4 skyrius

Algoritmai grafuose

4.1. Grafų teorijos uždaviniai

4.1.1. Grafai

Tegul turime viršūnių aibę $V = \{v_1, v_2, \ldots, v_N\}$ (angl. *vertex*) ir briaunų aibę $E = \{e_1, e_2, \ldots, e_K\}$, briauna (angl. *edge*) yra viršūnių pora – $e_j = (v_{1j}, v_{2j})$. Paprasčiausias grafo pavyzdys yra šalies kelių žemėlapis: miestai ir gyvenvietės sudaro viršūnių aibę, o keliai – briaunų aibę.

Jei briaunos $e_j = (v_{1j}, v_{2j})$ ir $e_k = (v_{2j}, v_{1j})$ yra skirtingos (svarbi yra ir jungimo kryptis), tai jos vadinamos *orientuotomis*, o grafas, sudarytas iš tokių briaunų, yra vadinamas *orientuotuoju* grafu. Orientuotąją briauną dar vadiname *lanku*. Miesto keliuose irgi susiduriame su tokia situacija, kai gatvėje leidžiamas tik vienpusis eismas.

Dvi grafo viršūnės, sujungtos bent viena briauna, vadinamos *gretimomis* arba *kaimyninėmis*, priešingu atveju jos vadinamos nepriklausomomis. Pavyzdžiui, Vilnius ir Kaunas yra gretimi miestai, o Kaunas ir Utena yra nepriklausomos kelių žemėlapio viršūnės.

Dažniausiai neužtenka tik žinoti, ar du miestai yra sujungti keliu, bet svarbu ir tai, koks yra atstumas tarp šių miestų, kokia yra kelio danga, koks maksimalus greitis yra leidžiamas važiuojant šiuo keliu. Todėl grafo briaunoms gali būti priskirti realūs skaičiai, įvertinantys atstumą, laiką,

svorį ir panašius požymius. Toks grafas yra vadinamas *svertiniu*. Briaunos $e_j \in E$ įvertį (svorį) žymėsime $w(e_j)$ (svoris angliškai *weight*).

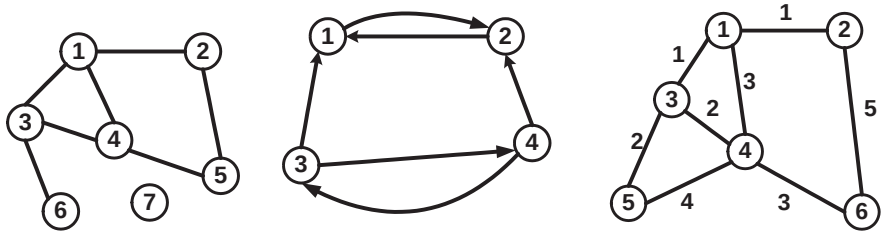
Viršūnės v kaimynų (angl. *neighbours*) aibę žymėsime

$$N(v) = \left\{ u : \ \ u \in V,\ (u,v) \in E \text{ arba } (v,u) \in E \right\}$$

ir vadinsime viršūnės *aplinka*.

Viršūnės v *laipsnis* $\deg(v)$ yra kaimynų skaičius. Jei viršūnė neturi kaimynų ($\deg(v) = 0$), tai ji vadinama *izoliuota*. Kai $\deg(v) = 1$, tai v vadinama *nusvirusia* viršūne. Orientuoto grafo atveju skiriame viršūnės *įėjimo* ir *išėjimo* puslaipsnius.

Svarbiausi grafų atvejai yra pavaizduoti 4.1 brėžinyje.



4.1 pav. Grafų pavyzdžiai: a) neorientuotas grafas, $|V| = 7$, $|E| = 7$, viršūnių v_1, v_3, v_4 laipsnis yra lygus 3, viršūnių v_2, v_5 laipsnis lygus 2, v_6 yra galinė viršūnė, v_7 yra izoliuota viršūnė, b) orientuotas grafas, $|V| = 4$, $|E| = 6$, c) svertinis grafas, $|V| = 6$, $|E| = 8$.

Viršūnių seka $p = \left\{ v_{i_0},\ v_{i_1},\ \ldots,\ v_{i_k} \right\}$ yra vadinama k – *keliu* (maršrutu, angl. *path*), jei sekos visos gretimos viršūnės yra sujungtos briaunomis, t.y.

$$(v_{i_j}, v_{i_{j+1}}) \in E, \ \ j = 0, 1, \ldots, k-1\,.$$

Ciklu vadiname k – kelią, kuriame pradinė viršūnė sutampa su paskutine $v_{i_0} = v_{i_k}$, o kitos viršūnės kelyje nesikartoja.

Grafas vadinamas *jungiu*, jei tarp bet kurių jo viršūnių egzistuoja kelias. Pavyzdžiui, jeigu turime kelių žemėlapį ir grafas yra jungus, tai iš bet kurio miesto ar gyvenvietės galima nuvažiuoti į kitą vietovę. Pavasarinių potvynių metu kai kurios gyvenvietės tampa nepasiekiamomis.

