra
    viršūnė,nudažyta p-qja spalva? }
    j := lst [u] + 1; t := false;
    { Jei t = false, tai viršūnė u galima dažyti p-qja spalva }
    while (j <= lst [u + 1]) and not t do
begin
    x := l [j];
    if d[x] = p then t:=true
    else j:=j+1;
end;
    if not t then
begin
    d [u] := p;

```

```

        sv := sv + 1;
    end;
end;
end;
end;

```

### **Euristinės algoritmas, paremtas principu "pirma viršūnė, o po to spalva"**

Šios euristikos idėja yra ta, kad pirmiausia pagal kažkokį kriterijų išrenkama grafo viršūnė, o po to ji dažoma, jei galima, viena iš anksčiau panaudotų spalvų; jei išrinktos viršūnės negalima nudažyti nei viena iš anksčiau panaudotų spalvų, tai ji dažoma nauja spalva.

#### **Viršūnės išrinkimo kriterijus**

**Apibrėžimas.** Viršūnės  $h$  laipsnis – tai skaičius spalvų, kuriomis šios viršūnės negalima dažyti, t.y. šiai viršūnei negalimų spalvų skaičius.

**Apibrėžimas.** Viršūnės  $k$  laipsnis – tai šiai viršūnei gretimų nenudažytų viršūnių skaičius.

Pirmiausia pasirinksime viršūnę, kurios  $h$  laipsnis yra didžiausias.

Jei yra kelios viršūnės su tuo pačiu didžiausiu  $h$  laipsniu, tai iš jų išsirinksime viršūnę, kurios  $k$  laipsnis yra didžiausias.

Jei ir šiuo atveju bus daugiau nei viena viršūnė, imsime viršūnę, kurios numeris mažiausias.

Tuo būdu algoritmas aprašomas taip.

```

begin
p := 0;
while "yra nenudažytų viršūnių" do
begin
    1) pagal aukščiau aprašytą kriterijų parenkama nenudažyta viršūnė v;
    2) jei galima, viršūnę v dažome viena iš anksčiau panaudotų spalvų
        (paprastai, pirmąja iš galimų), t.y. viena iš aibės {1,2,...,p} galimų
        spalvų;
    3) jei viršūnės v negalima nudažyti nei viena iš anksčiau panaudotų
        spalvų, tai p := p + 1, ir viršūnę v dažome p-qją spalva;
    4) visoms nenudažytoms viršūnėms perskaičiuojame h ir k laipsnius;
end;
end;

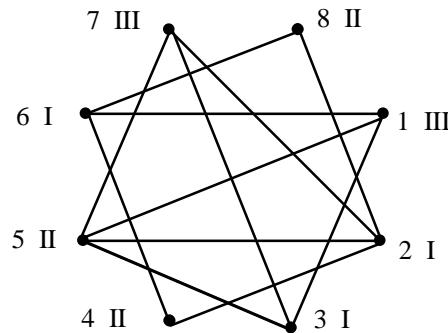
```

**Pavyzdys.** Panagrinėkime, kaip bus dažomos 2.11.4 pav. grafas. Skaičiavimai parodyti 2.11.1 lentelėje, o 2.11.5 pav. pavaizduotas nudažytas grafas.

**2.11.1 lentelė.** Grafo dažymas

NR	$h$ – laipsnis	$k$ – laipsnis	spalva	*
1	$\theta \pm 2$	$3 \pm 1$	III	5
2	0	4	I	1
3	$\theta \pm 2$	$3 \pm 1$	I	4
4	$\theta 1$	$\pm 1$	II	6
5	$\theta 1$	4 3	II	2
6	$\theta \pm 2$	$3 \pm 1$	I	7
7	$\theta \pm 2$	$3 \pm 1$	III	3
8	$\theta 1$	$\pm 0$	II	8

čia \* – viršūnės išrinkimo eilės numeris.



**2.11.5 pav.** Grafo dažymas laikantis principo “pirma viršūnė, o po to spalva”

Žemiau pateikta grafo dažymo, naudojant euristiką “pirma viršūnė, o po to spalva” procedūra.

*const c = 500;*

*type*

*mas = array [1..c] of integer;*

*matr = array [1..2, 1..c] of integer;*

*procedure grfdaz2(n, m : integer; b : matr; var p : integer; var d : mas);*

*{ Procedūra grfdaz2 dažo grafo viršūnes, remdamasi euristika “pirma viršūnė, o po to spalva”}.*

*Formalūs parametrai:*

*n – grafo viršūnių skaičius,*

$m$  – grafo briaunų (lankų) skaičius,  
 $b$  – grafo briaunų matrica ,  
 $p$  – spalvų skaičius,  
 $d [1..n]$  – viršūnių spalvų masyvas;  
 jei  $d [i] = k$ , tai reiškia, kad i-oji viršūnė dažoma k-qja spalva. }  
 var  $i, sv, max, maxka, v, u, j, x, sp$  : integer;  
 $t,st$  : boolean;  
 $L,lst,h,ka$  : mas;  
 begin  
**BLst** ( $n, m, b, L, lst$ );  
 { Pradinių reikšmių suteikimas darbo masyvams ir kintamiesiems }  
 for  $i := 1$  to  $n$  do  
     begin  
          $ka [i] := lst [i + 1] - lst [i]; \{ ka [i] – i\text{-tosios viršūnės laipsnis } \}$   
          $d[i] := 0;$   
          $h[i] := 0;$   
     end;  
      $p := 0; \{ p – spalvų skaičius \}$   
      $sv := 0; \{ sv – nudažytų viršūnių skaičius \}$   
     while  $sv < n$  do  
         begin  
             { Viršūnės išrinkimas }  
              $max := -1;$   
             for  $i := 1$  to  $n$  do  
                 if ( $d [i] = 0$ ) and ( $max < h [i]$ ) then  $max := h [i];$   
                  $maxka := -1;$   
                 for  $i := 1$  to  $n$  do  
                     if ( $d [i] = 0$ ) and ( $h [i] = max$ ) and ( $maxka < ka [i]$ ) then  
                         begin  
                              $maxka := ka [i];$   
                              $v := i;$   
                         end;  
                     { Kuria spalva dažyti viršūnę  $v$  ? }  
                      $sv := sv + 1; \{ Dažome viršūnę v \}$   
                      $t := false; \{ t = false, jei viršūnei v spalva neparinkta \}$   
                      $i := 1; \{ Bandome viršūnę v dažyti spalva i \}$   
                     while ( $i <= p$ ) and ( $not t$ ) do  
                         begin  
                             { Tikriname, ar viršūnei v gretimų viršūnių tarpe yra viršūnė, kuri  
                             nudažyta i-aja spalva }  
                              $j := lst [v] + 1;$

```

st := false; { Jei st = false, tai reiškia, kad viršūnei v gretimų
viršūnių tarpe spalvos i nėra }
while (j <= lst [v + 1]) and (not st) do
begin
    u := L [j];
    sp:=d[u];
    if i = sp then st := true
    else j:=j+1;
end;
if st then i := i + 1 { Bandome naują spalvą }
else t:=true; { Viršūnė v gali būti dažoma i-qja spalva }
end;
if t then d [v] := i { Viršūnę v dažome i-qja spalva }
else begin
    p := p + 1; { Ivedame naują spalvą }
    d [v] := p; { Viršūnę v dažome nauja spalva }
end;
{ h ir ka laipsnių perskaičiavimas nenudažytoms viršūnėms, gretimoms
viršūnei v }
for i:=lst[v]+1 to lst[v+1] do
begin
    u := L [i];
    if d [u] = 0 { Viršūnė u , gretima viršūnei v, – nenudažyta } then
begin
        ka[u] := ka [u] – 1;{ Viršūnei u gretimų nenudažytų viršūnių
skaičius sumažėjo }
        j := lst [u] + 1;
        t := false; { Nei viena iš viršūnei u gretimų viršūnių
nenudažyta d [v] spalva }
        while (j <= lst [u + 1]) and ( not t ) do
begin
            x:=L[j];
            if (x <> v) and (d [x] = d [v]) then t := true
            else j := j + 1;
end;
        if not t { Viršūnei u gretimų nudažytų viršūnių x tarpe nėra
spalvos, lygios d[v] }

```

```

        then  $h[u] := h[u] + 1;$ 
    end;
end;
end;

```

Uždavinys, analogiškas grafo viršūnių dažymo uždaviniui, yra briaunu dažymo uždaviny.

Mažiausias skaičius spalvą, kuriomis grafo briaunas galima nudažyti taip, kad bet kurios dvi gretimos briaunas būtų nudažytos skirtinga spalva, vadinamas grafo **chromatine klase**.

Aišku, kad  $(n,m)$ -grafo  $G$  chromatinė klasė yra lygi šiam grafui atitinkančio briauninio grafo (žr. 2.5 paragrafą) chromatiniam skaičiui.

Vadinasi, aukščiau aptarti chromatino skaičiaus apskaičiavimo algoritmai tinkta ir grafo chromatinei klasei apskaičiuoti. Tik šiuo atveju reikia nagrinėti grafui  $G$  atitinkanti briauninį grafą.

Reikia pabrėžti, kad eilė praktinių uždavinių yra formuluojami kaip grafo dažymo uždaviniai. Šiemis uždaviniams būdinga tai, kad kai kurie objektai dėl vienokių ar kitokiu priežasčių negali būti jungiami į grupes.

**Pirmas pavyzdys.** Tarkime, kad per galimai trumpiausią laiką reikia perskaityti keletą paskaitų. Kiekvienos paskaitos trukmė yra 1 valanda ir kai kurias paskaitas skaito tas pats lektorius. Sudarykime grafą  $G$ , kurio viršūnės vaizduoja paskaitas, o dvi viršūnės yra gretimos, jei jų negalima skaityti vienu metu, t.y. jei jas skaito tas pats lektorius. Aišku, kad bet koks teisingas grafo nudažymas apibrėžia paskaitų skaitymo tvarkaraštį, t.y. paskaitos, atitinkančios viršūnėms, kurios nudažytos ta pačia spalva, gali būti skaitomos vienu metu. Vadinasi, grfo chromatinis skaičius bus lygus mažiausiam valandų skaičiui, kuris reikalingas perskaityti paskaitų ciklą.

**Antras pavyzdys.** Duota darbų  $V = \{v_1, v_2, \dots, v_n\}$  aibė ir mechanizmų  $S = \{s_1, s_2, \dots, s_m\}$  aibė. Visų darbų trukmės yra lygios, o darbams atliliki naudojami aibės  $S$  mechanizmai. Aišku, kad tas pats mechanizmas vienu metu negali būti naudojamas keliems darbams atliki. Mechanizmus reikia paskirstyti taip, kad bendras visų darbų atliki laikas būtų trumpiausias.

Sudarykime grafą  $G$ , kurio viršūnių aibė yra darbų aibė  $V$ . Viršūnės  $v_i$  ir  $v_j$  ( $i \neq j$ ) yra gretimos, jei šių darbų atlirkimui reikia vieno ir to paties mechanizmo. Aišku, kad teisingai nudažius grafą, darbai, nudažyti ta pačia spalva, gali būti vykdomi tuo pačiu metu.

**Chromatinio skaičiaus įverčiai** [EM90]. Kadangi chromatino skaičiaus apskaičiavimo uždavinys yra sunkus, tai svarbu žinoti chromatino skaičiaus įverčius.

Simboliais  $\delta(G)$  ir  $\Delta(G)$  atitinkamai pažymėkime grafo  $G$  mažiausią ir didžiausią viršūnių laipsnį, t.y.  $\delta(G) = \min_{v \in V} d(v)$ , o  $\Delta(G) = \max_{v \in V} d(v)$ . Tada jungiajam grafui  $G$  galima nurodyti tokius chromatinio skaičiaus įverčius.

1. Grafo  $G$  chromatinis skaičius tenkina nelygybę  $\gamma(G) \leq 1 + \Delta(G)$ .
2. **Bruksko teorema** (1941). Jei  $G$  – jungusis ir nepilnasis grafas, kuriam  $\Delta(G) \geq 3$ , tai  $\gamma(G) \leq \Delta(G)$ .
3.  $\gamma(G) \leq \min_{1 \leq i \leq n} \max(d(i) + 1, i)$  (Welsh, Powell).
4.  $\gamma(G) \geq \frac{n}{n - \frac{1}{2m} \sum_{i=1}^n d^2(i)}$  (Elphick).
5. **Teorema** (A.P.Eršovas, G.I.Kožuchinas, 1962).

$$\begin{aligned} & -\left[ -n \sqrt{\left[ \frac{n^2 - 2m}{n} \right]} \cdot \left( 1 - \left\{ \frac{n^2 - 2m}{n} \right\} \right) \middle/ \left( 1 + \left[ \frac{n^2 - 2m}{n} \right] \right) \right] \leq \\ & \leq \gamma(G) \leq \left[ \frac{3 + \sqrt{9 + 8(m-n)}}{2} \right], \end{aligned}$$

čia simbolis  $[\cdot]$  žymi skaičiaus sveikają dalį, o  $\{\cdot\}$  – skaičiaus trupmeninę dalį.

Šio paragrafo pabaigoje paminėsime G.Hadžigerio 1943 m. iškeltą hipotezę.

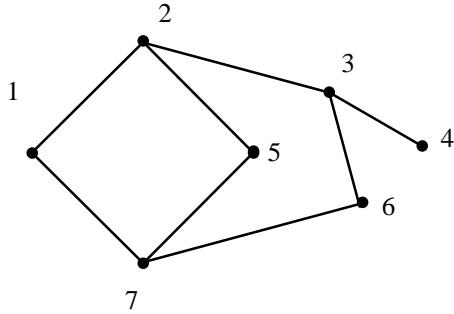
**Hadžigerio hipotezė.** Bet koks jungusis  $n$ -chromatinis grafas gali būti sutrauktas iš  $K_n$  grafą.

Ši hipotezė įrodyta, kai  $n \leq 5$ . Plačiau apie tai žr. lit. [EM90].

### 2.11.3. Nepriklausomumo skaičius

**Apibrėžimas.**  $(n, m)$ -grafo  $G = (V, U)$  viršūnių poaibis  $A$  ( $A \subseteq V$ ) yra **nepriklausomoji aibė** (kitur literatūroje – *vidiniai stabilioji aibė*), jei aibę  $A$  sudarančios viršūnės tarpusavyje nėra gretimos.

Pavyzdžiuui, 2.11.6 pav. grafui aibės  $A = \{1, 5, 4\}$ ,  $B = \{2, 7, 4\}$  yra nepriklausomosios.



#### 2.11.6 pav. Nepriklausomumo skaičius

Akivaizdu, jei  $A$  yra nepriklausomoji aibė, tai bet koks šios aibės poaibis taip pat bus nepriklausomas. Vadinasi, kiekvienam grafui galima sudaryti nepriklausomųjų aibių šeimą  $T$ .

**Apibrėžimas.** Nepriklausomoji aibė, kuri nėra nei vienos kitos nepriklausomosios aibės tikrinis poaibis, vadinama **maksimalija** nepriklausomaja aibe.

**Apibrėžimas.** Nepriklausomoji aibė, turinti didžiausią elementų skaičių, vadinama **didžiausiaja** nepriklausomaja aibe.

**Apibrėžimas.** Skaičius  $\alpha(G) = \max_{A \in T} |A|$ , t.y. didžiausios nepriklausomosios aibės elementų skaičius, vadintamas grafo nepriklausomumo skaičiumi.

Pavyzdžiu, 2.11.3.1 pav. grafui didžiausia nepriklausomoji aibė yra  $A = \{1,4,5,6\}$ , o nepriklausomumo skaičius  $\alpha(G) = 4$ .

Nepriklausomumo skaičiaus radimo uždavinys yra diskrečiojo programavimo uždavinys, ir, kaip ir chromatinio skaičiaus ieškojimo uždavinys, neturi efektyvių sprendimo algoritmų. Kitaip tariant, visų tikslų šio uždavinio sprendimo algoritmų veiksmų skaičius išreiškiamas eksponentiškai, grubiai kalbant, nuo grafo viršūnių skaičiaus.

Pavyzdžiu, grafo nepriklausomumo skaičiaus ieškojimo uždavinį galime formalizuoti taip.

Iveskime kintamuosius

$$x_i = \begin{cases} 1, & \text{jei } i\text{-oji viršūnė priklauso didžiausiai nepriklausomajai aibei,} \\ 0, & \text{priešingu atveju,} \end{cases}, \quad i = \overline{1, n}.$$

Tada reikia maksimizuoti tikslo funkciją:

$$z = \sum_{i=1}^n x_i \rightarrow \max ,$$

esant apribojimams

$$\sum_{i=1}^n a_{ij} \cdot x_i \leq 1, \quad j = \overline{1, m},$$

čia  $a_{ij}$  – grafo incidencijų matricos elementas; šis apribojimas reiškia, kad nepriklausomosios aibės viršūnės tarpusavyje nėra gretimos.

$$x_i \in \{0,1\}, \quad i = \overline{1, n} .$$

Kadangi šis uždavinys neturi tikslį efektyvių sprendimo algoritmų, tai jis sprendžiamas įvairiais euristiniais algoritmais.

Viena iš populiarusių euristikų yra “godus” algoritmas, kai, formuojant nepriklausomą aibę, kiekviename žingsnyje į šią aibę įtraukiame mažiausio laipsnio viršūnę. Ši euristinė algoritma galima užrašyti taip.

```

begin
    A := ∅;
    while "grafas turi viršūnių" do
        begin
            1) Rasti mažiausio laipsnio viršūnę v.
            2) A := A ∪ {v};
            3) Iš grafo pašalinti viršūnę v ir jai gretimas viršūnes, t.y. pašalinti
                viršūnių {v} ∪ N(v) aibę.
        end;
    end;

```

Aišku, kad šis algoritmas visada apskaičiuos maksimaliają, tačiau ne visada didžiausią nepriklausomą aibę.

Žemiau pateikta ši algoritma realizuojanti procedūra.

```

const c = 100;
type
    mas = array [1..c] of integer;
procedure neprk (n, m : integer; L, l st : mas; var alfa : integer; var a : mas);
{ Procedūra neprk apskaičiuoja grafo, nusakyto masyvais L ir lst,
  nepriklausomumo skaičių alfa ir didžiausią nepriklausomą aibę a.
Formalūs parametrai:
  n – grafo viršūnių skaičius,
  m – grafo briaunų (lankų) skaičius,
  L, lst – grafo nusakantys tiesioginių nuorodų masyvai,
  alfa – grafo nepriklausomumo skaičius,
}

```

```

 $a -$  didžiausia nepriklausoma aibė. }

var  $s, i, k, u, v, min : integer;$ 
     $p : boolean;$ 
     $r, d : mas;$ 
begin
    alfa := 0;
    { Visos viršūnės – nenudažyto }
for  $i := 1$  to  $n$  do  $d [i] := 1$ ;
 $p := true;$  { Nepriklausoma aibė – neapskaičiuota }
while  $p$  do
begin
    { Viršinių laipsnių apskaičiavimas }
    for  $i := 1$  to  $n$  do
        if  $d [i] = 1$  then
begin
             $s := 0;$ 
            for  $k := lst [i] + 1$  to  $lst [i + 1]$  do
begin
                 $u := l [k];$ 
                 $s := s + d [u];$ 
end;
                 $r [i] := s;$ 
end;
        end;
    end;
    { Rasti mažiausio laipsnio viršūnę }
     $min := n;$ 
    for  $i := 1$  to  $n$  do
        if ( $d [i] = 1$ ) and ( $min > r[i]$ ) then
begin
             $min := r [i];$ 
             $k := i;$ 
end;
    end;
    if  $min = n$  then  $p := false$ 
        else
begin
    { Viršūnę k talpiname į aibę a }
    alfa := alfa + 1;
     $a [alfa] := k;$ 
    { Šaliname viršūnę k ir visas jai gretimas viršūnes }
     $d [k] := 0;$ 
    for  $i := lst [k] + 1$  to  $lst [k + 1]$  do
begin

```

```

     $v := L[i];$ 
     $d[v] := 0;$ 
    end;
end;
end;

```

Eilė praktinių uždavinijų yra formuluojami kaip grafo nepriklausomumo skaičiaus ieškojimo uždaviniai.

**Pirmas pavyzdys** (Gausas). Ar galima šachmatų lentoje sustatyti 8 valdoves taip, kad jos nekirstų viena kitos?

**Pastaba.** Analogiškai uždavinį galima formuliuoti vietoj valdovių imant kitas šachmatų figūras, t.y. kokį didžiausią skaičių tų pačių šachmatų figūrų galima šachmatų lentoje sustatyti taip, kad jos nekirstų viena kitos.

Sudarykime 64 viršūnių grafi. Šio grafo viršūnės vaizduoja šachmatų lentos langelius. Viršunes  $u$  ir  $v$  jungsime briauna, jei iš šachmatų lentos lanelio  $u$  galime patekti į lanelį  $v$  valdovės éjimui.

Tada, jei šio grafo nepriklausomumo skaičius yra lygus 8, tai šis uždavinyis turi sprendinį, ir valdoves reikia statyti į didžiausios nepriklausomosios aibės viršunes atitinkančius šachmatų lentos langelius.

1854 m. Berlyno šachmatų žurnale buvo paskelbta 40 šio uždavinio sprendimių. Gausas galvojo, kad yra 76 šio uždavino sprendiniai. Iš tikro jų yra 92 ir juos galima sutraukti į 12 diagramų:

(7, 2, 6, 3, 1, 4, 8, 5), (6, 1, 5, 2, 8, 3, 7, 4), (5, 8, 4, 1, 7, 2, 6, 3),  
(3, 5, 8, 4, 1, 7, 2, 6), (4, 6, 1, 5, 2, 8, 3, 7), (5, 7, 2, 6, 3, 1, 4, 8),  
(1, 6, 8, 3, 7, 4, 2, 5), (5, 7, 2, 6, 3, 1, 8, 4), (4, 8, 1, 5, 7, 2, 6, 3),  
(5, 1, 4, 6, 8, 2, 7, 3), (4, 2, 7, 5, 1, 8, 6, 3), (3, 5, 2, 8, 1, 7, 4, 6).

Kaip šifruoti diagramas? Pavyzdžiui, pirmoji diagrama reiškia, kad, sunumeravus šachmatų lentos eilutes iš apačios į viršų skaičiais nuo 1 iki 8, valdoves statysime taip: 8-ijoje (viršutinėje) eilutėje valdovę statysime 7-ajame stulpelyje, 7-ijoje eilutėje valdovę statysime 2-ajame stulpelyje ir t.t. Kitos diagramos iššifruojamos taip pat.

Kiekviena diagrama, išskyrus paskutinią (3, 5, 2, 8, 1, 7, 4, 6), duoda 8 sprendinius, kurie gaunami pasukus šachmatų lentą  $90^\circ$ ,  $180^\circ$  ir  $270^\circ$  bei paémus kiekvieno varianto veidrodinį atspindį pagrindinės įstrižainės atžvilgiu.

Paskutinioji diagrama duoda tik 4 sprendinius, nes, pasukus lentą  $180^\circ$ , digrama transformuoja pati į save.

**Antras pavyzdys.** Tai maksimaliosios klikos radimo uždavinyis.

**Apibréžimas.**  $(n,m)$ -grafo  $G = (V, U)$  viršūnių poaibis vadinams klika, jei bet kokios dvi šio poaibio viršūnės yra gretimos, t.y. jei šio poaibio indukuotas poaibis yra pilnasis.

Klika yra maksimalioji, jei ji nėra didesnės klikos pografis.

Didžiausioji klika – tai klika, kurios viršūnių skaičius tarp visų klių yra didžiausias.

Didžiausiosios klikos viršūnių skaičius vadinamas grafo tankiu (arba klikiniu skaičiumi) ir žymimas  $\varphi(G)$ , t.y.  $\varphi(G) = \max_{A \in T} |A|$ , čia  $T$  – maksimaliųjų klių šeima.

Aišku, kad grafo  $G$  didžiausioji klika sutampa su grafo  $G$  papildomojo grafo  $\bar{G}$  didžiausiaja nepriklausomaja aibe, t.y.  $\varphi(G) = \alpha(\bar{G})$ .

**Pastaba.** Visų maksimaliųjų klių radimo algoritmas išnagrinėtas lit. [RND80, 389 – 396 p.p.].

Reikia pabrėžti, kad grafo maksimaliųjų klių skaičius gali augti eksponentiškai, priklausomai nuo viršūnių skaičiaus.

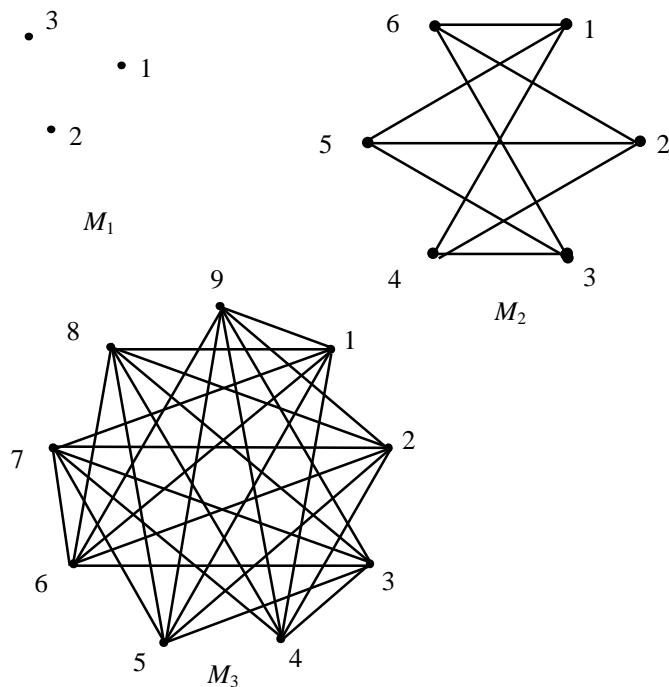
Pavyzdžiuui, panagrinėkime Muno ir Mozero gautą  $M_n$ . Šis grafas turi  $3n$  viršūnių, kurios išskaidytos triadomis:

$$\{1, 2, 3\}, \{4, 5, 6\}, \dots, \{3n-2, 3n-1, 3n\};$$

$M_n$  grafe kiekvienos triados viršūnės tarpusavyje nėra gretimos, tačiau kiekvienos triados viršūnės yra gretimos visoms likusioms viršūnėms. 2.11.7 pav. pavaizduoti  $M_1$ ,  $M_2$  ir  $M_3$  grafai.

Naudojant indukciją, galime irodyti, kad  $M_n$  turi  $3^n$  maksimaliųjų klių ir kiekviena klika turi  $n$  viršūnių.

Tai akivaizdu, kai  $n=1$  ir  $n=2$ . Tarkime, kad  $M_{n-1}$  turi  $3^{n-1}$  maksimaliųjų klių, turinčių po  $n-1$  viršūnę. Tada kiekviena iš trijų viršūnių, pridedamą formuoojant  $M_n$  grafą, duoda kliką su kiekviena  $M_{n-1}$  grafo klika. Vadinasi,  $M_n$  maksimaliųjų klių skaičius yra  $3 \cdot 3^{n-1} = 3^n$ , ir kiekviena klika turi  $n$  viršūnių.



**2.11.7 pav.** Muno ir Mozero grafai

**Trečias pavyzdys** (15-os mergaičių uždavinys). Pensione mokosi 15-ka megaičių, kurias pažymėkime taip: a, b, c, d, e, f, g, h, i, j, k, l, m, n, o. Mergaitės po tris kiekvieną dieną eina pasivaikščioti. Ar galima taip sudaryti savaitės pasivaikščiojimų tvarkaraštį, kad bet kurios dvi mergaitės į trejetą patektų ne daugiau kaip vieną kartą?

Keli, pasinaudodamas simetrija, rado šio uždavinio sprendinį, kuris pateiktas mergaičių pasivaikščiojimo tvarkaraščio lentelėje (žiūr 2.11.2 lentelę).

Norėdami išspręsti šį uždavinį, pirmiausia spręskime pagalbinį uždavinį: ar galima 15 mergaičių suskirstyti į 35 trejetus, kad bet kurios dvi mergaitės nepatektų į trejetą daugiau kaip vieną kartą?

### 2.11.2 lentelė. Mergaičių pasivaikščiojimo tvarkaraštis

Pirma-dienis	Antra-dienis	Trečia-dienis	Ketvir-tadienis	Penkta-dienis	Šesta-dienis	Sekma-dienis
a f k	a b l	a l m	a d o	a g n	a h j	a c i
b g l	c n o	b c f	b i k	b d j	b m n	b h o
c m h	d f l	d e h	c j l	c e k	c d g	d k m
d i n	g k h	g i o	e g m	f m o	e f i	e l n
e j o	i j m	j k n	f h n	h i l	k l o	f g j

Šiam uždavinuii spręsti sudarykime 455 viršūnių grafi. Kiekviena viršūnė vaizduoja  $C_{15}^3$  trejetą. Dvi viršūnės  $u$  ir  $v$  jungiamos briauna, jei  $u$  ir  $v$  trejetuose yra tos pačios dvi mergaitės. Raskime šio grafo nepriklausomumo skaičių ir šį skaičių atitinkančią nepriklausomają aibę. Kadangi kiekviena mergaitė negali būti daugiau kaip 7-iuose trejetuose, tai didžiausioji nepriklausomoji aibė turės  $15 * 7 * 1/3 = 35$  trejetus.

Turėdami šio pagalbinio uždavinio sprendinį, sudarykime grafi, kurio viršūnės vaizduoja sprendinio trejetus. Dvi viršūnės jungiamos briauna, jei ta pati mergaitė priklauso abiem trejetams. Jei šio grafo chromatinis skaičius lygus 7, tai trejai, nudažyti ta pačia spalva, yra mergaičių pasivaikščiojimų vienos dienos tvarkaraštis. Reikia pastebeti, kad ne kiekvienas pagalbinio uždavinio sprendinys duoda 15-os mergaičių uždavinio sprendinį.

#### Nepriklausomumo skaičiaus įverčiai

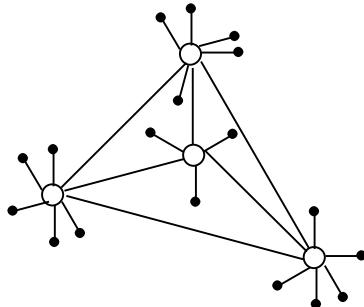
1.  $\alpha(G) \geq \sum_{v \in V} (1 + d(v))^{-1}$  (Wei).
2.  $\alpha(G) \geq \frac{n}{1 + d}$  (Myers, Liu), čia  $\bar{d} = \frac{2m}{n}$  – vidutinis viršūnės laipsnis.
3.  $\alpha(G) \geq \left\lceil \frac{2pn - 2m}{p(p+1)} \right\rceil$ ,  $p = 1 + \left\lfloor \frac{2m}{n} \right\rfloor$  (Jeršovas, Kožuchinas).
4.  $\alpha(G) \leq p^0 + \min\{p^-, p^+\}$ , čia  $p^+$ ,  $p^-$  ir  $p^0$  atitinkamai grafo gretumino matricos teigiamų, neigiamų ir nuliniai tikrinių reikšmių skaičius (D.Cvetkovičius, 1973).
5.  $\alpha(G) \geq \frac{n}{\max_{v \in V} d(v)}$  (Brooks).

Primename, kad šiose formulėse  $n$  – viršunių skaičius,  $m$  – briaunų skaičius, o  $d(v)$  –  $v$ -osios viršūnės laipsnis.

**Dar kartą apie grafo dažymą.** Chromatinis skaičius  $\gamma(G)$  ir nepriklausomumo skaičius  $\alpha(G)$  surišti nelygybe:  $\alpha(G) \cdot \gamma(G) \geq n$ .

Aišku, kad grafo viršunes galima išskaidyti į  $\gamma(G)$  nepriklausomųjų poaibų: poaibį sudaro viršūnės, nudažytos ta pačia spalva. Kadangi kiekvieno šio poaibio elementų skaičius neviršija  $\alpha(G)$ , tai  $\alpha(G) \cdot \gamma(G) \geq n$ . Kyla klausimas, ar tarp šių skaičių nėra glaudesnio ryšio? Gal chromatinij skaičių galima rasti tokiu būdu. Pirmiausia apskaičiuojame grafo  $G$  didžiausią nepriklausomają aibę  $A_1$  ir šios aibės viršunes nudažome pirmaja spalva. Po to randame pograffio, kurį indukuoja aibę  $V/A_1$ , didžiausią nepriklausomają aibę  $A_2$  ir šias viršunes dažome antraja spalva ir t.t., kol nudažome visas grafo viršunes.

Reikia pabrėžti, kad šis metodas gali apskaičiuoti neoptimalų chromatinij skaičių (žr. 2.11.8 pav.). Šiam grafui  $\gamma(G) = 4$ . Tamsiais taškais pažymėtos didžiausios nepriklausomosios aibės viršūnės. Jei visas jas nudažysime viena spalva, tai likusio grafo viršūnėms nudažyti dar teks panaudoti keturias naujas spalvas. Vadinas, aukščiau pateiktas metodas duos neoptimalų chromatinij skaičių.

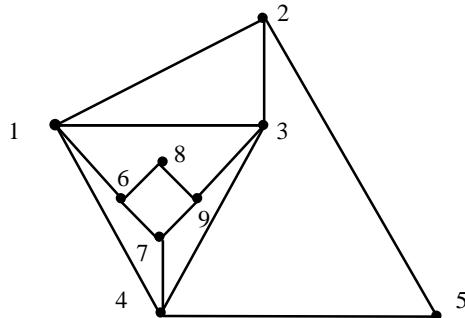


2.11.8 pav. Ryšys tarp  $\alpha(G)$  ir  $\gamma(G)$

#### 2.11.4. Dominavimo skaičius

**Apibrėžimas.**  $(n,m)$ -grafo  $G = (V,U)$  viršunių poaibis  $A$  ( $A \subseteq V$ ) yra **dominuojančioji aibė** (kitur literatūroje *išoriškai stabilioji aibė*), jei kiekviena grafo viršūnė, nepriklausanti poaibiu  $A$ , yra gretima bent vienai poaibio  $A$  viršūnei, t.y. jei visos likusios grafo viršūnės, nutolusios nuo bent vienos dominuojančios aibės viršūnės atstumu, lygiu vienetui.

Pavyzdžiu, 2.11.9 pav. grafui aibės  $\{1,2,7\}$ ,  $\{2,7\}$ ,  $\{1,3,5\}$ .



**2.11.9 pav.** Dominavimo skaičius

Akivaizdu, kad jei  $A$  yra dominuojančioji aibė ir  $A \subseteq B$ , tai  $B$  taip pat dominuojančioji aibė.

**Apibrėžimas.** Dominuojančioji aibė, kurios bet kuris tikrinis poaibis nėra dominuojančioji aibė, vadinama **minimaliąja** dominuojančiaja aibe.

**Apibrėžimas.** Dominuojančioji aibė, turinti mažiausią elementų skaičių, vadinama **mažiausiąja** dominuojančiaja aibe.

**Apibrėžimas.** Skaičius  $\beta(G) = \min_{A \in T}(A)$ , čia  $T$  – grafo  $G$  dominuojančiųjų aibių šeima, t.y. mažiausios dominuojančiosios aibės elementų skaičius, vadinamas grafo dominavimo skaičiumi.

2.11.9 pav. grafo dominavimo skaičius yra 2, o mažiausia dominuojančioji aibė –  $\{2,7\}$ .

Mažiausios dominuojančiosios aibės, o tuo pačiu ir dominavimo skaičiaus radimo uždavinys yra tipiškas padengimo uždavinys.

**Padengimo** uždavinį galima formuluoti taip. Duota aibė  $S = \{s_1, s_2, \dots, s_n\}$  ir aibės  $S$  poaibių šeima  $A = \{A_1, A_2, \dots, A_m\}$ . Rasti tokį mažiausią poaibių  $A_i$  rinkinį (padengimą), kad kiekvienas aibės  $S$  elementas  $s_k$  ( $k = \overline{1, n}$ ) priklauso ytū (būtų padengtas) bent vienam rinkinio poaibui  $A_i$ .

Formalai šį uždavinį galime užrašyti taip.

$$\text{Įveskime kintamuosius } x_i = \begin{cases} 1, & \text{jei } A_i \text{ įeina į padengimą,} \\ 0, & \text{priešingu atveju,} \end{cases} \quad i = \overline{1, m}.$$

Poaibių  $A_i$  šeimą vaizduokime  $(n \times m)$  formato matrica  $A = [a_{ij}]$ ,  $i = \overline{1, n}$ ,  $j = \overline{1, m}$ , čia

$$a_{ij} = \begin{cases} 1, & \text{jei } s_i \in A_j, \\ 0, & \text{priešingu atveju,} \end{cases}$$

t.y.  $j$ -asis matricos  $A$  stulpelis vaizduoja poaibį  $A_j$ . Tada padengimo uždavinys ekvivalentiškas tokiam optimizavimo uždavininiui:

$$\min z = \sum_{i=1}^m x_i$$

esant apribojimams:

- “kiekvienas elementas priklauso bent vienam į padengimą įeinančiam poaibiu”,
- $\sum_{j=1}^m a_{ij} \cdot x_j \geq 1, \quad i = \overline{1, n},$
- $x_i \in \{0,1\}, \quad i = \overline{1, n}.$

Dominavimo skaičiaus ieškojimo atveju aibė  $S$  yra grafo viršūnių aibė  $V$ . Kiekviena grafo viršūnė  $v \in V$  apibrėžia viršūnių poaibį  $A_v = \{v\} \cup N(v)$ ,  $v \in V$ . Reikia rasti tokį mažiausią poaibį  $A_v$  skaičių (viršūnių  $v$  aibę), kad kiekviena grafo viršūnė priklausytų bent vienam rinkinio poaibiu  $A_v$ .

Kadangi visi tikslūs padengimo uždavinio sprendimo metodai yra neefektyvūs, tai paprastai šis uždavinys sprendžiamas euristiniais algoritmais. Viena iš populiariausių euristikų yra “godaus” algoritmo euristika: “kol yra nepadengtų viršūnių, kiekviename žingsnyje į dominuojančiąją aibę (padengimą) įtrauksite tokią aibę  $A_v$ , kuriai priklauso didžiausias skaičius nepadengtų viršūnių, t.y. tokią viršūnę, kurios laipsnis yra didžiausias”.

Formalai šį algoritmą galime užrašyti taip.

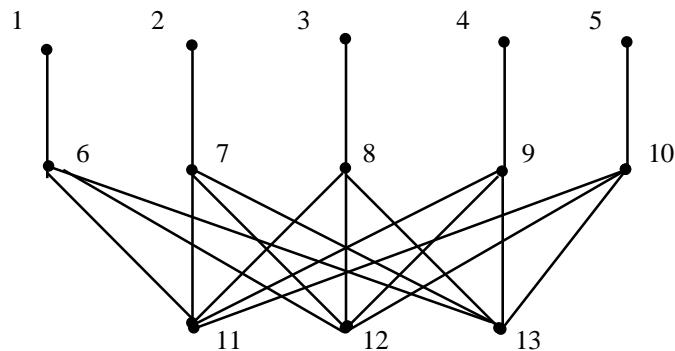
```

