4.3. Minimalaus dengiančio medžio radimas

Šiame skyriuje susipažinsime su minimalių dengiančių medžių radimo algoritmais. Pirmiausia sudarysime dvi taisykles, leidžiančias pasirinkti tas įvertintojo grafo G briaunas, kurios priklauso kuriam nors minimaliajam dengiančiam medžiui, ar nustatyti, kad briauna nepriklauso nei vienam iš jų. Po to parodysime, kaip jos naudojamos Primo, Kruskalo ir Boruvkos algoritmuose. Kadangi abiejų taisyklių teisingumas bus įrodytas bendruoju atveju, tai atskirų algoritmų teisingumo tirti jau nereikės.

4.3.1. Algoritmų sudarymo taisyklės

Turime neorientuotą įvertintąjį jungų grafą $G = (V, E)$. Visas briaunas skirstome į tris poaibius $E = B \cup R \cup W$:

- mėlynas briaunas $e \in B$, priklausančias pasirinktam minimaliam dengiančiam medžiui;
- raudonas briaunas $e \in R$, nepriklausančias nei vienam minimaliam dengiančiam medžiui;
- baltas briaunas $e \in W$, kurių priklausomumas minimaliam dengiančiam medžiui dar nenustatytas.

Priminsime, kad dengiančiame medyje yra $n - 1$ briauna, todėl minimalaus dengiančio medžio radimo algoritmai ir turi parinkti tiek mėlynų briaunų.

Tegul $S \subset V$, tada $(S, V \setminus S)$ yra vadinamas grafo G *pjūviu*. Briauna $e \in E$ *kerta* (angl. *crosses*) pjūvį, jei vienas jos galas priklauso S, o kitas $V \setminus S$.

Tegul A yra grafo briaunų aibės poaibis $A \subset E$, tada grafo pjūvis $(S, V \setminus S)$ yra suderintas su A, jei nei viena A briauna nekerta šio pjūvio. Pastebėsime, kad A briaunų galai gali priklausyti abiejoms aibėms S, $V \setminus S$, bet abi vienos briaunos viršūnės būtinai turi priklausyti tik vienai iš šių aibių.

Tarp briaunų, kertančių grafo pjūvį, surandame mažiausio svorio briaunas, jas vadinsime *lengvomis*.

Mėlynoji taisyklė. Imkime grafo G briaunų poaibį A, kuriam priklauso dalis kurio nors minimalaus dengiančio medžio briaunų. Sudarome grafo pjūvį $(S, V \setminus S)$, suderintą su A. Randame lengvas baltas pjūvio briaunas ir vieną iš jų nudažome mėlyna spalva.

Raudonoji taisyklė. Pasirinkime paprastą ciklą K, kuriame nėra raudonų briaunų. Tarp baltų jo briaunų randame didžiausio svorio briauną ir nudažome ją raudonai.

4.1 lema. *Kol bent viena jungaus įvertintojo grafo G briauna yra baltos spalvos, visada galima pritaikyti melynąją arba raudonąją taisyklę.*

Įrodymas. Kadangi visos mėlynos briaunos priklauso minimaliam dengiančiam medžiui, tai algoritmo vykdymo metu turime mišką, sudarytą iš atskirų mėlynų medžių. Imkime baltą briauną $e = (v, w) \in W$.

Jeigu ji priklauso vienam iš mėlynų medžių, tai egzistuoja kelias, jungiantis viršūnes v ir w. Tada prijungus naują briauną e, gauname paprastą ciklą, todėl e nudažome raudona spalva.

Tarkime, kad e jungia skirtingus mėlynuosius medžius

$$T_1 = (V_1, B_1), \quad T_2 = (V_2, B_2)\,.$$

Tada turime baltą briauną, kertančią grafo pjūvį $(V_1, V \setminus V_1)$, suderintą su mėlynų briaunų aibe B_1. Suradę tokio pjūvio lengvas briaunas, vieną jų nudažome mėlyna spalva. □

Turime tokią bendrą minimalaus dengiančio medžio radimo algoritmų schemą.