Nagrinėkime svertinį grafą. Kelio p ilgiu vadinsime skaičių

$$W(p) = \sum_{j=0}^{k-1} w\big((v_{i_j}, v_{i_{j+1}})\big)\,.$$

Tuo atveju, kai grafo briaunų svoriai nėra užduoti, kelio ilgiu vadiname kelio briaunų skaičių.

Trumpiausiu keliu, jungiančiu dvi grafo viršūnes a ir b, vadinsime kelią

$$p = \big\{a,\, v_{i_1},\, \ldots,\, v_{i_k}, b\big\},$$

tenkinantį sąlygą $W(p) \leqslant W(p')$, čia p' yra bet koks kitas kelias, jungiantis a ir b.

Jei visų briaunų svoriai yra teigiami skaičiai, tai trumpiausias kelias visada egzistuoja. Šį teiginį įrodome tokiais samprotavimais: kadangi briaunų svoriai yra teigiami skaičiai, tai trumpiausiame kelyje negali būti ciklų. Tada lieka baigtinis (nors gal būt ir labai didelis) skirtingų kelių skaičius, tarp jų ir išrenkame trumpiausią.

Grafas yra vadinamas *pilnu*, jei visos jo viršūnės tarpusavyje sujungtos briaunomis, t.y.:

$$N(v_j) = V \setminus \{v_j\}, \quad j = 1, 2, \ldots, |V|.$$

4.1.2. Pagrindiniai uždaviniai

1 uždavinys. Duotas grafas $G = (V, E)$. Reikia rasti trumpiausią kelią tarp dviejų jo viršūnių $a, b \in V$.

2 uždavinys. Reikia rasti trumpiausius kelius tarp viršūnės a ir visų kitų grafo viršūnių $v \in V$.

3 uždavinys. Jeigu grafas G yra orientuotas, tai reikia rasti trumpiausius kelius iš visų grafo viršūnių $v \in V$ iki duotosios viršūnės $a \in V$.

4 uždavinys. Kiekvienai grafo viršūnių porai $a, b \in V$ reikia rasti trumpiausią jas jungiantį kelią.

Aišku, kad išmokę spręsti 2 uždavinį, 3 uždavinį galėsime išspręsti tuo pačiu algoritmu, prieš tai pakeitę briaunų kryptis. Taigi lieka trys skirtingi uždaviniai. Atrodytų, kad užtenka išmokti spręsti 1 uždavinį, tada 2 ir 4 uždavinius spręsime, kaip seką paprastesnių pirmojo tipo uždavinių. Bet toks būdas nebūtinai yra geriausias: *pirma*, sudėtingesnio uždavinio tiesioginis sprendimo algoritmas gali būti daug efektyvesnis, *antra*, pamatysime, kad paprastesnį 1 uždavinį pavyksta išspręsti tik algoritmu, skirtu 2 uždavinio sprendimui. Pastarasis faktas yra pakankamai pamokantis: dažnai lengviau yra išspręsti tinkamai suformuluotą bendresnį uždavinį, nei rasti atskirojo uždavinio sprendinį.

5 uždavinys. Minimalus dengiantis medis. Kitas uždavinys labai dažnai sutinkamas planuojant komunikacinius tinklus, pavyzdžiui, kompiuterinį tinklą, jungiantį visus įstaigos kompiuterius. Tokį tinklą vaizduojame grafu, kurio viršūnių aibę V sudaro asmeniniai kompiuteriai, darbo stotys ir serveriai, o briaunų aibę E sudaro jungtys, jungiančios šiuos kompiuterius. Aišku, gautasis grafas turi būti jungiu, tik tada visi darbuotojai galės keistis informacija. Taip pat siekiame, kad komunikacinių linijų kaina būtų minimali, todėl reikia mažinti grafo briaunų skaičių.

Pirmiausia apibrėžiame svarbų grafo atvejį.

Medis yra jungus grafas, kuriame nėra ciklų. Aptarsime kai kurias medžių savybes, kurios ir charakterizuoja šią struktūrą.

4.1 teorema. *Tegul v ir w yra skirtingos medžio viršūnės, tada egzistuoja vienintelis jas jungiantis paprastas kelias.*

Įrodymas. Tarkime priešingai, kad egzistuoja du skirtingi keliai, jungiantys v ir w. Tada gauname, kad grafo briaunos sudaro ciklą, bet taip būti negali, nes grafas yra medis. □

4.2 teorema. *Medis, kuriame yra n viršūnių, turi $n-1$ briauną.*

Įrodymas. Jei turime tik vieną viršūnę, tai briaunų aibė yra tuščia. Pridėdami papildomą briauną turime pridėti ir naują viršūnę, nes priešingu atveju gausime ciklą. □

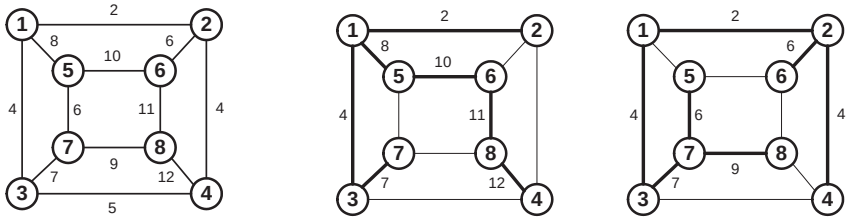
Tegul $G = (V, E)$ yra jungus grafas. Tada grafo G *dengiančiu* medžiu (angl. *spanning tree*) vadinsime medį $T = (V, E')$, kurio briaunų aibė E' yra grafo G briaunų aibės poaibis, t.y. $E' \subset E$.