begin
    A := ∅; {A – mažiausioji dominuojančioji aibė}
    while "grafas turi viršūnių" do
        begin
            1. Rasti didžiausio laipsnio viršūnę v.
            2. A := A ∪ {v}
            3. Iš grafо pašalinti viršūnių {v} ∪ N(v) aibę.
        end;
    end;

```

Nesunku pastebėti, kad šis algoritmas yra analogiškas didžiausios nepriklausomosios aibės apskaičiavimo algoritmui. Aišku, kad šis algoritmas visada apskaičiuos minimalią dominuojančiąją aibę, tačiau ji ne visada bus mažiausioji. Pavyzdžiui, 2.11.10 pav. grafui mažiausioji dominuojančioji aibė

yra  $\{6,7,8,9,10\}$ , o pagal algoritmą apskaičiuota minimalioji dominuojančioji aibė yra  $\{11,12,13,1,2,3,4,5\}$ .

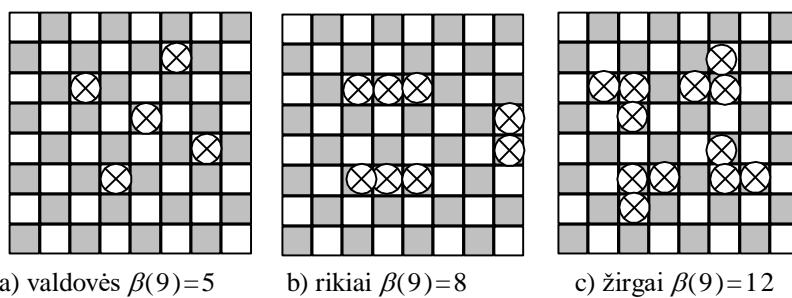


**2.11.10 pav.** “Godus” algoritmas neapskaičiuoja minimaliosios dominuojančiosios aibės

Aptarsime keletą pavyzdžių, kuriuos sprendžiant reikia apskaičiuoti mažiausią dominuojančiąjį aibę.

**Pirmas pavyzdys. Uždavinys apie sargybinius.** Tarkime, kad grafas  $G$  (pvz. žr. 2.11.9 pav.) yra  $N$ -miesto kalėjimo planas. Grafo viršūnės vaizduoja kameras, kuriose įkalinti pavojingi nusikaltėliai. Viršūnės  $u$  ir  $v$  jungiamos briauna, jei jas jungia tiesus koridorius. Reikia rasti mažiausią skaičių sargybinių, kad jie galėtų sekti visų kamero duris.

**Antras pavyzdys. Penkių valdovų uždavinys.** Kiek mažiausiai šachmatų lentoje reikia pastatyti valdovų (galima klausti, kiek rikių, kiek žirgų ir pan.), kad kiekvienas šachmatų lentoje langelis būtų kertamas.  
pav. a) parodytas mažiausias valdovų skaičius, b) – mažiausias rikių skaičius, c) – mažiausias žirgų skaičius.



**2.11.11 pav.** Šachmatų uždavinys

Aišku, kad sprendžiant šį uždavinį nagrinėsime 64 viršūnių grafą. Kiekviena viršūnė vaizduoja šachmatų lento langelį. Viršūnė  $u$  jungiama briauna su viršūne  $v$ , jei iš lanelio  $u$  galima patekti į lanelį  $v$  nurodytos figūros (valdovės, žirgo ir pan.) éjimu. Tada šiame grafe reikës rasti mažiausią dominuojančią aibę.

**Trečias pavyzdys.** Prancūzijos mugėse mègstamas toks žaidimas: ant stalo padëtas didelis baltas diskas (tarkime, šio disko spindulys lygus 1 m); ši diską reikia pilnai uždengti šešiais mažesniais (spindulio  $\rho < 1\text{m}$ ) raudonais diskais, kuriuos žaidéjas vieną po kito deda ant balto disko. Kartą padëtas raudonasis diskas nebejudinamas. Koks turi būti mažiausias raudonojo disko spindulys  $\rho$ , kad baltasis diskas būtų pilnai padengtas?

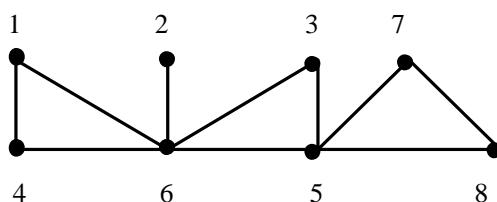
Šis uždavinys ekvivalentus uždaviniui: rasti mažiausią dominuojančią aibę begaliniam grafe  $(V, U)$ , čia  $V$  – baltojo disko taškų aibė, o viršūnės  $u$  ir  $v$  jungiamos briauna, jei atstumas tarp šių viršūnių  $d(v, u) < \rho$ .

**Ketvirtas pavyzdys.** Duota aibė gyvenviečių, sujungtų kelių tinklu. Kai kuriose gyvenvietėse reikia pastatyti buitines įmones taip, kad nuo bet kurios gyvenvietės buitinė įmonė būtų ne toliai nei duotas atstumas  $c$ . Kokiose gyvenvietėse statyti įmones, kad jų skaičius būtų mažiausias?

Sudarykime grafą, kurio viršūnės vaizduoja gyvenvietes. Dvi viršūnės jungiamos briauna, jei atstumas tarp jų yra mažesnis už  $c$ . Tada mažiausios dominuojančiosios aibës viršūnės ir nurodys gyvenvietes, kuriose turi būti statomos buitinės įmonės.

**Apibrëžimas.** Grafo viršūnių poaibis, kuris vienu metu yra ir nepriklausomas ir dominuojantis, vadinamas **grafo branduoliu**.

Aišku, kad grafo nepriklausomoji aibë yra maksimali (nebūtinai didžiausia) tada ir tikta tada, kada ji yra dominuojanti. Pavyzdžiu, 2.11.4.4 pav. grafo aibës  $\{1,2,3,7\}$ ,  $\{1,2,3,8\}$ ,  $\{2,3,5,7\}$ ,  $\{2,3,5,8\}$ ,  $\{4,7\}$ ,  $\{4,8\}$  yra grafo branduoliai.



**2.11.12 pav.** Grafo branduolys

Reikia pabrėžti, kad tiek nepriklausomosios aibės, tiek ir dominuojančiosios aibės pateikti generavimo algoritmai įgalina apskaičiuoti neorientuotojo grafo branduoli.

**Apibrėžimas.** Sakyime, kad viršūnė ir briauna dengia viena kitą, jei jos incidentiškos. Pavyzdžiui, briauna  $(u, v)$  dengia viršunes  $u$  ir  $v$ , o viršūnė  $u$  arba viršūnė  $v$  dengia briauną  $(u, v)$ .

**Apibrėžimas.** Grafo viršūnių poaibis  $V'$  vadinamas **viršūniniu denginiu**, jei kiekviena grafo briauna yra incidentiška bent vienai aibės  $V'$  viršūnei.

Denginys vadinamas **minimaliuoju**, jei bet koks jo tikrinis poaibis nėra denginys; denginys vadinams **mažiausiuoju**, jei jo elementų skaičius yra mažiausias tarp visų denginių. Šis elementų skaičius žymimas  $\beta_0(G)$  ir vadinamas **viršūninio denginio skaičiumi**.

Pavyzdžiui, 2.11.12 pav. grafui aibės  $X_1 = \{4,5,6,8\}$ ,  $X_2 = \{4,5,6,7\}$  ir  $X_3 = \{1,2,3,5,6,8\}$  yra denginiai.  $X_1$  ir  $X_2$  – mažiausi denginiai. Vadinasi,  $\beta_0(G) = 4$ .

Tarp denginio ir nepriklausomosios aibės yra glaudus ryšys, kurį nusako žemiau pateikta teorema.

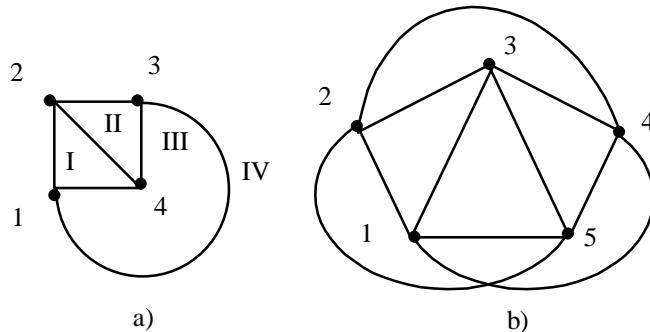
**Teorema.** Grafo  $G = (V, U)$  viršūnių poaibis  $A$  yra mažiausiasis (minimalusis) denginys tada ir tik tai tada, kai aibė  $\bar{A} = V \setminus A$  yra didžiausioji (maksimalioji) grafo  $G$  nepriklausomoji aibė. Vadinasi,  $\alpha(G) + \beta_0(G) = n$ .

Tuo būdu, visi nepriklausomumo skaičiaus įverčiai gali būti naudojami ir denginio įverčiui.

## 2.12. Plokštieji grafai

**Apibrėžimas.**  $(n, m)$ - grafas (bendru atveju multigrafas)  $G = (V, U)$  vadinamas plokščiuoju, jei jį galima plokštumoje pavaizduoti taip, kad briaunos kirstysi tik viršūnėse.

2.12.1 a) pav. pavaizduotas plokštasis grafas  $K_4$ , o 2.12.1 b) pav. – neplokštasis grafas  $K_5$ . Kad  $K_5$  nėra plokštasis grafas, įrodysime vėliau.



**2.12.1 pav.** Plokštieji grafa: a) plokštusis; b) neplokštusis

Reikia pažymėti, kad nustatymas faktu, ar grafas yra plokštusis ir grafo pavaizdavimas plokštumoje, kad jo briaunos kirstysi tik viršūnėse, yra praktiškai svarbus uždavinys: elektrinės schemas (spausdintas montažas) yra plokštieji grafa.

Aptarsime keletą plokščiojo grafo sąvokų.

**Grafo siena** – tai plokštumos dalis, apribota ciklu, kurioje nėra nei viršūnės, nei briaunos.

Ciklas, ribojantis grafo sieną, vadinamas **minimaliuoju ciklu**.

Dvi grafo sienos yra **gretimos**, jei jos turi bent vieną bendrą briauną.

Pavyzdžiuui,  $K_4$  grafas (žr. 2.12.1 a) turi keturias sienas. Jos pažymėtos romėniškais skaičiais, o jų minimalieji ciklai atitinkamai yra: 1,2,4,1 ; 2,3,4,2 ; 1,4,3,1 ir 1,2,3,1 . Ciklas 1,2,3,1 riboja begalinę IV sieną, kuri yra šio ciklo išorėje. Kiekvienas plokštusis grafas turi vieną begalinę sieną. Todėl plokščiojo grafo baigtinės sienos dar vadinamos **vidinėmis**, o begalinė siena – **išorine**.

**Teorema.**  $(n,m)$ - plokščiojo grafo baigtinių sienų minimalūs ciklai yra tiesiškai nepriklausomi ir sudaro bazę.

**Išvada.** (Oilerio formulė). Plokščiojo grafo sienų skaičius  $f$ , briaunų skaičius  $m$  ir viršūnių skaičius  $n$  susieti formule

$$f - m + n = 2,$$

kuri vadinama Oilerio formule.

Oilerio formulės teisingumas išplaukia iš to, kad plokščiojo grafo baigtinių sienų minimalūs ciklai yra nepriklausomi ir sudaro bazę. Kitaip tariant, plokščiojo grafo ciklomatinis skaičius  $v(G) = f - 1$ . Iš čia

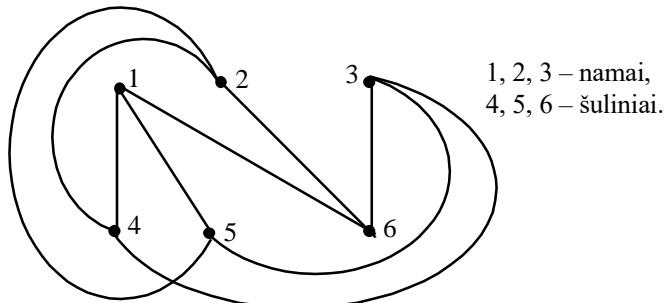
$$m - n + 1 = f - 1.$$

Tada

$$f - m + n = 2.$$

Remdamiesi Oilerio formule įrodysime, kad pilnasis dvidalis grafas  $K_{3,3}$  ir penkių viršūnių pilnasis grafas  $K_5$  nėra plokštieji.

**$K_{3,3}$  grafas nėra plokštusis.** Tarkime priešingai, kad  $K_{3,3}$  grafas yra plokštusis. ( $K_{3,3}$  grafas pavaizduotas 2.12.2. pav.).



**2.12.2 pav.**  $K_{3,3}$  grafas nėra plokštusis

Tada, remdamiesi Oilerio formule, galime apskaičiuoti šio grafo sienų skaičių:

$$f = m - n + 2 = 9 - 6 + 2 = 5.$$

Sudarysime pagalbinį incidencijų sienos – briaunos grafą. Šis grafas yra dvidalis  $(A, B, U)$ . Aibės  $A$  viršūnės vaizduoja  $K_{3,3}$  grafo sienas; aibės  $B$  viršūnės vaizduoja  $K_{3,3}$  grafo briaunas. Viršūnė  $a \in A$  jungiama briauna su viršūne  $b \in B$ , jei  $K_{3,3}$  briauna  $b$  yra incidentiška  $K_{3,3}$  sienai  $a$ .

Įvertinkime šio pagalbinio grafo briaunų skaičių  $m^*$ .

Aišku, kad  $m^* \leq 2m$ , čia  $m = K_{3,3}$  grafo briaunų skaičius (kiekviena grafo  $K_{3,3}$  briauna incidentiška nedaugiau kaip dviem sienom). Kita vertus,  $m^* \geq 4f$ , nes  $K_{3,3}$  grafo sienų minimalių ciklų ilgai nemažesni nei 4. Tarus, kad minimalaus ciklo ilgis yra 3, turėtume jungti arba du namus, arba du šulinius, o taip daryti negalima, nes grafas yra dvidalis ir nemultigrafas. Vadinas,  $4f \leq m^* \leq 2m$ . Iš šios nelygybės gausime, kad  $20 \leq m^* \leq 18$ . Gautas prieštaravimas rodo, kad prielaida buvo klaidinga.

**$K_5$  grafas nėra plokštusis.** Įrodymas, kad  $K_5$  grafas nėra plokštusis grafas, yra analogiškas, kaip ir įrodymas, kad  $K_{3,3}$  grafas nėra plokštusis.

Tarkime, kad  $K_5$  grafas yra plokštusis grafas. Tada pagal Oilero formulę apskaičiuosime jo sienų skaičių:

$$f = 10 - 5 + 2 = 7.$$

Kaip ir  $K_{3,3}$  atveju, sudarykime pagalbinį incidencijų sienos – briaunos grafa ir įvertinkime jo briaunų skaičių  $m^*$ . Nesunku suprasti, kad  $m^*$  tenkina nelygybę

$$3f \leq m^* \leq 2m,$$

t.y. kiekviena grafo  $K_5$  briauna incidentiška ne daugiau kaip dviem sienom, o trumpiausio minimalaus ciklo ilgis yra nemažesnis nei 3 ( $K_5$  nėra multigrafas).

Vadinasi,  $21 \leq m^* \leq 20$ . Gautas prieštaravimas rodo, kad  $K_5$  nėra plokštusis grafas.

Aptarsime porą plokščiojo grafo savybių. Viena iš jų gali būti dalinis kriterijus, nustatant, ar grafas yra plokštusis, t.y. būtina plokščiojo grafo sąlyga, o antroji – įrodant, kad bet kokį plokštujį grafa galima nudažyti 5-mis spalvomis.

**Pirmoji savybė.** Jei  $G$  – jungasis plokštusis  $(n,m)$ -grafas yra nemultigrafas, tai, kai  $n \geq 3$ ,  $m \leq 3n - 6$ .

**Irodymas.** Kadangi grafas  $G$  nėra multigrafas, tai kiekvienos sienos minimalaus ciklo ilgis yra nemažesnis nei 3. Pasinaudodami incidencijų sienos – briaunos grafu, gausime, kad  $3f \leq 2m$ . Iš šios nelygybės išplaukia, kad  $f \leq (2/3)m$ . Remdamiesi Oilerio formule, gausime:

$$\begin{aligned} f - m + n &= 2, \\ m &= f + n - 2, \\ m &\leq \frac{2}{3}m + n - 2, \\ m &\leq 3n - 6. \end{aligned}$$

**Antroji savybė.** Bet kuriame plokščiajame grafe  $G$ , kuris nėra multigrafas, yra bent viena viršūnė, kurios laipsnis nedidesnis nei 5.

**Irodymas.** Sudarykime grafo  $G$  incidencijų sienos – briaunos grafa. Tada šio grafo briaunų skaičius  $m^*$  tenkina nelygybę:

$$3f \leq m^* \leq 2m.$$

Iš šios nelygybės gauname, kad  $f \leq (2/3)m$ .

Sudarykime plokščiojo grafo  $G$  incidencijų viršūnės – briaunos grafa  $(A, B, U)$ . Viršūnių aibė  $A$  vaizduoja grafo  $G$  viršūnes, o viršūnių aibė  $B$  – grafo  $G$  briaunas. Viršūnė  $a \in A$  jungiama briauna su viršūne  $b \in B$ , jei grafo

$G$  viršūnė  $a$  incidentiška grafo  $G$  briaunai  $b$ . Tarkime, kad plokščiojo grafo  $G$  visų viršūnių laipsniai nemažesni nei 6. Tada pagalbinio incidencijų viršūnės – briuanos grafo briaunų skaičius  $m^{**}$  tenkins nelygybę:

$$6n \leq m^{**} \leq 2m.$$

Iš šios nelygybės gausime, kad  $n \leq \frac{1}{3}m$ . Remdamiesi Oilerio formule, gausime:

$$2 = f - m + n \leq \frac{2}{3}m - m + \frac{1}{3}m = 0.$$

Gautas prieštaravimas rodo, jog prielaida, kad visų grafo  $G$  viršūnių laipsniai yra nemažesni nei 6, yra klaudinga. Vadinasi, plokštusis grafas turi bent vieną viršūnę, kurios laipsnis nedidesnis už 5.

Kaip nustatyti, ar duotas grafas yra plokštusis? Atsakymą į šį klausimą pirmasis davė lenkų matematikas G.Kuratovskis 1930 m. (G.Kuratovski. Sur le probleme des courbes gauches en topologie. Fund. Math., 15-16 (1930), 271 p.)<sup>2</sup>. Čia pateikiame 1937 m. K.Vagnerio teoremą, kuri yra analogiška Kuratovskio – Pontriagino teoremai.

**Teorema** (G.Vagneris, 1937). Grafas  $G$  yra plokštusis tada ir tikai tada, kada jis neturi pografių, kuriuos galima sutraukti į  $K_{3,3}$  arba  $K_5$  grafus.

**Teorema** (Koršunov A.D. 1985) [EM90] Beveik nėra plokščiųjų grafų.

Reikia pabrėžti, kad minėtos teoremos padeda tik nustatyti faktą, ar grafas yra plokštusis. Tačiau atsakymas, kad grafas yra plokštusis, yra nekonstruktivus: lieka neaišku, kaip grafą pavaizduoti plokštumoje, kad jo briaunos kirstysi tik viršūnėse? Atsakymas į šį klausimą kaip tik ir turi pagrindinę praktinę vertę. Todėl labai svarbu turėti algoritmą, kuris leistų pavaizduoti grafą plokštumoje taip, kad briaunos kirstysi tik viršūnėse, jei grafas yra plokštusis, arba duotą atsakymą “ne”, jei, vaizduojant grafą plokštumoje, paažinkėtų, kad jis nėra plokštusis. Toks algoritmas detaliai išdėstytas literatūroje [RND80, 402 – 424 p.p.].

**Plokščiojo grafo dažymas.** Kaip buvo minėta aukščiau, plokščiojo grafo keturių spalvų hipotezę, t.y. hipotezę, kad bet kokio plokščiojo grafo viršūnes galima nudažyti 4-iomis spalvomis taip, kad gretimos viršūnės būtų nudažytos skirtinė spalva, iškėlė Augustas de Morganas savo laiške, rašytame 1852 m. spalio 23 d. serui Viljamui Rovanui Hamiltonui.

---

<sup>2</sup> Nepriklausomai tokią pat teoremą 1927 m. suformulavo rusų matematikas L.S.Pontriaginas. Tačiau ši teorema nebuvvo publikuota. Todėl teorema, nusakanti būtinas ir pakankamas sąlygas, kad grafas būtų plokštusis, dažnai vadinama Kuratovskio – Pontriagino teorema.

Pirmasis šią hipotezę 1880 m. bandė įrodyti A.Kempė. Tačiau 1890 m. R.Hivudas Kempės įrodymė rado klaidą<sup>3</sup>. Tais pačiais metais R.Hivudas suformulavo ir įrodė teoremą.

**Teorema.** Kiekvienas plokštusis grafas gali būti nudažytas 5-iomis spalvomis.

*Įrodymas.* Teorema įrodoma matematinės indukcijos metodu.

Teorema teisinga, jei plokščiojo grafo viršūnių skaičius nedidesnis už penkis.

Tarkime, kad ši teorema teisinga, kai plokščiojo grafo viršūnių skaičius yra  $n > 5$ .

Įrodysime, kad ši teorema teisinga, kai plokščiojo grafo viršūnių skaičius lygus  $n + 1$ .

Tarkime, kad kai plokščiojo grafo viršūnių skaičius yra  $(n + 1)$ . Kaip įrodėme aukšciau, tame egzistuoja bent viena viršūnė  $v_0$ , kurios laipsnis nedidesnis nei 5.

Panagrinėkime du atvejus.

1.  $|N(v_0)| \leq 4$ . Aišku, kad grafas  $G - v_0$  yra plokštusis, ir jį galima nudažyti 5-iomis spalvomis. Viršūnę  $v_0$  nudažysime viena iš 5-ių spalvų, kuri nepanaudota viršūnės  $v_0$  gretimų viršūnių dažyme.

2. Tarkime, kad  $|N(v_0)| = 5$ . Tada aibėje  $N(v_0)$  egzistuoja dvi negretimos viršūnės  $v_1$  ir  $v_2$ , nes, priešingu atveju, aibės  $N(v_0)$  indikuotas pografis būtų  $K_5$ , o tai reiškia, kad nagrinėjamas grafas nėra plokštusis.

Tarkime,  $G'$  yra grafas, gautas apjungus grafo  $G - v_0$  viršunes  $v_1$  ir  $v_2$ . Šią apjungtantą viršūnę pažymėkime raide  $v$ . Grafą  $G'$  galima nudažyti 5-iomis spalvomis. O tai reiškia, kad grafą  $G - v_0$  galima nudažyti 5-iomis spalvomis taip, kad viršūnės  $v_1$  ir  $v_2$  būtų nudažytos ta pačia spalva, t.y. viršūnės  $v$  spalva. Tada viršūnę  $v_0$  dažysime viena iš spalvų, nepanaudotų viršūnių  $N(v_0)$  dažyme.

## 2.13. Medžiai

Nagrinėsime neorientuotuosius jungiuosius grafus. Jei vienas jungiuoją grafų ribinis atvejis yra pilnieji grafai, t.y. jungieji grafai, turintys didžiausią briaunų skaičių, tai kitas ribinis atvejis yra jungieji grafai, turintys mažiausią briaunų skaičių. Tie grafai vadinami medžiais.

---

<sup>3</sup> Kaip buvo minėta aukšciau, keturių spalvų hipotezę K.Apelis, W.Hakenas ir J.Kochas įrodė 1976 m.

Istoriškai medžiai nepriklausomai buvo atrasti kelis kartus. 19 amžiuje Kirchhofas naudojo medžius tirdamas elektrines grandines. Vėliau matematikas Keli iš naujo apibréžė medžius, ištyrė jų savybes ir juos naudojo norėdamas apskaičiuoti sočiujų angliavandenilių izomerų skaičių. Po jo matematikas Žordanas medžius tyrė kaip gryna matematinį objektą.

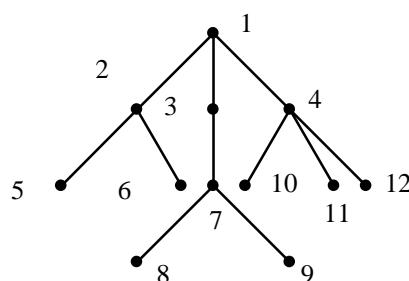
**Apibrėžimas.** Jungusis grafas, neturintis ciklų, vadinamas medžiu.

Medis turi 6 ekvivalenčias savybes, kiekviena iš kurių gali būti medžio apibrėžimas. Šias savybes nusako

**Teorema.** Jei  $(n,m)$ -grafas  $G$  yra medis, tai

- 1)  $G$  – jungusis ir neturi ciklų,
- 2)  $G$  – jungusis ir  $m = n - 1$ ,
- 3)  $G$  – neturi ciklų ir  $m = n - 1$ ,
- 4)  $G$  neturi ciklų, tačiau įvedus naujā briauną, jungiančią bet kokias dvi negretimas medžio viršunes, atsiranda vienintelis ciklas,
- 5) Grafas  $G$  yra jungusis, tačiau praranda šią savybę, pašalinus bet kurią briauną,
- 6) Bet kuri viršunių pora sujungta grandine ir tikta viena.

2.13.1 pav. pavaizduotas medis. Nesunku įsitikinti šių savybių teisingumu.



**2.13.1 pav.** Medis

2.13.1 pavaizduoto medžio 1-oji viršūnė vadinama **medžio šaknimi**. Viršūnės, kurios nutolę nuo šaknies atstumu  $k$  ( $k$  – natūralusis skaičius), vadinamos  **$k$ -tojo lygio viršūnėmis**. Medžio viršūnės, kurių laipsnis yra lygus 1, vadinamos **kabančiomis viršūnėmis**. 2.13.1 pav. grafo 2-oji, 3-ioji ir 4-oji viršūnės yra pirmo lygio viršūnės. 5-oji, 6-oji, 8-oji, 9-oji, 10-oji, 11-oji ir 12-oji viršūnės yra kabančios viršūnės.

### 2.13.1. Dengiančiojo medžio apskaičiavimo uždaviny

*Apibrėžimas.*  $(n, m)$ -grafo  $G$  dalinis grafas medis vadinamas **dengiančiuoju medžiu**.

*Teorema.* Būtina ir pakankama sąlyga, kad  $(n, m)$ -grafas  $G$  turėtų dalinių grafų medžių yra ta, kad grafas  $G$  būtų jungusis.

**Būtinumas.** Jei grafo dalinis grafas yra medis, tai grafas  $G$  yra jungusis, nes medis yra jungusis grafas.

**Pakankumas.** Grafas  $G$  yra jungusis. Ar grafas turi briauną, kuria pašalinus jungumas neišyra? Jei tokios briaunos nėra, tai pagal 5-ają savybę grafas  $G$  (dalinis grafas) yra medis. Jei tokia briauna yra, tai ją pašaliname ir klausimą kartojame iš naujo.

Šios teoremos įrodymas yra konstruktyvus, t.y. iš jo išplaukia dalinio grafo medžio konstravimo algoritmas. Tačiau reikia pabrėžti, kad šis algoritmas yra neracionalus: kiekvienam žingsnyje turime ieškoti briaunos, kuria pašalinus grafas nesuritų.

Žymiai racionalesni dengiančiojo medžio radimo algoritmai yra pagrįsti **paieška platyn** arba **paieška gilyn**. Tarkime, kad paieškos platyn ar gilyn metu iš viršūnės  $u$  atėjome į naują (neaplankytą) viršūnę  $v$ . Tada briauna  $(u, v)$  yra medžio briauna.

Reikia pastebeti, kad paiešką gilyn ar platyn galima baigti, kai į medžių ištrauksimė  $(n - 1)$  briauną, t.y. paiešką galima nutraukti nelaukiant jos natūralios pabaigos.

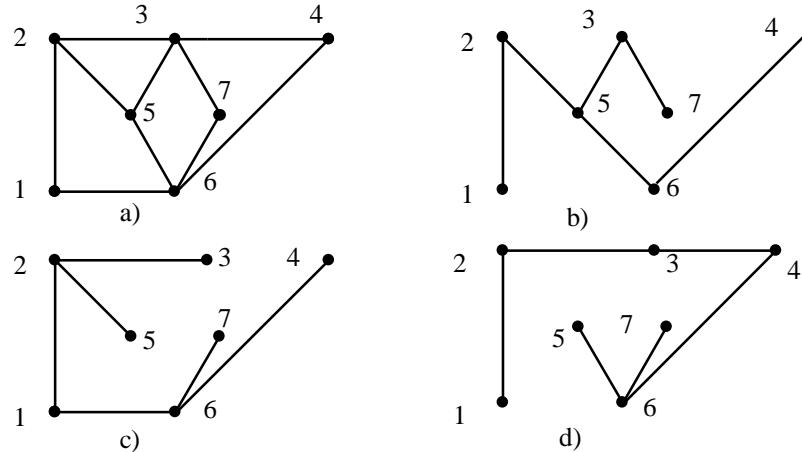
**Pavyzdys.** Panagrinėkime dengiančio medžio konstravimą visais trim metodais. 3.13.2 a) pav. pavaizduotam grafui.

2.13.2 b) paveiksle pavaizduotas dengiantis medis, apskaičiuotas metodu, išplaukiančiu iš teoremos įrodymo, kai iš pradinio grafo buvo pašalintos briaunos: (1,6), (2,3), (3,4), (6,7). Aišku, kad šių briaunų skaičius yra lygus  $m - n + 1$ , t.y. lygus grafo ciklomatiniams skaičiui. Kaip buvo minėta aukščiau, nagrinėjant nepriklausomų ciklų išskyrimo uždavinį, šios briaunos vadinamos **atvirkštinėmis briaunomis** arba **stygomis**, ir kiekviena iš jų išsaukia nepriklausomą ciklą, kuris sudarytas iš šios briaunos ir medžio briaunu.

2.13.2 c) paveiksle pavaizduotas dengiantis medis, gautas paieškos platyn iš 1-osios viršūnės metodu, laikant, kad kiekvienai viršūnai gretimos viršūnės išdėstytos jų numerių didėjimo tvarka.

Paieškos platyn metu iš 1-osios viršūnės aplankome naujas 2-ają ir 6-ają viršunes. Tuo pačiu briaunos (1,2) ir (1,6) yra medžio briaunos. Po to, paieškos platyn antrojo žingsnio metu iš 2-osios viršūnės aplankome naujas 3-iąją ir 5-ąją viršunes, o iš 6-osios viršūnės –naujas 4-ąją ir 7-ąją viršunes. Tuo

būdu briaunos (2,3), (2,5),(6,4) ir (6,7) yra medžio briaunos. Kadangi i  
medži įtraukėme  $(n-1)$ , t.y. šešias briaunas, paiešką nutraukiame.



### 2.13.2 pav. Dalinio grafo – medžio konstravimas

2.13.2 d) paveiksle pavaizduotas dengiantis medis, gautas paieškos gilyn iš 1-osios viršūnės metodu, laikantis taisyklės, kad pirmiausia eisime į gretimą viršūnę, kurios numeris yra mažiausias.

Iš pirmosios viršūnės ateiname į naujają 2-ają viršūnę. Vadinas, (1,2) yra medžio briauna. Iš 2-osios viršūnės, po bandymo eiti į 1-ają viršūnę, ateiname į naujają 3-iąją viršūnę. Tuo būdu (2,3) yra medžio briauna. Iš 3-iosios viršūnės ateiname į naujają 4-ają viršūnę ir gausime dar vieną medžio briauną (3,4). Iš 4-osios atėjome į naujają 6-ają viršūnę, o iš pastarosios į naujas 5-ają ir 7-ają viršūnes. Tuo būdu medis pasipildys briaunomis (4,6),(6,5) ir (6,7). Kadangi į medži įtraukėme  $(n-1)$ , t.y. šešias briaunas, paiešką gilyn nutraukiame.

2.8.1 ir 2.8.2 paragrafuose detaliai aptarėme paieškos gilyn ir paieškos platyn organizavimo algoritmus. Šie algoritmai su nedidele modifikacija pilnai tinkta aptartiems medžio konstravimo metodams realizuoti.

Dengiantys medžiai yra naudojami sprendžiant įvairius uždavinius, pavyzdžiu, konstruojant algoritmą, kurio dėka nustatome, ar grafas yra plokštusis (žr. [RND80]).

Čia aptarsime vieną svarbų praktinį uždavinį, susijusį su **hidrauliniu vandentiekio tinklo skaičiavimu**.

Tarkime, kad 2.13.2 a) paveiksle pavaizduotas grafas yra vandentiekio (dujotiekio) tinklas. Šio grafo viršūnės vaizduoja sutelktinius vandens vartotojus, o briaunos – vandentiekio vamzdžius (žr. [APS83]). Tarkime, kad žinant vandentiekio vamzdžių skersmenis bei vandens šaltinių debitą [ $\text{l/s}$ ] reikia kiekviename tarpe (briaunoje) apskaičiuoti vandens debitą, vandens tekėjimo greitį bei slėgio kritimą.

Procesai, vykstantys vandentiekio tinkle, aprašomi Kirchgofo dėsniais.

**Pirmasis Kirchgofo dėsnis.** Kiek vandens atiteka į mazgą (viršūnę), tiek ir išteka, atmetus jo suvartojojamą mazgę.

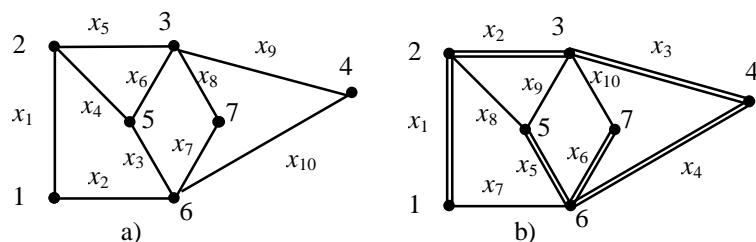
**Antrasis Kirchgofo dėsnis.** Slėgių kritimų suma žiede (nepriklausomame cikle) lygi nuliui.

Reikia pabrėžti, kad pirmasis Kirchgofo dėsnis užrašomas tiesine lygtimi, o antrasis – netiesine.

Tuo būdu, vandentiekio tinklo procesai aprašomi ( $n - 1$ ) tiesine lygtimi ( $n - (\text{viršūnių})$  mazgų skaičius), ir  $m - n + 1$  netiesine lygtimi.

Ši sistema sprendžiama taip. Pirmiausia apskaičiuojamas atskiras tiesinės dalies spredinys. Tam tikslui  $m - n + 1$  kintamasis parenkamas laisvai (remiantis inžineriniais samprotavimais), o likę  $(n - 1)$  kintamujų – apskaičiuojami iš  $(n - 1)$  tiesinių lygčių sistemos. Po to šis spredinys statomas į netiesines lygtis ir, jei tikslumas netenkinamas, spredinys iteracinių procedūros pagalba koreguojamas, taip, kad jis visą laiką tenkintų tiesines lygtis, o netiesinių lygčių netiktis artėtų į nulį.

Laisvai pažymėjus kintamuosius (debitus) ir surašius tiesines lygtis, sistema bus reta ir nenuliniai koeficientai matricoje bus išsidėstę netvarkingai.



**2.13.3 pav.** Sistemos nežinomųjų ir lygčių numeracija

2.13.3 a) pav. pavaizduotam tinklui tiesinės sistemos matrica bus tokia (varnelės žymi nenuliniaus koeficientus):

$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	$x_8$	$x_9$	$x_{10}$	Mazgo nr.
✓	✓									1
✓			✓	✓						2
				✓	✓		✓	✓		3
								✓	✓	4
		✓	✓		✓					5
	✓	✓				✓			✓	6

Kintamųjų  $x_7$ ,  $x_8$ ,  $x_9$  ir  $x_{10}$  reikšmes parinkus laisvai, teks spręsti šešių lygčių sistemą, kurios matrica yra kairiau dvigubos linijos. Norėdami išspręsti šią sistemą, pavyzdžiui, Gauso metodu, pirmiausia ją turėsime perskaičiuoti į trikampį pavidalą, o po to, atvirkštinio etapo metu, nuosekliai apskaičiuosime  $x_6$ ,  $x_5$ , ...,  $x_1$  reikšmes. Realių vandentiekio tinklų mazgų (viršūnių) skaičius yra šimtų eilės, o tarpų (briaunų) – tūkstančių eilės. Todėl laisvas nežinomujų ir lygčių sunumeravimas sprendimo eigoje išsauks dideles atminties ir laiko sąnaudas.

Kelkime klausimą: "Gal galima nurodyti tokią kintamųjų sunumeravimo eilę ir tokią lygčių surašymo tvarką, kad tiesinės sistemos dalies matrica būtų pseudotrikampė?" Atsakymas į šį klausimą teigiamas. Paieškos platyn arba gilyn metodu raskime dengiantį medį ir surašykime grafo briaunas ta tvarka, kaip jos buvo jungiamos į medį. Taip pat surašykime grafo viršunes ta tvarka, kuria jos buvo dengiamos konstruojant medį. Pavyzdžiui, paieškos gilyn metodu į medį briaunos buvo jungiamos tokia tvarka (1,2), (2,3), (3,4), (4,6), (6,5), (6,7), o viršūnės dengiamos taip: 1, 2, 3, 4, 6, 5 ir 7. Medžio briaunoms, jų surašymo tvarka, nuosekliai priskirkime kintamuosius, pradedant kintamuoju  $x_1$ . Kintamieji briaunoms, neprieklausančioms medžiui, priskiriami laisvai. Toks kintamųjų priskyrimas pavaizduotas 2.13.3 b) pav. Jame dviguba linija pažymėtos medžio briaunos. Tada, jei lygtis nuosekliai rašysime mazgams (viršūnėms), pradedant antruoju, ta tvarka, kokia jie buvo dengiami konstruojant medį, sistemos tiesinės dalies matrica bus pseudotrikampė. 2.13.3 b) pav.grafui ši mazgų tvarka būtų: 2, 3, 4, 6, 5 ir 7.

**Pastaba.** Remiantis medžio savybėmis, nesunku suvokti, kad taip sunumeravus kintamuosius ir lygtis, sistemos tiesinės dalies matrica bus pseudotrikampė.

Žemiau pateikta 3.13.3 b) grafo sistemos tiesinės dalies matrica. Matome, kad ji yra pseudotrikampė.

$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	$x_8$	$x_9$	$x_{10}$	Mazgo n.r
✓	✓						✓			2
	✓	✓					✓	✓		3
		✓	✓							4
			✓	✓	✓	✓				6
				✓			✓	✓		5
					✓				✓	7

Dabar, parinkus kintamujų  $x_7$ ,  $x_8$ ,  $x_9$  ir  $x_{10}$  reikšmes, iš karto gausime tiesinę lygčių sistemą su trikampe matrica, kurią pilnai nusako grafo gretumumo struktūra. Vadinas, šiuo atveju tiek sistemos saugojimui, tiek ir sprendimui atminties ir laiko sąnaudos bus minimalios.

### **Dengiančio medžio konstravimo algoritmai**

Aptarkime dengiančio medžio konstravimo algoritmus, pagrįstus paieška gilyn ir paieška platyn.

**Pirmasis algoritmas** (be rekursijos), pagrįstas paieška gilyn.

Duota:  $n$  – grafo viršūnių skaičius,

$m$  – grafo briaunų skaičius,

$L[1..2m]$  – grafo briaunų masyvas,

$lst[1..n+1]$  – grafo briaunų adresų masyvas.

Rasti: Dengiantį medį  $T$ , kuris nusakytas briaunų matrica  $B[1..2, 1..n-1]$ .

Procedūros vidiniai darbo masyvai ir kintamieji sutampa su paieškos gilyn (žr. 2.8.1.2 paragrafą) procedūros vidiniais darbo masyvais ir kintamaisiais.

*const c = 500;*

*type*

*mas = array [1..c] of integer;*

*matr = array [1..2, 1..c] of integer;*

*procedure medis1 (v, n, m : integer; L, lst : mas; var b : matr);*

{ Procedūra **medis1** konstruoja dengiantį medį paieškos gilyn iš viršūnės v metodu, kai grafas nusakytas L ir lst masyvais.

*Formalūs parametrai:*

*v – pradinė paieškos viršūnė,*

*n – grafo viršūnių skaičius,*

*m – grafo briaunų (lankų) skaičius,*

*L, lst – grafą nusakantys tiesioginių nuorodų masyvai,*

```

 $b$  – dengiančio medžio briaunų matrica. }

var  $i, k, u, is$  : integer;
 $t, p$  : boolean;
 $fst, prec$  : mas;
begin
{Incializacija}
for  $i := 1$  to  $n$  do begin
     $fst[i] := lst[i]+1; prec[i] := 0;$ 
end;
 $k := v;$ 
if  $fst[k] <= lst[k+1]$  then {yra nenagrinėtų briaunų, incidentiškų viršūnei  $k$ }
begin
     $t := false; p := true;$ 
     $prec[k] := k; \{ k – pradinė paieškos viršūnė \}$ 
    { Nagrinėti viršūnę  $k$  }
     $is := 0; \{ Medžio briaunų skaičius \}$ 
end
else {Viršūnė  $k$  yra arba izoliuota viršūnė, arba neturi išeinančių
lankų (orientuotojo grafo atveju); paieškos pabaiga }
 $t := true;$ 
while not  $t$  do { paieška nebaigta }
begin
{ Pirmyn }
while  $p$  do
begin
     $u := L[fst[k]];$ 
    if  $prec[u] = 0$  then {  $(k, u)$  – medžio briauna }
    begin
         $is := is + 1;$ 
         $b[1, is] := k;$ 
         $b[2, is] := u;$ 
        if  $is = n - 1$  then { Medis sukonstruotas }  $t := true;$ 
         $prec[u] := k; \{ j viršūnę u atėjome iš viršūnės k \}$ 
        if  $fst[u] <= lst[u+1]$  then { viršūnė u neišsemta }
         $k := u$ 
        else { viršūnė u išsemta }
    end
     $p := false;$ 
end
else
 $p := false; \{ viršūnė u nenauja \}$ 
end;

```

```

{ Atgal }
while not p and not t do
begin
    { Imama nauja, dar nenagrinėta briauna, incidentiška viršynei k }
    fst [k] := fst [k] + 1;
    if fst [k] <= lst [k + 1] then { tokia briauna egzistuoja } p := true
        else { viršūnė k išsemta }
    if prec [k] = k then { pradinė paieškos viršūnė išsemta;
        paieškos pabaiga } t := true
        else { grįžome į viršūnę, iš kurios buvome atėję į viršūnę k }
        k := prec [k];
    end;
end;
end;

```

**Antrasis algoritmas** (su rekursija), pagristas paieška gilyn.

Duota: Jungusis  $(n,m)$ -grafas  $G = (V, U)$ , nusakytas gretimumo struktūra:  
 $N(v)$  – aibė viršūnių, gretimų viršūnei  $v$ .

Rasti: Dengiantį medį  $(V, T)$ .

Iveskime masyvą  $naujas [1..n]$ ;  

$$naujas [v] = \begin{cases} \text{true}, & \text{jei } v - \text{nauja,} \\ \text{false}, & \text{priešingu atveju,} \end{cases} \quad v \in V .$$

```

procedure Medis2 (v);
    { Paieška gilyn sujungta su medžio briaunų radimu;
    kintamieji naujas ir T – globalieji}
begin
    naujas [v] := false;
    for u ∈ N (v) do
        if naujas [u] then {(v, u) – medžio briauna}
        begin
            T := T ∪ {(v, u)};
            Medis2 (u);
        end;
    end;
begin {pagrindinė programa}
    for v ∈ V do naujas [v] := true;
    T := ∅; {T – medžio briaunų aibė} r := bet kuri grafo viršūnė;
    Medis2 (r);
end;

```

### **Trečiasis algoritmas**, pagristas paieška platyn.

Duota: Jungusis  $(n,m)$ -grafas  $G = (V, U)$ , nusakytas gretumumo struktūra:

$N(v)$  – aibė viršūnių, gretimų viršūnei  $v$ .

Rasti: Dengiantį medį  $(V, T)$ .

Kaip ir antrajame algoritme, įveskime masyvą  $naujas [1..n]$ .

Tada medžio ieškojimo procedūra bus užrašoma taip:

```

procedure Medis3 ( $v$ );
{Paieška platyn sujungta su medžio briaunų radim.
Kintamieji naujas ir  $T$  – globalieji}
begin
    Eilė :=  $\emptyset$ ;
    Eilė  $\Leftarrow v$ ;
    naujas [ $v$ ] := false;
    while Eilė  $\neq \emptyset$  do
        begin
             $p \Leftarrow Eilė$ ;
            for  $u \in N(p)$  do
                if naujas [ $u$ ] then  $\{(p, u)\} – medžio briauna\}$ 
                begin
                     $T := T \cup \{(p, u)\};$ 
                    Eilė  $\Leftarrow u$ ;
                    naujas [ $u$ ] := false;
                end;
            end;
        end;
    end;

```

Pagrindinė programa sutampa su antrojo algoritmo pagrindine programa.

#### **2.13.2. Trumpiausio dengiančio medžio svoriniame grafe apskaičiavimo uždavinys**

**Apibrėžimas.** Jei  $(n,m)$ -grafo  $G = (V, U)$  kiekvienai briaunai  $(u, v)$  yra priskirtas svoris – realusis skaičius, tai grafas  $G$  vadinamas svoriniu grafu.

**Uždavinio formulavimas.** Duotas svorinis jungusis  $(n,m)$ -grafas  $G = (V, U)$ . Rasti trumpiausią dengiantį medį, t.y. dengiantį medį, kurio briaunų svorių suma būtų mažiausia tarp visų galimų dengiančių medžių.

**Pastaba.** Tolesniame dėstyme vietoje žodžių “briaunos svoris” naudosime žodžius “briaunos ilgis”.

Nors tai optimizavimo uždavinys, tačiau šio uždavinio tikslūs sprendimo algoritmai yra efektyvūs. Daugiausia naudojami du šio uždavinio sprendimo metodai: Kraskalo (Kruskal J.B.)<sup>4</sup> ir Primo (Prim R.C.)<sup>5</sup>.

Kraskalo algoritmas reikalauja  $O(m \log m)$  operacijų, o Primo (Deikstros) –  $O(n^2)$  operacijų. 1975 m. buvo sukurtas trumpiausio dengiančio medžio radimo algoritmas, reikalaujantis  $O(m \log \log n)$  operacijų (žr. Yao A.C.C. An  $O(m \log \log n)$  Algorithm for Finding Minimum Spanning Trees, Info. Proc. Let., 4, 1975, p.p. 21 – 23; arba Cheriton D., Tarjan R.E. Finding Minimum Spanning Trees, SIAM J. Comput., 5, 1976, p.p. 724 – 742).

Žemai aptarsime Kraskalo ir Primo metodus ir algoritmus.

**Kraskalo metodas.** Kraskalo metodą nusako teorema.

**Teorema.** Tarkime, kad  $G = (V, U)$  jungis pilnasis grafas ir visų jo briaunų ilgiai (svoriai) skirtinti. Tada egzistuoja vienintelis trumpiausias dengiantis medis, kuris konstruojamas taip: “iš likusių grafo  $G$  briaunų randame trumpiausią briauną ir ją įtraukiame į medį, jei ši briauna neišaukia ciklo su anksčiau paimtom medžio briaunom”.

**Pastaba.** Likusios grafo briaunos – tai grafo  $G$  briaunos, atmetus medžio briaunas ir briaunas, kurias bandėme įtraukti į medį.

**Irodymas.** Tarkime, kad  $H = (V, U_H)$  yra dengiantis medis, sukonstruotas teoremoje nurodytu metodu. Aišku, kad medžio  $H$  formavimas bus baigtas, kai bet kuri iš likusių grafo  $G$  briaunų iššauks ciklą su anksčiau paimtom medžio briaunom, t.y. su briaunom, priklausančiom aibei  $U_H$ . (žr. 4-ają medžio savybę). Tarkime, kad šis medis nėra trumpiausias. Tada egzistuoja trumpiausias dengianti medis  $T = (V, U_T)$  ir  $U_T \neq U_H$ . Kadangi  $U_T \neq U_H$ , tai tarkime, kad  $u_k$  – pirmoji iš aibės  $U_H$  briaunu, nepriklausanti aibei  $U_T$ , t.y. abiejų aibiu  $U_H$  ir  $U_T$  pirmieji ( $k - 1$ ) elementai sutampa, o  $u_k \in U_H$ , bet  $u_k \notin U_T$ .

Panagrinėkime briaunu aibę  $U_T \cup \{u_k\}$ . Aišku (žr. 4-ają medžio savybę), ši aibė turės vienintelį ciklą, kuriame bus bent viena briauna  $u_0 \notin U_H$ ; priešingu atveju medyje  $(V, U_H)$  yra ciklas. Aptarkime medį, kurį apibrėžia briaunu aibė  $W = U_T \cup \{u_k\} \setminus \{u_0\}$ . Grafas  $(V, W)$  tikrai yra medis, nes jis turi

<sup>4</sup> Kruskal J.B. On the shortest spanning subtree of a graph and the travelling salesman problem. Proc. Amer. Math. Soc., 1956, 7, p.p. 48 – 50.

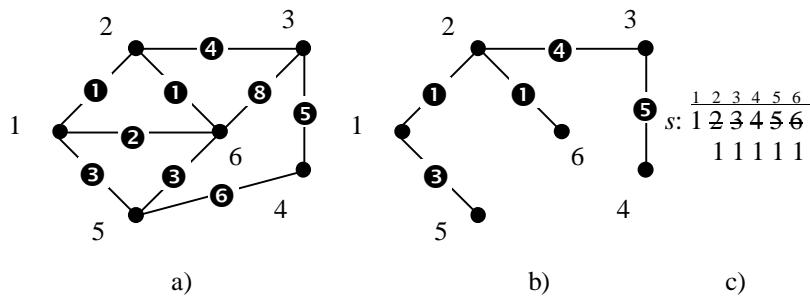
<sup>5</sup> Prim R.C. Shortest Connection Networks and Some Generalizations, Bell Syst. Tech. J., 36, 1957, p.p. 1389 – 1401. Dijkstra E. Two problems in Connexion with Graphs, Num. Math. 1, 1959, 259 – 271.

$(n-1)$  briauną ir neturi ciklų. Aišku, kad  $l(u_0)$  ( $u_0$ -osios briaunos ilgis) yra didesnis nei  $l(u_k)$  ( $u_k$ -osios briaunos ilgis). Iš tikro, kadangi briaunos  $u_1, u_2, \dots, u_{k-1}$  ir  $u_0$  priklauso aibei  $U_T$ , tai jos nesudaro ciklo. Formuojant medį  $H = (V, U_H)$  teoremoje nurodytu metodu,  $k$ -ajame žingsnyje iš visų likusių briaunų išrinkome trumpiausią briauną, kuri nesudarė ciklo su anksčiau paimtom briaunom  $u_1, u_2, \dots, u_{k-1}$ . Kadangi briauna  $u_0$  neiššaukia ciklo su briaunomis  $u_1, u_2, \dots, u_{k-1}$  ir šios briaunos neišrinkome, tai reiškia, kad  $l(u_0) > l(u_k)$ . Vadinasi, medžio  $(V, W)$  briaunų ilgių suma yra mažesnė (ilgesnę briauną  $u_0$  pakeitėme trumpesne briauna  $u_k$ ) nei, pagal prielaidą, paties trumpiausio dengiančio medžio  $T = (V, U_T)$  briaunų ilgių suma. Vadinasi, prielaida yra klaidinga ir medis  $H = (V, U_H)$  yra trumpiausias dengiantis medis.

**Pastaba.** Si teorema galioja ir tuo atveju, jei  $(n, m)$ - grafas  $G = (V, U)$  yra jungasis, bet nepilnasis ir kai kurių jo briaunų ilgiai yra lygūs. Šiuo atveju galime įsivaizduoti, kad grafas yra pilnasis, priimant, kad nesančių briaunų ilgiai yra begalybės.

Jei kai kurių briaunų ilgiai yra lygūs, tai bendru atveju gali egzistuoti keli ekvivalentūs sprendiniai. Tada to paties ilgio briaunas sunumeruose, ir jas bandysime jungti į medį jų numerių didėjimo tvarka. Pavyzdžiu, tarkime, kad  $l(u_1) = l(u_2) = l(u_3) = l$ , tada, jei konstruojant medį, trumpiausios briaunos ilgis bus lygus  $l$ , tai pirmiausia bandysim į medį įtraukti briauną  $u_1$ , po to – briauną  $u_2$ , o po to – briauną  $u_3$ .

**Pavyzdys.** Kraskalo metodu sukonstruokime trumpiausią dengiantį medį 2.13.4 pav. a) pavaizduotam grafui.



**2.13.4 pav.** Kraskalo metodas

2.13.4 b) pavaizduotas trumpiausias dengiantis medis, apskaičiuotas Kraskalo metodu, kai briaunas į medį bandėme traukti tokia tvarka: (1,2),

(2,6), (1,6), (1,5), (5,6), (2,3) ir (3,4). Pabrauktos briaunos į medį nebuvo įtrauktos, nes išsaukdavo ciklą su anksčiau paimtom medžio briaunom.

Aptarkime, kaip formaliai nustatyti, “*ar trumpiausia briauna iš likusių sudaro ciklą su anksčiau paimtom medžio briaunom?*”.

Tam tikslui įveskime masyvą  $s[1..n]$ , čia  $s_i$  yra jungiosios komponentės, kurias priklauso viršūnė  $i$ , numeris. Aišku, kad pradžioje, kai trumpiausią medį sudaro tuščiasis grafas, kiekviena viršūnė priklauso skirtingai jungajai komponentei, t.y., pradžioje  $s_i = i$ ,  $i = \overline{1, n}$ . Tarkime, kad konstruojant medį, trumpiausia briauna iš likusių yra briauna  $(u, v)$ , tada:

*if  $s[u] \neq s[v]$  then {briauna  $(u, v)$  neišsaukia ciklo su anksčiau paimtom medžio briaunom}*

```

begin
    briauną  $(u, v)$  įtraukti į medį.
    {perskaičiuoti jungiųjų komponenčių masyvo  $s$  elementus}
     $l := s[u]; k := s[v];$ 
    for  $i := 1$  to  $n$  do
        if  $s[i] = k$  then  $s[i] := l;$ 
    end;

```

2. 13. 4 c) pav. parodyta, kaip skaičiuojant kito jungiųjų komponenčių masyvo  $s$  turinys:

### **Kraskalo metodo algoritmas**

Duota:  $n$  – grafo viršūnių skaičius,  
 $m$  – grafo briaunų skaičius,  
 $b [1..2, 1..m]$  – jungiojo grafo briaunų matrica,  
 $c [1..m]$  – briaunų ilgių masyvas;  
 $c[j]$  yra briaunos  $(b[1,j], b[2,j])$  ilgis.

Rasti: Trumpiausią dengiantį medį, t.y. medžio briaunų matricą  $t [1..2, 1..n-1]$  ir jų ilgių masyvą  $d [1..n-1]$ .

Vidiniai darbo masyvai:

$s [1..n]$  – viršūnių jungiųjų komponenčių masyvas,  
 $p [1..m]$  – požymių masyvas;  
 $p[i] = \begin{cases} 0, & \text{jei } i - \text{oji briauna nenagrinėta,} \\ 1, & \text{priešingu atveju.} \end{cases}$

Žodžiai “briauna nagrinėta” reiškia, kad ji yra arba medžio briauna, arba ją bandėme įtraukti į medį.

```

const cc = 500;
type
    mas = array [1..cc] of integer;
    mas1 = array [1..cc] of real;
    matr = array [1..2, 1..cc] of integer;
procedure kraskalas (n, m : integer; b : matr; c : mas1; var t : matr;
                      var d : mas1);
{ Procedūra kraskalas apskaičiuoja jungiojo svorinio grafo, nusakyto briaunų
matrica, trumpiausią dengiantį medži.
Formalūs parametrai:
    n – grafo viršinių skaičius,
    m – grafo briaunų (lankų) skaičius,
    b – grafo briaunų matrica,
    c – briaunų ilgių masyvas,
    t [1..n-1] – trumpiausio dengiančio medžio briaunų matrica,
    d[1..n-1] – trumpiausio dengiančio medžio briaunų ilgių masyvas.}
var i, j, k, l, u, v, x, y : integer;
    sum, min : real;
    s, p : mas;
begin
for i := 1 to n do s [i] := i;
sum := 1;
for i := 1 to m do
begin
    begin
        p [i] := 0;
        sum := sum + c [i];
    end;
k := 0; { J medži įtrauktų briaunų skaičius }
while k < n - 1 do
begin
    begin
        { Rasti trumpiausią briauną iš likusių }
        min := sum;
        for i := 1 to m do
if (p [i] = 0) and (min > c [i]) then
begin
            min := c [i];
            l := i;
        end;
if min = sum then begin
            writeln ('Grafas – nejungusis');
            exit;

```

```

end;
{ Ar galima l-qją briauną įtraukti į medį? }
p [l] := 1; { l-oji briauna – nagrindėta }
u := b [1, l];
v := b [2, l];
if s [u] <> s [v] then
begin
    { Briauną (u, v) įtraukiame į medį }
    k := k + 1;
    t [1, k] := u; t [2, k] := v;
    d [k] := c [l];
    { Perskaičiuojame jungiamas komponentes }
    x := s [u]; y := s [v];
    for i := 1 to n do
        if s [i] = y then s [i] := x;
    end;
end;

```

end;

**Primo metodas.** Jei Kraskalas trumpiausią dengiantį medį apskaičiuoja taikydamas “godų” algoritmą, tai Primas šį uždavinį sprendžia “artimiausio kaimyno” metodu.

### Primo algoritmas

#### Nulinis žingsnis

- a)  $(n, m)$  - jungiojo grafo  $G = (V, U)$  visos viršūnės nenudažytos.
- b) Trumpiausias dengiantis medis  $T$  yra tuščiasis grafas, turintis  $n$  viršūnių.
- c) Pasirenkame bet kurią grafo  $G$  viršūnę ir ją nudažome.

#### Bendrasis žingsnis

while “grafas  $G$  turi nenudažytų viršūnių” do  
begin

- a) Randame trumpiausią briauną tarp nudažytų ir nenudažytų grafo  $G$  viršūnių. Tarkime, kad tokia briauna yra briauna  $(u, v)$ , čia  $u$  – nudažyta, o  $v$  – nenudažyta viršūnė.
- b) Briauną  $(u, v)$  įjungiamo į medį  $T$ .
- c) Nudažome grafo  $G$  viršūnę  $v$ .

end;

Kaip rasti trumpiausią briauną tarp nudažytų ir nenudažytų viršūnių?

Tam tikslui įveskime tris pagalbinius masyvus.

$$d[1..n], d[i] = \begin{cases} 0, & \text{jei grafo } G \text{ } i\text{-oji viršūnė nenudažyta,} \\ 1, & \text{priešingu atveju.} \end{cases}$$

$t[1..n]$  – trumpiausių briaunų tarp nudažytų ir nenudažytų viršūnių masyvas.

Tarkime, kad kažkuriamame bendrojo žingsnio etape trumpiausia briauna, jungianti nenudažytą viršūnę  $k$  su nudažtomis viršūnėmis yra briauna  $(k, s)$ , čia  $s$  – nudažyta viršūnė, gretima viršūnei  $k$ . Tada  $t[k] = l(k, s)$ , t.y. lygi briaunos  $(k, s)$  ilgiui.

Aišku, kad pradžioje (nuliniamame žingsnyje) visi  $t[k] = "begalybei"$ . ("Begalybė" šiam uždaviniui imamas skaičius  $\text{sum} = 1 + \sum_{u \in U} l(u)$ , čia  $l(u)$  – grafo  $u$ -osios briaunos ilgis).

$nr[1..n]$  – jei  $t[k] = l(k, s)$ , tai  $nr[k] = s$ . Kitaip tariant,  $nr[k]$  žymi numerį dažtos viršūnės, kuri riboja trumpiausią briauną tarp nedažytos viršūnės  $k$  ir dažytų viršūnių.

Aišku, kad pradžioje visi  $nr[k] = 0$ ,  $k = \overline{1, n}$ .

Tarkime, kad trumpiausios briaunos tarp nudažytų ir nenudažytų viršūnių ilgis kažkuriamame bendrojo žingsnio etape yra  $t_k = \min_{1 \leq i \leq n} (t_i / i$  - nenažyta viršūnė), o trumpiausia briauna –  $(k, nr[k])$ .

Itraukus briauną  $(k, nr[k])$  iš medij, viršūnė  $k$  nudažoma, o masyvai  $d$ ,  $t$ , ir  $nr$  perskaiciuojami taip:

```

 $d[k] := 1;$ 
 $\text{for } u \in N(k) \text{ do}$ 
     $\text{if } (d[u] = 0) \{ \text{viršūnė } u, \text{ gretima viršūnei } k, \text{ yra nenažyta} \} \text{ and}$ 
         $(t[u] > l(k, u) \{ \text{briaunos } (k, u) \text{ ilgis mažesnis už ilgi}$ 
             $\text{trumpiausios briaunos, jungiančios nenažyty viršūnę } u \text{ su}$ 
             $\text{kitomis nudažtomis viršūnėmis} \}$  then
                 $\text{begin}$ 
                     $t[u] := l(k, u);$ 
                     $nr[u] := k;$ 
                 $\text{end};$ 

```

Formaliai Primo algoritmą galima užrašyti taip.

Tarkime, kad  $(n,m)$ -grafas  $G$  yra jungasis ir nusakytas masyvais  $L[1..2m]$  ir  $lst[1..n+1]$ . Grafo  $G$  briaunų ilgai apibrėžti masyvu  $C[1..2m]$ , t.y. jei viršūnei  $k$  gretima viršūnė  $u$  yra patalpinta masyve  $L$  adresu  $l$ , tai briaunos  $(k,u)$  ilgis patalpintas masyve  $C$  tuo pačiu adresu  $l$ .

Apskaičiuoto trumpiausio medžio briaunos bus talpinamos matricoje  $B[1..2, 1..n-1]$ , o jų ilgai – masyve  $R[1..n-1]$ , t.y. briaunos  $(b[1,l], b[2,l])$  ilgis bus lygus  $R[l]$ .

```
const cc = 100;
type
    mas = array [1..cc] of integer;
    mas1 = array [1..cc] of real;
    matr = array [1..2, 1..cc] of integer;
var i, j, n, m : integer;
    c, r : mas1;
    b, bb : matr;
procedure BLClst (n, m : integer; b : matr; c : mas1 var L, lst : mas,
    var cl : mas1);
{ Procedūra BLClst perveda svorinio neorientuotojo grafo briaunų matricą į tiesioginių nuorodų masyvus L ir lst ir perskaičiuoja briaunų ilgio masyvą.
Formalūs parametrai:
```

$n$  – grafo viršūnių skaičius,  
 $m$  – grafo briaunų (lankų) skaičius,  
 $b$  – grafo briaunų matrica;  
 $(b[1,j], b[2,j])$  –  $j$ -toji grafo briauna;  
 $c$  – grafo briaunų ilgių masyvas;  
 $c[j]$  –  $j$ -osios briaunos ilgis.  
 $L, lst$  – grafo nusakantys tiesioginių nuorodų masyvai;  
 $cl$  – grafo briaunų ilgai, kai grafas užrašomas  $L$  ir  $lst$  masyvais.

Vidiniai kintamieji:

```
d[1..n] – viršūnių laipsnių masyvas;
d[i] –  $i$ -osios viršūnės laipsnis.
fst[1..n] – adresų masyvas;
pradžioje fst[i] = lst[i] + 1,  $i = 1, 2, \dots, n$ . fst[i] – tai adresas masyve  $L$ ,
kur turi būti talpinamas numeris pirmos viršūnės, gretimos viršūnei  $i$ . }
var i, j, k : integer;
    fst, d : mas;
begin
{ Viršūnių laipsnių apskaičiavimas. }
for i := 1 to n do d[i] := 0;
```

```

for i := 1 to 2 do    {*}
    for j := 1 to m do
        begin
            k := b [i, j];
            d [k] := d [k] + 1;
        end;
    { Masyvo lst formavimas }
    lst [1] := 0;
    for i := 1 to n do lst [i + 1] := lst [i] + d [i];
    { Masyvo L formavimas }
    for i := 1 to n do fst [i]:=lst [i] + 1;
    for j := 1 to m do
        begin
            k := b[1 ..j];
            L [fst [k]] := b [2, j];
            cl [fst [k]] := c [j];
            fst [k] := fst [k] + 1;
            k := b [2, j];      { **}
            L [fst [k]] := b [1, j]; { **}
            cl [fst [k]] := c [j];
            fst [k] := fst [k] + 1; { **}
        end;
    end;
procedure primo (n, m : integer; bb : matr; c : mas1; var b : matr; var r :
mas1);
{ Procedūra primo apskaičiuoja jungiojo svorinio grafo, nusakyto briaunų
matrica, trumpiausių dengiančių medžių.
Formalūs parametrai:
n – grafo viršūnių skaičius,
m – grafo briaunų (lankų) skaičius,
bb – grafo briaunų matrica,
c – briaunų ilgių masyvas,
b [1..n-1] – trumpiausio dengiančio medžio briaunų matrica,
r [1..n-1] – trumpiausio dengiančio medžio briaunų ilgių masyvas.}
var i, k, j, u, v, next : integer;
    sum, min : real;
    L, lst, d, nr : mas;
    cl, t : mas1;
begin
BLClst (n, m, bb, c, L, lst, cl);
sum := 1;

```

```

for i := 1 to m do sum := sum + c [i];
for i := 1 to n do
begin
  d [i] = 0;
  t [i] := sum;
  nr [i] := 0;
end;
next := n;
d [next] := 1;
k := 1; { Nudažytų viršūnių skaičius }
j := 0; { I medži trauktų briaunų skaičius }
while k <= n - 1 do
begin
{ Perskaičiuojame masyvus t ir nr }
for i := lst [next] + 1 to lst [next + 1] do
begin
  u := L [i];
  if (d [u] = 0) and (t [u] > cl [i]) then
begin
  t [u] := cl [i];
  nr [u] := next;
end;
end;
{ Randame trumpiausią briauną tarp nudažytų ir nenudažytų grafo G
viršūnių }
min := sum;
for i := 1 to n do
if (d [i] = 0) and (min > t [i]) then
begin
  min := t [i];
  v := i;
end;
if min = sum then begin
  writeln('Grafas – nejungasis');
  exit;
end;
{ Briauną (v, nr [v]) jutraukiame į medžio briaunų aibę }
j := j + 1;
b [1, j] := v; b [2, j] := nr [v];
r [j] := min;
next := v;

```

```

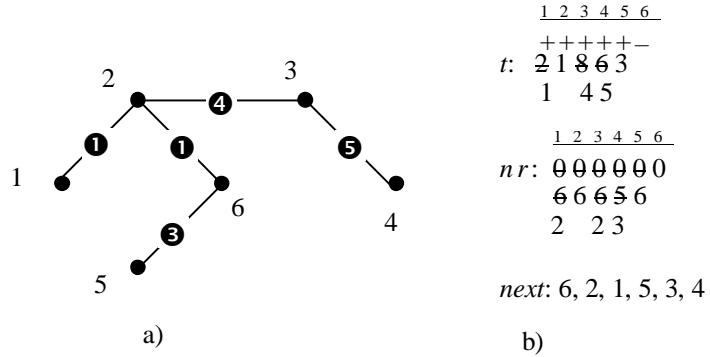
d [v] := 1;
k := k + 1;
end;
end;
begin
writeln;
writeln ('iveskite viršūnių skaičių');
read (n);
writeln ('iveskite briaunų skaičių');
read (m);
writeln ('iveskite briaunų matricą');
for i := 1 to 2 do
begin
for j := 1 to m do read (bb [i, j]);
readln;
end;
writeln ('iveskite briaunų ilgius');
for i := 1 to m do read (c [i]);
primo (n, m, bb, c, b, r);
writeln;
writeln ('Matr. b');
for i := 1 to 2 do
begin
for j := 1 to n - 1 do write (b [i, j] : 2);
writeln;
end;
writeln;
writeln ('Br. ilgiai');
for i := 1 to n - 1 do write (r [i] : 4 : 1);
end.

```

2.13.5 a) pav. pavaizduotas 2.13.4 a) pav. grafo trumpiausias dengiantis medis, kai pradžioje  $next = 6$ . Briaunų įtraukimo į medį tvarka buvo (6,2), (2,1), (6,5), (2,3) ir (3,4).

2.13.5 b) pav. parodyta, kaip skaičiuojant kito masyvų  $t$ ,  $nr$  ir kintamojo  $next$  turiniai.

**Pastaba.** Yra kitas, mažiau racionalus, nei aukščiau pateiktas trumpiausios briaunos tarp nudažytų ir nenudažytų viršūnių radimo būdas. Grafo briaunų aibę išrikiuokime jų ilgių didėjimo tvarka. Tada trumpiausia briauna tarp nudažytų ir nenudažytų viršūnių bus pirmoji sekos briauna, kuriai tik viena viršūnė nudažyta.



**2.13.5 pav.** Primo metodas

**Keli uždavinys.** Kaip buvo minėta anksčiau, Keli medžius naudojo norėdamas apskaičiuoti galimą cheminių junginių izomerų skaičių.

Keli uždavinys formuluojoamas taip: *plokštumoje duota n taškų. Keliais skirtingais būdais juos galima sujungti, kad gautasis grafas būtų medis?*

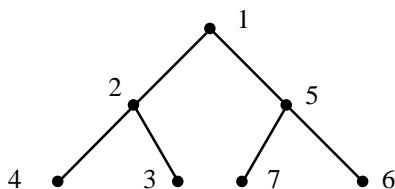
Tam, kad šį uždavinį galėtume išspręsti, pirmiausia aptarsime kompaktišką medžio užrašą – medžio kodą.

Tarkime,  $G = (V, U)$  yra medis. Šio medžio briaunas užrašykime laikydamiesi taisyklėj.

1. Rasti kabančią briauną, kurios kabančios viršūnės numeris yra pats mažiausias.
2. Užrašyti šią briauną. Rašant briauną, pirmiausia rašyti kabančios viršūnės numerij.
3. Ištrinti kabančią briauną drauge su kabančia viršūne.

Aišku, kad užrašydami medžio briaunas, taip elgsimės, kol medis turės bent vieną briauną.

Pavyzdžiu, 2.13.6 pav. medžio briaunų seka, sudaryta laikantis išvardintų taisyklėj, bus: (3,2), (4,2), (2,1), (1,5), (6,5) ir (5,7).



**2.13.6 pav.** Medžio kodas

Jei, laikantis šių taisyklių, medžio  $G = (V, U)$  briaunas surašysime iš eilės, tai gausime tokią briauną seką:  $(a_1, b_1), (a_2, b_2), \dots, (a_{n-2}, b_{n-2}), (a_{n-1}, b_{n-1})$ .

Remdamiesi šia seka, sudarykime aibę  $B = \{b_1, b_2, \dots, b_{n-2}\}$ . Ši aibė vienareikšmiškai nusako medį, t.y. kiekvienam medžiui atitiks vienintelė aibė  $B$  ir atvirkščiai, kiekviena aibė vieninteliu būdu nusakys medį. Šią aibę pavadinsime **medžio kodu**.

Nagrinėtam pavyzdžiui medžio kodas bus aibė  $B = \{2, 2, 1, 5, 5\}$ .

**Kaip, turint aibę  $B$ , rasti medžio briaunas?**

Tam tikslui elgsimės taip.

1. Sudarome aibę  $A = \{1, 2, 3, \dots, n\}$ .
2. Aibėje  $A$  ieškome pirmo elemento, nepriklausančio aibei  $B$ . Tarkime, tas elementas yra  $a_1$ . Tada  $(a_1, b_1)$  yra medžio briauna, čia  $b_1$  yra pirmasis aibės  $B$  elementas.
3.  $a_1$  šaliname iš aibės  $A$ , o  $b_1$  šaliname iš aibės  $B$ , t.y.  $A := A \setminus \{a_1\}$ ,  $B := B \setminus \{b_1\}$ .

Aišku, kad taip elgsimės tol, kol aibė  $B$  turės bent vieną elementą. Kai aibė  $B$  taps tuščia, tai aibėje  $A$  liks du elementai, kurie ir apibrėž paskutinę medžio briauną. Rašydami šią briauną pirmiausia rašysime mažesnį elementą.

Laikantis šių taisyklių, gausime tą pačią medžio briauną seką, kuria remdamiesi ir sudarėme medžio kodą.

Nagrinėtam pavyzdžiui turėsime:  $A = \{1, 2, 3, 4, 5, 6, 7\}$  ir  $B = \{2, 2, 1, 5, 5\}$ .

Pirmasis aibės  $A$  elementas, nepriklausantis aibei  $B$ , yra 3. Vadinas, (3,2) yra medžio briauna. Iš aibės  $A$  pašalinę elementą 3, o iš aibės  $B$  – elementą 2, gausime:  $A = \{1, 2, 4, 5, 6, 7\}$  ir  $B = \{2, 1, 5, 5\}$ . Dabar pirmasis aibės  $A$  elementas, nepriklausantis aibei  $B$ , yra 4. Vadinas, (4,2) yra medžio briauna. Po atitinkamų elementų pašalinimo iš aibų  $A$  ir  $B$ , gausime:  $A = \{1, 2, 5, 6, 7\}$ , o  $B = \{1, 5, 5\}$ . Šios aibės nusakys briauną (2,1), ir po perskaičiavimo aibės bus:  $A = \{1, 5, 6, 7\}$  ir  $B = \{5, 5\}$ . Iš jų gausime (1,5), ir po perskaičiavimo  $A = \{5, 6, 7\}$ , o  $B = \{5\}$ . Iš šių aibų gausime briauną (6,5), o po aibų  $A$  ir  $B$  perskaičiavimo gausime:  $A = \{5, 7\}$ , o  $B = \emptyset$ . Vadinas, paskutinioji briauna yra (5,7). Kaip matome, gavome tą pačią medžio briauną seką.

Iš šio nagrinėjimo darome išvadą, kad medžio kodas yra  $(n-2)$ -jų elementų iš aibės  $A = \{1, 2, 3, \dots, n\}$  rinkinys su pasikartojančiais elementais. Rinkinys nuo rinkinio skiriasi arba pačiais elementais, arba jų tvarka, t.y.

turime gretinius iš  $n$  elementų po  $n - 2$  su pasikartojimais:  $\overline{A_n^{n-2}}$ . Šiu junginių skaičius lygus  $n^{n-2}$ . Vadinasi, skirtingų Keli medžių yra  $n^{n-2}$ .

Pavyzdžiuui, kai  $n = 4$ , skirtingų Keli medžių bus  $4^2 = 16$ , kuriuos nusakys kodai:

1,1 ; 1,2 ; 1,3 ; 1,4 ; 2,1 ; 2,2 ; 2,3 ; 2,4 ;  
3,1 ; 3,2 ; 3,3 ; 3,4 ; 4,1 ; 4,2 ; 4,3 ; 4,4 .

Pavyzdžiuui, kodas – aibė  $B = \{2,3\}$  duos medj: (1,2) ; (2,3) ; (3,4), o kodas – aibė  $B = \{3,2\}$  duos medj: (1,3) ; (3,2) ; (2,4).

**Šeinerio uždavinys grafe.** Duotas jungusis svorinis grafas  $G = (V, U)$  ir viršūnių aibės  $V$  poaibis  $A$ . Rasti junguji pografi  $T$ , tenkinantį sąlygas:

- 1) poaibis  $A$  yra pografinio  $T$  viršūnių poaibis,
- 2) pografinio  $T$  briaunų suma turi būti mažiausia tarp visų pografių, tenkinančių 1) sąlygą.

Aišku, kad Šeinerio uždavinio sprendinys yra medis. Jis vadinamas Šeinerio medžiu. Aišku taip pat, kad Šeinerio medžio radimo uždavinys ekvivalentus uždaviniui: tarp visų grafo  $G$  indukuotų pografių, kuriems aibė  $A$  yra jų viršūnių poaibis, rasti minimalų medj

Néra žinomi efektyvūs šio uždavinio sprendimo metodai. Aišku, kad tikslų sprendinį duos pilnas grafo  $G$  indukuotų pografių, kuriems aibė  $A$  yra jų viršūnių poaibis, perrinkimas. Tačiau šis metodas reikalautų perrinkti  $2^{n-k}$  variantų, čia  $|V| = n$ , o  $|A| = k$ . Todėl sprendžiant šį uždavinį naudojamos įvairios euristikos. Aptarkime vieną iš euristikų, kuri iš esmės yra artimiausio kaimyno metoda.

Tarkime, duotas svorinis grafas  $G = (V, U)$  ir jo viršūnių poaibis  $A$  (poaibio  $A$  viršūnės vadinamos Šeinerio viršūnėmis). Tarkime, kad į Šeinerio tinklą jau įtrauktos Šeinerio viršūnės  $a_1, a_2, \dots, a_t$  ir viršūnės  $v_1, v_2, \dots, v_l$ , priklausančios aibei  $V \setminus A$ . Tada į tinklą įtrauksimė tą Šeinerio viršūnę  $a_k$ , kuriai  $d(a_k, u) = \min(d(a_k, v) / v \in \{a_1, a_2, \dots, a_t\} \text{ arba } v \in \{v_1, v_2, \dots, v_l\})$ , t.y. tą viršūnę, kuri nuo tinklo viršūnių yra arčiausiai.

*begin*

*Visos grafo  $G$  viršūnės – nenudažyto. Išrenkame Šeinerio viršūnę, nuo kurios atstumų iki likusių Šeinerio viršūnių suma yra mažiausia, t.y. viršūnę  $a$ , kuriai  $\sum_{v \in A} d(a, v) \rightarrow \min$ . Šeinerio viršūnę a nudažome.*

*while "yra nenudažytų Šeinerio viršūnių" do  
begin*

Tarp nenudažytų Šteinerio viršūnių randame tokią viršūnę, nuo kurios atstumas iki nudažytų grafo  $G$  viršūnių yra mažiausias. Tarkime, kad tai Šteinerio viršūnė "next", o jai artimiausia nudažyta grafo viršūnė yra  $v$ . (Aišku, kad nudažytos grafo viršūnės priklauso Šteinerio tinklui).

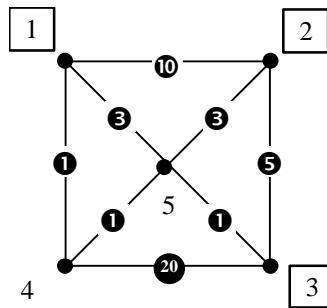
Tada i Šteinerio tinklą įtraukiame trumpiausią grandinę, jungiančią viršūnę "next" su viršūne  $v$ .

Visas nenudažytas šios grandinės viršūnes nudažome.

*end;*

*end;*

**Pavyzdys.** Panagrinėkime 2.13.7 pav. pavaizduotą grafą. Šio grafo Šteinerio viršūnės yra {1,2,3}. 2.13.7 pav. jos pažymėtos kvadratukais.



**2.13.7 pav.** Šteinerio uždavinys

$$\text{Trumpiausių kelių tarp Šteinerio viršūnių matrica } C = \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & 5 & 3 \\ 2 & 5 & 0 & 4 \\ 3 & 3 & 4 & 0 \end{array}.$$

Kadangi 3-ijai viršūnei atstumų iki likusių Šteinerio viršūnių suma yra mažiausia, tai išrenkame 3-iją viršūnę ir ją nudažome. Tada trumpiausios grandinės, jungiančios nenudažytas Šteinerio viršūnes su nudažtomis grafo  $G$  viršūnėmis, ilgis yra 3, o grandinė yra 1,4,5,3. Šią grandinę į Jungiame į Šteinerio tinklą, o 1-ają, 4-ają ir 5-ają viršūnes nudažome. Likusioji nenudažyta 2-oji Šteinerio viršūnė yra arčiausiai nudažytos 5-osios viršūnės. Todėl į Šteinerio tinklą įjungiamo trumpiausią grandinę – briauną (2,5) ir nudažome Šteinerio viršūnę – 2-ają viršūnę. Kadangi visos Šteinerio viršūnės nudažytos, tai Šteinerio tinklas sudarytas, ir jį sudarys briaunos (1,4); (4,5); (5,3) ir (2,5).

## 2.14. Optimalių kelių ieškojimas

Aptarsime kelis optimalių kelių grafe ieškojimo uždavinius:

- 1) trumpiausio kelio radimo uždavinį,
- 2) plačiausios siauros vietos (didžiausios keliamosios galios) radimo uždavinį,
- 3) uždavinį apie stiprinimą,
- 4) ilgiausio kelio radimo uždavinį,
- 5) trumpiausių kelių tarp visų viršūnių porų apskaičiavimo uždavinį.

### 2.14.1. Trumpiausio kelio radimo uždavinys

**Uždavinio formulavimas.** Duotas svorinis grafas  $G=(V, U, C)$ , čia  $C[1..m]$  – grafo briaunų svorių masyvas.

Rasti:

- 1) trumpiausius kelius nuo viršūnės  $s$  iki visų likusių grafo viršūnių,
- 2) trumpiausią kelią nuo viršūnės  $s$  iki viršūnės  $t$ .

Kadangi 2)-asis uždavinys yra 1)-ojo uždavinio atskirasis atvejas, tai čia nagrinėsime 1)-ajį uždavinį. 2)-ojo uždavinio sprendimo algoritmas pilnai sutaps su 1)-ojo uždavinio sprendimo algoritmu, išskyrus algoritmo pabaigos sąlygą.

**Pastaba.** Trumpiausią kelių ieškojimo besvoriniame grafe uždavinys aptartas 2.9 paragrafe.

Yra keli 1)-ojo uždavinio sprendimo metodai, iš kurių efektyviausias yra Deikstros algoritmas (žr. Dijkstra E.W. A note on two problems in connection with graphs, Numer. Math., 1959, 1, p.p. 259-271), kurį čia ir panagrinėsime.

Paprastai mus domina ne tiktais trumpiausio kelio ilgis, bet ir per kurias viršunes šis kelias eina. Tam tikslui įvesime du masyvus:

$d[1..n]$  – kelių ilgių masyvas, čia  $d[k]$  – trumpiausio kelio nuo viršūnės  $s$  iki viršūnės  $k$  ilgis, ir

$prec[1..n]$  – masyvas, parodantis per kurias viršunes kelai eina;

$$prec[u] = \begin{cases} k, & \text{jei trumpiausias kelias i viršūnę } u \text{ ateina iš viršūnės } k, \\ u, & \text{jei kelias prasideda viršūnėje } u. \end{cases}$$

### Deikstros algoritmas

*begin*

#### Paruošiamasis žingsnis

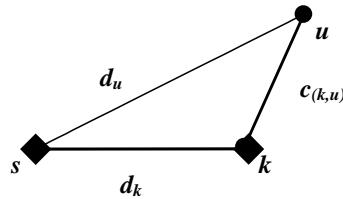
Visos svorinio grafo  $G=(V, U, C)$  viršūnės – nenudažytos.

$$sum = \sum_{i=1}^m c_i + 1; \{ begalybė \}$$

```

 $d[i] := sum; \quad prec[i] := 0; \quad i = \overline{1, n}.$ 
 $d[s] := 0; \quad \{d(s,s)=0\}$ 
 $prec[s] := s; \{ \text{visi keliai prasidesta viršūnėje } s \}$ 
Pagrindiniai skaičiavimai
while "yra nenudažytų viršūnių" do
begin
    {Išrinkti nenudažytą viršūnę k, iki kurios trumpiausio kelio nuo
    viršūnės s ilgis yra nusistovėjęs. Tam tikslui, tarp masyvo d elementų,
    atitinkančių nenudažytas viršūnes, reikia rasti mažiausią elementą, t.y.
     $d[k]=\min(d[v] / v \text{ nenudažyta grafo viršūnė})$ }
    min:=sum;
    for i:=1 to n do
        if "viršūnė i nenudažyta" and ( $\min > d[i]$ )
        then
            begin
                min:= d[i]; k:=i ;
            end;
        if min=sum
        then
            begin
                "grafas G – nejungusis"; stop;
            end
        else
            begin
                Nudažome viršūnę k;
                {Perskaičiuojame masyvų d ir prec elementus; jų
                perskaičiavimas pagristas 2.14.1 pav.}
                for u $\in N(k)$  do
                    if "u nenudažyta viršūnė" and  $d[u] > d[k] + c(k,u)$ 
                    { $c(k,u) – tai (k,u)$  briaunos ilgis}
                    then {iki viršūnės u radome naują kelį,
                        trumpesnį už iki šiol žinomą iki šios
                        viršūnės trumpiausią kelio ilgį  $d[u]$ }
                    begin
                         $d[u] := d[k] + c(k,u);$ 
                         $prec[u] := k$ 
                    end;
                end; {else}
            end; {while}
    end;

```



2.14.1 pav. Deikstros algoritmas

Kodėl formulė  $d[k] = \min_{v \in V} (d[v] / v - nenudažytos viršūnės)$  apskaičiuoja viršūnę, iki kurios trumpiausias kelias nuo viršūnės  $s$  – nusistovėjęs? Tarkime priešingai, kad iki viršūnės  $k$  yra dar trumpesnis kelias. Tuo atveju turi egzistuoti tokia nenudažyta viršūnė  $l$ , kad  $d[l] + c(l, k) < d[k]$ . Tačiau to negali būti, nes  $d[k] = \min_{v \in V} (d[v] / v - nenudažytos viršūnės)$ .

**Pastaba 1.** 2)-ajam uždavinui ciklo *while* antraštę reikėtų perrašyti taip: *while* “viršūnė  $t$  nenudažyta” *do*.

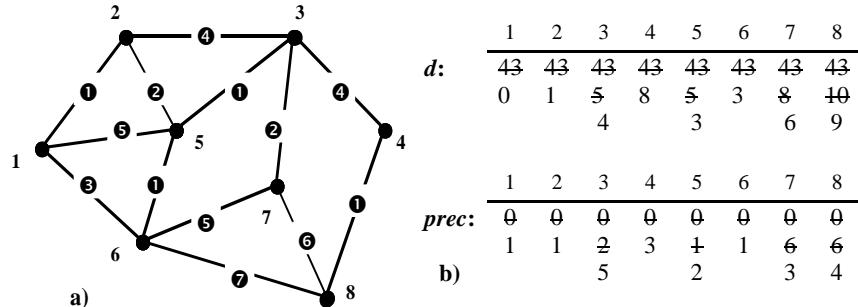
**Pastaba 2.** Deikstros algoritmas teisingai skaičiuos net ir tuo atveju, jei dalis grafo briaunų svorių bus neigiami skaičiai, esant sąlygai, kad grafas neturėtų ciklo, kurį sudarytų neigiamo svorio briaunos. Šiuo atveju reiktų modifikuoti tą pateikto Deikstros algoritmo vietą, kur kalbama apie masyvą  $d$  ir  $prec$  elementų perskaičiavimą. Nudažius viršūnę  $k$ , turėtume nagrinėti visas viršūnei  $k$  gretimas viršunes, tiek nenudažytas, tiek ir nudažytas. Jei rastume trumpesnį kelią iki nudažytos viršūnės, tai viršūnės spalvą nuvalytume, t.y. viršūnė taptų nenudažyta. Formaliai šį nagrinėjimą užrašytume taip:

```

for  $u \in N(k)$  do
    if  $d[u] > d[k] + c(k, u)$ 
        then
            begin
                 $d[u]:= d[k] + c(k, u);$ 
                 $prec[u]:= k;$ 
                if “viršūnė  $u$  buvo nudažyta”
                    then viršūnės  $u$  spalvą nutriname;
            end;
    
```

Algoritmas bus vykdomas, jei bus bent viena nenudažyta grafo viršūnė, t.y. algoritmo pabaigos požymis yra: “visos grafo viršūnės yra nudažytos”.

**Pavyzdys.** Panagrinėkime 2.14.2 pav. pavaizduoto svorinio grafo trumpiausių kelių nuo 1-osios viršūnės iki visų likusių apskaičiavimą Deikstros metodu. 2.14.2 pav. pavaizduotas masyvų  $d$  ir  $prec$  turinių kitimas. Šiam grafui parametru *sum* (begalybė) reikšmė yra 43.



2.14.2 pav. Trumpiausi keliai

Pavyzdžiu, trumpiausias kelias nuo 1-osios viršūnės iki 8-osios viršūnės yra 9, t.y.  $d(1,8)=9$ . Kelias eina per viršūnes: 8,4,3,5,2,1. Nesunku suvokti, kad masyvas  $prec$  nusako visų trumpiausią kelių, nuo viršūnės 1 iki visų likusių viršūnių, medį.

#### 2.14.2. Didžiausios keliamosios galios (plačiausios siauros vietas) apskaičiavimo uždavinys

**Uždavinio formulavimas.** Duotas svorinis grafas  $G=(V,U,C)$ , čia, kaip ir anksčiau,  $C[1..m]$  briaunų svorių masyvas. Tarkime, kad briauna vaizduoja kelią, jungiantį viršūnėms atitinkančias gyvenvietes. Tarkime, tame kelyje yra tiltas per upę, ir briaunos svoris reiškia to tilto keliamąją galią. Apskaičiuoti, kokius didžiausio svorio krovinius galima nuvežti iš viršūnės  $s$  iki

- 1) viršūnės  $t$ ;
- 2) visų likusių grafo viršūnių.

Kadangi, 1)-asis uždavinys yra 2)-ojo uždavinio atskirasis atvejas, tai čia nagrinėsime antrajį uždavinį. Pirmojo uždavinio sprendimo algoritmas skiriiasi nuo antrojo uždavinio sprendimo algoritmo tik pabaigos sąlyga. Jei antrojo uždavinio sprendimo algoritmo pabaigos sąlyga yra: "visos grafo viršūnės nudažytos", tai pirmojo – "viršūnė  $t$  nudažyta".

Didžiausios keliamosios galios apskaičiavimo uždavinio sprendimo algoritmas yra analogiškas Deikstros algoritmui.

Apibrėžkime masyvus  $d[1..n]$  ir  $prec[1..n]$ .

Masyvo  $d$  elementas  $d[i]$ - reiškia didžiausią svorį krovinio, kurį galima nugabenti iš viršūnės  $s$  iki viršūnės  $i$ . Pradžioje visi  $d[i]=0$ ,  $i = \overline{1, n}$ .

Masyvo  $prec$  elementas čia, kaip ir 2.14.1 paragrafe, reiškia:

$$prec[i] = \begin{cases} k, & \text{jei kelias į viršūnę } i \text{ ateina iš viršūnės } k, \\ i, & \text{jei kelias prasidesta viršūnėje } i. \end{cases}$$

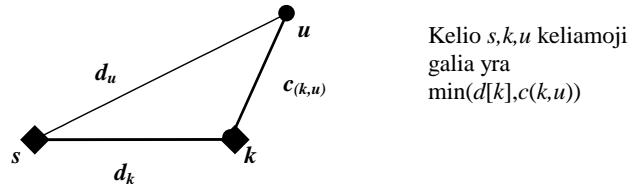
*Algoritmas*