Minimalaus dengiančio medžio radimo algoritmas

(1) $B = \emptyset$, $R = \emptyset$;
(2) **while** ($|B| <$ (n - 1)) {
    (3) Taikome *Mėlynąją* arba *Raudonąją* taisyklę;
    (4) **if** $\big(\ |R| == (|E| - n + 1)\ \big)$ $B = E - R$;
}

Įsitikinsime, kad naudodami šį algoritmą visada randame minimalų dengiantį medį).

4.4 teorema. *Tarkime, kad įvykdyta k algoritmo žingsnių ir egzistuoja minimalus dengiantis medis T, kuriam*

- *priklauso visos mėlynos briaunos,*
- *nepriklauso nei viena raudona briauna.*

Tada šis teiginys lieka galioti ir po eilinio algoritmo žingsnio, t.y. pritaikius mėlynąją arba raudonąją taisyklę.

4.3.2. Primo algoritmas

Algoritmas priklauso godžiųjų metodų klasei. Jį paskelbė *R.C. Prim*. Algoritmo idėja labai paprasta, kiekviename žingsnyje randame trumpiausią briauną $e = (u, v) \in E$, jungiančią jau parinktas minimalaus dengiančio medžio viršūnes U su likusiomis grafo viršūnėmis $V \setminus U$. Tada briauną e įtraukiame į medį $T = (U, B)$.

Apibrėžkime aibę briaunų, jungiančių aibės U viršūnes su likusiomis grafo viršūnėmis

$$D = \left\{ e \in E : \ e = (u, v), \ u \in U, \, v \in V \setminus U \right\}.$$

Primo algoritmas

(1) $U = \{v_1 \in V\}, \quad B = \emptyset$;
(2) **while** ($|U| < |V|$) {
(3) Randame briauną $e = (u, v) : \ w(e) = \min_{z \in D} w(z)$;
(4) $U = U \cup \{v\}, \quad B = B \cup e$;
}

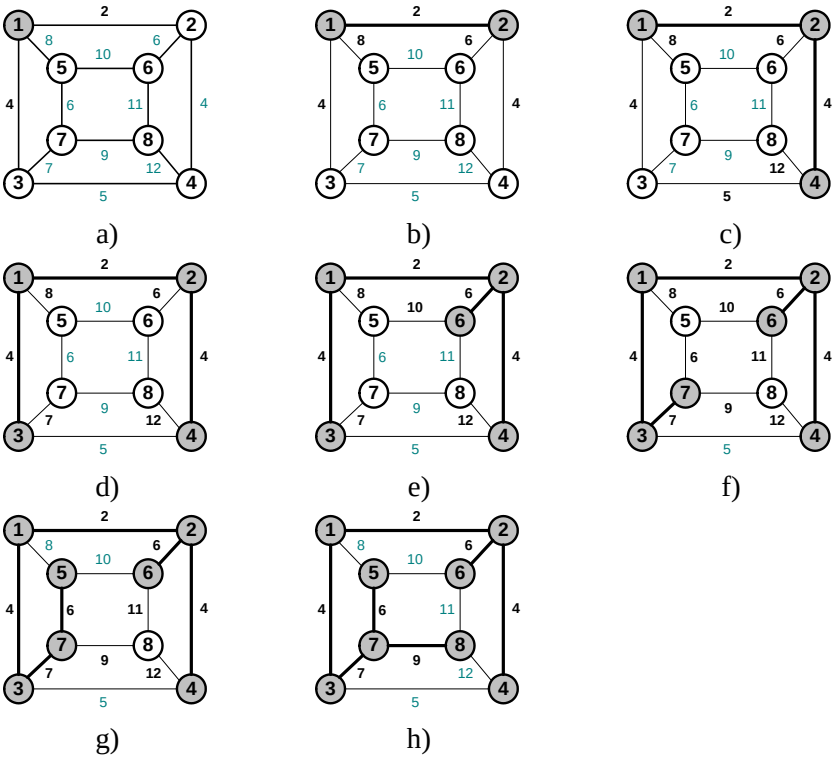
4.5 teorema. *Tegul G yra įvertintasis jungus grafas. Tada Primo algoritmu randame minimalųjį dengiantį medį.*

Įrodymas. Nagrinėkime grafo G pjūvį $(U, \, V \setminus U)$. Jis yra suderintas su jau parinktų briaunų aibe B, todėl briauna e yra lengva. Taigi įrodėme, kad Primo algoritme briaunos yra parenkamos naudojant mėlynąją taisyklę. □

4.4 pavyzdys. Minimalaus dengiančiojo medžio radimas Primo algoritmu. Imkime grafą, pavaizduotą 4.2 paveiksle. Minimalaus dengiančiojo medžio formavimas Primo algoritmu yra parodytas 4.6 paveiksle.