Aišku, kad grafo dengiantis medis nebūtinai yra vienintelis. Yra daug algoritmų, leidžiančių sukonstruoti dengiančius medžius. Susipažinsime tik su vienu paprastu algoritmu. Pasirenkame bet kurią grafo G viršūnę. Kadangi grafas yra jungus, tai randame naują viršūnę, sujungtą briauna su viena iš jau parinktų viršūnių. Šį ciklą kartojame tol, kol parenkame visas n viršūnes.

Uždavinys pasunkėja, kai grafas G yra įvertintasis. Tada reikia rasti *minimalų* dengiantį medį T, t.y. medį, kurio bendrasis briaunų svoris $W(T)$ yra mažiausias, čia

$$W(T) = \sum_{e \in E'} w(e)\,.$$

Dengiančių medžių pavyzdžiai yra pavaizduoti 4.2 brėžinyje.



4.2 pav. Dengiančių medžių pavyzdžiai: a) grafas G, b) dengiantis medis $W(T) = 54$, c) minimalus dengiantis medis $W(T) = 38$.

4.1.3. Grafų vaizdavimas

Duomenų struktūros, vaizduojančios grafą, parinkimas nėra labai paprastas. Būtina atsižvelgti į du svarbius kriterijus: saugomos informacijos apimtį ir veiksmų (metodų) su grafais atlikimo efektyvumą. Ypač dažnai reikia mokėti rasti viršūnes, kurios yra gretimos duotajai. Orientuoto grafo atveju dar skiriamos gretimos įeinančios ir išeinančios viršūnės.

Grafo viršūnių gretimumo matrica. Turime grafą $G = (V, E)$. Apibrėžiame $n \times n$ dydžio matricą

$$\mathbf{S} = \begin{pmatrix} s_{11} & s_{12} & \dots & s_{1n} \\ s_{21} & s_{22} & \dots & s_{2n} \\ \dots & \dots & \dots & \dots \\ s_{n1} & s_{n2} & \dots & s_{nn} \end{pmatrix},$$

kurios elementai yra tokie:

$$s_{ij} = \begin{cases} 1, & \text{jei } e_{ij} := (v_i, v_j) \in E, \\ 0, & \text{jei } e_{ij} \notin E. \end{cases}$$

Jei grafas yra svertinis, tai gretimumo matricoje saugome ir briaunų svorius:

$$s_{ij} = \begin{cases} w_{ij}, & \text{jei } e_{ij} \in E, \\ 0, & \text{jei } e_{ij} \notin E. \end{cases}$$

Jei grafas nėra orientuotas, tai jo gretimumo matrica **S** yra *simetrinė*:

$$s_{ij} = s_{ji}, \quad 1 \leqslant i, j \leqslant n\,.$$

Matricos **S** i-osios eilutės nenuliniai elementai apibrėžia viršūnes v_j, į kurias galima patekti iš v_i viršūnės. Atitinkamai, j-ojo stulpelio nenuliniai elementai apibrėžia viršūnes v_i, iš kurių galima patekti į v_j.

Saugomos informacijos apimtis yra n^2 skaičių, viršūnės v_i visas gretimas viršūnes randame atlikę n veiksmų. Ši duomenų struktūra ypač efektyvi, kai reikia patikrinti ar $e_{ij} \in E$, tokio veiksmo kaštai yra $\mathcal{O}(1)$ eilės dydis.

Suspausto formato matrica. Dažniausiai grafo viršūnių laipsnis (gretimų viršūnių skaičius) yra daug mažesnis už n. Todėl didesnioji grafo gretimumo matricos koeficientų dalis yra lygi nuliui ir toks informacijos saugojimo būdas nėra ekonomiškas. Tiesinėje algebroje matricos, kurių eilučių nenulinių koeficientų skaičius yra daug mažesnis už stulpelių skaičių, yra vadinamos *retomis* matricomis (angl. *sparse matrix*). Jų saugojimui naudojame įvairius informacijos suspaudimo būdus. Vieną jų pritaikysime ir grafo duomenų vaizdavimui.

Masyve A iš eilės surašome visų viršūnių gretimas viršūnes. Šio masyvo ilgis yra lygus grafo briaunų skaičiui $\dim E$. Masyvo R elementas r_i nurodo viršūnės v_i gretimų viršūnių sąrašo pradžią masyve A, taigi v_i kaimynų aibė yra

$$N(v_i) = \left\{ v_{a_j} : \quad r_i \leqslant j < r_{i+1} \right\}.$$

Masyvo R ilgis yra $n+1$, paskutinis elementas $r_{n+1} = |E|+1$ yra naudojamas apibrėžiant viršūnės v_n kaimynus.

Jeigu turime įvertintąjį grafą G, tai W masyve saugome atitinkamų briaunų svorius. Briaunos numeruojamos taip pat, kaip ir gretimos viršūnės A masyve.