```

begin
Paruošiamasis žingsnis
    Visos svorinio grafo  $G=(V,U,C)$  viršūnės – nenudažytos.
     $sum := \sum_{i=1}^m c_i + 1;$  {begalybė}
    for  $i:=to n$  do
        begin
             $d[i]:=0;$   $prec[i]:=0$ 
        end;
     $d[s]:= sum;$   $prec[s]:= s;$ 
Pagrindiniai skaičiavimai
    while “yra nenudažytų viršūnių” do
        begin
            {apskaičiuoti  $d[k] = \max_{v \in V} (d(v) / v - nenudažyta viršūnė)}$ 
             $max:=0;$ 
            for  $i:=1$  to  $n$  do
                if “viršūnė  $i$  nenudažyta” and ( $max < d[i]$ ) then
                    begin
                         $k:=i;$ 
                         $max:= d[i]$ 
                    end;
                if  $max=0$  then
                    begin
                        “grafas  $G$  - nejungusis”; stop
                    end
                else
                    begin
                        Nudažome viršūnę  $k$ ;
                        for  $u \in N(k)$  do
                            if “viršūnė  $u$  nenudažyta” and ( $d[u] < \min(d[k], c(k,u))$ )
                                then
                                    begin
                                         $d[u]:= \max(d[u], \min(d[k], c(k,u)))$ 
                                         $d[u]:= \min(d[k], c(k,u));$ 
                                         $prec[u]:= k$ 
                                    end;
                            end; {else}
                        end; {while}
                    end;

```

Masyvų  $d$  ir  $prec$  elementų perskaičiavimo pagrindimas parodytas 2.14.3 pav.



2.14.3 pav. Didžiausias svoris

Kelio  $s,k,u$  didžiausia keliamoji galia yra  $\min(d[k], c(k,u))$ . Vadinas, jei šio naujojo kelio galia yra didesnė už visų žinomų kelių, jungiančių viršūnę  $s$  su viršūne  $u$ , keliamają galią  $d[u]$ , tai,  $d[u]:=\min(d[k], c(k,u))$ , o  $prec[u]:=k$ , t.y. kelias iš viršūnė  $u$  ateina iš viršūnės  $k$ .

### 2.14.3. Uždavinys apie stiprinimą

**Uždavinio formulavimas.** Tarkime, kad  $G=(V,U,C)$  yra orientuotas grafas, neturintis ciklų ir turintis vieną viršūnę, neturinčią įeinančių lankų (minorantą) ir vieną viršūnę, neturinčią išeinančių lankų (mažorantą). Toks grafas vadinamas **tinkliniu grafu**. Tarkime, kad lanko svoris reiškia stiprinimą. Kelio stiprinimas, tai visų jo lankų stiprinimų sandauga. Tarp viršinių  $s$  ir  $t$  rasti kelią, kurio stiprinimas yra didžiausias.

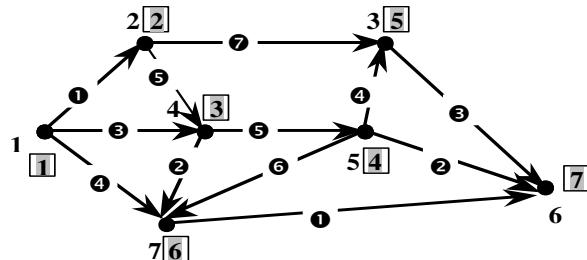
**Pavyzdys.** Tarkime, transportuojant prekę iš pagaminimo punkto iš pardavimo punktą, kiekvienam maršruto intervale prarandame dalį produkto. Koks turi būti prekės maršrutas, kad pervežimo nuostoliai būtų mažiausi.

Ši uždavinij galime suformuluoti, kaip kelio su didžiausiu stiprinimu uždavinij, jei kiekvienam lankui priskirsime dalį produkto, kuris be nuostolių pervežamas šiuo lanku.

Aišku, kad šio uždavinio sprendimo algoritmas yra visiškai toks pats, kaip ir didžiausios keliamosios galios uždavinio sprendimo algoritmas, išskyrus tai, kad pradžioje  $d[s]=1$ , o perskaičiuojant masyvą  $d$  ir  $prec$  elementus, kai nudažoma viršūnė  $k$  (žr. 2.14.3 pav.) elgsimės taip:

```
for  $u \in N(k)$  do
    if  $d[u] < d[k] * c(k,u)$ 
        then
            begin
                 $d[u]:=d[k]*c(k,u);$ 
                 $prec[u]:=k;$ 
                if "viršūnė u nudažyta" then "viršūnės spalvą nutriname"
            end;
```

**Pavyzdys.** Grafui, pavaizduotam 2.14.4 pav., tarp 1-osios ir 6-osios viršūnių raskime kelią, kurio stiprinimas būtų didžiausias.



2.14.4 pav. Didžiausias stiprinimas

Žemiau parodyta, kaip kito masyvų  $d$  ir  $prec$  turiniai. Žvaigždutės virš masyvo  $d$  elementų žymi kiek kartų šios viršūnės buvo dažytos: nubrauktos žvaigždutės žymi, kad viršūnė, kurią nusako elemento eilės numeris, pradžioje buvo nudažyta, o po to skaičiuojant jos spalva buvo nutrinta.

	d:							prec:						
	1	2	3	4	5	6	7	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	60	3	15	4	4		1	1	5	+	4	7	+
	100	5	25	30	6		90	5	2	4	5	4		7
						480	150				3	5		
						300						3		

Šis uždavinys būtų sprendžiamas žymiai paprasčiau, jei grafo viršūnės būtų sunumeruotos teisingai.

**Apibrėžimas.** Orientuotojo aciklinio grafo viršūnės *sunumeruotos teisingai*, jei kiekvienam lankui  $(i,j)$  galioja sąlyga  $i < j$ .

Vienas iš paprasčiausiu teisingo numeravimo metodų yra rangų metodas.

**Apibrėžimas.** Orientuotojo aciklinio grafo viršūnės *u rangas*, tai ilgiausias kelias (pagal lankų skaičių) nuo minorantos iki viršūnės *u*.

Pavyzdžiu, 2.14.4 pav. tinklinio grafo viršūnių rangai surašyti 2.14.1 lentelėje.

**2.14.1 lentelė.** Viršunių  
rangai

rangas	viršūnės Nr.
0	1
1	2
2	4
3	5
4	3,7
5	6

Aišku, kad to paties rango viršūnės nėra gretimos, todėl jos gali būti numeruojamos laisvai. Vadinasi, norint teisingai sunumeruoti grafą, reikia iš eilės einančiais natūralaisiais skaiciuais numeruoti viršunes, pradedant nulinio rango viršune. Laikantis šio principo, nagrinėjamam pavyzdžiuui gausime tokį teisingo sunumeravimo masyvą  $nr(1,2,5,3,4,7,6)$ ; jei  $nr[k]=l$ , tai  $k$  viršūnės teisingas numeris yra  $l$  (žr. 2.14.4 pav.). Teisingi numeriai pažymėti kvadrateliuose.

Tarkime, kad grafas sunumeruotas teisingai. Tada didžiausio stiprinimo algoritmas bus toks.

```

begin
    Paruošiamasis žingsnis
        for i:=1 to n do
            begin
                d[i]:=0;
                prec:=[i]:=0
            end;
        d[1]:=1; prec[1]:=1;
    Pagrindiniai skaičiavimai
        for k:=1 to n-1 do
            for u ∈ N(k) do
                if d[u] < d[k] * c(k,u) then
                    begin
                        d[u]:= d[k]* c(k,u);
                        prec[u]:= k;
                    end;
            end;
    end;
```

Žemiau parodyta, kaip remiantis teisinga grafo numeracija ir šiuo algoritmu, kito masyvų  $d$  ir  $prec$  turiniai.

$$\begin{array}{ccccccccc}
 & \square & \square & \square & \square & \square & \square & \\
 d: & \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} & & & & & & \\
 & 1 & 1 & 3 & 25 & 7 & 4 & 50 \\ & & & 5 & & 100 & 40 & 300 \\ & & & & & & & 150
 \end{array}
 \quad
 \begin{array}{ccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \\
 prec: & \begin{array}{ccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 1 & 4 & 3 & 2 & 4 & 4 \end{array} & & & & & & \\
 & 2 & 4 & 3 & 5 & & & & \\
 & & & & & & & 4
 \end{array}$$

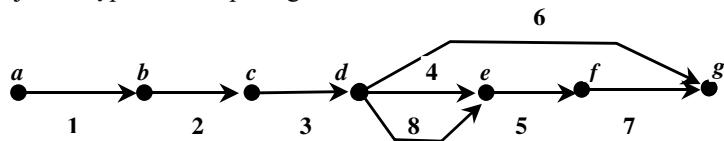
#### 2.14.4. Ilgiausio kelio uždavinys

2.14.3 paragafe apibrėžėme tinklinį grafą. Reikia pabrėžti, kad tinkliniai grafaip yra patogi priemonė įvairiems projektams vaizduoti. Pavyzdžiu, norint pastatyti naujā namą, turi būti atliki tokie darbai:

- |                               |                             |
|-------------------------------|-----------------------------|
| 1) sklypo išvalymas;          | 5) tinkavimas;              |
| 2) pamatų montavimas;         | 6) teritorijos sutvarkymas; |
| 3) sienų mūrijimas;           | 7) apdailos darbai;         |
| 4) elektros laidų pravedimas; | 8) stogo uždengimas.        |

Aišku, kad darbai turi eiliškumą, o taip pat reikalauja laiko sąnaudų bei materialinių resursų.

Namo statybos projektą galima pavaizduoti tinkliniu grafu (žr. 2.14.5 pav.). Grafo lankai vaizduoja darbus. Lanko svoris gali vaizduoti darbo trukmę arba šiam darbui atliki reikalingų materialinių resursų kiekį. 2.14.5 pav. skaičiai prie lankų vaizduoja anksčiau išvardintų darbų numerius. Viršūnės vaizduoja darbų pradžios ir pabaigos momentus.



2.14.5 pav. Namo statybos grafas

Galimi įvairūs uždavinių formulavimai.

1. Tarkime, kad lanko svoris yra lankui atitinkančio darbo trukmė. Rasti trumpiausią laiką, per kurį, turint neribotus resursus, projektas gali būti įvykdytas.

Aišku, kad trumpiausias projekto įvykdymo laikas yra lygus ilgiausio kelio (kritinio kelio) nuo minorantos iki mažorantos ilgiui. Darbai, priklausa kritiniam kelui, vadinami kritiniais darbais. Bet kokio kritinio darbo suvėlinimas iššaukia viso projekto įvykdymo vėlinimą.

2. Kaip organizuoti darbus, kad, esant ribotiemis resursams, projektas būtų įvykdytas galimai trumpiausiu laiku.

3. Kaip organizuoti darbus ir koks reikalingas minimalus resursų kiekis, kad projektas būtų įvykdytas per nurodytą laiką (aišku, laikas netrumpesnis nei kritinio kelio ilgis).

Čia aptarsime 1-ojo uždavinio sprendimą.

#### **Kritinio kelio tinkliniame grafe radimo algoritmas**

Apibrėžkime du masyvus  $d[1..n]$  ir  $prec[1..n]$ , kurių prasmė yra ta pati, kaip ir Deikstros algoritme (žr. 2.14.1 paragrafą), išskyrus tai, kad  $d[k]$  reiškia ilgiausio kelio nuo minorantos iki viršūnės  $k$  ilgi. Tarkime, tinklinis grafas

sunumeruotas teisingai (žr. 2.14.3 paragrafą). Tada kritinio kelio radimo algoritmas gali būti užrašytas taip.

*begin*

**Paruošiamasis žingsnis**

*for i:=1 to n do*

*begin*

*d[i]:=0;*

*prec[i]:=0*

*end;*

*prec[1]:=1;*

**Pagrindiniai skaičiavimai**

*for k:=1 to n-1 do*

*for u ∈ N(k) do*

*if d[u] < d[k] + c(k,u) then*

*begin*

*d[u]:= d[k]+c(k,u);*

*prec[u]:= k;*

*end;*

*end;*

Pavyzdžiui, 2.14.4 pav. Tinklinio grafo kritinio kelio skaičiavimas iliustruojamas žemiau pateiktais masyvais d ir prec, t.y. parodant šiu masyvų turinių kitimą.

<i>d:</i>	1	2	3	4	5	6	7	<i>prec:</i>	1	2	3	4	5	6	7
	0	0	0	0	0	0	0		0	0	0	0	0	0	0
1		3	11	8	4	43			1	4	3	2	4	4	
	6			15	8	18				2	4	3	5		
					17								4		

Šio tinklinio grafo kritinio kelio ilgis yra 18, ir šis grafas turi du kritinius kelius, nes, nagrinėjant 6-tajai viršunei gretimas viršunes, gauname, kad  $d[6]+c(6,7)=18$ . Tačiau, kelią, kurio ilgis lygus 18, iki 7-osios viršūnės jau buvome atradę iš 5-osios viršūnės. Tuo būdu masyvas prec žymi tik vieną kritinį kelią: 7,5,4,3,2,1.

Darbai, nepriklasantys kritiniams keliams, turi laiko rezervą, t.y. jei šio darbo atlikimo vėlinimas neviršija laiko rezervo, tai bendras projekto vykdymo laikas nepailgėja. Tuo tarpu darbų, priklausančių kritiniams keliams, laiko rezervai lygūs nuliui.

**Pastaba.** Darbų (operacijų) laiko rezervų apskaičiavimo algoritmai išnagrinėti literatūroje [Ma81].

Kadangi tinklinio grafo uždaviniai sprendžiami paprasčiau, kai grafas sunumeruotas teisingai, tai dabar aptarsime rangų apskaičiavimo ir teisingo numeravimo algoritmus.

### **Tinklinio grafo viršūnių rangų apskaičiavimas**

Metodo idėja yra labai paprasta. Minoranta yra nulinio rango viršūnė.  $k$ -ojo rango viršūnės yra viršūnes, kurioms jėjimo puslaipsnis tampa lygus nuliui, kai iš grafo pašalinamos viršūnės, kurių rangas mažesnis už  $k$ .

Ši metodą patogu realizuoti naudojant paiešką į plotį: į eilę talpiname tas viršūnei  $u$  (viršūnė  $u$  – pirmoji eilės viršūnė) gretimas viršūnes, kurioms perskaičiuotas jėjimo puslaipsnis lygus nuliui.

Tarkime, kad tinklinis grafas nusakytas briaunų matrica  $B[1..2,1..m]$ , čia  $(b[1,j],b[2,j])$  –  $j$ -tasis lankas ir  $b[1,j]$  – lanko pradžia, o  $b[2,j]$  – pabaiga. Tada viršūnių rangų apskaičiavimo procedūra gali būti užrašyta taip.

*const c = 200;*

*type mas= array[1..c] of integer;*

*matr= array[1..2,1..c] of integer;*

*procedure rangas ( n, m ,mn: integer; b:matr; var r: mas; var L1,lst1:mas);*

*{procedūra rangas apskaičiuoja tinklinio grafo viršūnių rangus}*

*{Formalūs parametrai*

*n – tinklinio grafo viršūnių skaičius,*

*m – tinklinio grafo lankų skaičius,*

*mn – minorantos numeris,*

*b – lankų matrica, b[1,j] – lanko pradžia, b[2,j] – pabaiga,*

*r[1..n] – rangų masyvas, jei r[u]=k, tai reiškia, kad viršūnės u rangas*

*yrat k;*

*masyvų L1 ir lst1 prasmės paaiškintos žemiau.}*

*{Pradžioje apskaičiuosime masyvus L1[1..m], lst1[1..n+1] ir L2[1..m] ir  
lst2[1..n+1].*

*Masyvo L1[k], k =  $\overline{lst1[u]+1, lst1[u+1]}$ , elementai, žymi viršūnes, iš  
kurias eina lankai iš viršūnės u.*

*Masyvo L2[k], k =  $\overline{lst2[u]+1, lst2[u+1]}$ , elementai žymi viršūnes, iš  
kurių ateina lankai iš viršūnės u. }*

*var i, j ,k ,u ,v, p, f, n1, s, x: integer;*

*S1, S2, L2, lst2, fst1, fst2, eilė, d :mas;*

*begin*

*{Viršūnių laipsnių apskaičiavimas}*

*for i:=1 to n do*

*begin*

*S1[i]:=0; S2[i]:=0;*

*end;*

*for j:=1 to m do*

*begin*

*k:=b[1,j];*

```

S1[k]:=S1[k]+1;
k:=b[2,j];
S2[k]:=S2[k]+1;
end;
{Masyvų lst1 ir lst2 formavimas}
lst1[1]:=0; lst2[1]:=0;
for i:=1 to n do
begin
    lst1[i+1]:= lst1[i]+ S1[i];
    lst2[i+1]:= lst2[i]+ S2[i];
end;
{Masyvų L1 ir L2 formavimas }
for i:=1 to n do
begin
    fst1[i]:= lst1[i]+ 1;
    fst2[i]:= lst2[i]+ 1;
end;
for j:=1 to m do
begin
    k:=b[1,j];
    L1[fst1[k]]:=b[2,j];
    fst1[k]:=fst1[k]+1;
    k:=b[2,j];
    L2[fst2[k]]:=b[1,j];
    fst2[k]:=fst2[k]+1
end;
{Rangų apskaičiavimas}
for i:=1 to n do d[i]:=1; {d[i]=1, jei i- oji viršūnė – nenudažyta;
d[i]=0, jei i-oji viršūnė – nudažyta}
n1:=n+1;
p:=1; f:=1; {eilė:=∅}
r[mn]:=0; d[mn]:=0;
{eilė ⊜ mn}
if p=n1 then p:=1
else p:=p+1;
if p=f then
begin
    writeln ('eilės persipildymas');
    exit;
end
else eilė[p]:=mn;

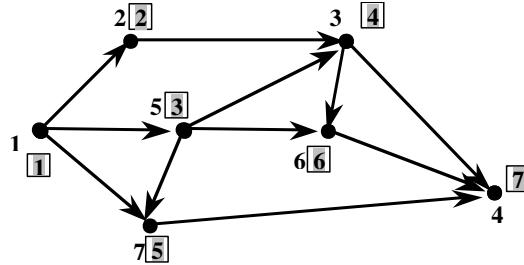
```

```

while f<>p { eilė ≠ ∅ } do
begin
{u ⊂ eilė}
if p=f then
begin
writeln ('eilė tuščia');
exit;
end
else
begin
if f = n1 then f:=1
else f:=f+1;
u:=eilė[f];
end;
for i:=lst1[u]+1 to lst1[u+1] do
begin
v:=L1[i];
{Ar viršūnės v jéjimo puslaipsnis lygus nuliui}
s:=0;
for j:= lst2[v]+1 to lst2[v+1] do
begin
x:= L2[j];
s:=s+d[x];
end;
if s=0 {Viršūnės v puslaipsnis lygus nuliui}
then
begin
r[v]:= r[u]+1; d[v]:=0;
end;
{eilė ⊂ v }
if p = n1 then p:=1
else p:=p+1;
if p = f then
begin
writeln ('eilės persipildymas'); exit;
end
else eilė[p]:=v;
end;{for i}
end; {while}
end; {rangas}

```

**Pavyzdys.** Panagrinėkime 2.14.6 pav. pavaizduotą tinklinį grafą, kurį nusako briaunų matrica  $B = \begin{pmatrix} 1 & 1 & 1 & 2 & 3 & 3 & 5 & 5 & 5 & 6 & 7 \\ 2 & 5 & 7 & 3 & 6 & 4 & 3 & 7 & 6 & 4 & 4 \end{pmatrix}$ .



2.14.6 pav. Tinklinio grafo viršūnių rangai

Pirmiausia bus apskaičiuoti masyvai:

$$L1: \frac{1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11}{2, \quad 5, \quad 7, \quad 3, \quad 6, \quad 4, \quad 3, \quad 7, \quad 6, \quad 4, \quad 4},$$

$$lst1: \frac{1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8}{0, \quad 3, \quad 4, \quad 6, \quad 6, \quad 9, \quad 10, \quad 11},$$

$$L2: \frac{1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11}{1, \quad 2, \quad 5, \quad 3, \quad 6, \quad 7, \quad 1, \quad 3, \quad 5, \quad 1, \quad 5},$$

$$lst2: \frac{1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8}{0, \quad 0, \quad 1, \quad 3, \quad 6, \quad 7, \quad 9, \quad 11}.$$

Į eilę viršūnės bus talpinamos tokia tvarka: 1, 2, 5, 3, 7, 6, 4 , o rangų masyvas bus tokš:

$$r: \frac{1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7}{0, \quad 1, \quad 2, \quad 4, \quad 1, \quad 3, \quad 2}.$$

#### Tinklinio grafo teisingo pernumeravimo algoritmas

Tarkime, kad tinklinis grafas nusakytas masyvais  $L1[1..m]$ ,  $lst1[1..n+1]$  ir žinomas viršūnių rangų masyvas  $r[1..n]$ , t.y. teisingo pernumeravimo algoritmo įėjimo parametrai sutampa su procedūros “**rangas**” išėjimo parametrais.

Algoritmo išėjimo parametrai bus masyvai  $L[1..m]$ ,  $lst[1..n+1]$ , čia  $L[i]$ ,  $i = lst[u] + 1, lst[u + 1]$ , teisingai sunumeruoto tinklinio grafo viršūnės, į kurias lankai eina iš viršūnės  $u$ .

Ivesime masyvus:  $nr[1..n]$ , čia  $nr[u]$  – naujas viršūnės  $u$  numeris ir  $nrs[1..n]$ , čia  $nrs[u]$  yra viršūnės  $u$  senas numeris.

```

procedure numeravimas (n, m : integer; L1, lst1: mas; r :mas;
                      var nr:mas; var L, lst:mas);
var i, k, t : integer;
    nrs : mas;
begin
{Masyvo nr[1..n] formavimas}
k:=0; {rango numeris};
t:=0; {viršūnės eilės numeris};
while t < n {kol nesunumeruotos visos viršūnės} do
begin
    for i:=1 to n do
        if r[i] = k then
            begin
                t:= t+1;
                nr[i]:= t;
                nrs[t]:=i;
            end;
        k:=k+1;
    end; {while}
{Masyvų L ir lst formavimas}
lst[1]:=0;
for i:=1 to n do
begin
    {Naujam numerui i randame seną numerį k}
    k:= nrs[i];
    {Apskaičiuojame masyvų L ir lst elementus}
    lst[i+1]:= lst[i]+lst1[k+1]-lst1[k];
    for t := lst1[k]+1 to lst1[k+1] do
        L[lst[i]+t-lst1[k]]:= nr[L1[t]];
    end;
end; {numeravimas}

```

**Pavyzdys.** Panagrinėkime 2.14.6 pav. Pavaizduoto tinklinio grafo viršūnių pernumeravimą. Po procedūros “**rangas**” gausime masyvus:

$$\begin{array}{c}
\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\
\hline
r: & 0, & 1, & 2, & 4, & 1, & 3, & 2 & , \\
\\
\begin{array}{ccccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{array} \\
\hline
L1: & 2, & 5, & 7, & 3, & 6, & 4, & 3, & 7, & 6, & 4, & 4 & , \\
\end{array}$$

$lst1:$   $\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 0, & 3, & 4, & 6, & 6, & 9, & 10, & 11 \end{array},$

Šie masyvai yra procedūros “**numeravimas**” jėjimo parametrai.

Pirmiausia apskaičiuosime masyvus  $nr$  ir  $nr_s$ .

$nr:$   $\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 1 & 2 & 4 & 7 & 3 & 6 & 5 \end{array},$      $nr_s:$   $\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 1 & 2 & 5 & 3 & 7 & 6 & 4 \end{array}.$

Masyvo  $nr$  elementai riša senuosius numerius su naujaisiais; pavyzdžiui,  $nr[5]=3$  reiškia, kad 5-osios viršūnės naujasis numeris yra 3. Masyvo  $nr_s$  elementai riša naujuosius numerius su senaisiais; pavyzdžiui,  $nr_s[4]=3$  reiškia, kad naujojo numero 4 senasis numeris yra 3. Po masyvų  $nr$  ir  $nr_s$  apskaičiavimo, bus apskaičiuoti masyvai  $L$  ir  $lst$ , kurie ir nusakys teisingai sunumeruotą tinklinį grafą (2.14.6 pav. teisingi numeriai parašyti kvadrateliuose):

$lst:$   $\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 0, & 3, & 4, & 7, & 9, & 10, & 11, & 11 \end{array}$       ir  
 $L:$   $\begin{array}{cccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ \hline 2, & 3, & 5, & 4, & 4, & 5, & 6, & 6, & 7, & 7, & 7 \end{array}.$

#### 2.14.5. Trumpiausių kelių tarp visų viršūnių porų apskaičiavimas

**Uždavinio formulavimas.** Duotas svorinis  $(n,m)$ -grafas  $G=(V,U,C)$ , čia  $C$  – briaunų svorių aibė. Rasti trumpiausius kelius tarp visų viršūnių porų.

Šio uždavinio sprendimas yra n-osios eilės kvadratinė matrica  $D=[d_{ij}]$   $i=\overline{1,n}$ ,  $j=\overline{1,n}$ , kurios elementas  $d_{ij}$  reiškia ilgį trumpiausio kelio tarp viršūnių  $i$  ir  $j$  poros. Jei be kelio ilgio mus domina ir per kurias viršūnes šis kelias eina, tai šalia matricos  $D$  reikia apskaičiuoti  $n$ -osios eilės matricą  $P$ , kurios elementai  $p_{ij}$  rodo, kuria kryptimi (i kuria viršūnę arba iš kurios viršūnės) iš viršūnės einama.

##### Deikstros metodas

Aišku, kad ši uždavinį galima spręsti Deikstros algoritmo pagalba. 2.14.1 paragrafe išnagrinėtas Deikstros algoritmas leidžia apskaičiuoti trumpiausius kelius nuo viršūnės  $s$  iki visų likusių viršūnių, t.y. Deikstros algoritmo pagalba randami masyvai  $d[1..n]$  ir  $prec[1..n]$ . Masyvo  $d$  elementas  $d_k$  reiškia ilgį trumpiausio kelio nuo viršūnės  $s$  iki viršūnės  $k$ , o masyvo  $prec$  elementai žymi trumpiausių kelių nuo viršūnės  $s$  iki likusių grafo viršūnių medži (žr. 2.14.1 paragrafą).

Aišku, kad masyvas  $d$  yra matricos  $D$   $s$ -toji eilutė, o masyvas  $prec$  – matricos  $P$   $s$ -toji eilutė. Vadinasi norint apskaičiuoti  $D$  ir  $P$  matricas, i Deikstros algoritmą reikės kreiptis, kai  $s$  kinta nuo 1 iki  $n$ , ir po kiekvieno

kreipinio masyvų  $d$  ir  $prec$  elementus reikės patalpinti į atitinkamas matricę  $D$  ir  $P$  eilutes. Aišku, kad  $d_{ij}$  reikš ilgį trumpiausio kelio, tarp  $i$  ir  $j$  viršūnių, o per kurias viršūnes šis kelias eina sužinosime taip: tarkime, kad  $p_{ij} = t_1, p_{it_1} = t_2, \dots, p_{it_{k-1}} = t_k, p_{it_k} = i$ ; tada trumpiausias kelias eis per viršūnes:  $i, t_k, t_{k-1}, \dots, t_2, t_1, j$ .

### Floido metodas

Floido metodo idėja yra labai paprasta. Tarkime, kad svorinio  $(n,m)$ -grafo  $G=(V,U,C)$  viršūnės sunumeruotos iš eilės einančiais natūraliaisiais skaičiais nuo 1 iki  $n$ .

**Pastaba.** Kalbėdami apie žymėtuosius grafus įsivaizdavome, kad grafo viršūnės būtent taip ir sunumeruotos.

Tada matrica  $D$  gaunama nuosekliai apskaičiuojant matricas  $D^0, D^1, \dots, D^m, \dots, D^n$ . Matricos  $D^m = [d_{ij}^m] \quad i = \overline{1, n}, \quad j = \overline{1, n}$  elementas reiškia ilgį trumpiausio kelio tarp  $i$  ir  $j$  viršūnių, kai tarpinėmis šio kelio viršūnėmis gali būti tik viršūnės su numeriais nuo 1 iki  $m$ . Priminsime, kad tarpine kelio viršūne vadinsime bet kurią jo viršūnę, nesutampančią su pradine ir galine šio kelio viršūne. Kalbamuoju atveju tai viršūnės, nesutampančios su viršūnėmis  $i$  ir  $j$ .

Jei tarp viršūnių  $i$  ir  $j$  nurodyto tipo kelio nėra, tai  $d_{ij}^m = \infty$ .

Pirmiausia apibrėžime matricą  $D^0$ . Šios matricos elementas  $d_{ij}^0$ , tai ilgis trumpiausio kelio tarp viršūnių  $i$  ir  $j$ , kuris neturi tarpinių viršūnių, t.y.  $d_{ij}^0 = c(i, j)$ , čia  $c(i, j)$  – briaunos (lanko)  $(i, j)$  ilgis. Aišku, kad  $d_{ii}^0 = 0, \quad i = \overline{1, n}$ , o  $d_{ij}^0 = \infty, \quad jei (i, j) \notin U$ , (briaunos (lanko)  $(i, j)$  nėra).

$$\text{Vadinasi } d_{ij}^0 = \begin{cases} c(i, j), & \text{jei } (i, j) \in U, \\ \infty, & \text{jei } (i, j) \notin U, \\ 0, & \text{jei } i = j, \end{cases} \quad \text{visiems } i = \overline{1, n}, \quad j = \overline{1, n}.$$

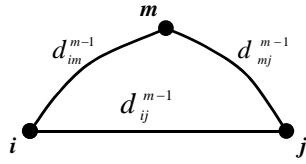
Kaip iš matricos  $D^{m-1}$  apskaičiuoti matricą  $D^m, m=1, 2, \dots, n$ ?

Panagrinėkime 2.14.7 pav. Šiame paveiksle

$d_{ij}^{m-1}$  žymi ilgį trumpiausio kelio tarp viršūnių  $i$  ir  $j$ , kai tarpinėmis šio kelio viršūnėmis yra viršūnės, kurių numeriai yra tarp 1 ir  $m-1$ ;

$d_{im}^{m-1}$  žymi ilgį trumpiausio kelio tarp viršūnių  $i$  ir  $m$ , kai tarpinėmis kelio viršūnėmis yra viršūnės su numeriais tarp 1 ir  $m-1$ ;

$d_{mj}^{m-1}$  žymi ilgį trumpiausio kelio tarp viršūnių  $m$  ir  $j$ , kai tarpinėmis kelio viršūnėmis yra viršūnės su numeriais tarp 1 ir  $m-1$ .



2.14.7 pav. Floido metodas

Kadangi  $d_{ij}^m$  žymi ilgi trumpiausio kelio nuo  $i$  iki  $j$ , kai to kelio tarpinėmis viršūnėmis yra viršūnės su numeriais nuo 1 iki  $m$ , tai aišku, kad šio kelio ilgis yra  $\min(d_{ij}^{m-1}, d_{im}^{m-1} + d_{mj}^{m-1})$ .

Vadinasi,

$$d_{ij}^m = \begin{cases} d_{ij}^{m-1}, & \text{jei } d_{ij}^{m-1} \leq d_{im}^{m-1} + d_{mj}^{m-1}, \\ d_{im}^{m-1} + d_{mj}^{m-1} & \text{priešingu atveju,} \end{cases} \text{ visiems } i = \overline{1, n}, \quad j = \overline{1, n}.$$

Kaip minėjome anksčiau, paprastai mus domina ne tik trumpiausio kelio tarp viršūnių  $i$  ir  $j$  ilgis, bet ir per kokias viršunes šis kelias eina. Tam tikslui apibrėžime matricas  $P^0, P^1, \dots, P^m, \dots, P^n$ . Matricos  $P^m$  elementas  $p_{ij}^m$  reiškia numerį viršūnės, į kurią tiesiogiai trumpiausias kelias veda iš viršūnės  $i$  į viršūnę  $j$ .

Aišku, kad

$$p_{ij}^0 = \begin{cases} j, & \text{jei } (i, j) \in U, \\ 0, & \text{jei } (i, j) \notin U, \end{cases} \text{ visiems } i = \overline{1, n}, \quad j = \overline{1, n}.$$

Kaip iš matricos  $P^{m-1}$  apskaičiuoti matricą  $P^m$ ? Iš 2.14.7 pav. nesunku suvokti, kad

$$p_{ij}^m = \begin{cases} p_{ij}^{m-1}, & \text{jei } d_{ij}^{m-1} \leq d_{im}^{m-1} + d_{mj}^{m-1}, \\ p_{im}^{m-1} & \text{priešingu atveju,} \end{cases} \text{ visiems } i = \overline{1, n}, \quad j = \overline{1, n}.$$

Kaip minėjome anksčiau, matricos  $D^n$  ir  $P^n$  atitinkamai žymi ilgius trumpiausių kelių tarp visų viršūnių porų ir trumpiausių kelių medžius.

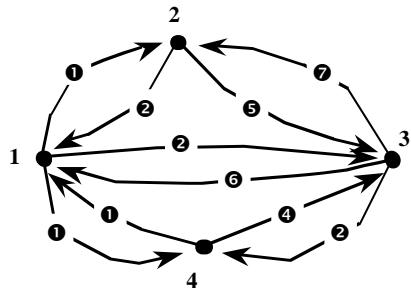
Kaip sužinoti, per kurias viršunes eina trumpiausias kelias, jungiantis  $i$  ir  $j$  viršunes?

Tarkime, kad  $p_{ij} = t_1, p_{t_1j} = t_2, \dots, p_{t_{k-1}j} = t_k$ , ir  $p_{tkj} = j$ ; tada trumpiausias kelias eina per viršunes:  $i, t_1, t_2, \dots, t_k, j$ .

**Pavyzdys.** Panagrinėkime orientuotą grafi, pavaizduotą 2.14.8 pav.

Floido metodu raskime atstumus tarp visų viršūnių porų.

Žemiau pateiktos matricos  $D^m$  ir  $P^m$ ,  $m = \overline{1, 4}$ , kurios apskaičiuotos pagal anksčiau pateiktas formules.



2.14.8 pav. Floido metodo iliustracija

$$\begin{array}{c}
 D^0 = \begin{array}{c|cccc}
 & 1 & 2 & 3 & 4 \\
 \hline
 1 & 0 & 1 & 2 & 1 \\
 2 & 2 & 0 & 7 & - \\
 3 & 6 & 5 & 0 & 2 \\
 4 & 1 & - & 4 & 4
 \end{array}, \quad P^0 = \begin{array}{c|cccc}
 & 1 & 2 & 3 & 4 \\
 \hline
 1 & 0 & 2 & 3 & 4 \\
 2 & 1 & 0 & 3 & 0 \\
 3 & 1 & 2 & 0 & 4 \\
 4 & 1 & 0 & 3 & 0
 \end{array} .
 \end{array}$$
  

$$D^1 = \begin{array}{c|cccc}
 & 1 & 2 & 3 & 4 \\
 \hline
 1 & 0 & 1 & 2 & 1 \\
 2 & 2 & 0 & 4 & 3 \\
 3 & 6 & 5 & 0 & 2 \\
 4 & 1 & 2 & 3 & 0
 \end{array}, \quad P^1 = \begin{array}{c|cccc}
 & 1 & 2 & 3 & 4 \\
 \hline
 1 & 0 & 2 & 3 & 4 \\
 2 & 1 & 0 & 1 & 1 \\
 3 & 1 & 2 & 0 & 4 \\
 4 & 1 & 1 & 1 & 0
 \end{array} .$$
  

$$D^2 = \begin{array}{c|cccc}
 & 1 & 2 & 3 & 4 \\
 \hline
 1 & 0 & 1 & 2 & 1 \\
 2 & 2 & 0 & 4 & 3 \\
 3 & 6 & 5 & 0 & 2 \\
 4 & 1 & 2 & 3 & 0
 \end{array}, \quad P^2 = \begin{array}{c|cccc}
 & 1 & 2 & 3 & 4 \\
 \hline
 1 & 0 & 2 & 3 & 4 \\
 2 & 1 & 0 & 1 & 1 \\
 3 & 1 & 2 & 0 & 4 \\
 4 & 1 & 1 & 1 & 0
 \end{array} .$$
  

$$D^3 = \begin{array}{c|cccc}
 & 1 & 2 & 3 & 4 \\
 \hline
 1 & 0 & 1 & 2 & 1 \\
 2 & 2 & 0 & 4 & 3 \\
 3 & 6 & 5 & 0 & 2 \\
 4 & 1 & 2 & 3 & 0
 \end{array}, \quad P^3 = \begin{array}{c|cccc}
 & 1 & 2 & 3 & 4 \\
 \hline
 1 & 0 & 2 & 3 & 4 \\
 2 & 1 & 0 & 1 & 1 \\
 3 & 1 & 2 & 0 & 4 \\
 4 & 1 & 1 & 1 & 0
 \end{array} .$$

$$D^4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 2 & 1 \\ 2 & 2 & 0 & 4 & 3 \\ 3 & 3 & 4 & 0 & 2 \\ 4 & 1 & 2 & 3 & 0 \end{array}, \quad P^4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 2 & 3 & 4 \\ 2 & 1 & 0 & 1 & 1 \\ 3 & 4 & 4 & 0 & 4 \\ 4 & 1 & 1 & 1 & 0 \end{array}.$$

Pavyzdžiu,  $d_{31}=3$  ir trumpiausias kelias iš 3-osios į 1-ają viršūnę eina: 3, 4, 1.  $d_{24}=3$ , o kelias eina: 2, 1, 4.

## 2.15. Maršrutai

Šiame paragafe nagrinėsime Oilerio ir Hamiltono maršrutus: grandines ir ciklus. Šie klausimai iškyla ne tiktais sprendžiant įvairius galvosūkius, bet ir praktiniuose uždaviniuose.

Pavyzdžiu, buitinio aptarnavimo mašina turi aplankytį kokio tai rajono visas gatves. Koks turi būti jos maršrutas? Tai Oilerio maršruto uždavinys.

Su Hamiltono maršratais susijęs garsus komivojažerio arba keliaujančio pirklio uždavinys, kuris formuluoamas taip. Turime  $n$  miestų, o matrica  $C=[c_{ij}]$   $i=\overline{1,n}$ ,  $j=\overline{1,n}$  yra atstumų matrica:  $c_{ij}$  – atstumas tarp miestų  $i$  ir  $j$ . Pirklys, išėjęs iš 1-ojo miesto turi apeiti visus miestus po vieną kartą ir grįžti į 1-ajį miestą. Koks turi būti pirklio maršrutas, kad jo ilgis būtų trumpiausias?

### 2.15.1. Oilerio maršrutai

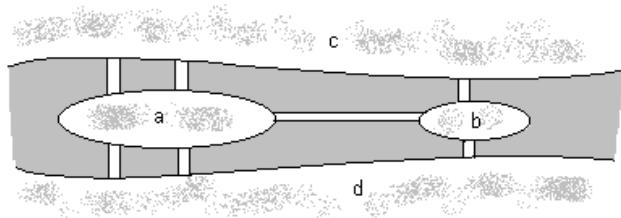
Visa, kas bus kalbama šiame paragafe, tinkta ir multigrafams.

**Apibrėžimas.** Maršrutas (kelias), apeinantis visas grafo briaunas po vieną kartą, vadinamas **Oilerio maršruto**. Jei pradinė ir galinė maršruto viršūnės nesutampa, tai tokis maršrutas vadinamas **Oilerio grandine**, priešingu atveju – **Oilerio ciklu**.

Grafas, turintis Oilerio maršrutą, vadinamas Oilerio grafu.

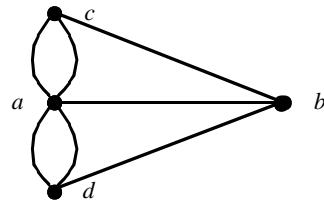
Reikia pastebėti, kad Oilerio maršruto radimo uždavinys yra pirmasis grafų teorijos uždavinys. Tai uždavinys apie Karaliaučiaus tiltus, kurį 1736 m. sėkmungai išsprendė Oileris.

Per Karaliaučių teka upė Prēglis, kurioje yra dvi salos. Šios salos su krantais ir tarpusavyje yra sujungtos tiltais (žr. 2.15.1 pav.).



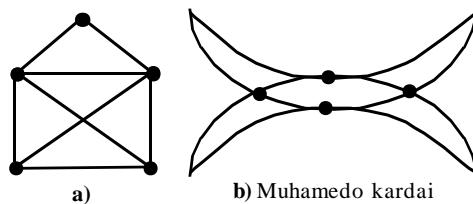
2.15.1 pav. Karaliaučiaus tiltai

Ar galima, išėjus iš namų, apeiti visus tiltus po vieną kartą ir vėl grįžti į namus. Kitaip tariant, ar multigrafas, pavaizduotas 2.15.2 pav. turi Oilerio maršrutą?



2.15.2 pav. Karaliaučiaus tiltų grafas

Su Oilerio maršruti yra susiję įvairūs galvosūkiai. Pavyzdžiui, ar galima, neatitraukus rankos nuo popieriaus, nupiešti figūras pavaizduotas 2.15.3 a) ir b) pav.?



2.15.3 pav. Oilerio maršrutai

Kaip minėjome, ar grafas turi Oilerio maršrutą, išsprendė Oileris įrodydamas teoremą.

**Teorema** (Oilerio teorema, 1736). Būtina ir pakankama sąlyga, kad grafas  $G$  turėtų Oilerio maršrutą yra:

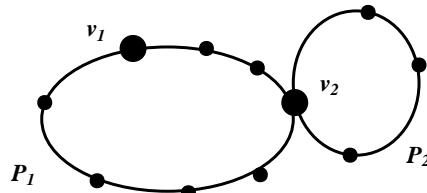
- 1)  $G$  turi būti jungusis,
- 2) visų jo viršunių laipsniai turi būti arba lyginiai, arba  $G$  turi turėti tik dvi nelyginio laipsnio viršunes.

Pirmuoju atveju grafas turi Oilerio ciklą, o antruoju – grandinę, prasidedančią ir besibaigiančią nelyginio laipsnio viršunėse.

Pateiksime šios teoremos įrodymą, kadangi jis yra konstruktyvus: iš įrodymo išplaukia Oilerio maršruto konstravimo algoritmas.

**Būtinumas.** Tarkime  $G$  – Oilerio grafas. Oilerio ciklas eidamas per kiekvieną viršūnę, patenka į viršūnę viena briauna, o išeina – kita briauna. Tai reiškia, kad kiekviena grafo  $G$  viršūnė incidentiška lyginiam Oilerio ciklo briaunu skaičiui. Kadangi Oilerio ciklas apima visas grafo briaunas, tai reiškia, kad grafas yra jungusis ir, kad kiekvienos grafo viršūnės laipsnis yra lyginis.

**Pakankamumas.** Tarkime, kad visų grafo  $G$  viršunių laipsniai yra lyginiai. Pradėkime kelionę iš viršūnės  $v_1$  ir, eidami grafo briaunomis, laikysimės taisyklos: “niekada neiti ta pačia briauna”. Tai tolygu taisyklei: “pereitą briauną – ištriname”. Atėję į bet kurį viršūnę  $v \neq v_1$ , iš jos visada galėsime išvykti, nes viršūnės  $v$  laipsnis yra lyginis. Jei patekome į viršūnę, iš kurios negalime išvykti, tai reiškia, kad esame pradinėje kelio viršūnėje  $v_1$ . Vadinas mūsų kelias sudarė ciklą  $P_1$ . Pašalinę šį ciklą iš grafo  $G$ , gausime grafą  $G'$ , kurio kiekvienos viršūnės laipsnis yra arba lyginis, arba lygus nuliui. Jei visų viršunių laipsniai yra lygūs nuliui, tai reiškia, kad  $P_1$  yra Oilerio ciklas. Priešingu atveju nagrinėsime grafą  $G'$ . Kadangi grafas  $G$  yra jungusis grafas, tai grafaip  $P_1$  ir  $G'$  turi bent vieną bendrą viršūnę  $v_2$ . Pradėję kelionę grafe  $G'$  iš viršūnės  $v_2$  ir laikydamiesi tos pačios taisyklos, sukonstruosime ciklą  $P_2$ . Iš ciklų  $P_1$  ir  $P_2$  sukonstruosime bendrą ciklą taip: iš viršūnės  $v_1$  ciklo  $P_1$  briaunomis nukeliausime į viršūnę  $v_2$ , po to apeisime ciklą  $P_2$  ir po to iš viršūnės  $v_2$  ciklo  $P_1$  briaunomis ateisime į viršūnę  $v_1$  (žr. 2.15.4 pav.).



2.15.4 pav. Oilerio ciklo konstravimas

Pašalinkime šį ciklą iš grafo  $G$ . Jei po ciklo pašalinimo grafas yra tuščiasis, tai šis ciklas yra Oilerio ciklas. Priešingu atveju, šiame grafe atliksime analogiškus veiksmus. Ir taip elgsimės iki sukonstruosime ciklą, einantį per visas grafo  $G$  briaunas.

**Teorema** (R.Reidas, 1962). Beveik nėra Oilerio grafų. [EM90].

Primename, kad savoka “beveik nėra grafų” apibrėžiama taip. Simboliu  $GP(n)$  pažymekime skaičių  $n$ -viršūnių grafų, turinčių Oilerio maršrutą, t.y. Oilerio grafų skaičių. Simboliu  $G(n)$  pažymekime visų  $n$ -viršūnių grafų skaičių. Tada  $\lim_{n \rightarrow \infty} \frac{GP(n)}{G(n)} = 0$ , ir sakome, kad “beveik nėra Oilerio grafų”.

### Oilerio ciklo konstravimo algoritmas

Duota: Jungusis grafas  $G$ , kurio visų viršūnių laipsniai yra arba lyginiai, arba  $G$  turi dvi nelyginio laipsnio viršunes. Tarkime, kad grafas  $G$  nusakytas gretumumo struktūra:  $N(v)$  – tai aibė viršūnių gretimų viršūnei  $v$ .

Rasti: Sukonstruoti Oilerio maršrutą.

Įrodyme pateiktą Oilerio maršruto konstravimą patogu realizuoti naudojant du stekus: *STEK* ir *Oiler*.

Priminsime, kad stekas, tai tiesinis sąrašas, kuriame naujo elemento patalpinimas ir pašalinimas atliekamas viename sąrašo gale. Kitaip dar stekas vadinamas sąrašu LIFO (nuo angliskų žodžių “last in first out” (paskutinis į steką – pirmas iš jo)).

Steką patogu organizuoti vienmačiu masyvu *STEK* [1..n]. Ji charakterizuoja vienas parametras – rodyklė “*top*”, kuri reiškia paskutinio elemento sąraše adresą.

**Stekas tuščias** {žymime *STEK*:=  $\emptyset$ }

*top*:=0;

**Elemento patalpinimas į steką** {žymime *STEK* $\leftarrow v$ }.

*top*:= *top*+1;

*if top > n then* “steko persipildymas”

*else STEK*[*top*]:= *v*;

**Elemento pašalinimas iš steko** {žymime *v* $\leftarrow STEK$ }.

*if top = 0 then* “stekas tuščias”

*else begin*

*v*:= *STEK*[*top*];

*top*:=*top*-1

*end*;

**Elemento nuskaitymas iš steko** {žymime *v*:= *top(STEK)*}.

*if top=0 then* “stekas tuščias”

*else v*:=*STEK*[*top*];

**Algoritmo veikimo principas.** Tarkime  $v_1$  yra viršūnė, iš kurios pradėsime brėžti Oilerio ciklą. Šią viršūnę patalpinkime į steką *STEK*. Iš viršūnės  $v_1$  keliausime per grafą, talpindami kelio viršunes į steką *STEK* ir ištrindami kelio briaunas. Jei atėjome į viršūnę, iš kurios negalime išeiti, tai

reiškia, kad esame pradinėje kelio viršūnėje (jei grafo  $G$  visų viršūnių laipsniai buvo lyginiai) arba kitoje nelyginio laipsnio viršūnėje (jei  $v_1$  laipsnis buvo nelyginis). Šią viršūnę patalpinsime į steką *Oiler* ir bandysime keliauti iš priešpaskutinės kelio viršūnės. Kelionė galima, nes iš grafo  $G$  pašalinome ciklą, einantį per viršūnę  $v_1$ , ir likusių viršūnių laipsniai yra arba lyginiai arba lygūs nuliui.

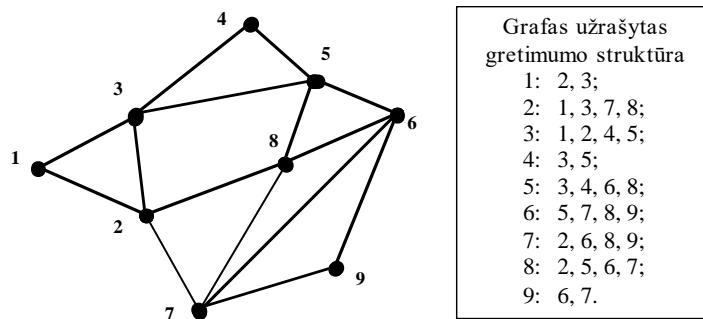
Taip elgsimės iki stekas *STEK* taps tuščias. Tada viršūnės patalpintos steke *Oiler* ir nusakys Oilerio maršrutą.