Algoritmo sudėtingumo analizė. Primo algoritmo skaičiavimų apimtis esminiai priklauso nuo duomenų struktūrų, kuriose saugome informaciją apie grafą G ir minimalų dengiantį medį T.

Tarkime, kad grafą G vaizduojame jo viršūnių gretimumo matrica. Masyve d saugome informaciją apie legviausių briaunų, jungiančių dar neparinktas viršūnes su minimalaus dengiančio medžio viršūnėnėmis, svorius. Tada Primo algoritmo (3) žingsnio realizacija (lengviausios kertančios briaunos paieška) trunka $\mathcal{O}(|V|)$ veiksmų. Masyvo d reikšmių patikslinimas, atsižvelgiant į ką tik parinktą naują viršūnę, irgi atliekamas per $\mathcal{O}(|V|)$ veiksmų. Vykdydami algoritmą (2) ciklą kartojame $|V|-1$ kartą, todėl tokios paprasčiausios Primo algoritmo realizacijos apimtis yra $\mathcal{O}(|V|^2)$ veiksmų.



4.6 pav. Minimalaus dengiančiojo medžio radimas Primo algoritmu: a) pradinis grafas, b – h) minimalaus dengiančiojo medžio pomedis po kiekvieno algoritmo žingsnio. Riebiu šriftu pažymėti svoriai tų nepasirinktų briaunų, kurios nagrinėjamos eiliniame algoritmo žingsnyje

Jeigu grafo viršūnių gretimumo matrica yra reta (t.y. $|E| = \mathcal{O}(|V|)$), tai naudosime sudėtingesnes duomenų struktūras. Tegul Q yra prioritetinė eilė, kurioje saugomos dar neparinktos grafo G viršūnės, pradžioje $Q = V$. Viršūnės $v \in Q$ vieta eilėje priklauso nuo įverčio $d(v)$ reikšmės. Algoritme dažnai tenka tikrinti ar duotoji viršūnė jau parinkta, todėl naudojame papildomą masyvą, kurio elementai yra loginės konstantos T ir F, parodančios ar $v \in Q$. Tada tokio tikrinimo sudėtingumas yra tik $\mathcal{O}(1)$ veiksmų.

Taip pat apibrėžiame funkciją π, kurios argumentai yra grafo viršūnės. Jei $v \in B$, tai $\pi(v)$ reikšmė yra viršūnės v tėvas minimaliame dengiančiame medyje T, jei $v \in Q$, tai $\pi(v)$ reikšmė yra jau parinkta viršūnė $u \in B$, su kuria jungiančios briaunos $e = (u, v)$ svoris yra lygus $d(v)$.

Realizuodami algoritmą nesaugome minimalaus dengiančiojo medžio briaunų,

nes jas randame panaudodami funkciją π ir eilę Q

$$B = \left\{ \big(v, \pi(v)\big) : \quad v \in V \setminus \{s\} \setminus Q \right\},$$

čia s yra medžio šaknis (pradinė viršūnė).

Pateiksime patikslintą Primo algoritmą.