4.1 pavyzdys. Grafų vaizdavimas suspaustu formatu. Pateiksime grafų, pavaizduotų 4.1 brėžinyje, suspausto formato matricas:

$$A_1 = (2, 3, 4, 1, 5, 1, 4, 6, 1, 3, 5, 2, 4, 3),$$
$$R_1 = (1, 4, 6, 9, 12, 14, 15);$$

$$A_2 = (2, 1, 2, 4, 1, 3), \quad R_2 = (1, 2, 3, 5, 7);$$

$$A_3 = (2, 3, 4, 1, 6, 1, 4, 5, 1, 3, 5, 6, 3, 4, 2, 4),$$
$$W_3 = (1, 1, 3, 1, 5, 1, 2, 2, 3, 2, 4, 3, 2, 4, 5, 3),$$
$$R_3 = (1, 4, 6, 9, 13, 15, 17).$$

4.2. Trumpiausio kelio radimas

Pirmiausia spręsime 2 uždavinį, t.y. rasime trumpiausius kelius nuo viršūnės v iki visų kitų įvertintojo grafo viršūnių $w \in V$.

4.2.1. Deikstros algoritmas

Šį efektyvų algoritmą pasiūlė *E. Dijkstra*.

Masyve D saugome trumpiausių kelių iki kiekvienos viršūnės ilgius. Masyvas P yra naudojamas optimalaus maršruto atstatymui, jo i-ojo elemento reikšmė $p_i = k$ parodo, kad į v_i viršūnę patenkame iš v_k viršūnės.

Tegul S yra aibė viršūnių, iki kurių jau radome trumpiausią kelią. Pradžioje šiai aibei priklauso tik pradinė viršūnė v. Vykdydami algoritmą kiekviename žingsnyje aibę S papildome viena nauja viršūne. Aibėje Q saugome viršūnes, iki kurių trumpiausias kelias dar nežinomas. Aišku, taupant kompiuterio atmintį, galima apsiriboti tik vienos iš aibių S arba Q naudojimu, nes $Q = V \setminus S$, tačiau algoritmo realizacija yra efektyvesnė, kai parenkame tinkamą duomenų struktūrą aibės Q saugojimui.

Tarsime, kad pradinė yra v_1 viršūnė.

Deikstros algoritmas

```
(1)  for ( i = 2;  i ⩽ n ;  i++ ) {
         (2)  if ( (v_1, v_i) ∈ E )  d_i = w_1i;
         (3)  else  d_i = ∞;
         (4)  p_i = 1;
     }
(5)  S = {v_1};   Q = V \ S;
(6)  for ( i = 1;  i < n ;  i++ ) {
         (7)  randame  v_k ∈ Q:  d_k ⩽ d_j, ∀v_j ∈ Q;
         (8)  if  ( d_k = ∞ ) stop  // Grafas nejungus
         (9)  S = S ∪ v_k;   Q = Q \v_k;
         (10) for ( v_j ∈ N(v_k)∩ Q ) {
                  (11)  d = d_k + w_kj
                  (12)  if ( d_j > d ) {
                            (13)  d_j = d;
                            (14)  p_j = k;
                        }
              }
     }
```

Aišku, kad po pirmo žingsnio randame viršūnę v_k, iki kurios kelias iš v_1 yra trumpiausias (tai kaimyninė viršūnė). Po to nagrinėjame visas naujosios viršūnės dar neparinktas kaimynes v_j ir palyginame dviejų kelių ilgius: geriausio žinomo iki šiol ir naujo, kai pirmiausia trumpiausiu keliu einame į v_k viršūnę, o iš jos pasiekiame v_j.

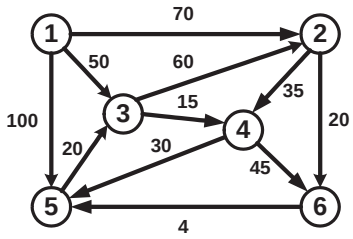
Šiame algoritme naudojame *godaus* metodo principą: kiekviename žingsnyje pasirenkame geriausią lokalų sprendinį. Kaip matėme, godusis algoritmas nebūtinai garantuoja gautojo sprendinio globalų optimalumą. Kitame poskyryje įrodysime, kad Deikstros algoritmu tikrai randame trumpiausius kelius.

1 uždavinio sprendimas. Jei užtenka rasti trumpiausią kelią tik iki vienos viršūnės w, tai Deikstros algoritme pakeičiame (6) sąlygą tokia:

$$(6)\ \textbf{while}\ (\, w \notin S\,)\ \{$$

Tačiau blogiausiu atveju teks atlikti $N-1$ algoritmo žingsnį, nes w gali būti parinkta pati paskutinė.

4.2 pavyzdys. Trumpiausių kelių radimas orientuotatame grafe. Turime orientuotą svorinį grafą, pavaizduotą 4.3 brėžinyje.