```

begin
    STEK:=∅ ; Oiler:=∅ ;
    v:= bet kuri grafo viršūnė, jei visų grafo viršūnių laipsniai yra
        lyginiai arba nelyginio laipsnio viršūnė, jei grafas turi dvi
        nelyginio laipsnio viršūnes.
    STEK≤=v;
    while STEK ≠ ∅ do
        begin
            v:= top(STEK); {v = viršutinis steko elementas; rodyklė top
                neperskaičiuojama}
            if N(v) ≠ ∅
                then
                    begin
                        u:= pirmoji aibės N(v) viršūnė;
                        STEK ≤= u;
                        {pašalinti (v,u) briauną}
                        N(v):= N(v) \ {u} ;
                        N(u):= N(u) \ {v} ;
                    end
            else { N(v) = ∅ }
                begin
                    v≤= STEK ;
                    Oiler ≤= v;
                end;
            end;
        end;
    end;

```

**Pavyzdys.** Remdamies algoritmu, sukonstruokime Oilerio ciklą grafui, pavaizduotam 2.15.5 pav.



2.15.5 pav. Oilerio ciklo pavyzdys

Kadangi grafo visų viršūnių laipsniai yra lyginiai, tai kelionę (Oilerio ciklo konstravimą) pradėsime iš 1-osios viršūnės. Tada stekų: *STEK* ir *Oiler* turiniai bus tokie:

*STEK*: 1, 2, 3, 4, 4, 5, 3, 6, 7, 2, 8, 5, 6, 9, 7, 8

*Oiler*: 1, 3, 5, 8, 7, 9, 6, 8, 2, 7, 6, 5, 4, 3, 2, 1.

Nubraukti elementai steke STEK žymi momentus, kai keliones metu atėjome į viršunes, iš kurių nebuvo galima išeiti.

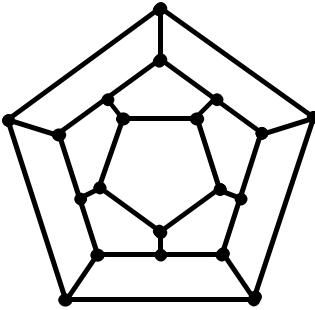
**Pastaba.** Praktikoje tenka spręsti tokį uždavinį. Duotas svorinis grafas  $G$ , kuris nėra Oilerio grafas. Kokias šio grafo briaunas pakartoti, kad jis taptų Oilerio grafu ir Oilerio ciklo ilgis būtų trumpiausias. Čia šio uždavinio nenagrinėsime, pažymédami, kad jo sprendimas pateiktas literatūroje [K78].

## 2.15.2. Hamiltono maršrutai

**Apibrėžimas.** Maršutas (kelias) apeinantis visas grafo viršunes po vieną kartą vadinamas Hamiltono maršrutu. Jei pradinė ir galinė maršuto viršūnės sutampa, tai šis maršutas vadinamas Hamiltono ciklu; priešingu atveju – Hamiltono grandine.

Grafas turintis Hamiltono maršrutą vadinamas Hamiltono grafu.

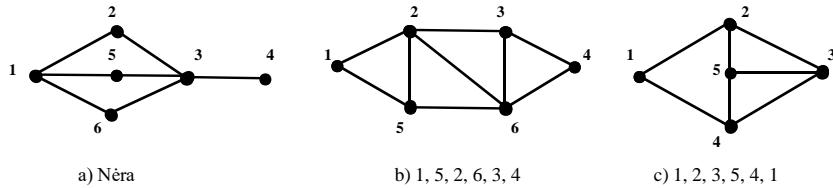
Hamiltono grafo sąvoka susijusi su žymaus airių matematiko W. Hamiltono vardu, kuris 1859 m. pasiūlė žaidimą, pavadintą “Kelionė aplink pasaulį”. Kiekvienai iš 20-ies dodekaedro viršūnių priskiriamas koks nors žymaus pasaulio miesto vardas. Reikia, einant iš vieno miesto į kitą, aplankytį visus miestus po vieną kartą ir grįžti į pradinį miestą. (žr. 2.15.6 pav.)



2.15.6 pav. Kelionė aplink pasaulį

Skirtingai nei Oilerio grafo atveju nėra žinoma nei vienos paprastos būtinės ir pakankamos sąlygos, kuri pasakytu ar grafas yra Hamiltono grafas. Negana to, nėra žinoma nei vieno algoritmo, nustatančio, ar grafas yra Hamiltono grafas ir kurio veiksmų skaičius būtų išreikiamais polinomu nuo viršunių skaičiaus  $n$ . Kitaip tariant, Hamiltono grafo nustatymo uždavinys priklauso  $NP$  uždavinių klasei [GD82].

2.15.7 pav. pateikti trys grafa, iš kurių grafas a) nėra Hamiltono grafas, grafas b) turi Hamiltono grandinę, o grafas c) – Hamiltono ciklą.



2.15.7 pav. Hamiltono maršrutai

Aišku, jei grafas yra pilnasis, tai jis yra Hamiltono grafas.

Pateiksime trejetą teoremą, nusakančią pakankamas sąlygas, kad grafas būtų Hamiltono grafas.

**Teorema** (V. Hvatalas, 1972) [EM90]. Grafas su viršunių laipsnių seka  $d_1 \leq d_2 \leq \dots \leq d_n$  yra Hamiltono grafas, jei bet kuriam sveikajam  $k$ , tenkinančiam sąlygą  $1 \leq k < \left\lfloor \frac{n}{2} \right\rfloor$  teisinga implikacija  $(d_k \leq k) \Rightarrow (d_{n-k} \geq n - k)$ .

**Teorema** (O. Ore, 1960) [EM90]. Jei  $n$  viršunių ( $n \geq 3$ ) grafo  $G$  bet kuriai negretimų viršunių porai  $u$  ir  $v$  teisinga nelygybė:  $d(u) + d(v) \geq n$ , tai  $G$  – Hamiltono grafas, čia  $d(u), d(v)$  – viršunių  $u$  ir  $v$  laipsniai.

**Teorema** (G. Dirakas, 1952) [EM90]. Jei  $n$  viršūnių grafo  $G$  kiekvienos viršūnės laipsnis nemažesnis nei  $\frac{n}{2}$ , tai  $G$  - Hamiltono grafas.

**Teorema** (Prepielica V.A., 1969) [EM90]. Beveik visi grafai yra Hamiltono grafai.

Aptarkime Hamiltono maršrutų radimo uždavinį:

- 1) nustatyti ar duotasis grafas  $G$  yra Hamiltono grafas ir
- 2) duotajam grafui  $G$  rasti visus Hamiltono maršrutus.

Aišku, kad 1)-asis uždavinys yra 2)-ojo uždavinio atskirasis atvejas, todėl čia nagrinėsime 2)-ąjį uždavinį.

Aišku, kad visus Hamiltono maršrutus rasime išnagrinėję visas maršrutų galimybes, kurį skaičius yra  $n!$ . Kitaip tariant, kiekvienas  $n$  – elementų kelinys gali būti Hamiltono maršrutas, t.y. kiekvienam keliiniui reikės patikrinti ar jis nusako Hamiltono maršrutą. Šio algoritmo sudėtingumas yra  $O(n \cdot n!)$ . Kadangi faktorialas auga greičiau nei eksponentinė funkcija, tai šis kelias yra labai neracionalalus.

Aptarkime rationalesnį Hamiltono maršrutų apskaičiavimo algoritmą, nors ir šio algoritmo veiksmų skaičius apribotas eksponente. Šis algoritmas pagrįstas “paieška gilyn su grįžimu” (angl. backtracking).

Šio metodo esmė yra labai paprasta: jei paieškos gilyn metu viršūnė  $u$  tampa išsemta, tai pamirštame, kad šioje viršūnėje buvome, t.y. viršūnė  $u$  tampa nauja. Šis metodas įgalina perrinkti visus galimus grafo maršrutus.

Žemiau pateikiame ši metodą realizuojantį algoritmą.

Tarkime, kad grafas  $G=(V,U)$  yra jungusis ir nusakytas masyvais  $L$  ir  $lst$  (žr. 2.7 paragrafą). Paiešką pradėsime iš 1-osios viršūnės. Paieškos gilyn metu aplankytas viršūnes talpinsime į masyvą  $X[1..n+1]$ . Kitaip tariant, masyvo  $X$  elementai apibrėž aplankytujų viršūnių maršrutą. Jei paieškos gilyn metu masyve  $X$  yra  $n$  elementų, ir iš viršūnės  $x_n$  atėjome į 1-ąją viršūnę, tai reiškia, kad radome Hamiltono ciklą. Jei viršūnei  $x_n$  gretimų viršūnių tarpe nėra 1-osios viršūnės, tai masyvo  $X$  elementai nusakys Hamiltono kelią. Žemiau pateiktoje procedūroje pastaroji salyga netikrinama.

```

const c = 500;
type mas = array[1..c] of integer;
        matr = array[1..2,1..c] of integer;
var X, fst, prec: mas;
procedure hamil (n,m:integer; L,lst:mas; var alfa:boolean);
{ Procedūra hamil apskaičiuoja Hamiltono ciklus paieškos gilyn iš
  pirmosios viršūnės metodu, kai grafas nusakytas L ir lst masyvais. Rasti
  ciklai atspausdinami. }

```

*Formalūs parametrai:*

*n* - grafo viršūnių skaičius,  
*m* - grafo briaunų (lankų) skaičius,  
*L, lst* - grafa nusakantys tiesioginių nuorodų masyvai. }

```

var i, k, u, v : integer;
j : integer;
t,p : boolean;
fst, prec : mas;
x : mas;

begin
{Inicializacija}
alfa:=false;
v:=1;
for i:=1 to n do
begin
fst[i]:=lst[i]+1;
prec[i]:=0;
end;
k:=v;
if fst[k] <= lst[k+1]
then {yra nenagrinėtu briaunu, incidentiškų viršūnei k }
begin
t:=false;
p:=true;
prec[k]:=k; {k-pradinė pateškos viršūnė}
{ Nagrinėti viršūnę k }
j:=1;
x[j]:=k;
end
else { viršūnė k yra arba izoliuota viršūnė, arba neturi
išeinančių lankų (orientuotojo grafo atveju); pateškos pabaiga }
t:=true;
while not t do { pateška nebaigta}
begin
{Pirmyn}
while p do
begin
u:=L[fst[k]];
if prec[u]=0 then {viršūnė u nauja}
begin
j:=j+1;

```

```

x[j]:=u;
prec[u]:=k; {i viršūnė u atejome iš viršūnės k}
if fst[u] <= lst[u+1] then
    {viršūnė u neišsemta}
    k:=u
else {viršūnė u išsemta}
    p:=false;
end
else
begin
{ write(u:3); }
p:=false; {viršūnė u nenauja}
if (j = n) and (u = 1)
then { Radome Hamiltono ciklą }
begin
alfa:=true;
x[n+1]:=1;
writeln('Ciklas');
for i:=1 to n+1 do
    write(x[i]:3);
writeln;
end;
end;
end;
{Atgal}
while not p and not t do
begin
{Imama nauja dar nenagrinėta briauna,
incidentiška viršūnei k}
fst[k]:=fst[k]+1;
if fst[k] <= lst[k+1] then {tokia briauna egzistuoja}
    p:=true
else {viršūnė k išsemta}
    if prec[k]=k then {pradinė paieškos viršūnė
        išsemta; paieškos pabaiga.}
        t:=true
    else {grįžome į viršūnę, iš kurios buvome atėję į
        viršūnę k }
begin
u:=prec[k];
prec[k]:=0;

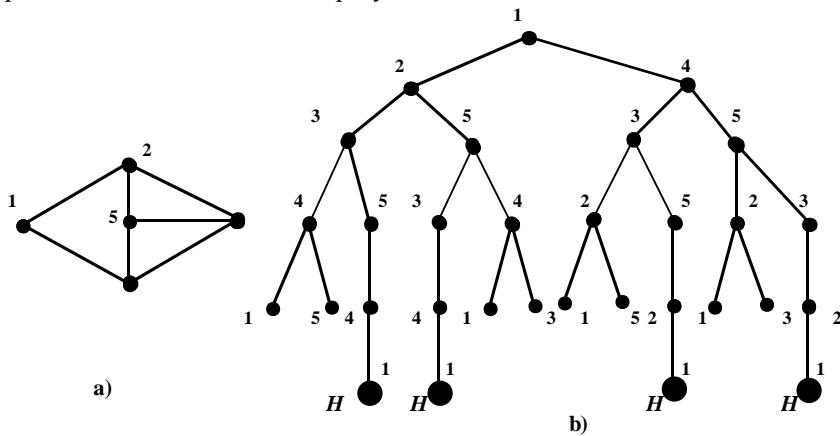
```

```

        fst[k]:=lst[k]+1;
        k:=u;
        j:=j-1;
    end;
end;
end;

```

**Pavyzdys.** Panagrinėsime 2.15.7 c) pav. grafą. Aprašytois procedūros pagalba gausime tokius Hamiltono ciklus: 1, 2, 3, 5, 4, 1; 1, 2, 5, 3, 4, 1; 1, 4, 3, 5, 2, 1; 1, 4, 5, 3, 2, 1. 2.15.8 a) pav. pavaizduotas 2.15.7 c) pav. grafas, 2.15.8 b) pav. pavaizduotas paieškų medis, t.y. paieškos gilyn su grįžimu perrinkti maršrutai. Raide "H" pažymėti Hamiltono ciklai.



2.15.8 pav. Hamiltono ciklų ieškojimas

Reikia pažymeti, nors pateiktos procedūros veiksmų skaičius yra žymiai mažesnis nei pilno perrinkimo veiksmų skaičius, kuris lygus  $O(n \cdot n!)$ , tačiau, nepalankiausiu atveju, ir šios procedūros veiksmų skaičius auga eksponentiškai nuo viršūnių skaičiaus. Tai bus teisinga ir tuo atveju, jei algoritmą nutrauksime, radę pirmajį Hamiltono ciklą. Jei grafas neturi Hamiltono ciklo, tai ir šiuo atveju reikės peržiūrėti visus galimus kelius.

#### Keliaujančio pirklio uždavinys

Kaip buvome minėję, su Hamiltono ciklu yra susijęs keliaujančio pirklio uždavinys. Priminsime šio uždavinio formulavimą.

Turime  $n$  miestų ir žinoma atstumų tarp šių miestų matrica  $C = [c_{ij}]$   $i = \overline{1, n}$ ,  $j = \overline{1, n}$ , čia  $c_{ij}$  – atstumas tarp miestų  $i$  ir  $j$ . Pirklys, išejęs

iš 1-ojo miesto, turi apeiti visus miestus po vieną kartą ir grįžti į pirmajį miestą.  
Koks turi būti pirklio maršrutas, kad jo ilgis būtų trumpiausias.

Šis uždavinys taip pat priklauso NP uždavinių klasei ir visų tikslų šio uždavinio sprendimo algoritmu sudėtingumas išreiškiamas eksponente nuo kintamujų skaičiaus  $n$ . Todėl, praktiškai sprendžiant šį uždavinį, naudojami įvairūs euristiniai algoritmai. Dažniausiai naudojamos euristikos yra:

- 1) artimiausio kaimyno metodas ir
- 2) miestų įterpimo metodas.

**Artimiausio kaimyno metodas.** Šis metodas pagrįstas taisykle: "jei pirklys yra mieste  $k$ , tai pirklys toliau kelias į artimiausią miestui  $k$  neaplankytą miestą  $l$ ; aišku, -  $l \neq 1$ , jei yra kita galimybė". Po šio veiksmo iš matricos  $C$  pašalinama  $k$ -oji eilutė (iš miesto  $k$  pirklys daugiau neišeis) ir  $l$ -asis stulpelis (iš  $l$ -ajį miestą pirklys niekada nebeužėis).

**Pavyzdys.** Taikydami artimiausio kaimyno metodą, apskaičiuokime pirklio maršrutą, esant tokiai atstumų matricai:

$$C = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 10 & 7 & 4 & 3 \\ 2 & 10 & 0 & 7 & 2 & 6 \\ 3 & 7 & 7 & 0 & 5 & 4 \\ 4 & 4 & 2 & 5 & 0 & 8 \\ 5 & 3 & 6 & 4 & 8 & 0 \end{array}.$$

Iš pirmojo miesto eisime į 5-ąjį miestą, nes  $c_{15} = \min_{\substack{1 \leq j \leq 5 \\ j \neq 1}} (c_{1j} / c_{1j} \neq 0)$ . Iš atstumų matricos pašalinę 1-ąją eilutę ir 5-ąjį stulpelį, gausime:

$$C = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 \\ \hline 2 & 10 & 0 & 7 & 2 \\ 3 & 7 & 7 & 0 & 5 \\ 4 & 4 & 2 & 5 & 0 \\ 5 & 3 & 6 & 4 & 8 \end{array}.$$

Dabar, iš 5-ojo miesto eisime į 3-ąjį, nes  $c_{53} = \min_{\substack{j \neq 1 \\ j \neq 5}} (c_{5j} / c_{5j} \neq 0)$ . Iš perskaičiuotos atstumų matricos pašaliname 5-ąją eilutę ir 3-ąjį stulpelį. Gauname:

$$C = \begin{array}{c|ccccc} & 1 & 2 & 4 \\ \hline 2 & 10 & 0 & 2 \\ 3 & 7 & 7 & 5 \\ 4 & 4 & 2 & 0 \end{array}.$$

Toliau iš 3-ojo miesto keliausime į 4-ąjį miestą, nes  $c_{34} = \min_{j \neq 1} (c_{3j} / c_{3j} \neq 0)$ . Pašalinus 3-ąją eilutę ir 4-ąjį stulpelį, gausime:

$$C = \begin{array}{c|cc} & 1 & 2 \\ \hline 2 & 10 & 0 \\ 4 & 4 & 2 \end{array}$$

Tada iš 4-ojo miesto keliausime į 2-ąjį miestą, o iš 2-ojo į 1-ąjį.

Tuo būdu, artimiausio kaimyno metodu apskaičiuotas maršrutas yra: 1, 5, 3, 4, 2, 1, ir jo ilgis yra  $c_{15} + c_{53} + c_{34} + c_{42} + c_{21} = 3+4+5+2+10=24$ .

**Ierpimo metodas.** Metodo idėja yra labai paprasta. Pirklio maršrutą konstruosime nuosekliai, pradėdami nuo ciklo  $v_1, v_2, v_1$ , ir kiekvienam žingsnyje šį ciklą praplēsime įterpdami po vieną naują miestą, t.y. po antrojo žingsnio turėsime ciklą, jungiantį tris miestus, po trečiojo – keturis miestus ir t.t., kol po  $(n-1)$ -ojo žingsnio turėsime ciklą, einantį per visus miestus. Praplēsdami ciklą, t.y. įterpdami naują miestą, elgsimės taip, kad praplēsto ciklo ilgio padidėjimas būtų minimalus. Be to pradinį ciklą  $v_1, v_2, v_1$  parinksime taip, kad jo ilgis būtų mažiausias iš visų galimų to ilgio ciklų.

Formaliai aprašysime įterpimo metodą.

#### Pradinio ciklo parinkimas.

1. Apskaičiuoti  $p = \min_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n \\ i \neq j}} (c_{ij} + c_{ji})$ .

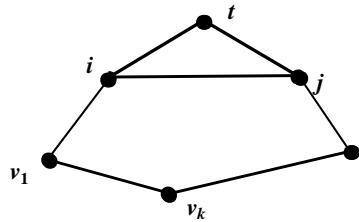
**Pastaba.** Jei atstumų matrica yra simetrinė, tai pakanka rasti  $\min_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n \\ i \neq j}} c_{ij}$ .

2. Tarkime, kad  $p = c_{kl} + c_{lk}$ . Tada pradinis ciklas yra  $k, l, k$  (žr. 2.15.9 pav.).



2.15.9 pav. Pirklio pradinis ciklas

**Ciklo praplėtimas.** Tarkime po  $(k-1)$ -ojo žingsnio turime ciklą  $v_1, v_2, v_3, \dots, i, j, \dots, v_k, v_1$ . Norédami praplēsti ciklą, bandysime įterpti vieną iš likusių miestų tarp dviejų gretimų ciklo miestų  $i$  ir  $j$ . Iš visų galimų  $(i, j)$  porų pasirinksime tokią porą ir tokį įterpiamą miestą, kad ciklo pailgėjimas būtų minimalus, t.y. reikia apskaičiuoti  $d = \min_{\forall (i,j) \in (i,j) \in \text{ciklui } t \in V - \text{ciklas}} (c_{it} + c_{tj} - c_{ij})$  (žr. 2.15.10 pav.).



2.15.10 pav. Miesto įterpimas

Tarkime, kad mažiausia  $d$  reikšmė yra miestų  $(i,j)$  porai, o įterpiamas miestas -  $t$ . Tada gausime ciklą  $v_1, v_2, \dots, i, t, j, \dots, v_k, v_1$ .

**Pavyzdys.** Sukonstruokime pirklio maršrutą, esant tai pačiai atstumų matricai  $C$  (žr. artimiausio kaimyno metodą).

$$C = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 10 & 7 & 4 & 3 \\ 2 & 10 & 0 & 7 & 2 & 6 \\ 3 & 7 & 7 & 0 & 5 & 4 \\ 4 & 4 & 2 & 5 & 0 & 8 \\ 5 & 3 & 6 & 4 & 8 & 0 \end{array}$$

Pradinis ciklas bus:  $2, 4, 2$  (žr. 2.15.11 a) pav.). Simboliu  $d_{i,l,j}$  pažymėkime ciklo pailgėjimą, kai tarp miestų  $i$  ir  $j$  įterpiamas miestas  $l$ . Tada

$$\begin{aligned} d_{2,1,4} &= c_{21} + c_{14} - c_{24} = 10 + 4 - 2 = 12, \\ d_{2,3,4} &= c_{23} + c_{34} - c_{24} = 7 + 5 - 2 = 10, \\ d_{2,5,4} &= c_{25} + c_{54} - c_{24} = 6 + 8 - 2 = 12. \end{aligned}$$

Kadangi matrica  $C$  yra simetrinė, tai, terpiant miestą tarp 4-ojo ir 2-ojo, gausime tuos pačius atstumus. Vadinas, 3-ajį miestą įterpsime tarp 2-ojo ir 4-ojo miestų. Gausime ciklą  $2, 3, 4, 2$  (žr. 2.15.11 b) pav.).

Apskaičiuosime

$$\begin{aligned} d_{2,1,3} &= c_{21} + c_{13} - c_{23} = 10 + 7 - 7 = 10, \\ d_{2,5,3} &= c_{25} + c_{53} - c_{23} = 6 + 4 - 7 = 3, \\ d_{3,1,4} &= c_{31} + c_{14} - c_{34} = 7 + 4 - 5 = 6, \\ d_{3,5,4} &= c_{35} + c_{54} - c_{34} = 4 + 8 - 5 = 7, \\ d_{4,1,2} &= c_{41} + c_{12} - c_{42} = 4 + 10 - 2 = 12, \\ d_{4,5,2} &= c_{45} + c_{52} - c_{42} = 8 + 6 - 2 = 12. \end{aligned}$$

Iš šių skaičių mažiausias yra 3. Vadinas, pailgėjės ciklas yra:  
 $2, 5, 3, 4, 2$  (žr. 2.15.11 c) pav.).

Apskaičiuosime

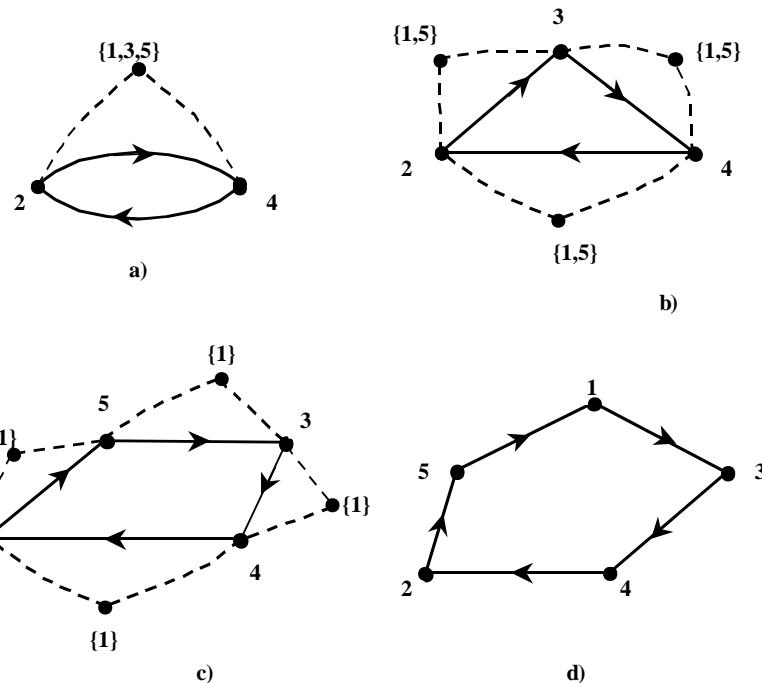
$$d_{2,1,5} = c_{21} + c_{15} - c_{25} = 10 + 3 - 6 = 7,$$

$$d_{5,1,3} = c_{51} + c_{13} - c_{53} = 3 + 7 - 4 = 6,$$

$$d_{3,1,4} = c_{31} + c_{14} - c_{34} = 7 + 4 - 5 = 6,$$

$$d_{4,1,2} = c_{41} + c_{12} - c_{42} = 4 + 10 - 2 = 12.$$

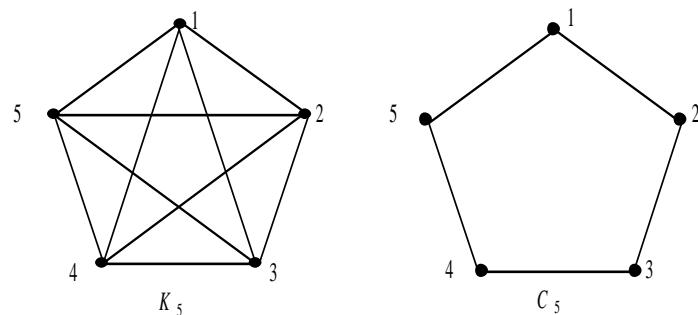
Yra du lygūs mažiausiai skaičiai. Pasirenkame pirmajį  $d_{5,1,3}$ . Tada pirklio maršutas yra:  $2, 5, 1, 3, 4, 2$  ir jo ilgis yra  $c_{25} + c_{51} + c_{13} + c_{34} + c_{42} = 6 + 3 + 7 + 5 + 2 = 23$  (žr. 2.15.11 d) pav.). Šis pirklio maršutas yra trumpesnis už maršutą, gautą artimiausio kaimyno metodu. Tas paaiškinama tuo, kad įterpimo metode buvo atliktas didesnis perrinkimas.



2.15.11 pav. Pirklio maršruto konstravimas įterpimo metodu

## 2.16. Jungumas

Jungusis grafas – tai grafas, kuriame bet kuri viršūnių pora sujungta grandine. Pavyzdžiui, (žr. 2.16.1 pav.)  $K_5$  ir  $C_5$  yra jungieji grafa, tačiau intuityviai jaučiame, kad  $K_5$  yra labiau jungus nei  $C_5$ .



2.16.1 pav. Jungumas

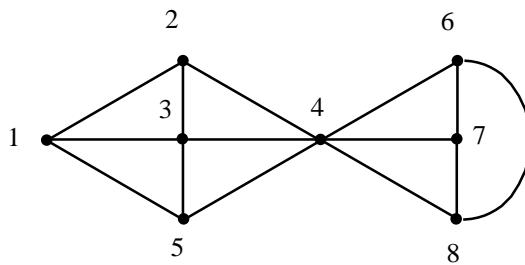
Grafo jungumo klausimas yra aktualus praktikoje. Tarkime, reikia suprojektuoti kompiuterių tinklą. Tinklą sudaro informacijos saugojimo ir apdorojimo centrai, kurie jungiami ryšio kanalais. Informacijos pasikeitimas tarp dviejų centrų vyksta arba tiesiogiai per šiuos centrus jungiantį ryšio kanalą (jei toks yra), arba per kitus centrus ir juos jungiančius ryšio kanalus. Aišku, kad tinklo grafas, kurio viršūnės vaizduoja centrus, o dvi viršūnės jungiamos briauna, jei tarp viršūnėms atitinkančių centrų yra ryšio kanalas, turi būti jungusis. Labai svarbus tinklo parametras yra jo patikimumas: tinklas turi funkcionuoti net ir tuo atveju, jei sugenda keli centrali arba ryšio kanalai. Ši parametras kiekybiškai galima įvertinti per žemiau įvestas sąvokas.

**Viršūninio jungumo skaičius.** Tai mažiausias skaičius viršūnių, kurias pašalinus, grafas  $G$  tampa arba nejungiuojančiu grafu. Šis skaičius žymimas  $\kappa(G)$ .

**Briauninis jungumo skaičius.** Tai mažiausias skaičius briaunų, kurias pašalinus, grafas  $G$  tampa nejungiuojančiu grafu. Šis skaičius žymimas  $\lambda(G)$ .

Pavyzdžiui,  $\kappa(K_n) = n - 1$ ,  $\kappa(C_n) = 2$ .

2.16.2 pav. grafui  $\kappa(G) = 1$  (pašalinus 4-ąją viršūnę, grafas suyra), o  $\lambda(G) = 3$  (pašalinus briaunas (4, 6), (4, 7), (4, 8) grafas suyra).

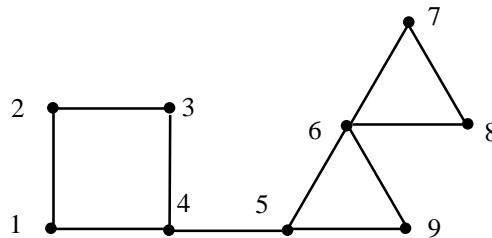


**2.16.2 pav.** Grafo jungumo kiekybiniai įverčiai

Grafo  $G$  viršūnė  $v$  vadinama **sąlyčio tašku**, jei  $G - v$  turi daugiau jungiųjų komponenčių nei grafas  $G$ .

Grafo  $G$  briauna  $(x, y)$  vadinama **tiltu**, jei, ją pašalimus, gautasis grafas turi daugiau jungiųjų komponenčių nei grafas  $G$ .

Pavyzdžiu, 2.16.3 pav grafas turi tris sąlyčio taškus (viršūnės 4, 5, 6) ir vieną tiltą (briauna  $(4, 5)$ ).



**2.16.3 pav.** Grafo sąlyčio taškai ir tiltai

Grįžtant prie paragrafo pradžijoje pateikto pavyzdžio, nesunku pastebėti, kad grafo viršūninio jungumo ir briauninio jungumo skaičius parodo tinklo atsparumą centrui ir ryšio kanalų gedimams, o sąlyčio taškai bei tiltai parodo labiausiai pažeidžiamas tinklo vietas.

**Teorema.** Kiekvienam grafui

$$\kappa(G) \leq \lambda(G) \leq \delta(G),$$

čia  $\delta(G) = \min_{v \in V} d(v)$ , o  $d(v)$  –  $v$ -osios viršūnės laipsnis.

**Teorema.** Beveik kiekvienam grafui<sup>6</sup> teisinga lygybė  
 $\kappa(G) = \lambda(G)$ .

Grafas vadinamas ***k-jungiuoju***, jei  $\kappa(G) \geq k$  ir ***briaunomis k-jungiuoju***, jei  $\lambda(G) \geq k$ .

Jei  $\kappa(G) = 2$ , tai grafas vadinamas ***dviryšiu***.

2.16.2 pav. pavaizduotas grafas yra 1-jungusis ir briaunomis – 3-jungusis. Aišku, kad šis grafas turi pografius, kurie yra labiau jungieji nei pats grafas. Pavyzdžiui, pografis, kurį indukuoja viršūnių {1,2,3,4,5} aibė, yra 3-jungusis. Tokių pografių apibūdinimui įvedama *k*-jungiosios komponentės sąvoka.

**Grafo *k*-jungioji komponentė** – tai maksimalus *k*-jungusis pografis. Jis dažnai vadinamas ***k-komponente***.

Pavyzdžiui, 2.16.4 pav. grafas  $G_1$  turi dvi 2-jungišias komponentes (pografių, kuriuos indukuoja viršūnių {1,2,3,4} ir {3,5,6,7} aibės), o grafas  $G_2$  turi dvi 3-komponentes (pografių, kuriuos indukuoja viršūnių {1,2,3,4,5,6,7} ir {4,6,8,9,10,11,12,13} aibės). Tačiau reikia pabrėžti, kad abu grafai  $G_1$  ir  $G_2$  yra 1-jungieji. Be to, nesunku pastebėti, kad dvi 2-komponentės turi vieną bendrą viršūnę (Grafe  $G_1$  – tai 3-ioji viršūnė), o dvi 3-komponentės turi dvi bendras viršunes (grafe  $G_2$  – tai 4-oji ir 6-oji viršūnės).

**Teorema.** Dvi skirtinges grafo  $G$  *k*-komponentės turi ne daugiau nei  $(k-1)$  bendrų viršūnių.

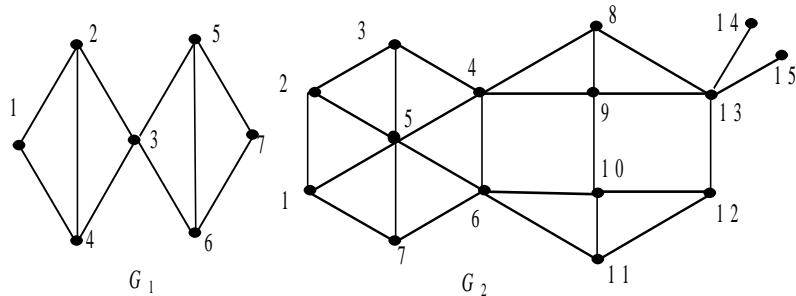
**Apibrėžimas.** Dvi  $(a,b)$ -grandinės vadinamos nesusikertančiomis (viršūnėmis nesusikertančiomis), jei jos neturi bendrų viršūnių, išskyrus  $a$  ir  $b$ .

**Teorema** (H.Witney, 1932). Grafas yra *k*-jungusis tada ir tikai tada, kai bet kuri nesutampančią viršūnių pora  $(a,b)$  sujungta ne mažiau kaip *k* viršūnėmis nesusikertančių grandinių.

**Apibrėžimas.** Sakoma, kad grafo  $G$  viršūnių poaibis  $S$  skiria viršunes  $a$  ir  $b$ , jei grafe  $G - S$  viršūnės  $a$  ir  $b$  priklauso skirtinges jungiosioms komponentėms.

---

<sup>6</sup> Primename sąvoką “beveik kiekvienam grafui”. Simboliu  $\varphi P(n)$  pažymėkime skaičių  $n$ -viršūninių grafų, turinčių savybę  $P$ , o simboliu  $\varphi(n)$  – visų  $n$ -viršūninių grafų skaičių. Tada sakysime, kad “beveik visi grafai turi savybę  $P$ ”, jei  $\lim_{n \rightarrow \infty} \frac{\varphi P(n)}{\varphi(n)} = 1$ , o jei  $\lim_{n \rightarrow \infty} \frac{\varphi P(n)}{\varphi(n)} = 0$ , tai sakome, kad “beveik nėra grafų, turinčių savybę  $P$ ”.



#### 2.16.4 pav. Grafo $k$ -komponentės

**Teorema** (K.Mengeras, 1927). Mažiausias skaičius viršūnų, skiriančiu dvi negretimas viršunes  $a$  ir  $b$ , yra lygus didžiausiam skaičiui poromis nesusikertančių grandinių, jungiančių  $a$  ir  $b$  viršunes.

Iveskime skiriančių briaunų sąvoką.

**Apibrėžimas.** Briaunu aibė  $R$  skiria grafo  $G$   $a$  ir  $b$  viršunes, jei grafe  $G - R$  viršūnės  $a$  ir  $b$  priklauso skirtingoms jungiamosioms komponentėms.

**Teorema.** Mažiausias skaičius briaunų, skiriančių grafo  $G$  viršunes  $a$  ir  $b$ , yra lygus didžiausiam briaunomis nesusikertančių grandinių, jungiančių  $a$  ir  $b$  viršunes, skaičiui.

#### 2.16.1. Dviryšiai grafai

Tiek grafų teorijoje, tiek ir praktikoje svarbią vietą užima 2-jungieji grafai, kurie vadinami dviryšiais grafais.

Kaip buvo minėta aukščiau, grafas  $G = (V, U)$  yra dviryšis, jei bet kurią nesutampančią viršunių  $a$  ir  $b$  porą jungia bent dvi viršūnėmis nesusikertančios grandinės.

Galima pateikti ir kitą dviryšio grafo apibrėžimą: grafas  $G = (V, U)$  vadinamas dviryšiu, jei jis neturi sąlyčio taškų.

**Apibrėžimas.** Didžiausias galimas grafo  $G$  pografis, neturintis sąlyčio taškų, vadinamas **dvigubo jungumo komponente** arba **bloku**.

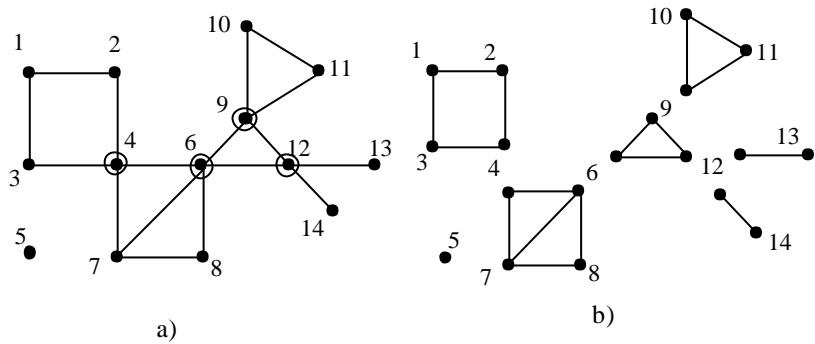
Grafo dvigubo jungumo komponenčių bei sąlyčio taškų ieškojimo uždavinys turi svarbią reikšmę. Pavyzdžiu, visi komunaliniai ir transporto tinklai negali turėti sąlyčio taškų, t.y. mažiausiai jie turi būti dviryšiai.

Dvigubo jungumo komponenčių išskyrimas grafe padeda spręsti nepriklausomų ciklų radimo uždavinį bei nustatyti, ar grafas yra plokštusis.

Aptarsime grafo  $G = (V, U)$  dvigubo jungumo komponenčių (dviryšių komponenčių, blokų) radimo uždavinį.

**Sąlyčio taško savybė.** Neorientuotojo jungiojo grafo  $G$  viršūnė  $a$  yra sąlyčio tašku tada ir tikai tada, kada egzistuoja tokios kitos dvi grafo viršūnės  $x$  ir  $y$ , kad bet kuris kelias tarp viršūnių  $x$  ir  $y$  eina per viršūnę  $a$ .

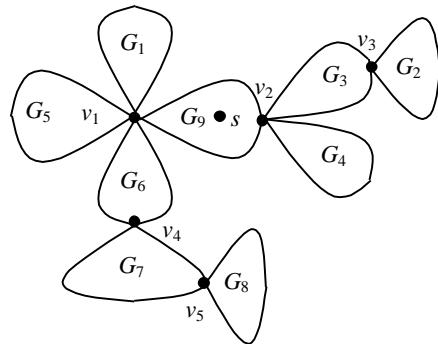
Pavyzdžiu, 2.16.5 a) pav. Pavaizduotas grafas  $G$ , o 2.16.5 b) pav. – to grafo dvigubo jungumo komponentės.



**2.16.5 pav.** Grafo dvigubo jungumo komponentės

Ši sąlyčio taško savybė įgalina panaudoti paieškos gilyn metodą, apskaičiuojant grafo  $G$  dvigubo jungumo komponentes – blokus.

Panagrinėkime pavyzdį (žr. 2.16.6 pav.).



**2.16.6 pav.** Grafo blokinė schema

2.16.6 pav. schematiškai vaizduoja junguji grafa, susidedantį iš 9-ių dvigubo jungumo komponenčių  $G_i$ ,  $i = \overline{1, 9}$  bei turintį 5-is sąlyčio taškus  $v_1, v_2, v_3, v_4$ , ir  $v_5$ .

Tarkime, kad paiešką gilyn pradėjome iš 9-ojo bloko  $G_9$  viršūnės  $s$ , ir visas aplankytas viršunes (briaunas) talpiname į steką STEK. Pradėję paiešką iš viršūnės  $s$ , per viršūnę  $v_2$  galime patekti į bloką  $G_4$ . Remiantis paieškos gilyn savybe, kad per viršūnę  $v_2$  grįšime, kai bus aplankytos visos  $G_4$  viršūnės, galime teigti, kad  $G_4$  sudaro viršūnės (briaunos), kurios buvo aplankytos tarp iėjimo ir grįžimo per viršūnę  $v_2$ .

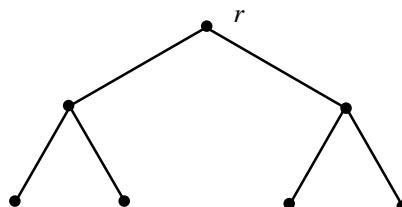
Su kitomis dvigubo jungumo komponentėmis reikalas sudėtingesnis. Iškeliavę iš viršūnės  $s$ , galime patekti į jungumo komponentę  $G_3$ , o iš  $G_3$  per viršūnę  $v_3$  – patekti į komponentę  $G_2$ . Tačiau ir šiuo atveju, kai aplankytos viršūnės (briaunas) saugomos steke STEK, grįžtant per viršūnę  $v_3$ , visos aplankytos bloko  $G_2$  viršūnės bus steko viršuje (nuo steko viršaus iki viršūnės  $v_3$ ). Pašalinus iš steko šias viršunes, steko viršuje liks bloko  $G_3$  viršūnės, kurios buvo aplankytos iki patekome į bloką  $G_2$ . Kai grįšime į viršūnę  $v_2$ , steko viršuje bus visos bloko  $G_3$  viršūnės.

*Vadinasi*, jei mokėtume atpažinti *sąlyčio taškus*, tai naudodami *paiešką gilyn* ir saugodami aplankytas viršunes (briaunas) *steke* ta tvarka, kuria jos buvo aplankytos, galime rasti *dvigubo jungumo komponentes*: *viršūnės (briaunos)*, *esančios steko viršuje grįžimo per sąlyčio tašką metu, sudaro dvigubo jungumo komponentę*.

Aptarkime formalią sąlyčio taškų radimo sąlygą.

Tarkime, kad  $G = (V, U)$  – jungasis neorientuotas grafas. Tarkime, kad  $T$  – jį dengiantis medis su šaknimi  $r$ , sukonstruotas paieškos gilyn iš viršūnės  $r$  metodu.

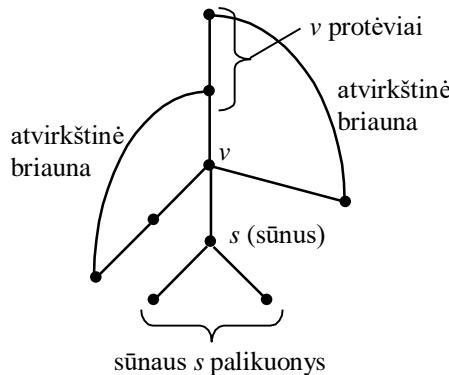
**1 lema.** Šaknis  $r$  yra sąlyčio taškas tada ir tikta tada, kai  $r$  turi daugiau nei vieną sūnų (žr. 2.16.7 pav.).



2.16.7 pav. Dengiančio medžio  $T$  šaknis  $r$  – sąlyčio taškas

**2 lema.** Viršūnė  $v \neq r$  yra grafo salyčio taškas tada ir tikai tada, kai yra nors vienas viršūnės  $v$  sūnus  $r$ , kuris neturi atvirkštinių briaunų, jungiančių jį patį arba bet kurį jo palikuonį su viršūnės  $v$  protėviu (žr. 2.16.8 pav.).

Priminsime, kad grafo  $G$  atvirkštinės briaunos yra visos briaunos, nepriklausančios dengiančiojo medžio  $T$  briaunų aibei.



**2.16.8 pav.** Dengiančio medžio  $T$  viršūnė  $v$  – salyčio taškas

Salyčio taškams rasti kiekvienai grafo viršūnei  $v$  įvesime du parametrus:  $nr[v]$  ir  $low[v]$ .

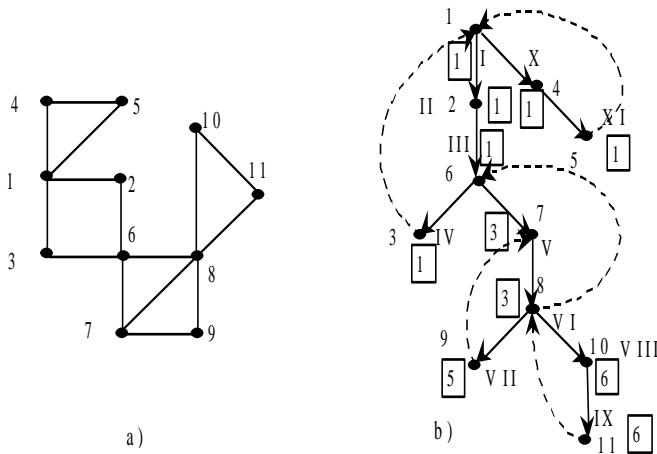
Parametras  $nr[v]$  yra viršūnės  $v$  aplankymo eilės numeris, atliekant paiešką gilyn iš viršūnės  $r$ .

Parametras  $low[v]^7$  – tai mažiausias aplankymo numeris viršūnės  $x$ , į kurią galime patekti iš viršūnės  $v$  grandine, sudaryta iš nulio arba daugiau dengiančio medžio briaunų ir nedaugiau kaip vienos atvirkštinės briaunos (žr. 2.16.9 pav.).

2.16.9 a) pav. pavaizduotas grafas  $G$ , o 2.16.9 b) pav. pavaizduotas tą grafą dengiantis medis, sukonstruotas paieškos gilyn iš 1-osios viršūnės metodu. Medžio briaunos pavaizduotos ištisine, o atvirkštinės briaunos – punktryrine linija. Romėniškais skaiciiais pažymėti viršūnių aplankymo eilės numeriai, o parametru  $low[v]$  reikšmės apibrėžtos.

---

<sup>7</sup> Parametrą  $low[v]$  galima apibrėžti ir kitais žodžiais: parametras  $low[v]$  – tai mažiausias aplankymo numeris viršūnės  $x$ , į kurią galime patekti iš viršūnės  $v$  arba bet kurio jos palikuonio ne daugiau kaip vienos atvirkštinės briaunos pagalba.



**2.16.9 pav.** Parametru  $nr[v]$  ir  $low[v]$  reikšmės

Pasinaudodami parametrais  $nr[v]$  ir  $low[v]$  galima 2 lemos kriterijų perrašyti kaip teoremą.

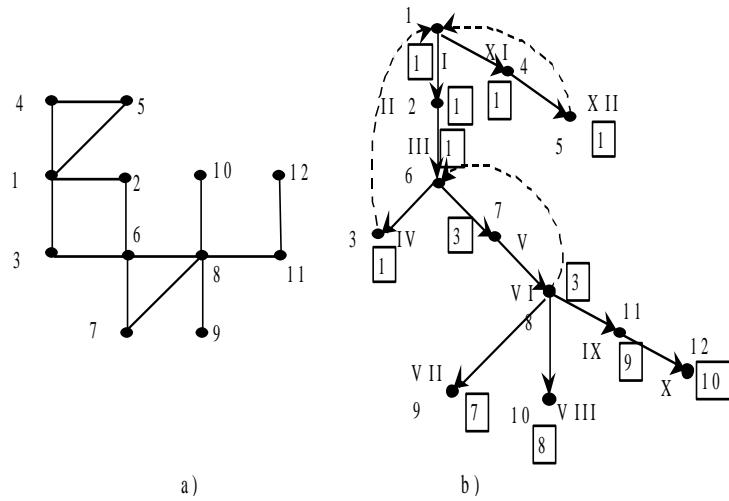
**Teorema.** Viršūnė  $v \neq r$  yra grafo  $G$  sąlyčio taškas tada ir tikai tada, kai  $v$  turi sūnų  $s$ , kuriam  $low[s] \geq nr[v]$ .

Pavyzdžiu, 2.16.9 a) pav. grafui sąlyčio taškai yra: 1-oji viršūnė (medžio šaknis, turinti du sūnus), 6-oji viršūnė (ji turi sūnų  $s = 7$ , kuriam  $low[7] = nr[6]$ ) ir 8-oji viršūnė (ji turi sūnų  $s = 10$ , kuriam  $low[10] = nr[8]$ ).

Bendresnis pavyzdys pateiktas 2.16.10 paveiksle. Čia 2.16.10 a) pav. duotas grafas, o 2.16.10 b) – jį dengiantis medis, sukonstruotas naudojant paiešką gilyn iš 1-osios viršūnės. Simbolių prie medžio reikšmės tos pačios, kaip ir 2.16.9 b) pav. Remiantis auksčiau pateiktu nagrinėjimu, nesunku apskaičiuoti, kad šio grafo sąlyčio taškai yra: 1-oji viršūnė (šaknis, turinti du sūnus), 6-oji viršūnė (sūnaus  $s = 7$  parametras  $low[7] = nr[6]$ ), 8-oji viršūnė (turi du sūnus  $s = 9$  ir  $s = 10$ , kurių  $low[s] > nr[8]$ ) ir 11-oji viršūnė (sūnaus  $s = 12$   $low[12] > nr[11]$ ).

Parametrą  $low[v]$  galima apibrėžti rekurentiškai:

$low[v] = \min (nr[v]; low[s], \text{ čia } s - \text{viršūnės } v \text{ sūnus}; nr[w], \text{ čia } (v, w) - \text{atvirkštinė briauna})$ .



**2.16.10 pav.** Salyčio taškų apskaičiavimas

Remiantis šiuo apibrėžimu,  $low[v]$  reikšmė apskaičiuojama taip.

1. Kai viršūnė v paieškos gilyne metu yra aplankoma pirmą kartą, tai  $low[v] := nr[v]$ .
2. Kai nagrinėjama atvirkštinė briauna  $(v, w)$ , tai  $low[v] := \min(low[v], nr[w])$ .
3. Kai grįžtame į viršūnę v, pilnai apėjus visas briaunas, incidentiškas sūnui s, tai  $low[s] := \min(low[v], low[s])$ .

Kai grįžtame į viršūnę v, pilnai apėjus visas briaunas, incidentiškas sūnui s, tai  $low[s]$  reikšmė nusistovi. Jei šiuo metu  $low[s] \geq nr[v]$ , tai, pagal teoremą, v yra salyčio taškas.

Žemiau pateikta dvigubo jungumo komponenčių ieškojimo procedūra, kurios pagrindą sudaro paieškos gilyn algoritmas, išnagrinėtas 2.8.1.2 paragafe, ir kuris papildytas aukščiau aptartu salyčio taškų salygos tikrinimu bei blokų formavimu.