Primo algoritmas

(1) $Q = V$;
(2) **for** ($v \in Q$) $d(v) = \infty$;
(3) $d(s) = 0$; $\pi(s) =$ NULL;
(4) **while** ($Q \neq \emptyset$) {
(5) Iš Q išimame pirmąją eilėje viršunę u: $Q = Q \setminus \{u\}$;
(6) **for** ($v \in N(u)$)
(7) **if** $\big($ $(v \in Q)$ &&($\mathrm{w}((u,v)) < d(v)$) $\big)$ {
(8) $\pi(v) = u$, $d(v) = \mathrm{w}((u,v))$;
}
}

Jeigu prioritetinę eilę Q realizuojame naudodami piramidę, tai eilės tvarkymo kaštai po (5) algoritmo žingsnio yra $\mathcal{O}(\log |V|)$ veiksmų. (4) ciklą kartojame $|V|$ kartų, todėl viso atliekame $\mathcal{O}(|V| \log |V|)$ veiksmų.

Prioritetinėje eilėje saugomų viršūnių įverčių tikslinimo operaciją reikės atlikti daugiausią $|E|$ kartų, vieno tikslinimo veiksmo kaštai yra $\mathcal{O}(\log |V|)$, todėl viso atliekame $\mathcal{O}(|E| \log |V|)$ veiksmų. Kadangi $|V| < |E|$, tai Primo algoritmo skaičiavimų apimtis yra $\mathcal{O}(|E| \log |V|)$.

4.3.3. Kraskalo algoritmas

Tai irgi gobšusis algoritmas. Paaiškinsime pagrindinę algoritmo idėją. Visas grafo briaunas surūšiuojame jų svorio didėjimo tvarka:

$$w(e_1) \leqslant w(e_2) \leqslant \ldots \leqslant w(e_m)\,.$$

Pradžioje turime dengiančių medžių mišką, kurį sudaro tik grafo viršūnės. Tada iš eilės tikriname kiekvieną briauną. Įvertiname dvi galimybes:

1. Jei nagrinėjamos briaunos e_i abu galai priklauso tam pačiam mėlynam medžiui, tai egzistuoja kelias, jungiantis šias viršūnes. Kadangi cikle nėra raudonų briaunų, tai briauną e_i nudažome raudona spalva, t.y. jos neįtraukiame į minimalų dengiantį medį.

2. Jei briaunos galai priklauso skirtingiems medžiams T_1 ir T_2, tai ją nudažome mėlyna spalva, o medžius sujungiame.

Kraskalo algoritmas

(1) Grafo briaunas išdėstome jų svorio didėjimo tvarka;
(2) $F = \{T_1, T_2, \ldots, T_n\}$, $T_i = \big(\{v_i\}, \emptyset\big)$;
(3) i=0, $B = \emptyset$;
(4) **while** ($|B| < |V| - 1$) {
  (5) **if** ($e_i = (u, v)$ galai priklauso skirtingiems medžiams) {
    (6) $B = B \cup e_i$;
    (7) Sujungiame abu pomedžius į vieną medį;
  }
  (8) i = i+1;
}

4.6 teorema. *Tegul G yra įvertintasis jungus grafas. Tada Kraskalo algoritmu randame minimalųjį dengiantį medį.*

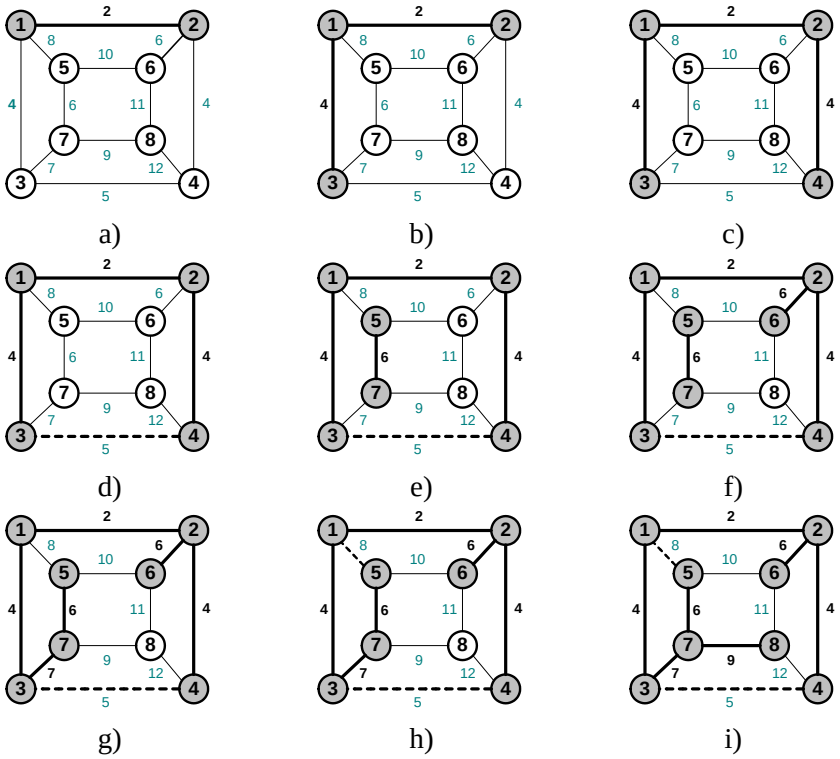
Įrodymas. Jeigu eiliniame žingsnyje briaunos e_i abu galai priklauso tam pačiam mėlynam medžiui, tai jos neįtraukiame į minimalųjį dengiantį medį. Kadangi šiai briaunai galime taikyti raudonąją taisyklę, tai šiuo atveju teoremos teiginys teisingas.