4.3 pav. Orientuotas įvertintasis grafas

Rasime trumpiausius kelius iš viršūnės v_1 iki visų likusių grafo viršūnių. Pradinės aibės S ir masyvų D, P reikšmės yra tokios:

$$S = \{v_1\}, \quad D = (\mathbf{0},\, 70,\, 50,\, \infty,\, 100,\, \infty),$$
$$P = (\mathbf{1},\, 1,\, 1,\, 1,\, 1,\, 1)\,.$$

Algoritmo vykdymo eiga yra tokia:

$$\mathbf{i = 1}: \quad S = \{v_1,\, v_3\},$$
$$D = (\mathbf{0},\, 70,\, \mathbf{50},\, 65,\, 100,\, \infty), \quad P = (\mathbf{1},\, 1,\, \mathbf{1},\, 3,\, 1,\, 1)\,,$$

$$\mathbf{i = 2}: \quad S = \{v_1,\, v_3,\, v_4\},$$
$$D = (\mathbf{0},\, 70,\, \mathbf{50},\, \mathbf{65},\, 95,\, 110), \quad P = (\mathbf{1},\, 1,\, \mathbf{1},\, \mathbf{3},\, 4,\, 4)\,,$$

$$\mathbf{i = 3}: \quad S = \{v_1,\, v_3,\, v_4,\, v_2\},$$
$$D = (\mathbf{0},\, \mathbf{70},\, \mathbf{50},\, \mathbf{65},\, 95,\, 90), \quad P = (\mathbf{1},\, \mathbf{1},\, \mathbf{1},\, \mathbf{3},\, 4,\, 2)\,,$$

$$\mathbf{i = 4}: \quad S = \{v_1,\, v_3,\, v_4,\, v_2,\, v_6\},$$
$$D = (\mathbf{0},\, \mathbf{70},\, \mathbf{50},\, \mathbf{65},\, 94,\, \mathbf{90}), \quad P = (\mathbf{1},\, \mathbf{1},\, \mathbf{1},\, \mathbf{3},\, 6,\, \mathbf{2})\,,$$

$$\mathbf{i = 5}: \quad S = \{v_1,\, v_3,\, v_4,\, v_2,\, v_6,\, v_5\},$$
$$D = (\mathbf{0},\, \mathbf{70},\, \mathbf{50},\, \mathbf{65},\, \mathbf{94},\, \mathbf{90}), \quad P = (\mathbf{1},\, \mathbf{1},\, \mathbf{1},\, \mathbf{3},\, \mathbf{6},\, \mathbf{2})\,,$$

Tada, trumpiausias kelias, jungiantis v_1 ir v_5, yra $p = (v_1, v_2, v_6, v_5)$, o jo ilgis $|p| = 94$. Jeigu būtų reikėję rasti trumpiausią kelią iki v_4 viršūnės, tai algoritmas užsibaigtų po antrojo žingsnio, o $p = (v_1, v_3, v_4)$.

Algoritmo sudėtingumo įvertinimas. Vykdydami Deikstros algoritmą (6) ciklą kartojame $n-1$ (t.y. $|V|-1$) kartą. Kiekviename žingsnyje randame aibės Q viršūnę, iki kurios žinomo kelio ilgis yra trumpiausias. Šios operacijos sudėtingumas priklauso nuo duomenų struktūros, realizuojančios aibę Q. Jei naudojame masyvą, tai veiksmų skaičius yra $\mathcal{O}(|Q|)$. Todėl trumpiausių kelių ilgių paieška viso reikalauja $\mathcal{O}(|V|^2)$ veiksmų. Kiekviena grafo briauna yra analizuojama tik vieną kartą, todėl kelių ilgiai yra perskaičiuojami $|E|$ kartų ir atliekame $\mathcal{O}(|E|)$ veiksmų. Taigi Deikstros algoritmo skaičiavimų apimtis yra $\mathcal{O}(|V|^2+|E|) = \mathcal{O}(|V|^2)$ veiksmų (čia pasinaudojome nelygybe $|E| < \frac{1}{2}|V|^2$).

Jeigu grafo G gretimumo matrica yra reta, t.y. $|E| = m|V|$, $m \ll |V|$ (daugelyje taikomųjų uždavinių m yra nedidelė konstanta), tai pagrindinė skaičiavimų dalis tenka trumpiausio kelio paieškai aibėje Q. Šį uždavinį spręsime efektyviau, kai Q yra piramidė. Piramidės formavimo kaštai yra $\mathcal{O}(|V|)$ veiksmų, o trumpiausių kelių ilgių perskaičiavimo ir piramidės struktūros išsaugojimo skaičiavimų apimtis yra $\mathcal{O}(|V|\log|V|)$ veiksmų. Taigi modifikuoto Deikstros algoritmo apimtis yra $\mathcal{O}(|V|\log|V|+|E|)$ veiksmų.

Deikstros algoritmo teisingumo įrodymas. Imkime kelią p iš grafo viršūnės a iki viršūnės v, nepriklausančios aibei S:

$$p = (a,\, w_1,\, w_2,\, \ldots,\, w_k,\, v)\,.$$

Jeigu visos tarpinės viršūnės priklauso aibei S, t.y. $w_j \in S, j = 1, \ldots, k$, tai toks kelias vadinamas S – *specialiuoju*.