```
const c = 500;
type
    mas = array [1..c] of integer;
    matr = array [1..2, 1..c] of integer;
procedure blok (n, m : integer; L, lst : mas; var st : mas);
```

{ Procedūra **blok** randa jungiojo grafo dvigubo jungumo komponentes (blokus) paieškos gilyn iš pirmosios viršūnės metodu, kai grafas nusakytas L ir lst masyvais. Rasti sąlyčio taškai ir blokų briaunų aibės atspausdinamos.

Formalūs parametrai:

```

n – grafo viršūnių skaičius,
m – grafo briaunų (lankų) skaičius,
L, lst – grafą nusakantys tiesioginių nuorodų masyvai;
st – sąlyčio taškų masyvas:
    jei st [i] = 1, tai viršūnė i yra sąlyčio taškas,
    jei st [i] = 0, tai viršūnė i nėra sąlyčio taškas. }

var i, k, u : integer;
s, sk, tau : integer;
t, p, pirma : boolean;
fst, prec : mas;
stek : mas;
nr, low : mas;

begin
{ Inicializacija }
pirma := false;
v := 1;
nr [v] := 1;
low [v] := 1;
sk := 1;
tau := 0;
for i := 1 to n do begin
    fst [i]:= lst [i] + 1;
    prec [i] := 0;
    st [i] := 0;
end;
k := v;
if fst [k] <= lst [k + 1] then {yra nenagrinetu briaunu, incidentisku viršunei k }
begin
    t := false;
    p := true;
    prec [k] := k; {k-pradinė paieškos viršūnė }
    { Nagrinėti viršūnę k }
end
else {viršūnė k yra arba izoliuota viršūnė, arba neturi išeinančių
lankų (orientuotojo grafo atveju); paieškos pabaiga }
    t := true;
while not t do { paieška nebaigta }
```

```

begin
{Pirmyn}
while p do
begin
    u := L [fst [k]];
    if prec [u] = 0 then {viršūnė u nauja}
    begin
        { Negrinėti viršūnę u }
        sk := sk +1;
        nr [u] := sk;
        low [u] := nr [u];
        tau := tau + 1;
        stek [tau] := k;
        tau := tau + 1;
        stek [tau] := u;
        prec [u] := k;{ i viršūnę u atėjome iš viršūnės k}
        iffs t [u] <= lst [u + 1] then {viršūnė u neišsemta}
        k := u
            else {viršūnė u išsemta}
        p := false;
    end
    else
begin
    if (prec [k] <> u) and (nr [k] > nr[u]) then
    { Briauna (k, u) - atvirkštinė briauna }
    begin
        tau := tau + 1;
        stek [tau] := k;
        tau := tau + 1;
        stek [tau] := u;
        { low[k]=min (low [k], nr[u]) }
        if nr [u] < low [k] then low [k] := nr [u];
        end;
        p := false; {viršūnė u nenauja}
    end;
end;
{Atgal}
while not p and not t do
begin
    {Imama nauja, dar nenagrindėta briauna, incidentiska viršūnei k}
    fst [k] := fst [k] + 1;

```

```

if fst [k] <= lst [k + 1] then {tokia briauna egzistuoja}
begin
    if k = 1 then
        { pirmoji viršūnė turi daugiau nei vieną sūnų }
        pirma := true;
        p := true;
    end
    else {viršūnė k išsemta}
if prec [k] = k then { pradinė paieškos viršūnė išsemta;
paieškos pabaiga }
t := true
else { grįžome į viršūnę,
iš kurios buvome atėję į viršūnę k }
begin
    s := k;{ viršūnė s išsemta }
    k := prec [k];
    { low [k] = min (low [k], low [s]) }
    if low [s] < low [k] then low[k]:=low[s];
    if ((k >> 1) and (low [s] >= nr [k])) or ((k = 1) and pirma) then
    { k - sąlyčio taškas }
    begin
        st[k] := 1;
        writeln ('sql. taškas = ', k : 3);
    end;
    if low [s] >= nr [k] then
    begin
        { Spausdinti bloką }
        writeln ('blokas');
        while (stek [tau] <> s) or (stek [tau - 1] <> k) do
        begin
            write (stek [tau] : 3, '-' );
            stek [tau - 1] : 3);
            tau := tau - 2;
            writeln;
        end;
        write (stek [tau] : 3, '-' );
        stek [tau - 1] : 3);
        tau := tau - 2;
    end;
end;

```

```

        writeln;
    end;
end;
end;
end;
end;

```

## 2.17. Srautai tinkluose ir giminingi uždaviniai

### 2.17.1. Pagrindinės sąvokos

Aptarsime kelis praktinių uždavinių pavyzdžius.

**Pirmas pavyzdys.** Tarkime, kad turime tinklą automobilinių kelių, kuriais galima nuvykti iš punkto  $A$  į punktą  $B$ . Keliai gali kirstis tarpiniuose punktuose. Kiekvienai kelio atkarpai yra žinomas maksimalus skaičius automobilių, kuriuos ši kelio atkarpa gali praleisti per laiko vienetą (kelio pralaidumas). Koks didžiausias skaičius automobilių per laiko vienetą gali nuvažiuoti iš punkto  $A$  į punktą  $B$ ? Šis automobilių skaičius vadinamas nagrinėjamo kelio tinklo automobilių srauto dydžiu. Paparastai mus domina ne vien tik koks didžiausias automobilių skaičius gali nuvykti iš  $A$  į  $B$ , bet ir kiekvieno kelio ruožo srautas, t.y. koks skaičius automobilių vyksta šiuo kelio ruožu.

Aišku, kad galimas ir kitas klausimas: kiek ir kokių kelių pralaidumas reikia padidinti, kad maksimalus automobilių srautas per ši kelių tinklą padidėtų nurodytu dydžiu?

**Antras pavyzdys.** Tarkime, kad naftos verslovė  $A$  naftotiekių tinklui yra sujungta su perdirbimo įmone  $B$ . Taip pat žinome kiekvieno naftotiekio tarpo maksimalų pralaidumą – debitą (naftos kiekį per laiko vienetą). Kokį kiekį naftos per laiko vienetą galime perpumpuoti šiuo tinklui?

**Trečias pavyzdys.** Turime informacijos perdavimo tinklą. Žinomas kiekvienos ryšio linijos pralaidumas. Kokį didžiausią informacijos kiekį šiuo tinklu galima perduoti iš punkto  $A$  į punktą  $B$ ?

Šie ir daugelis kitų praktinių uždavinių nagrinėjami srautų tinkluose teorijos metodais. Šiame paragrade spręsime pagrindinį šios teorijos uždavinį – **maksimalus srauto radimo uždavinį**.

**Apibréžimas.** Tinklas, tai pora  $S = (G, C)$ , kur pirmasis poros elementas  $G = (V, U)$  yra bet koks orientuotas grafas, kuriame išskirtos dvi viršūnės  $s$  ir  $t$ , iš kurių pirmoji viršūnė  $s$ , neturinti įeinančių lankų, vadinama **šaltiniu**, o antroji –  $t$ , neturinti išeinančių lankų, vadinama **nuotakiu**. Antrasis poros

elementas  $C$  yra funkcija  $U \rightarrow R$ , kuri kiekvienam lankui  $(u, v)$  priskiria neneigiamą realųjį skaičių  $c(u, v)$ , kuris vadinamas ***lanko pralaidumu***.

***Apibrėžimas.*** Tarkime, kad duota funkcija  $f : U \rightarrow R$ . Tada funkcijos  $f$  ***divergencija*** viršūnėje  $v$  vadinamas dydis  $\text{div}_f(v)$ , apibrėžiamas formule:

$$\text{div}_f(v) = \sum_{u:(v,u) \in U} f(v, u) - \sum_{u:(u,v) \in U} f(u, v). \quad (2.17.1)$$

Jei  $f(u, v)$  interpretuojame kaip srautą iš viršūnės  $u$  į viršūnę  $v$ , tai skaičius  $\text{div}_f(v)$  parodo “srauto kiekį”, kuris išeina iš viršūnės  $v$ . Šis skaičius gali būti ***teigiamas*** (jei iš viršūnės  $v$  daugiau išeina negu įėjina, t.y. viršūnė  $v$  yra papildomas šaltinis), ***neigiamas*** (jei i viršūnė  $v$  daugiau įėjina negu išeina, t.y. srautas kaupiasi viršūnėje  $v$ ) ir ***lygus nuliui***.

***Apibrėžimas.*** Funkcija  $f : U \rightarrow R$  vadinama ***srautu*** tinkle  $S$ , jeigu:

1. kiekvienam grafo  $G$  lankui  $(u, v)$  galioja nelygybė

$$0 \leq f(u, v) \leq c(u, v), \quad (2.17.2)$$

2. kiekvienos viršūnės  $v$ , išskyryus  $s$  ir  $t$ , divergencija lygi nuliui, t.y.

$$\text{div}_f(v) = 0, \quad v \in V \setminus \{s, t\}. \quad (2.17.3)$$

Skaičius  $W(f) = \text{div}_f(s)$  vadinamas ***srauto dydžiu***.

Pagal ši apibrėžimą nagrinėjame srautą, kuris nesikaupia ir neatsiranda nei vienoje tarpinėje grafo  $G$  viršūnėje (išskyryus viršūnes  $s$  ir  $t$ ).

Toliau spręsime maksimalaus srauto radimo uždavinį: *rasime tokią funkciją  $f : U \rightarrow R$ , kuri tenkintų (2.17.2) ir (2.17.3) apribojimus ir kuriai skaičius  $W(f) = \text{div}_f(s)$  būtų didžiausias.* Aišku, kad funkcijos  $f$  reikšmės  $f(u, v)$  kiekvienam lankui  $(u, v)$  apibrėš maksimalų srautą per ši lanką.

Maksimalaus srauto apskaičiavimo uždavinys yra tiesinio programavimo uždavinys, todėl jam gali būti taikomi tiesinio programavimo uždavinii sprendimo metodai. Tačiau, įvertinant uždavinio specifiką, yra sukurti žymiai efektyvesni šio uždavinio sprendimi metodai, kuriuos toliau ir nagrinėsime.

***Pjūvio apibrėžimas.*** Tinklo  $S$  pjūviu  $P(A)$ , kurį apibrėžia grafo  $G$  viršūnių tikrinis poaibis  $A$ , t.y.  $A \subset V$  ir  $A \neq \emptyset$  ir  $A \neq V$ , vadiname grafo  $G$  tokiai lankų  $(u, v) \in U$  aibę, kad  $u \in A$ , o  $v \in V \setminus A$ .

Kitaip tariant,

$$P(A) = U \cap (A \times (V \setminus A)).$$

Aišku, kad bet kokiam tinklo  $S$  srautui  $f$  pjūvio srautas  $P(A)$  yra lygus pjūvį sudarančių lankų sumai, t.y.

$$f(A, V \setminus A) = f(P(A)) = \sum_{e \in P(A)} f(e).$$

**Lema.** Jei  $s \in A$ , o  $t \in V \setminus A$ , tai bet kokiam srautui  $f$  iš  $s$  į  $t$  galioja lygybė

$$W(f) = f(A, V \setminus A) - f(V \setminus A, A) \quad (2.17.4)$$

**Įrodymas.** (2.17.4) lygybės teisingumas išplaukia iš (2.17.1) ir (2.17.3) formulų.

Susumuokime (2.17.3) lygtis visoms grafo viršūnėms  $v \in A$ . Šią sumą sudarys nariai  $f(u, v)$ , turintys pliuso arba minuso ženklą.

Jei abi viršūnės  $u$  ir  $v$  priklauso aibei  $A$ , tai  $f(u, v)$  turės pliuso ženklą lygtyje  $\text{div}_f(u) = 0$  ir minuso ženklą lygtyje  $\text{div}_f(v)$ , ir šio dėmens neturės nei viena lygtis  $\text{div}_f(r) = 0$ , kai  $r \neq u$  ir  $r \neq v$ . Vadinas, sumuodami šiuos dėmenis, gausime 0.

Jei  $u \in A$ , o  $v \in V \setminus A$ , tai kiekvienas narys  $f(u, v)$  sumoje pasikartos vienintelį kartą su ženklu plius lygtyje  $\text{div}_f(u) = 0$ , ir visumoje gausime  $f(A, V \setminus A)$ . Analogiški dėmenys  $f(u, v)$ , kai  $u \in V \setminus A$ , o  $v \in A$ , duos (2.17.4) formulės narį  $f(V \setminus A, A)$ . Kita vertus, nagrinėjama suma lygi  $\text{div}_f(s) = W(f)$ , nes  $\text{div}_f(v) = 0$  visiems  $v \in A \setminus \{s\}$ .

**Išvada.**  $\text{div}_f(s) = -\text{div}_f(t)$ .

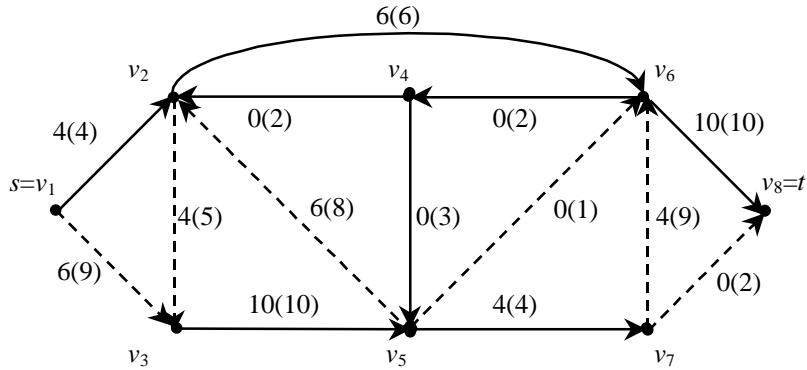
Imdami  $A = V \setminus \{t\}$  iš (2.17.4) formulės gausime:

$$\begin{aligned} \text{div}_f(s) &= W(f) = f(V \setminus \{t\}, \{t\}) - f(\{t\}, V \setminus \{t\}) = \\ &= -(f(\{t\}, V \setminus \{t\}) - f(V \setminus \{t\}, \{t\})) = -\text{div}_f(t). \end{aligned}$$

Gauta tapatybė įrodo intuityviai aiškų faktą: *nuotakio srautas yra lygus šaltinio srautui*.

Įrodyta lema sako tai, kad tinklo srautas gali būti matuojamas bet kuriame pjūvyje, kuris atskiria  $s$  ir  $t$  viršunes.

**Pavyzdys.** 2.17.1 pav. pavaizduotas tinklas. Skaičiai prie lankų apibrėžia srautą  $f$ . Lankų pralaidumą žymi skliausteliuose parašyti skaičiai. Panagrinėkime pjūvius, kuriuos nusako viršūnių  $A = \{v_1 = s, v_2, v_6, v_7\}$  ir  $B = \{v_1 = s, v_2, v_3\}$  poaibiai.



**2.17.1 pav.** Srautas tinkle: 1-osios lemos ir didinančiosios grandinės iliustracija. Punktyru išskirta didinančioji grandinė.

Aibės  $A$  viršūnių divergencija yra:

$$W(f) = \text{div}_f(v_1) = f(v_1, v_2) + f(v_1, v_3),$$

$$\text{div}_f(v_2) = f(v_2, v_3) + f(v_2, v_5) - f(v_1, v_2) - f(v_5, v_2) - f(v_4, v_2) = 0,$$

$$\text{div}_f(v_5) = f(v_6, v_8) + f(v_6, v_4) - f(v_2, v_6) - f(v_5, v_6) - f(v_7, v_6) = 0,$$

$$\text{div}_f(v_7) = f(v_7, v_6) + f(v_7, v_8) - f(v_5, v_7) = 0.$$

Šias lygtis sudėję, gausime:

$$\begin{aligned} & -W(f) + f(v_1, v_2) + f(v_1, v_3) + f(v_2, v_3) + f(v_2, v_6) - f(v_1, v_2) - \\ & - f(v_5, v_2) - f(v_4, v_2) + f(v_6, v_8) + f(v_6, v_4) - f(v_2, v_6) - \\ & - f(v_5, v_6) - f(v_7, v_6) + f(v_7, v_6) + f(v_7, v_8) - f(v_5, v_7) = 0. \end{aligned}$$

Sutraukę panašius narius, gausime:

$$\begin{aligned} W(f) &= f(v_1, v_3) + f(v_2, v_3) + f(v_6, v_8) + f(v_6, v_4) + f(v_7, v_8) \\ &\quad - f(v_5, v_2) - f(v_4, v_2) - f(v_5, v_6) - f(v_5, v_7) = \\ &= f(A, V \setminus A) - f(V \setminus A, A). \end{aligned}$$

Istatę srauto reikšmes, gausime:

$$W(f) = 6 + 4 + 10 + 0 + 0 - 6 - 0 - 0 - 4 = 10.$$

Pjūviui, kurį nusako poaibis  $B = \{v_1, v_2, v_3\}$ , analogiškai gausime:

$$W(f) = f(v_2, v_6) + f(v_3, v_5) - f(v_5, v_2) - f(v_4, v_2).$$

Istatę srauto reikšmes, gausime:

$$W(f) = 6 + 10 - 6 - 0 = 10.$$

Nesunku pastebeti, kad šie pavyzdžiai iliustruoja lemos įrodymą bei lemos išvados teisingumą.

**Apibrėžimas. Pjūvio**, kurį nusako viršunių poaibis  $A$ , t.y. pjūvio  $P(A)$  pralaidumas yra skaičius, apibrėžiamas formule

$$c(A, V \setminus A) = \sum_{e \in P(A)} c(e).$$

Kitaip tariant, pjūvio  $P(A)$  pralaidumas yra lygus pjūvių sudarančių lankų pralaidumų sumai.

**Apibrėžimas.** Minimaliu pjūviu vadinsime pjūvį  $P(A)$  ( $s \in A, t \in V \setminus A$ ), kurio **pralaidumas** yra **mažiausias** tarp visų galimų pjūvių, atskiriančių  $s$  ir  $t$  viršunes.

Vienas iš pagrindinių srautų tinkluose rezultatų yra nusakomas Fordo ir Falkersono teorema.

**Teorema** (Fordas (Ford L.R.) ir Falkersonas (Fulkerson D.R.) 1962). Bet kokio srauto iš viršunės  $s$  į viršunę  $t$  dydis neviršija minimalaus pjūvio, skiriančio šias viršunes, pralaidumo; be to egzistuoja tokis srautas, kurio dydis yra lygus tokio minimalaus pjūvio pralaidumui.

**Įrodymas.** Tarkime, kad  $P(A)$  – minimalus pjūvis. Remiantis lema, bet kokiam srautui  $f$  gausime:

$$\begin{aligned} W(f) &= f(A, V \setminus A) - f(V \setminus A, A) \leq f(A, V \setminus A) = \\ &= \sum_{e \in P(A)} f(e) \leq \sum_{e \in P(A)} c(e) = c(A, V \setminus A). \end{aligned}$$

Įrodymas, kad maksimalaus srauto dydis yra lygus minimalaus pjūvio pralaidumui, yra sudėtingesnis faktas. Šis įrodymas išplauks iš maksimalaus srauto konstravimo algoritmo, kuris nagrinėjamas kitame paragrale.

### 2.17.2. Maksimalaus srauto konstravimo algoritmas

Visi žinomi maksimalaus srauto konstravimo algoritmai pagrįsti nuosekliu srauto didinimo metodu, o visų srauto didinimo metodų pagrindą sudaro **didinančių grandinių** teorija.

**Apibrėžimas.** Tinklo  $S$  lankas  $e$  iš viršunės  $u$  į viršunę  $v$  vadinas **leistinuoju** lanku srauto  $f$  atžvilgiu, jeigu

- a)  $e = (u, v)$  ir  $f(e) < c(e)$
- arba
- b)  $e = (v, u)$  ir  $f(e) > 0$ .

Jei galioja salyga a), tai tokį lanką vadinsime **suderintuoju** (soglasovannaja duga). Jei galioja salyga b), tai tokį lanką vadinsime **nesuderintuoju** (nesoglasovannaja duga).

**Apibrėžimas.** Ilgio  $l$  didinančioji grandinė iš viršūnės  $s$  į viršūnę  $t$  srauto  $f$  atžvilgiu yra seka, sudaryta iš besikeičiančia tvarka surašytų (poromis skirtingu) viršūnių ir lankų:

$$v_0, e_1, v_1, e_2, v_2, \dots, v_{l-1}, e_l, v_l, \quad (2.17.5)$$

Čia  $v_0 = s$ ,  $v_l = t$  ir kiekvienam  $1 \leq i \leq l$  lankas  $e_i$  srauto  $f$  atžvilgiu yra leistinasis lankas iš viršūnės  $v_{i-1}$  į viršūnę  $v_i$ .

Pavyzdžiu, 2.17.1 pav. pavaizduoto tinklo ir srauto per jį atžvilgiu, grandinė:

$$\begin{aligned} &v_1, (v_1, v_3), v_3, (v_2, v_3), v_2, (v_5, v_2), v_5, \\ &(v_5, v_6), v_6, (v_7, v_6), v_7, (v_7, v_8), v_8, \end{aligned} \quad (2.17.6)$$

yra didinančioji grandinė, kurios ilgis yra 6.

Žinodami (2.17.5) pavidalo didinančiąją grandinę, srautą  $f$  galime padidinti dydžiu

$$\delta = \min \{ \Delta(e_i), 1 \leq i \leq l \}$$

čia

$$\Delta(e_i) = \begin{cases} c(e_i) - f(e_i), & \text{jei } e_i \text{ yra suderintasis lankas,} \\ f(e_i), & \text{jei } e_i \text{ yra nesuderintasis lankas.} \end{cases}$$

Tam tikslui (pavyzdžiu, žiūrint į 2.17.1 pav.) reikia didinančiosios grandinės kiekvieno sederintojo lanko srautą padidinti dydžiu  $\delta$ , o kiekvieno nesuderintojo lanko srautą sumažinti tuo pačiu dydžiu  $\delta$ , t.y.

$$f'(e_i) = \begin{cases} f(e_i) + \delta, & \text{jei } e_i \text{ yra didinančiosios grandinės} \\ & \text{suderintasis lankas,} \\ f(e_i) - \delta, & \text{jei } e_i \text{ yra didinančiosios grandinės} \\ & \text{nesuderintasis lankas,} \\ f(e_i), & \text{jei } e_i \text{ neprisklauso didinančiajai} \\ & \text{grandinei.} \end{cases}$$

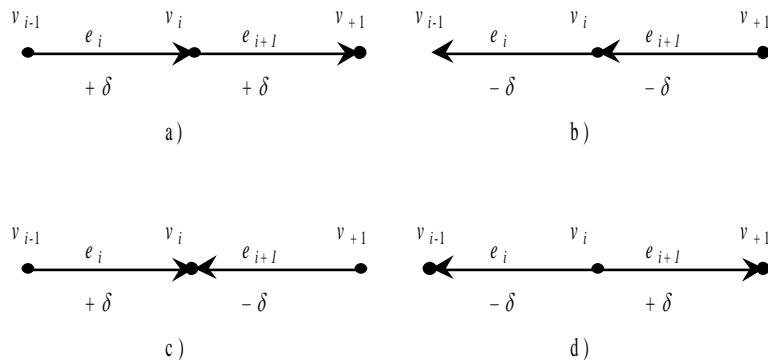
Aišku, kad taip apskaičiuota funkcija  $f'$  yra srautas, nes  $0 \leq f'(e) \leq c(e)$ ,  $e \in U$ ; ir  $\operatorname{div}_{f'}(v_i) = 0$ ,  $v_i \in V \setminus \{s, t\}$ .

Iš tikro, jei  $v_i$  nepriklauso didinančiajai grandinei, tai  $\operatorname{div}_f(v_i) = \operatorname{div}_{f'}(v_i) = 0$ .

Tarkime, kad  $v_i$  priklauso didinančiajai grandinei. Tada (žr. 2.17.2 pav.):

- 1) jei  $e_i$  ir  $e_{i+1}$  yra sederintieji lankai, tai srautas, įtekantis į viršūnę  $v_i$  ir ištekantis iš jos padidėja tuo pačiu dydžiu  $\delta$ ; tuo būdu  $\operatorname{div}_{f'}(v_i) = 0$  (žr. 2.17.2 a) pav.),

- 2) jei  $e_i$  ir  $e_{i+1}$  yra nesuderintieji lankai, tai srautas, įtekantis ir ištekantis iš viršūnės  $v_i$  sumažėja tuo pačiu dydžiu  $\delta$ ; vadinasi  $\text{div}_{f'}(v_i) = 0$  (žr. 2.17.2 b) pav.),
- 3) jei  $e_i$  yra sederintasis lankas, o  $e_{i+1}$  – nesuderintasis lankas, tai srautas įeinančiu į viršūnę  $v_i$  lanku  $e_i$  padidėja dydžiu  $\delta$ , o įeinančiu lanku  $e_{i+1}$  sumažėja tuo pačiu dydžiu; vadinasi  $\text{div}_{f'}(v_i) = 0$  (žr. 2.17.2 c) pav.),
- 4) jei  $e_i$  yra nesuderintasis lankas, o  $e_{i+1}$  – sederintasis lankas, tai išeinančiu iš viršūnės  $v_i$  lanku  $e_i$  srautas sumažėja dydžiu  $\delta$ , o išeinančiu iš viršūnės lanku  $e_{i+1}$  padidėja tuo pačiu dydžiu; tuo būdu  $\text{div}_{f'}(v_i) = 0$  (žr. 2.17.2 d) pav.).



**2.17.2 pav.** Didinančiosios grandinės lankų, incidentiškų viršūnei  $v_i$ , srauto perskaičiavimas

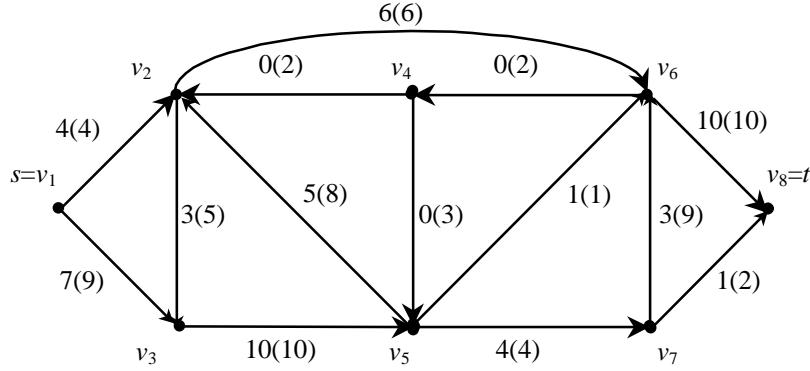
Srauto  $f'$  dydis padidėja dėmeniu  $\delta$ :

$$W(f') = \text{div}_{f'}(s) = \text{div}_f(s) + \delta = W(f) + \delta.$$

**Pavyzdys.** Panagrinėkime tinklą su srautu  $f$ , kuris pavaizduotas 2.17.1 pav. (2.17.6) seká yra šio tinklo ir srauto per jį atžvilgiu didinančioji grandinė. Šiai grandinei

$$\delta = \min\{\Delta(e_i) \mid 1 \leq i \leq 6\} = 1.$$

Vadinasi, 2. 17.1 pav. pavaizduoto tinklo srautą galima padidinti dydžiu 1, t.y. nuo 10 iki 11. 3.17.3 pav. pavaizduotas tas pats 2.17.1 pav. tinklas ir perskaičiuotas srautas.



**2.17.3 pav.** Tinklo, pavaizduoto 2.17.1 pav., srautas po modifikacijos

Įrodysime teoremą, kuri pagrindžia maksimalaus srauto apskaičiavimo, naudojant didinacijas grandines, metodą.

**Teorema.** Žemiau pateiktos trys sąlygos yra ekvivalenčios:

- 1) srautas iš  $s \downarrow t$  maksimalus,
- 2) neegzistuoja didinančios grandinės srauto  $f$  atžvilgiu,
- 3) egzistuoja pjūvis, kurį nusako toks viršūnių poaibis  $A$  ( $A \subset V$ ), skiriantis viršunes  $s$  ir  $t$  ( $s \in A$ ,  $t \in V \setminus A$ ), kad  $W(f) = c(A, V \setminus A)$ .

**Įrodymas.** 1)  $\Rightarrow$  2). Jei srautas maksimalus, tai aišku, kad neegzistuoja didinančios grandinės šio srauto atžvilgiu, nes, priešingu atveju, esant didinancijai grandinei, srautą būtų galima padidinti.

2)  $\Rightarrow$  3). Tarkime, kad srautui  $f$  neegzistuoja didinančiosios grandinės. Apibrėžkime viršūnių aibę  $A$  ( $A \subseteq V$ ), kurią sudaro viršūnės tokios grandinės

$$v_0, e_1, v_1, e_2, v_2, \dots, v_{k-1}, e_k, v_k,$$

kad  $k \geq 0$ ,  $v_0 = s$ ,  $v_k = t$  ir  $e_i$  – leistinasis lankas iš viršūnės  $v_{i-1}$  į viršūnę  $v_i$ ,  $1 \leq i \leq k$ . Aišku, kad  $s \in A$ , o  $t \notin A$ , nes šiuo atveju egzistotų didinančioji grandinė srauto  $f$  atžvilgiu. Panagrinėkime pjūvį  $P(A)$ . Iš aibės  $A$  ir sudegintojo lanko apibrėžimo išplaukia, kad kiekvienam pjūvio lankui turi galioti lygybę  $f(e) = c(e)$ . Vadinas,  $f(A, V \setminus A) = c(A, V \setminus A)$ .

Analogiškai, remiantis aibės  $A$  ir nesudegintojo lanko apibrėžimu, gausime, kad  $f(V \setminus A, A) = 0$ . Remiantis aukščiau įrodyta lema, gausime

$$W(f) = f(A, V \setminus A) - f(V \setminus A, A) = c(A, V \setminus A).$$

3  $\Rightarrow$  1) išplaukia iš to faktu, kad srauto dydis nevišja  $c(A, V \setminus A)$ .

**Pavyzdys.** Nesunku patikrinti, kad srautas 2.17.3 pav. yra maksimalus. „Prisotintas“ pjūvis  $P(A)$ , atsirandantis teoremos įrodyme, yra  $A = \{v_1, v_2, v_3, v_5\}$ .

Remiantis pateiktais teoriniais samprotavimais, galima sudaryti paprastą maksimalaus srauto apskaičiavimo algoritmą: pradėdami nuo bet koksio, pavyzdžiui, nulinio srauto ( $f(e) = 0$ ,  $e \in U$ ), ieškosime didinančiosios grandinės ir, jei tokia grandinė egzistuoja, srautą didinsime; priešingu atveju srautas  $f$  yra maksimalus.

Tačiau čia iškyla klausimas, ar algoritmo žingsnių skaičius yra baigtinis. Fordas ir Falkersonas<sup>8</sup> davė neigiamą atsakymą. 1962 m. jie pateikė pavyzdį tokio tinklo, kuriame galima taip parinkinėti didinančiasias grandines, kad procesas niekada nepasibaigtų. Be to, srauto dydis visą laiką bus ketvirtadaliu mažesnis už maksimalaus srauto dydį.

Tačiau 1972 m. Edmondsas (Edmonds J.) ir Karpas (Karp R.M.)<sup>9</sup> įrodė, kad, jei kiekviename žingsnyje srautą didinsime trumpiausios didinančiosios grandinės atžvilgiu, tai maksimalus srautas bus sukonstruotas, panaudojant ne daugiau kaip  $m \cdot n / 2$  grandinių (čia  $n = |V|$ ,  $m = |U|$ ). Rasti trumpiausią didinančiąją grandinę patogu, naudojant paiešką platyn (žr. 2.9 paragrafą). Ivertinant tai, kad trumpiausios grandinės konstravimo algoritmo sudėtingumas yra  $O(m+n)$ , galime ivertinti viso maksimalaus srauto apskaičiavimo algoritmo sudėtingumą, –  $O(mn(m+n))$ .

Čia nenagrinėsime šio algoritmo detalių, kadangi toliau panagrinėsime efektyvesnį 1970 m. Dinico (Dinic E.A.)<sup>10</sup> pasiūlytą maksimalaus srauto apskaičiavimo algoritmą, kurio sudėtingumas yra  $O(n^3)$ .

Dinico metodas susideda iš fazų. Jo efektyvumas pagrįstas tuo, kad, nepriklausomai nuo tinklo lankų pralaidumo, fazų skaičius niekada neviršija  $(n-1)$ , čia  $n$  – tinklo viršunių skaičius [Lip88].

---

<sup>8</sup> Ford L.R., Fulkerson D.R. Flows in networks. Princeton Univ. Press, Princeton, N.J. 1962. Yra vertimas į rusų kalbą. Ford L.R., Falkerson D.R. Potoki v setiach., Moskva: Mir, 1966.

<sup>9</sup> Edmonds J., Karp R.M. Theoretical improvements in algorithmic efficiency for network flow problems. J.ACM, 1972, 19, p.p. 248-264.

<sup>10</sup> Dinic E.A. Algoritm rešenija zadači o maksimalnom potoke v seti so stepennoj ocenkoj. Dokl. Akademii nauk SSSR, serija Mat. Fiz., 1970, 194, 4, s. 754-757.

**Panagrinėkime  $k$ -osios ( $k \leq n - 1$ ) fazės veiksmus.** Tarkime, kad fazės pradžioje turime tinklą  $G$  ir Jame apibrėžtą srautą  $f$ . Tokį tinklą žymėsime simboliu  $G_f$ .

Remdamiesi tinklu  $G_f$ , sudarome pagalbinį tinklą, kuris neturi kontūrų (ciklų) ir kurio struktūra vaizduoja visas trumpiausias didinančiasias tinklo  $G_f$  grandines. Pažymėkime šį tinklą simboliu  $S_f$ . Tinklas  $S_f$  konstruojamas paieškos platyn grafe  $G_f$  metodu, ir į tinklą  $S_f$  įtraukiame tinklo  $G_f$  srauto  $f$  atžvilgiu leistinieji lankai. Tuo būdu į tinklą  $S_f$  įeina šaltinis  $s$ , nuotakis  $t$  ir grafo  $G_f$  leistinieji lankai  $(u, v)$ , čia viršunė  $u$  nutolusi nuo šaltinio  $s$  atstumu  $d$ , o viršunė  $v$  – atstumu  $d+1$ ,  $0 \leq d \leq l$ , o  $l$  yra tinklo  $S_f$  ilgis, t.y. atstumas nuo  $s$  iki  $t$  grafe  $G_f$ . Šio lanko pralaidumą žymėsime  $c_f(u, v)$  ir apibrėšime formule

$$c_f(u, v) = \begin{cases} c(u, v) - f(u, v), & \text{jei } (u, v) \text{ yra suderintasis leistinas } G_f \text{ lankas,} \\ f(v, u), & \text{jei } (u, v) \text{ yra nesuderintasis leistinas } G_f \text{ lankas,} \\ c(u, v) - f(u, v) + f(v, u), & \text{jei lankas } (u, v) \text{ vienu metu ir} \\ & \text{suderintasis, ir nesuderintasis leistinas } G_f \text{ lankas.} \end{cases}$$

Sudarius pagalbinį tinklą  $S_f$ , toliau Jame apskaičiuojamas **pseudomaksimalus srautas**.

**Apibrėžimas.** Ilgio  $l$  tinklo  $S_f$  pseudomaksimalus srautas – tai toks tinklo  $S_f$  srautas  $f^*$ , prie kurio tinkle  $S_f$  neegzistuoja nei viena  $l$  ilgio didinančioji grandinė; kitaip tariant, bet kokiam tinklo  $S_f$  iš viršūnės  $s$  į viršūnę  $t$  keliui

$$v_0, v_1, \dots, v_l \quad (v_0 = s, v_l = t)$$

egzistuoja toks lankas  $(v_i, v_{i+1})$ ,  $0 \leq i < l$ , kad  $f^*(v_i, v_{i+1}) = c_f(v_i, v_{i+1})$ .

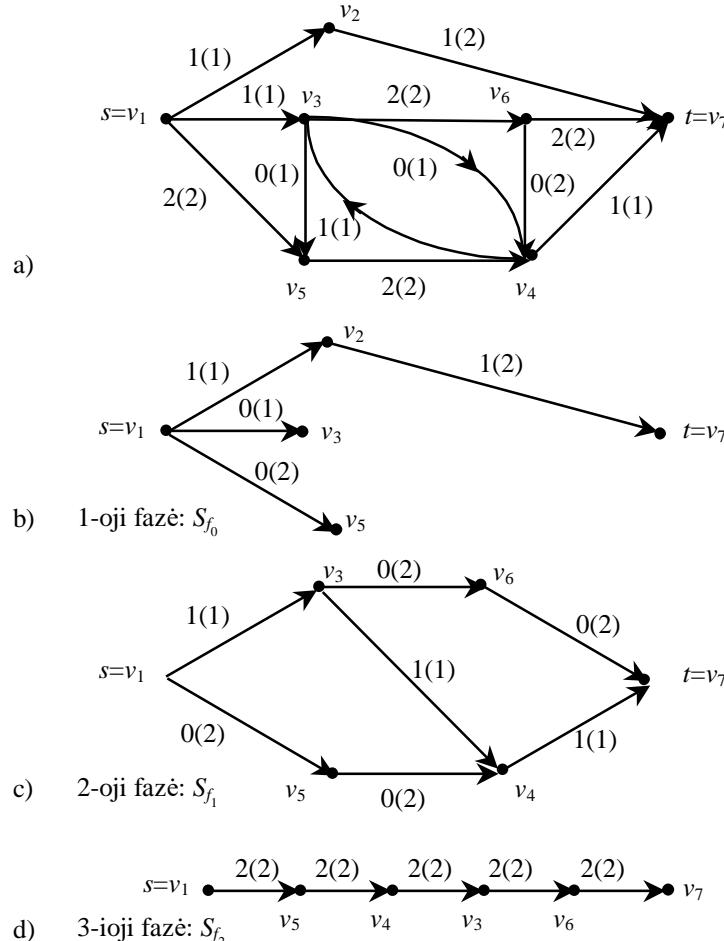
Po to pseudomaksimalus srautas “perkeliamas” į pagrindinį tinklą  $G_f$ : srautas  $f^*(u, v)$  sumuojamas su  $G_f$  srautu  $f(u, v)$ ; jei ši suma iššaukia lanko  $(u, v)$  “persipildymą”, t.y. suma viršija  $c(u, v)$ , tai šis srauto “perteklius” kompensuojamas sumažinant srautą  $f(v, u)$ . Tuo būdu viršūnės  $u$  divergencija  $\text{div}_f(u)$  padidėja dydžiu  $f^*(u, v)$ , o viršūnės  $v$  divergencija  $\text{div}_f(v)$  tuo pačiu dydžiu sumažėja. Nesunku įsitikinti, kad tokią visų lankų srautų modifikacija apibrėžia naują tinklo  $G_f$  srautą  $f'$ , kurio dydis  $W(f') = W(f) + W(f^*)$ . Šiuo veiksmu ir baigiamo k-oji fazė.

Maksimalaus srauto konstravimas baigiamas, kai iš tinklo  $G_f$  nebegalima sukonstruoti pagalbinio tinklo  $S_f$ .

Kaip bus matyti iš toliau pateiktų tinklo  $S_f$  bei jo pseudomaksimalaus srauto apskaičiavimo algoritmu, vienos fazės sudėtingumas yra  $O(n^2)$ .

Kadangi, kaip parodyta literatūroje [Lip88], fazinių skaičių neviršija  $(n-1)$ , tai Dinico metodo sudėtingumas yra  $O(n^3)$ .

**Pavyzdys.** Panagrinėkime tinklą, pavaizduotą 2.17.4 a) paveiksle. Prie lanko parašytas skaičius reiškia lanko maksimalų srautą, o skaičius skliausteliuose rodo lanko pralaidumą.



**2.17.4 pav.** Maksimalaus srauto ieškojimas Dinico metodu

Pradėję nuo tinklo nulinio srauto  $f_0$ , pirmiausia apskaičiuojame pagalbinį be kontūrų tinklą  $S_{f_0}$  (žr. 2.17.4 b) pav.). Po to rastajį pseudomaksimalų srautą perkeliame į tinklą  $G_f$ , gaudami srautą  $f_1$ , t.y. tinklą  $G_{f_1}$ .

Toliau konstruojame pagalbinį tinklą  $S_{f_1}$  (žr. 2.17.4 c) pav.), ir rastajį pseudomaksimalų srautą perkeliame į tinklą  $G_{f_1}$ . Gauname srautą  $f_2$  (tinklą  $G_{f_2}$ ).

Paskutinėje fazėje konstruojame pagalbinį tinklą  $S_{f_2}$  (žr. 2.17.4 d) pav.). Kai šio tinklo pseudomaksimalų srautą perkeliame į  $G_{f_2}$ , gauname srautą  $f$ , pavaizduotą 2.17.4 a) pav., kuris yra maksimalus tinklo  $G$  srautas, kadangi  $W(f) = 4 = c(\{s\}, V \setminus \{s\})$  (žr. 2.17.1 paragrafo Fordo ir Falkersono teorema).

2.17.4 c) pav. gerai iliustruoja tą faktą, kad pseudomaksimalus srautas nebūtinai yra maksimalus srautas. Iš tikro, šiame paveikslėlyje pavaizduotą pseudomaksimalų srautą galime padidinti panaudoję ilgio 5 didinančiąją grandinę:

$$v_1, (v_1, v_5), v_5, (v_5, v_4), v_4, (v_3, v_4), v_3, (v_3, v_6), v_6, (v_6, v_7), v_7.$$

Taip pat pažymėsime, kad tinklo  $S_{f_2}$  (žr. 2.17.4 d) pav.) lankas  $(v_4, v_3)$  atsirado iš pagrindinio tinklo  $G$  dvių priešpriešais nukreiptų lankų.

Aptarkime pagalbinio tinklo  $S_f$  ir jo pseudomaksimalaus srauto apskaičiavimo algoritmus.

#### ***Pagalbinio tinklo $S_f$ apskaičiavimo algoritmas***

Tarkime, kad pradinio tinklo grafas nusakytas gretumumo struktūromis  $N[v]$  ir  $PN[v]$ ,  $v \in V$ .

$N[v]$  – tai aibė viršūnių, į kuriuos iš viršūnės  $v$  išeina lankai.

$PN[v]$  – tai aibė viršūnių, iš kurių į viršūnę  $v$  įeina lankai.

Lankų pralaidumą žymėsime masyvu  $c[u, v]$ ,  $u, v \in V$ , o faktinają srautą  $f$  – masyvu  $f[u, v]$ ,  $u, v \in V$ . Tokia pralaidumo bei srauto vaizdavimo struktūra atminties požiūriu néra racionali, tačiau prie šios struktūros algoritmas tampa vaizdesnis.

Kintamieji, vaizduojantys pagalbinį tinklą  $S_f$ , yra analogiški pradinio tinklo kintamiesiems ir jie žymimi tais pačiais vardais, pridedant prie jų pradinę raidę  $S$ . Tuo būdu,  $SV$  žymi tinklo  $S_f$  viršūnių aibę;  $SN[v]$  ir  $SPN[v]$ ,  $v \in SV$ , žymi  $S_f$  gretumumo struktūrą;  $Sc$  ir  $Sf$  atitinkamai žymi  $S_f$  lankų

pralaidumas ir srautą. Masyvo  $d$  elementas  $d[v]$ ,  $v \in SV$ , žymi tinklo  $S_f$  viršūnės  $v$  nuotoli nuo šaltinio  $s$ .

Kaip buvo minėta aukščiau, tinklas  $S_f$  konstruojamas paieškos platyn iš viršūnės  $s$  tinkle  $G_f$  metodu. Paieškos platyn procese aplankytos viršūnės  $v$  talpinamos į eilę. Kiekviena iš eilės paimta viršūnė  $u$  nagrinėjama du kartus: pirmą kartą – ieškomi iš jos išeinantys leistinieji suderintieji lankai  $(u, v)$ ; antrą kartą – ieškomi leistinieji nesuderintieji lankai  $(v, u)$ .

Jei tinkle  $G_f$  vienu metu lankas  $(u, v)$  yra sederintasis ir nesuderintasis leistinasis lankas, t.y.  $(u, v)$  – leistinasis sederintasis lankas, o  $(v, u)$  – leistinasis nesuderintasis lankas, tai į tinkle  $S_f$  itraukto lanko  $(u, v)$  pralaidumas yra lygus šių lankų pralaidumų srauto  $f$  atžvilgiu sumai.