Nagrinėkime antrąjį atvejį, kai briaunos e_i galai priklauso skirtingiems medžiams $T_1 = (V_1, B_1)$ ir T_2. Tada konstruojame grafo G pjūvį $(V_1,\ V \setminus V_1)$. Jis yra suderintas su jau parinktų briaunų aibe B, o briauna e_i kerta šį pjūvį ir yra lengva, nes briaunos surūšiuotos jų svorio didėjimo tvarka. Taigi Kraskalo algoritme briaunos įtraukiamos į minimalų dengiantį medį naudojant mėlynąją taisyklę. □

4.5 pavyzdys. Minimalaus dengiančiojo medžio radimas Kraskalo algoritmu. Imkime grafą, pavaizduotą 4.2 paveiksle. Minimalaus dengiančiojo medžio formavimas Kraskalo algoritmu yra parodytas 4.7 paveiksle.

Algoritmo sudėtingumo analizė. Mums reikia įvertinti sudėtingumą trijų pagrindinių operacijų:

1. Surūšiuoti grafo briaunas jų svorių didėjimo tvarka.
2. Duota viršūnė v, reikia rasti medį T_i, kuriam ji priklauso.
3. Sujungti du pomedžius į vieną medį.



4.7 pav. Minimalaus dengiančiojo medžio radimas Kraskalo algoritmu: a – i) minimalaus dengiančiojo medžio pomedis po kiekvieno algoritmo žingsnio. Storos juodos linijos žymi mėlynas briaunas, punktyrinės linijos – raudonas briaunas

Grafo briaunas rūšiuojame kuriuo nors greituoju algoritmu, taigi atliekame $\mathcal{O}(|E|\log|E|)$ veiksmų. Likusias dvi operacijas atliekame $\mathcal{O}(|E|)$ kartų. Kitame poskyryje susipažinsime su aibių paieškos ir dviejų aibių sujungimo algoritmais, kurių bendroji skaičiavimų apimtis yra tik $\mathcal{O}\big(|E|\,o(\log|E|)\big)$ veiksmų.

Todėl Kraskalo algoritmo sudėtingumas yra $\mathcal{O}(|E|\log|E|)$ o didžioji skaičiavimų dalis yra skirta grafo briaunų rūšiavimui.

4.3.4. Aibės ir algoritmai

Kraskalo algoritme (ir daugelyje kitų algoritmų) svarbi vieta yra skirta pagrindinių aibių veiksmų atlikimui. Todėl šiame poskyryje susipažinsime su keliais efektyviais tokių operacijų realizavimo algoritmais.

Turime nesusikertančių aibių rinkinį

$$A = \{A_1, A_2, \ldots, A_N\}, \quad A_i \cap A_j = \emptyset, \;\; i \neq j\,.$$

Kiekvieną aibę $A_i = A_i(x_i)$ charakterizuoja vienas jos elementas $x_i \in A_i$.