Teisinga tokia teorema, kuri pagrindžia godaus metodo naudojimą sprendžiant 2 uždavinį apie trumpiausius kelius nuo duotosios grafo viršūnės iki visų kitų viršūnių:

4.3 teorema. *Tegul G yra įvertintasis grafas ir visų jo briaunų svoriai yra neneigiami skaičiai. Pradinę viršūnę pažymėkime $v_1 \in V$. Po kiekvieno Deikstros algoritmo žingsnio yra teisingi šie du teiginiai:*

a) jei $v_j \in S$, tai d_j yra trumpiausio kelio nuo v_1 iki šios viršūnės ilgis,

b) jei $v_j \in Q$, tai d_j yra trumpiausio S–specialaus kelio nuo v_1 iki šios viršūnės ilgis.

Įrodymas. Teoremą įrodysime matematinės indukcijos metodu. Pažymėkime S_{i-1} ir S_i aibę S prieš ir po i–ojo algoritmo žingsnio. Atitinkamai, kelių ilgius iki v_j viršūnės žymėsime $d_{j,i-1}$ ir $d_{j,i}$.

Pirmiausia įsitikinsime, kad abu teiginiai yra teisingi prieš pirmąjį žingsnį. Kadangi $S_1 = \{v_1\}$, tai visos likusios viršūnės priklauso Q_1. Taigi a teiginio teisingumo nereikia tikrinti nei vienai viršūnei. S_1–specialiuoju keliu yra briaunos, išeinančios iš pradinės viršūnės v_1, todėl ir b teiginys yra teisingas.

Tarkime, kad abu teiginiai yra teisingi prieš k-ąjį algoritmo žingsnį. Įrodysime, kad jie lieka teisingais ir atlikus šio žingsnio pertvarkymus. Tegul jo metu minimalų kelio ilgį turėjo v_k viršūnė, todėl $S_k = S_{k-1} \cup v_k$.

Jei $v_j \in S_{k-1}$, tai keliais iki viršūnės nebuvo pakeistas. Toks kelias buvo optimalus pagal indukcinę prielaidą, todėl jis liko trumpiausiu ir po k-ojo žingsnio. Lieka įsitikinti, kad ir kelias iki v_k, kuris taip pat nepakito šio žingsnio metu, yra trumpiausias. Remiantis indukcine prielaida jis yra trumpiausias S_{k-1}–specialusis kelias. Tarkime priešingai, kad egzistuoja trumpesnis kelias p nuo v_1 iki v_k, toks, kad

$$|p| < d_{k,k} = d_{k,k-1}\,.$$

Jis jau negali būti S_{k-1}–specialiuoju keliu, todėl jame yra viršūnių nepriklausančių S_{k-1}, tegul $\tilde{v}_1$ yra pirmoji tokia viršūnė:

$$p = \left\{v_1,\, v_{i_1},\, \ldots,\, v_{i_l},\, \tilde{v}_1,\, \ldots,\, \tilde{v}_m, \ldots, v_k\right\}.$$

Tada kelio dalis nuo v_1 iki $\tilde{v}_1$ yra S_{k-1}–specialusis kelias (nebūtinai trumpiausias)

$$\tilde{p} = \left\{v_1,\, v_{i_1},\, \ldots,\, v_{i_l},\, \tilde{v}_1\,\right\}.$$

Tegul $\tilde{v}_1 = v_j$. Gauname tokius p ilgio įverčius:

$$|p| \;\geqslant\; |\tilde{p}\,| \geqslant d_{j,k-1}\,.$$

Paskutinė nelygybė seka iš indukcinės prielaidos, kad $d_{j,k-1}$ yra ilgis trumpiausio S_{k-1}–specialaus kelio iki viršūnės $\tilde{v}_1$. Kadangi $\tilde{v}_1 \notin S_{k-1}$, tai $d_{k,k-1} \leqslant d_{j,k-1}$, priešingu atveju, vykdydami Deikstros algoritmo k-ąjį žingsnį, būtume pasirinkę $\tilde{v}_1$ viršūnę. Bet tada gauname, kad

$$p \geqslant d_{k,k-1},$$

o tai prieštarauja prielaidai, jog p yra trumpesnis kelias. Taigi a teiginys išlieka teisingas ir baigus k-ojo žingsnio pertvarkymus.

Dabar nagrinėsime b teiginį. Tegul $v_j \in Q_k$. Tada k-ajame žingsnyje perskaičiuojame trumpiausio S_k–specialaus kelio ilgį

$$d_{j,k} = \min\left(d_{j,k-1},\, d_{k,k-1} + w_{kj}\right)$$

ir pasirenkame trumpesnį iš dviejų galimų kelių.