```

procedure psa;
{Pagalbinio be kontūrų tinklo  $S_f$  konstravimas.
Kintamieji  $V, N, PN, c, f, SV, SN, SPN, Sc, Sf, d, s, t$  – globalieji}
begin
    for  $u \in V$  do {inicjalizacija}
        begin
             $d[u] := \infty;$ 
             $SN[u] := \emptyset; SPN[u] := \emptyset;$ 
            for  $v \in V$  do  $Sc[u, v] := 0;$ 
            for  $v \in V$  do  $Sf[u, v] := 0;$  {nulinio srauto inicializacija}
        end;
    Eilė :=  $\emptyset; SV := \emptyset; d[s] := 0;$ 
    {Paieška platyn iš šaltinio  $s$ }
    Eilė  $\Leftarrow s;$ 
    while  $Eilė \neq \emptyset$  do
        begin
             $u \Leftarrow Eilė;$ 
             $SV := SV \cup \{u\};$ 
            for  $v \in N[u]$  do {suderintujų lankų iš viršūnės  $u$  paieška}
                if ( $d[u] < d[v] \leq d[t]$ ) and ( $f[u, v] < c[u, v]$ ) then
                    begin
                        if  $d[v] = \infty$  then {viršūnė  $v$  – nauja}  $Eilė \Leftarrow v;$ 
                         $d[v] := d[u] + 1;$ 
                        {Laukas  $(u, v)$  įjungiamas į tinkle  $S_f$ }
                         $SN[u] := SN[u] \cup \{v\};$ 
                    end;
            end;
        end;
    end;

```

```

 $SPN[v] := SPN[v] \cup \{u\};$ 
 $Sc[u, v] := c[u, v] - f[u, v];$ 
end;
for  $v \in PN[u]$  do {nesuderintųjų lankų paieška}
if ( $d[u] < d[v] \leq d[t]$ ) and ( $f[v, u] > 0$ ) then
begin
if  $d[v] = \infty$  then {viršūnė  $v$  – nauja} Eilė  $\Leftarrow v$ ;
 $d[v] := d[u] + 1$ ;
if  $Sc[u, v] = 0$  then {Lankas  $(u, v)$  dar neįjungtas į tinklą}
Lanką  $(u, v)$  įtrauktī į tinklą}
begin
 $SN[u] := SN[u] \cup \{v\}$ ;
 $SPN[v] := SPN[v] \cup \{u\}$ ;
end;
 $Sc[u, v] := Sc[u, v] + f[v, u]$ ;
end;
end; {while}
end; {psa}.

```

Tarkime, po procedūros *psa* įvykdymo  $d[t] = l$ . Jei  $l = \infty$ , tai reiškia, kad paieškos platyn metu nuotakio viršūnė  $t$  nebuvo pasiekta. Vadinasi, tinkle  $G_f$  nuo  $s$  iki  $t$  nėra didinančiosios grandinės. Tada, remiantis 2.17.2 paragrafo teorema, galime teigti, kad srautas  $f$  yra maksimalus.

Jei  $l < \infty$ , tai trumpiausios didinančiosios grandinės iš  $s$  į  $t$  ilgis yra  $l$ . Be to, visų kelių nuo  $s$  iki  $t$  ilgiai tinkle  $S_f$  yra lygūs  $l$ , ir jie žymi  $l$  ilgio didinančiasias grandines tinkle  $G_f$ . Reikia pažymėti, kad, pasiekus nuotakį  $t$ ,  $d[t]$  tampa lygus  $l < \infty$ , ir viršūnės  $v$ , kurioms  $d[v] \geq l$ , procedūroje *psa* nenagrinėjamos, kadangi nei viena tokia viršūnė negali priklausyti  $l$  ilgio keliui, jungiančiam  $s$  ir  $t$ .

Nesunku įvertinti procedūros *psa* sudėtingumą. Kiekviena viršūnė  $v$  į eilę talpinama ir šalinama ne daugiau kaip vieną kartą. Kiekvienas viršūnei  $v$  incidentiškas lankas analizuojamas vieną kartą, be to, analizės veiksmų skaičius apribotas konstanta. Vadinasi, bendras veiksmų skaičius be inicializacijos yra  $O(n+m)$ . Inicializacijos veiksmų skaičius yra  $O(n^2) > O(n+m)$ . Tuo būdu, procedūros *psa* sudėtingumas yra  $O(n^2)$ .

Belieka aptarti tinklelo  $S_f$  pseudomaksimalaus srauto efektyvaus apskaičiavimo algoritmą. Žemiau pateiktas Malchotros, Kumaro ir Mahešvario

(Malhotra V.M., Kumar P.M., Makshvari S.N. An  $O(n^3)$  algorithm for finding maximum flows in networks. Information Processing Lett. 1978, 7, s. 277 – 278) pseudomaksimalus srauto apskaičiavimo algoritmą, kurio sudėtingumas yra  $O(n^2)$ .

#### **Tinklo $S_f$ pseudomaksimalus srauto apskaičiavimo procedūra maxpsa**

Šio algoritmo aprašymui įveskime tinklo **viršūnės potencialo** sąvoką.

**Apibrėžimas.** Tinklo viršūnės  $v$  potencialas yra maksimalus srauto kiekis, kurį galima praleisti per viršūnę  $v$ . Viršūnės  $v$  potencialas  $P(v)$  apibrėžiamas formule

$$P(v) = \min(Pin(v), Pout(v)),$$

$$Pin(v) = \begin{cases} \sum_{u: u \rightarrow v} c(u, v), & \text{jei } v \neq s, \\ \infty, & \text{jei } v = s, \end{cases}$$

$$Pout(v) = \begin{cases} \sum_{u: v \rightarrow u} c(v, u), & \text{jei } v \neq t, \\ \infty, & \text{jei } v = t. \end{cases}$$

Žemiau pateikta **maxpsa** procedūra pirmiausia apskaičiuoja pagalbinio tinklo  $S_f$  visų viršūnių potencialus ir nulinio potencijalo viršūnes patalpina į steką STEK. Aišku, kad nulinio potencijalo viršūnes drauge su joms incidentiškais lankais galima pašalinti iš tinklo  $S_f$ , ir šis pašalinimas neturės įtakos jokiam tinklo  $S_f$  srautui. Tokių viršūnių, o tiksliau joms incidentiškų lankų pašalinimas gali paveikti kitų tinklo  $S_f$  viršūnių potencialus, ir jie gali tapti lygūs nuliui. Jei taip atsitinka, tai naujos nulinio potencijalo viršūnės šalinamos iš tinklo.

Jei, pasibaigus nulinio potencijalo viršūnių šalinimo procesui, tinklas  $S_f$  neturi nenulinio potencijalo viršūnių, tai tinklo  $S_f$  pseudomaksimalus srautas surastas. Priešingu atveju randame mažiausio potencijalo viršūnę  $r$  ir jos potencialą  $p$ . Tada iš viršūnės  $r$  į viršūnę  $t$  ir iš viršūnės  $s$  į viršūnę  $r$  konstruojame  $p$  dydžio srautą. Tuo būdu apskaičiuojamas  $p$  dydžio srautas iš saltinio  $s$  į nuotaką  $t$ . Šis srautas naudos tik leistinuosius suderintuosius lankus.

Aptarsime, kaip konstruojamas  $p$  dydžio srautas iš viršūnės  $r$  į viršūnę  $t$ . Srauto iš viršūnės  $s$  į viršūnę  $r$  konstravimas yra analogiškas.

Kaip buvo minėta,  $p$  dydžio srautas iš viršūnės  $r$  į viršūnę  $t$  naudos tik suderintuosius leistinuosius lankus, ir ji patogiausia konstruoti naudojant paiešką platyn iš viršūnės  $r$ . Įsivaizduokime, kad viršūnėje  $r$  patalpintas

krovinys  $Q[r] = p$ , ir mes norime “perkelti” šį krovinį į viršūnę  $t$ . Kiekviena paieškos platyn metu aplankyta viršūnė  $v$  “perkelia” krovinį  $Q[v]$  į viršūnes  $u$ , į kurias eina lankai iš viršūnės  $v$ . Be to, į viršūnę  $u$  “perkeliamo” krovonio dalis yra lygi lanko  $(v, u)$  pralaidumui, t.y.  $Sf[v, u] = Sc[v, u]$ . Aišku, kad tokie lankai gali būti šalinami iš tinklo. Lankų šalinimas iššaukia viršūnių potencialų perskaiciavimą, ir, jei viršūnės  $v$  potencialas tampa lygus nuliui, tai viršūnė  $v$  talpinama į steką STEK.

Tuo būdu, krovinys  $Q[r] = p$  iš viršūnės  $r$ , kuri nuo šaltinio  $s$  nutolusi atstumu  $d$ , pirmiausia bus “perkeltas” į viršūnes, nutolusias nuo šaltinio  $s$  atstumu  $d+1$ , po to, iš šių viršūnių, į viršūnes, nutolusias nuo šaltinio  $s$  atstumu  $d+2$  ir t.t. iki visas krovinys bus “perkeltas” į viršūnę  $t$ . Kadangi  $p$  yra mažiausias viršūnės potencialas, tai toks krovonio “perkėlimas” visada yra galimas.

Kaip minėjome, srautas iš  $s$  į  $r$  konstruojamas analogiškai.

$p$  dydžio srauto iš viršūnės  $s$  į  $t$  konstravimu baigiasi pagrindinio ciklo veiksmai, ir, kaip buvo minėta aukščiau, pagrindinis ciklas bus kartojamas, kol tinklas  $S_f$  turės nenulinio potencijalo viršūnių.

```

procedure maxpsa;
{Pseudomaksimalaus srauto pagalbiname tinkle  $S_f$  apskaičiavimas
Malchotros, Kumaro ir Mahešvaro metodu.
Kintamieji SV, SN, SPN, Sc, Sf, s ir t yra globalieji.}
begin
    STEK := ∅; { Steke STEK bus saugomas nulinio potencijalo viršūnės. }
    for v ∈ SV do {viršūnės v potencijalo apskaičiavimas}
        begin
            Pin [v] := 0; Pout [v] := 0;
            if v = s then Pin [v] := ∞
                else
                    for u ∈ SPN [v] do
                        Pin [v] := Pin [v] + Sc [u, v];
            if v = t then Pout [v] := ∞
                else
                    for u ∈ SN [v] do
                        Pout [v] := Pout [v] + Sc [v, u];
            P [v] := min (Pin [v], Pout [v]);
            if P [v] = 0 then STEK ← v;
        end;
    for v ∈ SV do Q [v] := 0; { Krovinių inicializacija tinklo  $S_f$  viršūnėse }

```

```

 $XN := SV; \{ Aibėje XN bus saugomas tinklo S_f nenulinio potencialo viršūnės \}$ 
 $while XN \neq \emptyset do \{ Pagrindinis ciklas \}$ 
 $begin$ 
 $\{ Nulinio potencialo viršūnių šalinimas \}$ 
 $while STEK \neq \emptyset do$ 
 $begin$ 
 $v \leftarrow STEK; XN := XN \setminus \{v\};$ 
 $\{ Lanką, įeinančią i viršūnę v, šalinimas \}$ 
 $for u \in SPN[v] do \{ Lanko (u, v) šalinimas \}$ 
 $begin$ 
 $Pout[u] := Pout[u] - (Sc[u, v] - Sf[u, v]);$ 
 $SN[u] := SN[u] \setminus \{v\};$ 
 $SPN[v] := SPN[v] \setminus \{u\};$ 
 $if P[u] \neq 0 then \{ Potencialo P[u] modifikacija \}$ 
 $begin$ 
 $P[u] := min(Pin[u], Pout[u]);$ 
 $if P[u] = 0 then STEK \leftarrow u;$ 
 $end;$ 
 $end; \{ for \}$ 
 $\{ Lanką, išeinančią iš viršūnės v, šalinimas \}$ 
 $for u \in SN[v] do \{ Lanko (v, u) šalinimas \}$ 
 $begin$ 
 $Pin[u] := Pin[u] - (Sc[v, u] - Sf[v, u]);$ 
 $SPN[u] := SPN[u] \setminus \{v\};$ 
 $SN[v] := SN[v] \setminus \{u\};$ 
 $if P[u] \neq 0 then \{ P[u] modifikacija \}$ 
 $begin$ 
 $P[u] := min(Pin[u], Pout[u]);$ 
 $if P[u] = 0 then STEK \leftarrow u;$ 
 $end;$ 
 $end; \{ for \}$ 
 $end; \{ while STEK \neq \emptyset \}$ 
 $\{ XN - tai nenulinio potencialo viršūnių aibė \}$ 
 $if XN \neq \emptyset then \{ Srautas dar nepseudomaksimalus \}$ 
 $begin$ 
 $\{ Minimalaus potencialo viršūnės r apskaičiavimas \}$ 
 $p := \infty;$ 
 $for v \in XN do$ 
 $if P[v] < p then$ 
 $begin$ 

```

```

        r := v;
        p := P [r];
        end;
{ Dydžio p srauto iš viršūnės r į viršūnę t konstravimas }
Eilė := ∅; Eilė := r; Q [r] := p;
repeat
    v ⇐ Eilė;
    Pin [v] := Pin [v] - Q [v];
    Pout [v] := Pout [v] - Q [v];
    P [v] := P [v] - Q [v];
    if P [v] = 0 then STEK ⇐ v;
    if v = t then Q [v] := p
        else
            begin { Viršūnės v "iškrova" }
            u := pirmoji aibės SN [v] viršūnė;
            while Q [v] > 0 do
                begin
                    if Q [u] = 0 then Eilė := u;
                    delta := min (Q [v], Sc [v, u] - Sf [v, u]);
                    Sf [v, u] := Sf [v, u] + delta;
                    Q [v] := Q [v] - delta;
                    Q [u] := Q [u] + delta;
                    if Sf [v, u] = Sc [v, u] then { Lanko (v, u)
                        šalinimas }
                        begin
                            SN [v] := SN [v] \ {u};
                            SPN [u] := SPN [u] \ {v};
                        end;
                    if Q [v] > 0 then
                        u := po viršūnės u einanti kita aibės SN [v]
                        viršūnė;
                    end; { while Q [v] > 0 }
                end; { viršūnės v "iškrovos" pabaiga }
            end; { srautas iš viršūnės r į t surastas }

{ Dydžio p srauto iš šaltingo s į viršūnę r konstaravimas }
Eilė := ∅; Eilė := r; Q [r] := p;
repeat
    v ⇐ Eilė;

```

```

if  $v \neq r$  then {  $P[v]$  dar nesumažintas }
begin
     $Pin[v] := Pin[v] - Q[v];$ 
     $Pout[v] := Pout[v] - Q[v];$ 
     $P[v] := P[v] - Q[v];$ 
    if  $P[v] = 0$  then STEK  $\Leftarrow v;$ 
end;
if  $v = s$  then  $Q[v] := 0$ 
else
begin { Viršūnės  $v$  "iškrova" }
     $u :=$  pirmoji aibės  $SPN[v]$  viršūnė;
    while  $Q[v] > 0$  do
begin
    if  $Q[u] = 0$  then Eilė  $\Leftarrow u;$ 
     $delta := min(Q[v], Sc[u, v] - Sf[u, v]);$ 
     $Sf[u, v] := Sf[u, v] + delta;$ 
     $Q[v] := Q[v] - delta;$ 
     $Q[u] := Q[u] + delta;$ 
    if  $Sf[u, v] = Sc[u, v]$  then { Lanko ( $u, v$ )
        šalinimas }
begin
         $SPN[v] := SPN[v] \setminus \{u\};$ 
         $SN[u] := SN[u] \setminus \{v\};$ 
end;
    if  $Q[v] > 0$  then
         $u :=$  po viršūnės  $u$  einanti kita aibės
         $SPN[v]$  viršūnė;
    end; { while  $Q[v] > 0$  }
end; { viršūnės  $v$  "iškrovos" pabaiga }

until  $v = s$ ; { srautas iš viršūnės  $s$  į  $r$  surastas }

end; { if  $XN \neq \emptyset$  }
end; { pagrindinio ciklo while  $XN \neq \emptyset$  pabaiga }
end; { procedūros maxpsa pabaiga }

```

Aptarkime procedūros **maxpsa** sudėtingumą. Pradinis viršūnių potencialų apskaičiavimas reikalauja  $O(n+m)$  veiksmų, kadangi kiekvienas grafo lankas analizuojamas ne daugiau kaip du kartus: apskaičiuojant  $Pout[v]$  ir  $Pin[v]$ .

Pagrindinio ciklo nulinio potencailo viršūnių šalinimo sudėtingumas taip pat yra  $O(n + m)$ , kadangi kiekviena viršūnė į steką STEK talpinama vieną kartą, o šalinant viršūnę iš steko pašalinamos visos jai incidentiškos briaunos, tuo būdu kiekvienas lankas šalinamas ne daugiau kaip vieną kartą.

Kiekviena pagrindinio ciklo iteracija išsaukia ne mažiau kaip vienos nulinio potencailo viršūnės atsiradimą (viršūnės  $r$  potencialas visada bus lygus nuliui). Vadinasi, pagrindinio ciklo pasikartojimų skaičius neviršija  $n$ .

Pagrindiniame cikle, be minėtų nulinio potencailo viršūnių šalinimo veiksmų, naudojant paiešką platyn konstruojamas  $p$  dydžio srautas iš  $r \in t$  ir iš  $s \in r$ . Šių dalių sudėtingumas yra lygus lankų analizavimo skaičiaus eilei.

Lanko analizė gali būti “naikinamoji”, kai srautas per lanką yra lygus to lanko pralaidumui, ir toks lankas šalinamas iš tinklo.

Lanko analizė gali būti “nenaikinamoji”, kai srauto per lanką dydis yra mažesnis už to lanko pralaidumą. Toks lankas iš tinklo nešalinamas ir gali būti analizuojamas kartojant pagrindinį ciklą.

Aišku, kad per visus pagrindinio ciklo pasikartojimus “naikinamuojų” būdu analizuojame  $O(m)$  lankų, o kiekviename pagrindiniame cikle “nenaikinamuojų” būdu analizuojame ne daugiau kaip  $n$  lankų (po vieną kiekvienai aplankytai viršūnei). Vadinasi, per visus pagrindinio ciklo pasikartojimus “nenaikinamuojų” būdu bus analizuota ne daugiau kaip  $n^2$  lankų. Tuo būdu, bendras pagrindinio ciklo veiksmų skaičius yra  $O(m + n^2)$ , t.y.  $O(n^2)$ .

Iš viso nagrinėjimo darome išvadą, kad procedūros **maxpsa** sudėtingumas yra  $O(n^2)$ .

Dabar aprašytas **psa** ir **maxpsa** procedūras galima sukomponuoti į vieną bendrą maksimalaus srauto apskaičiavimo algoritmą.

#### **Maksimalaus srauto apskaičiavimo algoritmas**

Duota:      tinklas, nusakytas gretimumo struktūromis  $N(v)$  ir  $PN(v)$ ,  
 $v \in V$  ; laukų pralaidumu  $c[u, v]$ ,  $u, v \in V$  ; šaltiniu  $s$  ir nuotakiu  $t$ .

Rezultatas: maksimalus srautas  $f[u, v]$ ,  $u, v \in V$ .

```

begin
    for  $u \in V$  do
        for  $v \in V$  do  $f[u, v] := 0$ ; { Pradinis nulinis srautas }.
        repeat
            psa; { Pagalbinio be kontūrų tinklo  $S_f$  formavimas }
            if  $d[t] \neq \infty$  then { srautas  $f$  – nemaksimalus }
```

```

begin
  maxpsa; { Pseudomaksimalaus srauto tinkle  $S_f$  apskaičiavimas }
  { Pseudomaksimalaus srauto perkėlimas į pagrindinį tinklą }
  for  $u \in SV$  do {  $SV -$  tinklo  $S_f$  viršūnių aibė }
  for  $v \in SV$  do
    begin
       $f[u, v] := f[u, v] + Sf[u, v];$ 
      if  $f[u, v] > c[u, v]$  then
        begin
           $f[v, u] := f[v, u] - (f[u, v] - c[u, v]);$ 
           $f[u, v] := c[u, v];$ 
        end;
    end;
  end; { fazės pabaiga }