Nagrinėkime tokius pagrindiniuss aibių veiksmus:

- *MakeSet(x)* – sukuria naują aibę, kuriai priklauso vienintelis elementas x. Kadangi visos A rinkinio aibės turi būti nesusikertančios, tai x negali priklausyti jokiai kitai jau egzituojančiai aibei A_i.
- *FindSet(x)* – randa aibę A_j, kuriai priklauso elementas x ir grąžina nuorodą į šį elementą.
- *UnionOfSets(x, y)* – sujungia dvi aibes, kurioms priklauso elementai x ir y. Operacija atliekama, jei šios aibės yra skirtingos. Abi senos aibės yra sunaikinamos, o naujosios aibės pagrindiniu elementu parenkamas kuris nors jai priklausantis elementas.

Aibę realizuosime panaudodami medžio duomenų struktūrą, jo šaknis bus pagrindiniu aibės elementu. Kiekvienas elementas saugo nuorodą į savo tėvą.

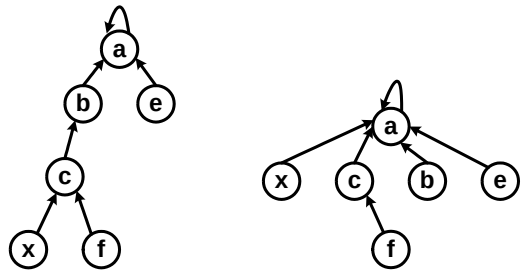
Tada *MakeSet(x)* sukuria naują medį, jį sudaro tik vienas elementas x, kurio tėvo rodyklė yra nukreipta į x.

Paprasčiausia *FindSet(x)* operacijos realizacija yra tokia – iš elemento x, eidami briaunomis, rodančiomis į elemento tėvą, pasiekiame medžio šaknį. Tačiau toks algoritmas yra neefektyvus, jei medis nėra subalansuotas. Todėl pateiksime sudėtingesnį algoritmą, kuriuo ne tik surandame aibę, kuriai priklauso elementas, bet ir sutrumpiname kelius nuo paieškos metu aplankytų viršūnių iki medžio šaknies.

Pažymėkime $p(x)$ elemento x tėvą.

FindSet(x)
begin
(1) **if** ($x \neq$ p(x)) p(x) = FindSet (p(x));
(2) **return** p(x)
end FindSet

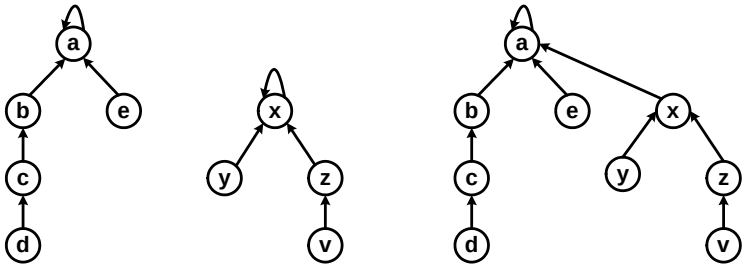
Algoritme naudojame rekursiją, jei x nėra medžio šaknis, tai ieškome kuriai aibei priklauso x tėvas. Po to pakeičiame rodyklės $p(x)$ reikšmę – ji nukreipiama į aibės pagrindinį elementą. Algoritmo darbas pavaizduotas 4.8 paveiksle.



a) b)

4.8 pav. Aibės paieška $FindSet(x)$ algoritmu: a) medis prieš vykdant x paiešką, b) pertvarkytas medis po paieškos algoritmo įvykdymo

Paprasčiausia *UnionOfSets(x, y)* operacijos realizacija yra tokia: vieno iš medžių šaknies tėvo rodyklę (priminsime, kad ji rodo į tą patį elementą) nukreipiame į kitos aibės šaknį (žr. 4.9 paveikslą)



a) b)

4.9 pav. Dviejų nesusikertančių aibių jungimas paprasčiausiu $UnionOfSets(x, y)$ algoritmu: a) du medžiai prieš vykdant sujungimą, b) medis po aibių sujungimo

Vykdydami įvairius algoritmus, pvz. Kraskalo algoritmą, ne tik sujungiame aibes, bet vėliau dar daug kartų tikriname, kokiai aibei priklauso vienas ar kitas elementas. Naujojo medžio aukštis bus mažiausias, jei sujungdami medžius dar atsižvelgsime ir į jų rangus. Naujai sukurtos aibės vienintelio elemento rangas yra lygus nuliui. Medžio šaknies rangas keičiasi tik tada, kai jungiamų medžių šaknų rangai yra vienodi. Tada naujai gauto medžio šaknies rangas didinamas vienetu.