Turime dvi galimybes:

1. Egzistuoja trumpiausias S_k – specialusis kelias, kuris neina per viršūnę v_k.

2. Visi trumpiausi S_k – specialieji keliai eina per viršūnę v_k.

Nagrinėkime *pirmąjį* atvejį. Tada trumpiausias S_k – specialusis kelias yra ir trumpiausias S_{k-1} – specialusis kelias, todėl $d_{j,k} = d_{j,k-1}$. Deikstros algoritme tokį variantą ir pasirenkame, nes, jei būtų išpildyta nelygybė

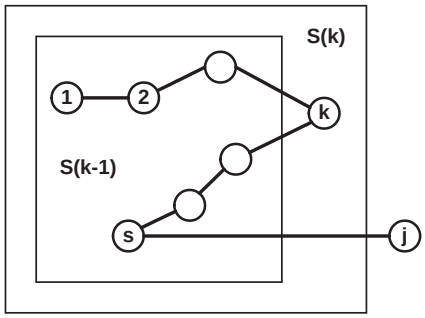
$$d_{k,k-1} + w_{k,j} < d_{j,k-1},$$

tai rastume trumpesnį S_k – specialųjį kelią iki viršūnės v_j ir jis eitų per v_k, o tai prieštarauja mūsų prielaidai. Taigi šiuo atveju b teiginys lieka teisingas ir po atliktų pertvarkymų.

Nagrinėkime *antrąjį* atvejį. Įrodysime, kad v_k visada yra *paskutinė* S_k – specialaus kelio viršūnė. Tarkime priešingai, kad $v_s \in S_k$ yra paskutinė kelio viršūnė. Kadangi $v_s \neq v_k$, tai $v_s \in S_{k-1}$. Tada teisinga nelygybė

$$d_{s,k-1} \leqslant d_{k,k-1},$$

nes priešingu atveju v_k viršūnė būtų pasirinkta anksčiau už v_s.



4.4 pav. Antrojo atvejo analizė: trumpiausias S_k – specialusis kelias

Trumpiausią S_k – specialųjį kelią p išskaidome į tris dalis (žr. 4.4 paveikslą): S_{k-1} – specialųjį kelią p_1, jungiantį pradinę viršūnę v_1 ir v_k, kelią p_2, jungiantį v_k ir v_s, bei briauną e_{sj}. Aišku, kad p_1 yra trumpiausias S_{k-1} – specialusis kelias.

Įvertiname kelio p ilgį:

$$\begin{aligned} |p| &= |p_1| + |p_2| + |e_{sj}| \geqslant |p_1| + |e_{sj}| \\ &= d_{k,k-1} + w_{sj} \geqslant d_{s,k-1} + w_{sj}\,. \end{aligned}$$

Taigi radome dar vieną S_k – specialųjį kelią, trumpiausiu būdu jungiantį v_1 su v_s ir po to tiesiogine briauna e_{sj} sujungtą su v_j. Jo ilgis yra nedidesnis už $|p|$ ir šis kelias neina per viršūnę v_k, o tai prieštarauja prielaidai, kad *visi* trumpiausi S_k – specialieji keliai eina per v_k. Deikstros algoritme tada ir pasirenkame naują trumpiausią kelią, taigi b teiginys lieka teisingas ir po atliktų pertvarkymų. □

4.2.2. Floido algoritmas

Šiame poskyryje spręsime 4 uždavinį, kai reikia rasti trumpiausius kelius tarp visų įvertintojo grafo G viršūnių porų. Ir šį uždavinį galime spręsti Deikstros algoritmu, kurį kartojame n kartų su vis kita pradine viršūne. Tokio metodo skaičiavimų apimtis yra $\mathcal{O}\big(|V|^2 \log|V| + |V|\,|E|\big)$.

Priminsime, kad Deikstros algoritmas priklauso godžiųjų metodų klasei. Susipažinsime su Floido (Floyd) algoritmu, kuriame panaudotas *dinaminio programavimo* metodas.

Metodo idėja yra paprasta. Pažymėkime D_k matricą, kurios koeficientai $d_{ij}(k)$ apibrėžia ilgį trumpiausio kelio nuo viršūnės v_i iki viršūnės v_j ir šiame kelyje nėra viršūnių, kurių indeksas didesnis už k. Pradinės matricos D_0 koeficientai yra tokie:

$$d_{ij}(0) = \begin{cases} 0, & \text{kai } i = j, \\ w_{ij}, & \text{kai } e_{ij} \in E, \\ \infty, & \text{kai } e_{ij} \notin E. \end{cases}$$

Toliau skaičiuojame matricas $D_1, D_2, \ldots, D_n$. Sudarant matricą D_k galimi du atvejai:

1. Trumpiausio kelio visų tarpinių viršūnių numeriai yra mažesni už k, tada teisinga lygybė
$$d_{ij}(k) = d_{ij}(k-1).$$
2. Trumpiausias kelias eina per viršūnę v_k, tada jo ilgis yra lygus atkarpų nuo v_i iki v_k ir nuo v_k iki v_j ilgių sumai (visų tarpinių viršūnių numeriai yra mažesni už k):
$$d_{ij}(k) = d_{ik}(k-1) + d_{kj}(k-1)\,.$$

Kadangi nežinome, kuris iš šių dviejų kelių yra trumpesnis, tai gauname matricos D_k koeficientų skaičiavimo formulę

$$d_{ij}(k) = \min\big(d_{ij}(k-1),\, d_{ik}(k-1) + d_{kj}(k-1)\big)\,.$$

Pastebėsime, kad kai kurių koeficientų nereikia skaičiuoti, nes teisingos tokios lygybės

$$\begin{aligned}
&d_{ii}(k) = 0\,,\\
&d_{ik}(k) = \min\big(d_{ik}(k-1),\, d_{ik}(k-1) + d_{kk}(k-1)\big) = d_{ik}(k-1)\,,\\
&d_{kj}(k) = \min\big(d_{kj}(k-1),\, d_{kk}(k-1) + d_{kj}(k-1)\big) = d_{kj}(k-1)\,.
\end{aligned}$$

Optimalų kelią saugome matricoje P, kurios koeficientas p_{ij} yra lygus trumpiausio kelio nuo v_i iki v_j tarpinių viršūnių didžiausiam numeriui.