  until  $d[t] = \infty$ ; { srautas  $f -$  maksimalus }
end; { algoritmo pabaiga }

```

**Išvada.** Iš procedūrų *psa* ir *maxpsa* sudėtingumo analizės bei to, kad fazių skaičius neviršija  $(n-1)$  išplaukia, kad pateikto maksimalaus srauto algoritmo sudėtingumas yra  $O(n^3)$ . Tuo pačiu šių algoritmų analizė parodė, kad bet kokiam tinklui egzistuoja maksimaus srautas. O tai ir yra Fordo ir Falkersono teoremos pilno įrodymo trūkstama grandis.

Ši paragrafą baigsite akivaizdžia, bet svarbia pateikta maksimalaus srauto radimo algoritmo savybe.

**Teorema.** Jeigu tinklo visų lankų pralaidumai yra sveikieji skaičiai, tai maksimalus srautas, apskaičiuotas pagal pateiktą algoritmą, yra sveikaskaitinis, t.y.  $f(u, v)$  yra sveikasis skaičius kiekvienam tinklo lankui  $(u, v)$ .

### 2.17.3. Maksimalaus suporavimo uždavinys dvidaliame grafe

**Apibrėžimas.** Suporavimas neorientuotajame grafe  $G = (V, U)$  – tai tokis grafo  $G$  briaunų poaibis  $M (M \subseteq U)$ , kad bet kokios dvi poaibio  $M$  briaunos tarpusavyje nėra gretimos, t.y. bet kokios dvi poaibio  $M$  briaunos nėra incidentiškos vienai ir tai pačiai viršūnei.

Jei briauna  $(u, v) \in M$ , tai sakome, kad  $M$  suporuoją  $u$  ir  $v$  viršūnes.

Jei viršūnė  $v$  nėra incidentiška nei vienai poaibio  $M$  briaunai, tai viršūnė  $v$  vadinama *laisvaja viršūne*, priešingu atveju – *suporuotaja*.

Maksimalaus suporavimo neorientuotame grafe uždavinys yra plačiai paplitęs, ir yra žinomi efektyvūs šio uždavinio sprendimo algoritmai. Visi jie pagrįsti alternuojančių grandinių teorija (teorija čeredujuščichsia cepei).

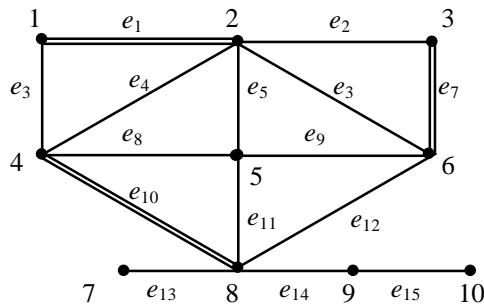
Tarkim,  $M$  – grafo  $G$  suporavimas.

**Apibrėžimas.** Grafo  $G$  grandinė, kurią sudaro biaunos, pakaitomis nepriklausančios ir priklausančios poaibiu  $M$ , vadinama suporavimo  $M$  atžvilgiu alternuojančia grandine.

Grandinė, kurios ilgis lygus 1, pagal apibrėžimą taip pat yra alternuojančia.

Alternuojančios grandinės briaunos, priklausančios suporavimui  $M$ , vadinamos **tamsiomis briaunomis**, o nepriklausančios  $M$  – **šviesiomis briaunomis**.

**Pavyzdys.** Panagrinėkime 2.17.5 pav. pavaizduotą grafą.



**2.17.5 pav.** Suporavimo uždavinys

Aibė  $M = \{e_1, e_7, e_{10}\}$  yra suporavimas grafe  $G$ ; grandinė  $P = (7, 8, 4, 1, 2, 5)$  yra suporavimo  $M$  atžvilgiu alternuojančia grandinė;  $e_1 = \{1, 2\}$ ,  $e_{10} = \{4, 8\}$  – tamsiosios grandinės  $P$  briaunas;  $e_7 = \{1, 4\}$ ,  $e_5 = \{2, 5\}$ ,  $e_{13} = \{7, 8\}$  – šviesiosios grandinės  $P$  briaunas; aibės  $\{1, 2, 3, 4, 6, 8\}$  ir  $\{5, 7, 9, 10\}$  – atitinkamai suporuotujų ir laisvųjų viršinių aibės.

Aišku, kag jei grafe  $G$  suporavimo  $M$  atžvilgiu egzistuoja grandinė, jungianti dvi laisvasias viršunes, tai grafe  $G$  galima sukonstruoti suporavimą, turintį didesnį briaunų skaičių nei  $M$ . Aišku, kad tokios alternuojančios grandinės šviesiųjų briaunų skaičius yra vienetu didesnis nei tamsiuju. Pašalinę iš  $M$  visas tamsiasias grandinės briaunas, ir į  $M$  įvedę visas šviesiasias tos grandinės briaunas, gausime suporavimą, kuriame briaunų skaičius bus vienetu didesnis nei  $M$ . Dėl tos priežasties alternuojančiai grandinės, jungianti dvi laisvasias viršunes, vadinama didinančiaja grandine.

**Teorema** (K.Beržas). Neorientuotojo grafo  $G$  suporavimas  $M$  yra didžiausias tada ir tik tai tada, kai šiame grafe suporavimo  $M$  atžvilgiu nėra didinančiosios grandinės.

Šiai teoremai iliustruoti panagrinėkime aukšciau 2.17.5 pav. pateiktą grafą. Imkime suporavimą  $M = \{e_4, e_7, e_{10}\}$  ir didinančiąją grandinę  $P = \{7, 8, 4, 1, 2, 5\} = \{e_{13}, e_{10}, e_3, e_1, e_5\}$ . Dabar galima sudaryti didesnį suporavimą  $M' = \{M \setminus \{e_1, e_{10}\}\} \cup \{e_3, e_5, e_{13}\} = \{e_3, e_5, e_7, e_{13}\}$ .

Suporavimas  $M'$  taip pat nėra didžiausias, nes šio suporavimo atžvilgiu egzistuoja grandinė  $P = \{9, 10\} = \{e_{15}\}$ .

Gausime suporavimą  $M'' = M' \cup \{e_{15}\} = \{e_3, e_5, e_7, e_{13}, e_{15}\}$ , kuris yra didžiausias, nes jo atžvilgiu grafe nėra didinančiosios grandinės.

Tuo būdu, Beržo teorema nusako tokią didžiausio suporavimo radimo strategiją.

1. Randame bet kokį pradinį suporavimą  $M$ .

Pavyzdžiu, pradinis suporavimas gali būti kuri grafo briauna.

Didesnio pradinio suporavimo galima ieškoti, naudojant “godaus” algoritmo strategiją:

- a) rasti briauną, kurios incidentiškų viršunių laipsnių suma yra mažiausia, ir ištraukti šią briauną iš suporavimą;
- b) iš grafo pašalinti iš suporavimą ištrauktą briauną drauge su jai gretimomis briaunomis.

Aišku, kad punktai a) ir b) bus kartojami tol, kol grafas  $G$  turės bent vieną briauną.

2. Nuosekliai konstruoti suporavimą seką

$M_1 = M, M_2, M_3, \dots, M_k, M_{k+1}, \dots$ , kurioje  $M_{k+1}$  gaunamas iš  $M_k$  radus grafe  $G$  suporavimo  $M_k$  atžvilgiu didinančiąją grandinę.

Kadangi  $|M_{k+1}| = |M_k| + 1$ , tai sekos ilgis bus nedidesnis nei  $\lfloor n/2 \rfloor$ .

Todėl norint rasti efektyvų šio uždavinio sprendimo algoritmą, pagrįstą išnagrinėta strategija, reikia sukonstruoti efektyvų didinančiosios grandinės radimo algoritmą. Nors tokie efektyvūs algoritmai egzistuoja, čia apsiribosime didžiausio suporavimo dvidaliame grafe nagrinėjimu.

Dažniausiai suporavimo uždavinys sprendžiamas dvidaliuose grafuose (žr. 2.2, 2.10 paragrafus). Tai klasikinis kombinatorikos uždavinys, žinomas “uždavinio apie sutuoktinį poras” vardu.

Primename, kad  $(n, m)$ -grafas  $G = (V, U)$  yra dvidalis, jei jo viršunių aibę  $V$  galima išskaidyti į du poaibius  $X$  ir  $Y$  taip, kad visų grafo  $G$  briaunų galai

priklausytų skirtiniams poaibiams. Todėl dvidalis grafas žymimas  $G = (X, Y, U)$ .

“Uždavinyse apie sutuoktinę porą” turi tokią interpretaciją. Tarkime,  $X$  ir  $Y$  atitinkamai yra vaikinų ir merginų aibės. Sudarykime dvidalį grafi  $G = (X, Y, U)$ , čia  $(x, y) \in U$ , jei jaunuolis  $x$  draugauja su mergaite  $y$ . Tada kiekvienas suporavimas  $M$  atitinka aibę galimų sutuoktinę porą, kiekviena iš kurių sudaryta iš tarpusavyje draugaujančių jaunuolio ir merginos; be to, kiekvienas žmogus dalyvauja ne daugiau kaip vienoje poroje.

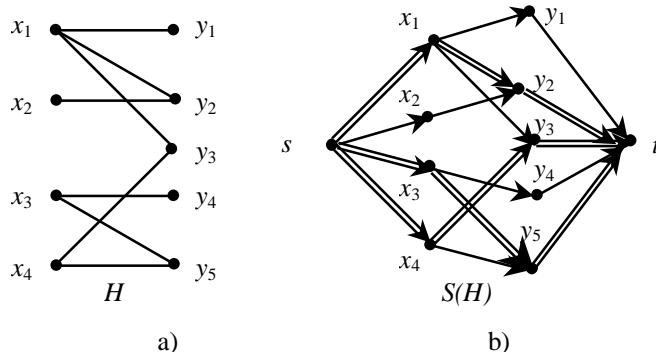
Pasirodo, kad maksimalaus suporavimo uždavinį dvidaliame grafe galima lengvai suvesti į maksimalaus srauto tinklo ieškojimo uždavinį.

Tarkime,  $H = (X, Y, U)$  – dvidalis grafas. Sudarykime tinklą  $S(H)$ , kuris turėtų šaltinį  $s$ , nuotaką  $t$  ( $s \neq t$  ir  $s, t \notin X \cup Y$ ), viršunių aibę  $V^* = \{s, t\} \cup X \cup Y$ , lankų aibę

$$\begin{aligned} E^* = & \{(s, x), x \in X\} \cup \{(y, t), y \in Y\} \\ & \cup \{(x, y) : x \in X \wedge y \in Y \wedge (x, y) \in U\} \end{aligned}$$

ir kiekvieno lanko  $u, v \in E^*$  pralaidumas  $c(u, v) = 1$ .

Pavyzdys. 2.17.6 a) pav. pavaizduotas dvidalis grafas  $H$ , o 2.17.6 b) pav. – jam atitinkantis tinklas  $S(H)$ .



**2.17.6 pav.** Dvidalis grafas  $H$  ir jam atitinkantis tinklas  $S(H)$ . Suporavimas  $M = \{(x_1, y_1), (x_2, y_2), (x_3, y_5), (x_4, y_3)\}$  ir jam atitinkantis srautas  $f_M$ .

**Teorema.** Kiekvienam grafo  $H$  k galios suporavimui  $M = \{(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)\}$ , ( $x_i \in X$ ,  $y_i \in Y$  visiems  $i = \overline{1, k}$ )

egzistuoja vienintelis tinklo  $S(H)$  dydžio  $k$  sveikaskaitinis (0,1) srautas  $f_M$  apibrėžiamas taip:

$$f_M(s, x_i) = f_M(x_i, y_i) = f_M(y_i, t) = 1 \text{ visiems } i = \overline{1, k}, \\ f_M(e) = 0 \text{ visiems likusiems tinklo lankams } e;$$

ir atvirkščiai, kiekvienam tinklo  $S(H)$  dydžio  $k$  (0,1) srautui  $f$  atitinka vienintelis grafo  $H$  suporavimas  $M_f$ ,  $|M_f| = k$ , apibrėžiamas taip:

$$M_f = \{(x, y) : x \in X \wedge y \in Y \wedge f(x, y) = 1\}.$$

2.17.6 pav. pavaizduotas suporavimas  $M = \{(x_1, y_2), (x_3, y_5), (x_4, y_3)\}$  grafe  $H$  ir jam atitinkantis srautas  $f_M$  tinkle  $S(H)$ .

Ši teorema įgalina maksimalaus srauto apskaičiavimo algoritmus, išnagrinėtus 2.17.6 paragrafe, panaudoti apskaičiuojant maksimalų suporavimą dvidaliame grafe.

Naudodami Dinico metodą, maksimalų suporavimą galima apskaičiuoti atlikus  $O(n^3)$  veiksmų. Tačiau, atsižvelgiant į tinklo  $S(H)$  specifiką, yra sukurti efektyvesni maksimalaus suporavimo apskaičiavimo algoritmai. Čia panagrinėsime Hopcrofto ir Karpo algoritmą, kurio sudėtingumas yra  $O(n^{5/2})$  (žr. Hopcroft J., Karp R.M. An  $n^{5/2}$  algorithm for maximum Matchings in bipartite graphs. SIAM J. Comput., 1973, 2, s. 225-231).

Šis algoritmas naudoja bendrą Dinico metodo schemą. Vienok, dėl tinklo  $S(H)$  specifiko galima sumažinti tiek fazų skaičių, tiek ir sukurti efektyvesnį pseudomaksimalaus srauto kiekvienoje fazėje apskaičiavimo algoritmą.

Pirmiausia pašymėsime, kad tinklo  $S(H)$  kiekvienos didinančiosios grandinės ilgis yra nelyginis skaičius, ir, iš jos atmetus pirmają ir paskutinią grandis, ši grandinė bus alternuojanti, prasidedanti ir pasibaigianti šviesiaja briauna (leistinaišiais suderintaisiais lankais).

**Apibrėžimas.** Duotajam suporavimui  $M$  ilgio  $l = 2k + 1$ ,  $k > 0$  iš  $X \setminus Y$  grandinę  $P \subseteq U$ :

$$P = \{(x_0, y_1), (y_1, x_1), (x_1, y_2), \dots, (y_k, x_k), (x_k, y_{k+1})\},$$

kurioje visos viršūnės  $x_0, x_1, \dots, x_k, y_1, y_2, \dots, y_{k+1}$  yra skirtingos,  $x_0$  – laisvoji  $X$  aibės viršūnė, o  $y_{k+1}$  – laisvoji  $Y$  aibės viršūnė, o kiekvienu antroji briauna priklauso  $M$ , t.y.

$$P \cap M = \{(y_1, x_1), (y_2, x_2), \dots, (y_k, x_k)\}$$

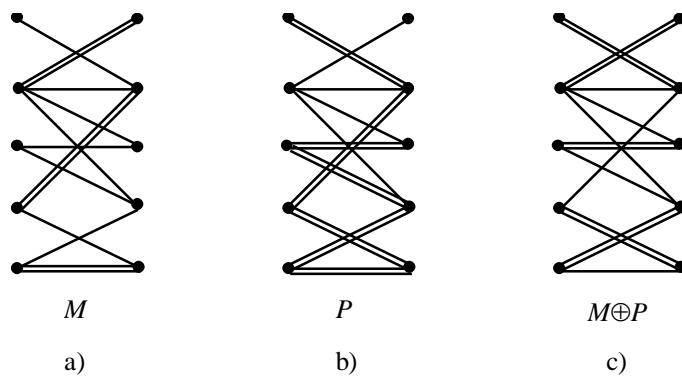
vadinsime alternuojančia grafo  $H$  grandine.

Aišku, kad alternuojančią grandinę galima nusakyti viršūnių  $x_0, x_1, \dots, x_k, y_1, y_2, \dots, y_{k+1}$  seka. Ši seka vieninteliu būdu apibrėžia srauto  $f_M$  atžvilgiu didinančiąją grandinę:

$$\begin{aligned} s, (s, x_0), x_0, (x_0, y_1), y_1, (y_1, x_1), x_1, (x_1, y_2), y_2, \dots, \\ y_k, (y_k, x_k), x_k, (x_k, y_{k+1}), y_{k+1}, (y_{k+1}, t), t. \end{aligned}$$

Be to, šios grandinės iššauktas srauto  $f_M$  padidėjimas nusako suporavimą, gautą susumavus aibes  $A$  ir  $P$  moduliu 2:  $M \oplus P$ .

Pavyzdžiu, 2.17.7 a) pav. pavaizduotas dvidalis grafas ir suporavimas  $M$ ; 2.17.7 b) pav. pavaizduota didinančioji alternuojančioji grandinė  $P$ , o 2.17.7 c) pav. pavaizduotas suporavimas  $M \oplus P$ .



**2.17.7 pav.** Suporavimo  $M$  padidinimas, panaudojant alternuojančią grandinę  $P$

Aišku, kad ir dvidalio grafo  $H$  atveju yra teisinga Beržo teorema: grafo  $H$  suporavimas  $M$  yra didžiausias tada ir tikta tada, kai šiame grafe suporavimo  $M$  atžvilgiu nėra didinančiosios grandinės.

Aprašysime Hopkrofto ir Karpo algoritmą. Šio algoritmo schema yra analogiška Dinico algoritmo schemai: kiekvienoje fazėje tinklui  $S(H)$  apskaičiuosime pagalbinį bekontūrinį tinklą, jo pseudomaksimalų srautą ir jį “perkelisime” į pagrindinį tinklą. Kaip buvo minėta aukšciau, dėl tinklo  $S(H)$  specifikos šie algoritmai yra efektyvesni nei Dinico metode naudojamas analogiškas algoritmas, o fazijų skaičius – mažesnis.

Pirmiausia aptarsime pagalbinio bekontūrinio tinklo apskaičiavimo algoritmą **PGA**.

Tarkime,  $H = (X, Y, U)$  yra dvidalis grafas, o  $M$  – suporavimas šiame grafe. Grafa  $H$  pakeiskime orientuotuoju grafu  $H_M$  tokiu būdu: kiekvienai

suporavimo  $M$  briaunai  $e \in M$  suteikime orientaciją nuo  $Y$  į  $X$ , o visoms likusioms briaunoms – orientaciją nuo  $X$  į  $Y$ . Aišku, kad tada kelias nuo laisvosios viršūnės  $x \in X$  į laisvąją viršūnę  $y \in Y$  bus srauto  $M$  atžvilgiu didinančioji grandinė.

Procedūra **PGA** pradžioje konstruoja bekontūrinio tinklo lankus nuo viršūnės  $s$  į visas laisvąsias aibes  $X$  viršunes. Po to iš šių laisvųjų viršūnių grafe  $H_M$  organizuojama paieška platyn. Jei paieškos platyn metu pasiekiamame laisvąją viršūnę  $y \in Y$ , tai į bekontūrinį tinklą įtraukiame lanką  $(y, t)$ . Be to, nuo to momento, kai į tinklą įtraukiamas pirmasis toks lankas, daugiau nenagrinėjamos viršūnės, kurios nutolę nuo viršūnės  $s$  nemažiau kaip kelio nuo  $s$  iki  $t$  ilgis.

Panagrinékime pseudomaksimalų srautą procedūros **PGA** sukonstruotame pagalbiname tinkle. Šiame pagalbiname  $(l+2)$  ilgio tinkle  $l$  ilgio (atmetame 1-ajį ir paskutinį lankus) alternuojančios didinančiosios grandinės visų viršūnių potencialai yra lygūs 1. Pagal šią grandinę padidinus srautą, visos grandinės viršūnės eliminuojamos iš tolesnio nagrinėjimo. Vadinas, pagalbinio tinklo pseudomaksimalų srautą apibréš didžiausia aibė ilgio  $l$  poromis nesusikertančių alternuojančių grandinių. Tokių grandinių ieškojimą realizuojama procedūra **fazė**, kuri naudoja paieškos gilyn iš viršūnės  $s$  pagalbiname tinkle metodą. Naujos aplankytos viršūnės talpinamos į steką STEK. Kiekvieną kartą, kai pasiekiamame viršūnę  $t$ , stike esančios viršūnės apibréžia alternuojančią didinančią grandinę. Tada visos viršūnės, išskyrus viršūnę  $s$ , šalinamos iš steko kartu didinant srautą (suporavimą), kurį nusako masyvas *pora*. Žemiau pateiki procedūrų **PGA**, **fazė** ir **pagrindinės programos** tekstai.

### *Hopkrofta ir Karpo didžiausio suporavimo algoritmas*

Duota: Dvidalis grafas  $H = (X, Y, U)$ , nusakytas gretumumo struktūra  $N(x)$ ,  $x \in X$ , čia  $N(x)$  – aibė viršūnių, gretimų viršūnei  $x$ .

Rasti: Didžiausią suporavimą, kurį nusako masyvas *pora*  $[1..n]$ ,  $n = |X| + |Y|$ , kurios elementas

$$pora[v] = \begin{cases} u, & \text{jei } pora(u, v) \text{ priklauso didžiausiam suporavimui,} \\ 0, & \text{jei viršūnė } v - \text{laisva,} \\ & v \in V = X \cup Y. \end{cases}$$

*procedure PGA;*

{ Pagalbinio bekontūrinio tinklo  $S(H)$  konstravimas.

*Kintamieji*  $V$ ,  $SV$ ,  $X$ ,  $Y$ ,  $pora$ ,  $N$ ,  $SN$ ,  $s$  ir  $t$  – globalieji. Čia, kaip ir maksimalaus srauto apskaičiavimo algoritme, (žr. 2.17.2 paragrafą)  $SV$  – tinklo  $S$  ( $H$ ) viršūnių aibė, o  $SN$  – tinklo  $S$  ( $H$ ) gretumino struktūra.

*begin*

```

for  $u \in V \cup \{s, t\}$  do { inicializacija }
  begin
     $d[u] = \infty$ ; { viršūnės v nuotolis nuo viršūnės s }
     $SN[u] := \emptyset$ ;
  end;
  { Laisvąsias viršūnes  $x \in X$  patalpinti į eilę ir į SN[s]. }
   $Eilé := \emptyset$ ;  $SV := \emptyset$ ;
   $SV := SV \cup \{s\}$ ; { SV – pagalbinio tinklo viršūnių aibė }
   $d[s] := 0$ ;
  for  $x \in X$  do
    if  $pora[x] = 0$  then { x – laisvoji viršūnė }
    begin;
       $Eilé \leftarrow x$ ;
       $SN[s] := SN[s] \cup \{x\}$ ;
       $d[x] := 1$ ;
    end;
  { Paieška platyn iš laisvųjų viršūnių  $x \in X$  }
  while  $Eilé \neq \emptyset$  do
    begin
       $u \leftarrow Eilé$ ;
       $SV := SV \cup \{u\}$ ;
      if  $u \in Y$  then
        if  $pora[u] = 0$  then { u – laisvoji viršūnė; prijungti lanką (u, t). }
        begin
           $SN[u] := SN[u] \cup t$ ;
          if  $d[t] = \infty$  then
            begin
               $SV := SV \cup \{t\}$ ;
               $d[t] := d[u] + 1$ ;
            end;
        end
        else {  $pora[u] \neq 0$  }
        begin
           $x := pora[u]$ ;
          if  $d[t] = \infty$  then { prijungti lanką (u, x) }
        end
      end
    end
  end

```

```

begin
    Eilė  $\Leftarrow$  x;
    SN [u] := SN [u]  $\cup$  {x};
    d [x] := d [u] + 1;
end;

begin
    end
else { u  $\notin$  Y, t.y. u  $\in$  X }
for y  $\in$  N [u] do
    if d [u] < d [y] then { d [y] = d [u] + 1 arba d [y] =  $\infty$ . }
    begin
        if d [y] =  $\infty$  then { y – nauja viršūnė } Eilė  $\Leftarrow$  y;
        SN [u] := SN [u]  $\cup$  {y};
        d [y] := d [u] + 1;
    end;
end;
end; { while }
end; { PGA }

procedure fazė;
{ Turimo suporavimo padidinimas, remiantis didžiausia aibe poromis
nesusikertančių kelių iš s į t pagalbiniame tinkle S (H).
Kintamieji SV, pradžia, SN, pora, s ir t – globalieji.
Procedūra fazė – tai modifikuota paieškos gilyn procedūra gylis3 (žr. 2.8.1.3
paragrafą). }
begin
    for u  $\in$  SV do { inicializacija }
    begin
        naujas [u] := true;
        p [u] := pradžia [u]; { pradžia [u] = rodyklė į sąrašo pradžią, –
        pirmajį elementą; p [u] = rodyklė į pirmajį analizuojamąjį sąrašo
        SN [u] elementą }
    end;
    { Paieška gilyn iš viršūnės s pagalbiniame tinkle (žr. 2.8.1.3 paragrafą) }
    STEK :=  $\emptyset$ ; STEK  $\Leftarrow$  s; naujas [s] := false;
    while STEK  $\neq$   $\emptyset$  do { pagrindinis ciklas }
    begin
        u := top (STEK); { u = viršutinis steko elementas }
        { Sąraše SN [u] ieškome pirmos naujos viešūnės }
        if p [u] = nil then { sąraše SN [u] naujų viršūnių nėra } b := false
            else b := not naujas [p [u].viršūnė];
        while b do

```

```

begin
     $p[u] := p[u] \uparrow.p_{\text{edsakas}}$ ;
    if  $p[u] = \text{nil}$  then  $b := \text{false}$ 
        else  $b := \text{not naujas } [p[u] \uparrow.\text{viršūnė}]$ ;
end;
if  $p[u] \neq \text{nil}$  then {
    Sąraše  $SN[u]$  randame naują viršūnę
        if  $p[u] \uparrow.\text{viršūnė} = t$  then {
            radome alternuojančią grandinę
            { Turimo suporavimo didinimas, remiantis steke saugoma
            alternuojančia grandine }
            while top (STEK)  $\neq s$  do
                begin
                     $y \Leftarrow STEK$ ;
                     $x \Leftarrow STEK$ ;
                    pora  $[x] := y$ ; pora  $[y] := x$ ;
                end; { while top (STEK)  $\neq s$  }
                { Steke liko tik šaltinis  $s$  }
                else {  $p[u] \uparrow.\text{viršūnė} \neq t$  }
                    begin
                         $v := p[u] \uparrow.\text{viršūnė}$ ;
                         $STEK \Leftarrow v$ ;
                        naujas  $[v] := \text{false}$ ;
                    end
            else {  $p[u] = \text{nil}$ . Sąraše  $SN[u]$  néra naujų viršūnių }
             $u \Leftarrow STEK$ ; { Šalinamas viršutinis steko elementas }
        end; { while  $STEK \neq \emptyset$  }
    end; { fazė }

begin { pagrindinė programa }
    for  $v \in V$  do pora  $[v] := 0$ ;
    { Inicializacija. Suporavimas – tuščioji aibė. }
    PGA; { Pradinio pagalbinio tinklo sudarymas }
    while  $t \in SV$  do
        begin
            fazė;
            PGA;
        end;
    end;

```

Reikia pažymeti, kad didelis Hopcrofto ir Karpo algoritmo efektyvumas gaunamas todėl, kad fazių skaičius yra nedidesnis nei  $\sqrt{n}$  [Lip88].

## 2.18. Grafo viršūnių laipsnių sekos

Paragrafe 2.2 apibrėžėme viršūnės  $v$  laipsnį  $\rho(v)$  ir grafo viršūnių laipsnių seką  $\rho = \rho(v_1), \rho(v_2), \dots, \rho(v_n)$ . Čia plačiau panagrinėsime kokias grafo savybes nusako jo viršūnių laipsnių seka, ką bendro turi grafai, turintis vienodas viršūnių laipsnių sekas, kaip sukonstruoti grafą pagal duotą viršūnių laipsnių seką, ir turintį vienokias ar kitokias savybes.

### 2.18.1. Grafinės sekos

Įveskime keletą apibrėžimų.

Sveikų neneigiamų skaičių seką  $d_1, d_2, \dots, d_n$  vadinsime ***n-seka*** ir žymėsime  $d$ . Jei egzistuoja grafas su viršūnių laipsnių seka  $\rho$ , sutampančia su  $n$ -seka  $d$ , tai šią  $n$ -seką vadinsime ***grafinė n-seka***, o ją atitinkantį grafą - ***sekos d realizacija***.  $n$ -seką vadinsime ***taisyklinga n-seka***, jei tenkinamos dvi sąlygos:

$$1) \quad n-1 \geq d_1 \geq d_2 \geq \dots \geq d_n,$$

$$2) \quad \sum_{i=1}^n d_i = \text{lyginis skaičius}.$$

Grafinė  $n$ -seka yra taisyklinga.

Kartais  $d$ -seką patogu užrašyti taip:  $d = (c_1^{k_1}, c_2^{k_2}, \dots, c_p^{k_p})$ , kur  $c_i$  - tarpusavyje skirtinė skaičiai, o laipsnio rodiklis nurodo, kiek kartų skaičius  $c_i$  kartoja się sekoje  $d$ . Pavyzdžiu,  $d = (5, 2^2, 1^6) = (5, 2, 2, 1, 1, 1, 1, 1)$ .

Bendru atveju grafinė seka turi daug realizacijų ir nustatyti jų skaičių yra sunku. Pavyzdžiu, grafinė  $n$ -seka  $d = (2^4, 1^4)$  turi penkias realizacijas (žr. 2.18.1 pav.). Čia

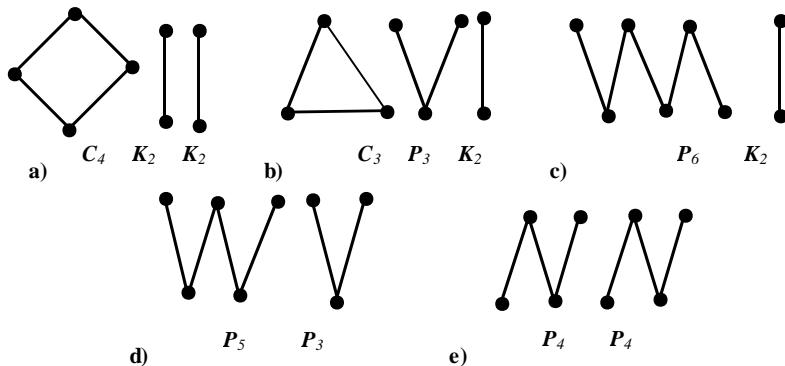
$C_n$  –  $n$ -viršūninis paprastasis ciklas,

$K_n$  –  $n$ -viršūninis pilnas grafas,

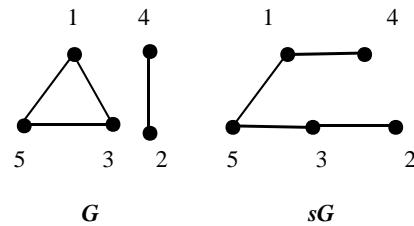
$P_n$  –  $n$ -viršūninė paprastoji grandis.

Galime nustatyti ryšį tarp kelių tos pačios grafinės sekos realizacijų. Įveskime ***perjungimo*** sąvoką. Tegul duotas grafas  $G = (V, U)$ ,  $a, b, c, d \in V$ ,  $(a, b), (c, d) \in U$ . Viršūnės  $a$  ir  $c$ , bei  $b$  ir  $d$  tarpusavyje briaunomis nesujungtos. Tada sakome, kad grafe  $G$  galimas ***perjungimas***  $s = (ab, cd)$ , kuri atlikus gaunamas grafas  $sG = G - (a, b) - (c, d) + (a, c) + (b, d)$ . Pavyzdžiu, 2.18.2 pav. parodyti grafai  $G$  ir  $sG$ , kur  $s = ((1, 3), (4, 2))$ .

Akivaizdu, kad perjungimas nekeičia grafo viršūnių laipsnių sekos. Perjungimo prasmę nusako žemiau pateikta teorema.



**2.18.1 pav.** Penkios grafinės  $n$ -sekos  $d = (2^4, 1^4)$  realizacijos



**2.18.2 pav.** Grafas  $G$  ir po perjungimo  $s=((1,3),(4,2))$  gautas grafas  $sG$ .

**Teorema.** Bet kuri grafinės sekos realizacija gaunama iš bet kurios kitos tos grafinės sekos realizacijos atlikus atitinkamą perjungimą seką.

Šios teoremos įrodymas remiasi tokia lema.

**Lema.** Tarkime  $G$  – grafas, kurio viršūnės sunumeruotos skaičiais 1, 2, ...,  $n$  taip, kad viršūnių laipsniai sudaro nedidėjančią seką  $d_i, i = \overline{1, n}$ , tai yra  $V=\{1,2,\dots,n\}, \rho(i)=d_i, d_i \geq d_{i+1}, i = \overline{1, n-1}$ . Tada bet kuriai grafo viršūnei  $i$  egzistuoja tokia perjungimų  $s_1, s_2, \dots, s_t$  seka, kad grafo  $H=s_1, s_2, \dots, s_t G$  viršūnės  $i$  gretimų viršūnių aibė  $N_H(i)$  sutampa su sekos  $d_i, i = \overline{1, n}$  didžiausio laipsnio viršūnių aibe, t.y.  $N_H(i) = \begin{cases} \{1, 2, \dots, d_i\}, & \text{jei } i > d_i, \\ \{1, 2, \dots, i-1, i+1, \dots, d_i+1\}, & \text{jei } i \leq d_i \end{cases}$

Kartais, nors ir retai, grafą vienareikšmiškai nusako jo viršūnių laipsnių seka. Jeigu visas grafinės sekos realizacijos yra izomorfiniai grafai, tai ši grafinė seka vadinama **unigrafine seka**, o jos realizacija - **unigrafa**. Pavyzdžiui, paprastasis ciklas  $C_5$  yra unigrafas.

### 2.18.2. Grafinės sekos kriterijai

Jau minėjome, kad grafinę seką visada galima laikyti taisyklinga  $n$ -seka. Tačiau atvirkštinis teiginys nėra teisingas –  $n$ -sekos taisyklingumas yra būtina, bet nepakankama sąlyga, kad  $n$ -seka būtų grafine. Pavyzdžiu, seka  $(3^2, 1^2)$  nėra grafinė seka.

**Havelo – Chakimi kriterijus.** Tegul  $d$  – taisyklinga  $n$ -seka,  $n > 1$ . Pažymėkime  $c^i$ ,  $i = \overline{1, n}$  ( $n-1$ )-seką, gautą iš  $n$ -sekos  $d$  išbraukus  $i$ -ajį narį:

$$c^i = (c_1, c_2, \dots, c_{n-1}), \quad c_k = \begin{cases} d_k, & \text{jei } k < i, \\ d_{k+1}, & \text{jei } k \geq i. \end{cases}$$

Iš sekos  $c^i$  gaukime seką  $d^i$  sumažindami vienetu pirmuosius  $d^i$  narius, t.y.

$$d^i = (f_1, f_2, \dots, f_{n-1}), \quad f_k = \begin{cases} c_k - 1, & \text{jei } k \leq d_i, \\ c_k, & \text{jei } k > d_i. \end{cases}$$

Tada  $d^i$  vadinsime *išvestine seka*.

Pavyzdžiu, jeigu  $d = (3, 3, 1, 1)$ , tai  $d^1 = d^2 = (2, 0, 0)$ ,  
 $d^3 = d^4 = (2, 3, 1)$ .

**Teorema.** (Havelas V. 1955 Chakimi S. 1962). Tegul  $d$  – taisyklinga  $n$ -seka ( $n > 1$ ). Jeigu kuriam tai indeksui  $i$ ,  $i = \overline{1, n}$ , išvestinė seka  $d^i$  yra grafinė seka, tai ir  $d$  seka yra grafinė seka. Jei seka  $d$  yra grafinė seka, tai ir kiekviena išvestinė seka  $d^i$  ( $i = \overline{1, n}$ ) yra grafinė seka.

**Erdeš – Gallai kriterijus.** Ši grafinės sekos kriterijų nusako tokia teorema.

**Teorema.** (Erdeš P. Gallai T. 1960). Taisyklinga  $n$ -seka yra grafinė tada ir tik tada, kai kiekvienam  $k = \overline{1, n-1}$  teisinga nelygybė

$$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min\{k, d_i\}.$$

Įrodyta, kad jei ši nelygybė galioja kai  $k = \overline{1, k(d)}$ , čia  $k(d) = \max\{i : d_i \geq i\}$ , tai ji galioja ir visoms likusioms  $k$  reikšmėms. Tai supaprastina Erdeš – Gallai kriterijaus taikymą  $n$ -sekos testavimui.

### 2.18.3. Grafinės sekos realizacijos algoritmas

Aprašytasis Havelo -Chakimi kriterijus leidžia atpažinti, ar duotoji taisyklinga  $n$ -seka yra grafinė ir rasti vieną iš sekos realizacijų. Pavadinkime ši algoritmą  $I$ -procedūra.

Duota:  $d$  – taisyklinga  $n$ -seka, t.y.

$$n-1 \geq d_1 \geq d_2 \geq \dots \geq d_n ,$$

$$\sum_{i=1}^n d_i = 2m \quad \text{lyginis skaičius.}$$

Rasti:  $d$  sekos grafinę realizaciją arba įsitikinti, kad ši seka tokios realizacijos neturi.

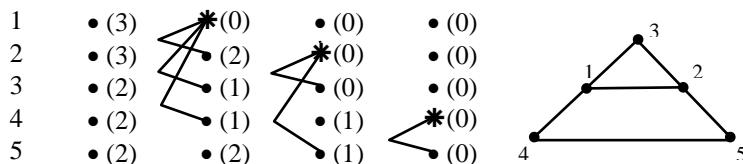
Tegul:  $V=\{1,2,\dots,n\}$  – būsimo grafo viršunių aibė, kur kiekvienai aibės viršūnei  $v$  ( $v = \overline{1, n}$ ) skirta atitinkama  $d(v)$  reikšmė,

$S(v)$  – viršūnių  $V$  poailbis, sudarytas iš tų viršūnių  $u \neq v$ , kurioms  $d(u) > 0$  reikšmės yra maksimalios. Aibės  $S(v)$  elementų skaičius yra lygus  $d(v)$  ir, bendru atveju ši aibė nusakoma nevienareikšmiškai.

Taigi  $I$ -procedūros eilinis žingsnis būtų toks:

- 1) fiksuoji bet kurią viršūnę  $v$ , kuriai  $d(v) > 0$  (viršūnė  $v$  vadina vedančiąja viršūne);
- 2) sudaryti vieną iš galimų aibų  $S(v)$ ;
- 3) vedančiąją viršūnę  $v$  sujungti briaunomis su visomis aibės  $S(v)$  viršūnėmis;
- 4) pakeisti  $d$  reikšmes vedančiajai viršūnei  $v$  ir kiekvienai aibės  $S(v)$  viršūnei  $u$ , nustatant  $d(v)=0$  ir  $d(u)=d(u)-1$ . Jei vykdant šį žingsnį kuri nors  $d$  reikšmė tampa neigiamą, reiškia seka neturi grafinės realizacijos.

**Pavyzdys.** Pritaikykime  $I$ -procedūrą tokiai  $n$ -sekai  $d = (3^2, 2^3)$ . 2.18.3 pav. kiekvienas stulpelis vaizduoja vieną  $I$ -procedūros žingsnį, parenkantį vedančiąją viršūnę (pažymėta žvaigždute), jai pavaldžiujų viršūnių aibę  $S(v)$  ir kiekvienos viršūnės  $d$  reikšmės kitimą (pažymėta skliausteliuose). Čia vedančiosios viršūnės - {1, 2, 4} ir  $S(1)=\{2, 3, 4\}$ ,  $S(2)=\{5, 3\}$ ,  $S(4)=\{5\}$ .



**2.18.3 pav.** Grafinė sekos  $d = (3^2, 2^3)$  realizacija taikant  $I$ -procedūrą

#### 2.18.4. Maksimaliai jungi grafinės sekos realizacija

Maksimaliai jungi grafinės sekos realizacija, yra tas grafas iš visų galimų grafinių realizacijų, kuriame briauninio jungumo skaičius  $\lambda(G)$  yra maksimalus.

Tegul  $d$  – taisyklinga grafinė  $n$ -seka. Kadangi bet kuriam grafui  $G$  galioja nelygybė  $\lambda(G) \leq \delta(G)$  (kur  $\delta(G)$  - minimalus grafo viršūnių laipsnis), stengsimės rasti grafinę realizaciją, kurioje  $\lambda(G) = d_n$ .

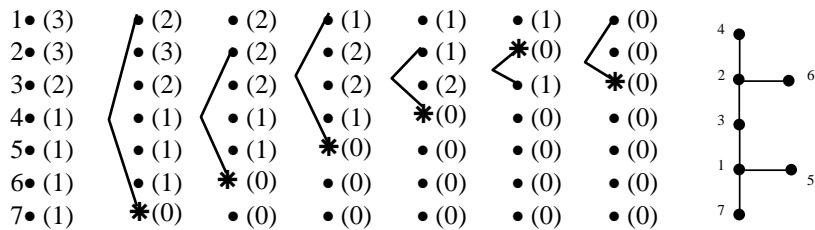
Pradžioje stengsimės sudaryti jungiąją grafinės sekos realizaciją.

**Teorema.** Taisyklinga grafinė seka gali būti realizuota jungiuoju grafu tada ir tik tada, kai  $d_n > 0$  ir teisinga nelygybė  $\sum_{i=1}^n d_i \geq 2(n-1)$ . Jeigu tenkinamos šios sąlygos  $I$ -procedūra, kiekviename žingsnyje vedančiaja viršūne parinkdama viršūnę su minimalia teigiamą  $d$  reikšme, sukonstruos junguojį grafą.

**Pastaba.** Kai  $d_n > 1$  teoremos nelygybė tenkinama automatiškai.

**Teorema.**  $n$ -seka  $d$  gali būti realizuota medžiu tada ir tik tada, kada ji neturi nulinį reikšmių ir galioja lygybė  $\sum_{i=1}^n d_i = 2(n-1)$ . Modifikuota  $I$ -procedūra, pavadinimine ja  $Im$ -procedūra, kiekviename žingsnyje vedančiaja viršūne parinkdama viršūnę su minimalia teigiamą  $d$  reikšme, sukonstruos grafą – medži.

Pavyzdžiui, seka  $d = (3^2, 2, 1^4)$  tenkina teoremoje pateiktą lygybę  $\sum_{i=1}^7 d_i = 2 \cdot (7-1)$  įstačius  $d$  reikšmes gauname  $3+3+2+1+1+1+1=12$ . Taigi, seka gali būti realizuota medžiu. Tai atliekančios  $Im$ -procedūros žingsniai ir sukonstruotas medis pavaizduoti 2.18.4 pav. Čia minimalios teigiamos  $d$  reikšmės paieška vykdoma einant nuo sekos  $d$  pabaigos į jos pradžią.



2.18.4 pav.  $Im$ -procedūra konstruojanti medži iš sekos  $d = (3^2, 2, 1^4)$

Pateikta teorema leidžia atpažinti, ar duotosios taisyklingos  $n$ -sekos realizacija gali būti medis ir modifikuoti  $l$ -procedūrą šio medžio konstravimui. Ši algoritmą pavadinome ***Im***-procedūra.

Tegul  $d$  – taisyklinga  $n$ -seka,

$V=\{1,2,\dots,n\}$  – būsimo grafo viršūnių aibė, kur kiekvienai aibės viršūnei  $v$  ( $v = \overline{1, n}$ ) skirta atitinkama  $d(v)$  reikšmė,

$ved\_v \in V$  – vedančiųjų viršūnių aibė  $|ved\_v| = n-1$ , t.y. algoritmo vykdymo eigoje parenkama viršūnė, turinti einamuoju momentu mažiausią  $d$  reikšmę, ir jungiama su kita viršūne (pavaldziaja), einamuoju momentu turinčia didžiausią reikšmę sekoje  $d$ ; algoritmo eigoje  $d$  sekos reikšmės kinta: parinkus vedančiąjį ir jai pavaldziajį viršunes jų abiejų  $d$  reikšmės sumažinamos vienetu; kadangi einamuoju momentu sekoje  $d$  gali būti kelios minimalios reikšmės, renkant vedančiąją viršūnę, ir kelios maksimalios reikšmės, renkant pavaldziajį viršūnę, sekos grafinė realizacija - medis nėra nusakomas vienareikšmiškai;

$pav\_v \in V$  – pavaldziujujų viršūnių aibė  $|pav\_v| = n-1$ , kiekvienai vedančiajai viršūnei iš aibės  $ved\_v$  parinkta pavaldzioji viršūnė įrašoma į aibę  $pav\_v$ .

$seka$  – kintamasis, įgaunantis reikšmes *TRUE*, jei galioja teoremos sąlygos  $\sum_{i=1}^n d_i = 2(n-1)$  ir  $d_n > 0$ , arba *FALSE* priešingu atveju, kas reiškia, kad pagal duotąjį seką negali būti sukonstruotas medis.

```
type mas= array [1..100] of integer;
procedure Im (n: integer; d: mas; var ved_v, pav_v:mas; var seka:BOOLEAN );
  var i,j,sum,max,min,w:integer;
begin
```

```
    sum:=0;
    if d[n]<>0 then
      begin
        for i:=1 to n do
          sum:=sum+d[i];
        if sum=2*(n-1)
        then
          begin
            seka:=TRUE;
            for w:=1 to n-1 do
              begin
```

{parinksime valdančiąjų viršūnę su  $\min d[i] <> 0$ , (minimalios  $d$  reikšmės pradėkime ieškoti nuo sekos  $d$  pabaigos, kur ir yra mažiausios  $d$  reikšmės)}

```
    min:=n; j:=0;
```

```

for i:=n downto 1 do
    if(d[i]<min)and(d[i]<>0)
        then begin min:=d[i]; j:=i end;
    ved_v[w]:=j;
    d[j]:=d[j]-1;
    {parinksime pavaldžių viršūnę su max d[i]}
    max:=0; j:=0;
    for i:=1 to n do
        if d[i]>max
            then begin max:=d[i]; j:=i end;
        pav_v[w]:=j;
        d[j]:=d[j]-1;
    end;
end
else seka:=FALSE
end;

```

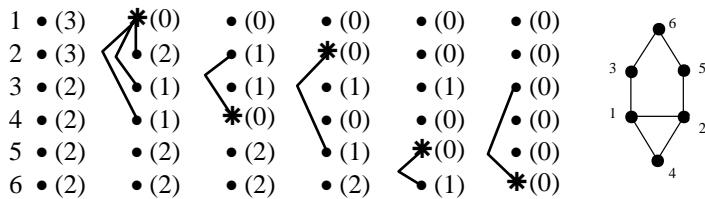
#### 2.18.5. Hamiltono grafo konstravimas pagal grafinę seką

**Teorema.** (Tshangfeizen V. 1978) Jei egzistuoja taisyklingos  $n$ -sekos realizacija  $d$  turinti Hamiltono grandinę prasidedančią  $d_1$  laipsnio viršūnėje, tai tokį grafą realizuos taip modifikuota  $lh$ -procedūra (pavadinkime ją  $lh$ -procedūra): pirmame žingsnyje vedančiaja viršūne imama  $d_1$  laipsnio viršūnė, o kiekviename sekantiame žingsnyje viršūnė, turinti mažiausią teigiamą  $d$  reikšmę iš aibės  $S(v)$ , kur  $v$ -vedančioji priešbuvisio žingsnio viršūnė, o  $S(v)$ -viršūnei v pavaldžiuju viršūnių aibė.

**Teorema.** (Tshangfeizen V. 1978) Kad taisyklingos  $n$ -sekos  $d$  realizacija būtų grafas, turintis Hamiltono ciklą, turi būti tenkinamos tokios dvi sąlygos:

- 1)  $d_i > 1, i = \overline{1, n};$
- 2) seka  $d$  gali būti realizuota Hamiltono grandine, prasidedančia  $d_1$  laipsnio viršūnėje.

Paveiksle 2.18.5 pav. parodyta sekos  $d = (3^2, 2^4)$  realizacija Hamiltono grafu,  $lh$  procedūroje pirmaja vedančiaja viršūne parinkus pirmają viršūnę. Gauta Hamiltono grandinė: 1, 4, 2, 5, 6, 3, kurią sudaro vedančiųjų viršūnių aibė  $\{1, 4, 2, 5, 6\}$  ir paskutinei vedančiajai viršūnei pavaldi viršūnė  $S(6)=3$ . Kadangi ši viršūnė yra ir pirmajai vedančiajai viršūnei (pradinei grandies viršūnei) pavaldžių viršūnių aibėje  $S(1)=\{2, 3, 4\}$ , gautoji realizacija turi ir Hamiltono ciklą: 1, 4, 2, 5, 6, 3, 1.

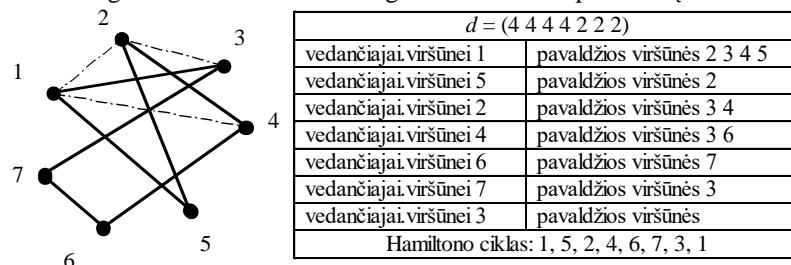


**2.18.5 pav.** Sekos  $d = (3^2, 2^4)$  realizacija Hamiltono grafu taikant  $Ih$ -procedūrą

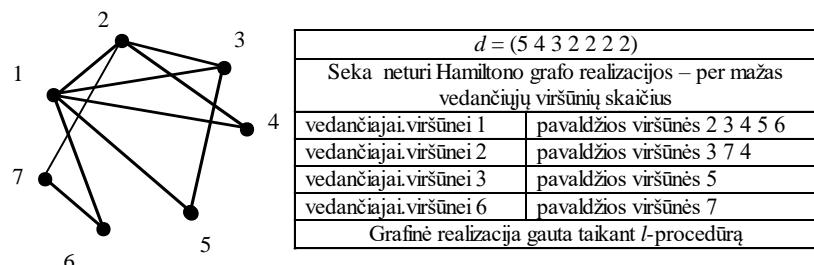
Nuo anksčiau aprašytoj  $I$ -procedūros  $Ih$ -procedūra skiriasi tuo, kad:

- nefiksuojamas vedančiųjų viršunių skaičius – jų turi būti  $n$ , kad susidarytų Hamiltono grandinę. Tiesa, čia paskutinė viršūnė vedančiųjų viršunių sąraše yra priešpaskutinės vedančiosios viršūnės pavaldžioji viršūnė, kuri jau nebeturai jai pavaldžių viršunių;
- nurodoma viršūnė  $v$ , nuo kurios pradedama ieškoti Hamiltono grandinės (ciklo). Pagal pirmają šio skyrelio teoremą, norint sužinoti, ar grandinė iš viso yra, pradedama nuo pirmos viršūnės.

Paveiksluose 2.18.6 pav. ir 2.18.7 pav. parodyti dar du bandymai realizuoti grafines sekas Hamiltono grafais taikant  $Ih$ -procedūrą.



**2.18.6 pav.** Sekos  $d = (4^4, 2^3)$  realizacija Hamiltono grafu taikant  $Ih$ -procedūrą



**2.18.7 pav.** Seka  $d = (5, 4, 3, 2^4)$  neturi Hamiltono realizacijos