Pažymėkime $rank(x)$ elemento x rangą.

UnionOfSets(x, y)
begin
(1) u = FindSet(x), v = FindSet(y);
(2) **if** (u $\neq$ v)
(3) **if** (rank(u) $>$ rank(v)) p(v) = u;
(4) **else** {
(5) p(u) = v;
(6) **if** (rank(u) == rank(v)) rank(v) = rank(v) +1
}
end UnionOfSets

Tarkime, kad viso atlikome m operacijų su aibėmis, tarp jų n kartų vykdėme *MakeSet(x)* veiksmą, tada blogiausio atvejo sudėtingumas yra $\mathcal{O}\big(m \log n\big)$.

4.3.5. Boruvkos algoritmas

Tai irgi gobšusis algoritmas. Paaiškinsime pagrindinę algoritmo idėją. Pradžioje, panašiai kaip Kraskalo algoritme, turime dengiančių medžių mišką F, kurį sudaro tik grafo viršūnės. Eiliniame algoritmo žingsnyje randame po vieną lengviausią briauną, išeinančią iš kiekvieno medžio. Kadangi kai kurios briaunos gali sutapti, tai užtenka nagrinėti miško poaibį $F_1 \subset F$, kad kiekviena lengva briauna būtų pasirinkta tik vieną kartą. Visas šias briaunas įtraukiame į minimalų dengiantį medį bei sujungiame kai kuriuos pomedžius. Procesą kartojame tol, kol parenkame $|V| - 1$ briauną.

Boruvkos algoritmas

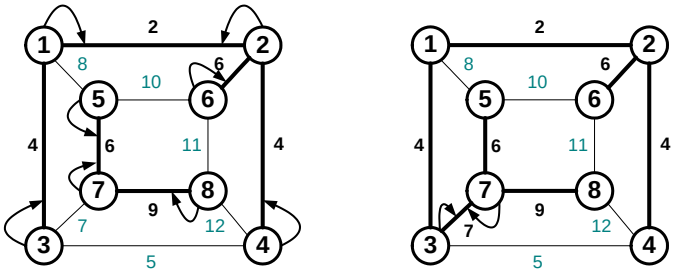
(1) $F = \{T_1, T_2, \ldots, T_n\}$, $T_i = \big(\{v_i\}, \emptyset\big)$, $B = \emptyset$;
(2) **while** ($|B| < |V| - 1$) {
(3) **for** ($T_i \in F$)
(4) Randame lengviausią išeinančią briauną e_i;
(5) Sudarome skirtingų e_i briaunų aibę L;
(6) $B = B \cup L$;
(7) Modifikuojame mišką F;
}

4.7 teorema. *Tegul G yra įvertintasis jungus grafas. Tada Boruvkos algoritmu randame minimalų dengiantį medį.*

Įrodymas. Imkime medį $T_i = (V_i, B_i)$ ir sudarykime grafo pjūvį $(V_i, V \setminus V_i)$, kuris yra suderintas su jau parinktų briaunų aibe B. Nauja parinkta briauna e_i

kerta šį pjūvį, taigi ji yra lengva. Toks teiginys yra teisingas visiems medžiams $T_i \in F_1$, taigi Boruvkos algoritme briaunos įtraukiamos į minimalų dengiantį medį naudojant mėlynąją taisyklę. □

4.6 pavyzdys. Minimalaus dengiančiojo medžio radimas Boruvkos algoritmu. Imkime grafą, pavaizduotą 4.2 paveiksle. Minimalaus dengiančiojo medžio formavimas Boruvkos algoritmu yra parodytas 4.10 paveiksle.



4.10 pav. Minimalaus dengiančiojo medžio radimas Boruvkos algoritmu