Floido algoritmas

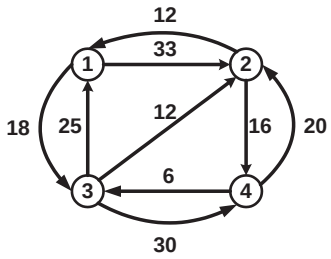
```
(1)  for ( i = 1; i ⩽ n ; i++ )
         (2)  for ( j = 1; j ⩽ n ; j++ ) {
                  (3)  if ( i == j )  d_ii = 0 ;
                  (4)  else if ( (v_i, v_j) ∈ E )  d_ij = w_ij;
                                   (5)  else  d_ij = ∞;
                  (6)  p_ij = 0;
              }
(7)  for ( k = 1; k ⩽ n ; k++ ) {
         (8)  for ( i = 1; i ⩽ n ; i++ )
         (9)  for ( j = 1; j ⩽ n ; j++ )
                  (10)  if ( (i ≠ k) && (j ≠ k) && (i ≠ j) ) {
                            (11)  d = d_ik + d_kj
                            (12)  if ( d_ij > d ) {
                                      (13)  d_ij = d;
                                      (14)  p_ij = k;
                                  }
                        }
     }
```

Pateiksime ir algoritmą, kuris atspausdina trumpiausio kelio tarpinių viršūnių numerius.

Trumpiausio kelio spausdinimo algoritmas

```
Path ( i, j )
begin
 (1)  k = p_ij ;
 (2)  if  ( k ≠ 0 ) {
        (3)  Path ( i, k );
        (4)  Spausdiname  k;
        (5)  Path ( k, j );
      }
end Path
```

4.3 pavyzdys. Trumpiausių kelių radimas Floido algoritmu. Turime orientuotą svorinį grafą, pavaizduotą 4.5 brėžinyje.



4.5 pav. Orientuotas įvertintasis grafas

Rasime trumpiausius kelius tarp visų grafo viršūnių. Pradinės masyvų D ir P reikšmės yra tokios:

$$D_0 = \begin{pmatrix} 0 & 33 & 18 & \infty \\ 12 & 0 & \infty & 16 \\ 25 & 12 & 0 & 30 \\ \infty & 20 & 6 & 0 \end{pmatrix}, \quad P_0 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Floido algoritmo vykdymo eiga yra tokia:

$$D_1 = \begin{pmatrix} 0 & 33 & 18 & \infty \\ 12 & 0 & \mathbf{30} & 16 \\ 25 & 12 & 0 & 30 \\ \infty & 20 & 6 & 0 \end{pmatrix}, \quad P_1 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

$$D_2 = \begin{pmatrix} 0 & 33 & 18 & \mathbf{49} \\ 12 & 0 & 30 & 16 \\ \mathbf{24} & 12 & 0 & \mathbf{28} \\ \mathbf{32} & 20 & 6 & 0 \end{pmatrix}, \quad P_2 = \begin{pmatrix} 0 & 0 & 0 & \mathbf{2} \\ 0 & 0 & 1 & 0 \\ \mathbf{2} & 0 & 0 & \mathbf{2} \\ \mathbf{2} & 0 & 0 & 0 \end{pmatrix},$$

$$D_3 = \begin{pmatrix} 0 & \mathbf{30} & 18 & \mathbf{46} \\ 12 & 0 & 30 & 16 \\ 24 & 12 & 0 & 28 \\ \mathbf{30} & \mathbf{18} & 6 & 0 \end{pmatrix}, \quad P_3 = \begin{pmatrix} 0 & \mathbf{3} & 0 & \mathbf{3} \\ 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 2 \\ \mathbf{3} & \mathbf{3} & 0 & 0 \end{pmatrix},$$

$$D_4 = \begin{pmatrix} 0 & 30 & 18 & 46 \\ 12 & 0 & \mathbf{22} & 16 \\ 24 & 12 & 0 & 28 \\ 30 & 18 & 6 & 0 \end{pmatrix}, \quad P_4 = \begin{pmatrix} 0 & 3 & 0 & 3 \\ 0 & 0 & \mathbf{4} & 0 \\ 2 & 0 & 0 & 2 \\ 3 & 3 & 0 & 0 \end{pmatrix}.$$

Algoritmo sudėtingumo įvertinimas. Vykdydami Floido algoritmo (7) ciklo vieną žingsnį atliekame $\mathcal{O}(|V|^2)$ veiksmų. Šio ciklo ilgis yra $n = |V|$ žingsnių, todėl Floido algoritmo apimtis yra $\mathcal{O}(|V|^3)$.