4.4. Specialieji grafų algoritmai

Šiame skyriuje susipažinsime su grafo viršūnių peržiūros algoritmais ir jų taikymais sprendžiant įvairius informatikos uždavinius. Kiekvieną grafo viršūnę galime pasiekti ir tiesiogiai, naudodami jos adresą, saugomą viršūnių masyve. Čia nagrinėsime grafo apėjimo algoritmus, kai iš vienos viršūnės galime patekti tik į jai gretimas viršūnes. Judėdami grafo briaunomis turime aplankyti visas likusias grafo viršūnes, be to kiekvieną viršūnę nagrinėjame tik vieną kartą. Susipažinsime su dviem svarbiausiais metodais:

- paieškos gilyn metodu,
- paieškos platyn metodu,

įvertinsime jų sudėtingumą ir parodysime, kaip šie metodai naudojami sprendžiant topologinio rūšiavimo ir trumpiausio kelio radimo uždavinius.

4.4.1. Topologinio rūšiavimo algoritmai

Topologinio rūšiavimo uždavinį suformulavome skyriuje, kuriame nagrinėjome rūšiavimo algoritmus. Tačiau naujasis uždavinys gerokai skiriasi nuo įprastinio skaičių ar abecėlinio rūšiavimo.

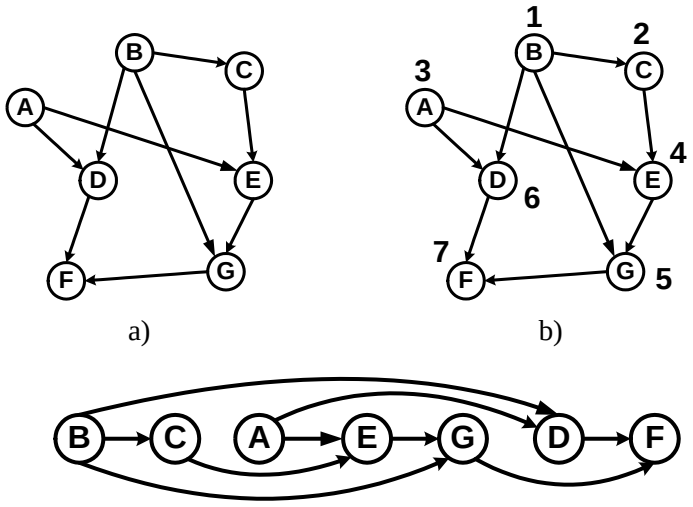
Pirmiausia pateiksime tikslesnį topologinio rūšiavimo uždavinio formulavimą. Turime orientuotą grafą $G = (V, E)$, kuriame nėra ciklų. Grafo viršūnes reikia sužymėti taip, kad kiekviena briauna jungtų mažesnio numerio viršūnę su didesnio numerio viršūne.

Topologinio rūšiavimo uždavinio sprendimo pavyzdys yra pateiktas 4.11 paveiksle.

Paieškos gilyn metodas

Šios paieškos strategija yra paprasta: iš duotosios viršūnės einame į jai gretimą, paieškos metu dar neaplankytą grafo viršūnę. Jei tokių viršūnių nėra, tai grįžtame vieną žingsnį atgal ir ieškome naujo kelio iš tėvo viršūnės. Taip surandame visas viršūnes, kurias galima pasiekti iš pasirinktos pradinės viršūnės. Jei grafas nėra jungus, tai algoritmą kartojame imdami naują dar neaplankytą pradinę viršūnę. Kadangi paieškos metu pirmiausia aplankome labiausiai nutolusias viršūnes, tai metodą vadiname *paieškos gilyn* metodu (angl. *depth first search*).

Kiekviena grafo viršūnė gali būti vienoje iš trijų būsenų (būsenas žymėsime skirtingomis spalvomis). Pradžioje visos viršūnės yra *neaplankytos* ir dažomos *balta* spalva. Kai viršūnė v pirmą kartą aplankoma, ji tampa *nenauja* ir dažoma



4.11 pav. Grafo viršūnių topologinis rūšiavimas: a) pradinis grafas, b) surūšiuotas grafas, c) grafo viršūnių išdėstymas tiesėje

pilka spalva. Laiko momentą, kada ji tapo nenauja, saugome masyvo elemente $d(v)$ (angl. *discovered*). Viršūnė nudažoma *juoda* spalva, kai išnagrinėjamos visos iš jos išeinačios briaunos, tokios viršūnės yra vadinamos *išsemtomis*. Laiko momentą, kada viršūnė tapo juoda, saugome masyvo elemente $f(v)$ (angl. *finished*).

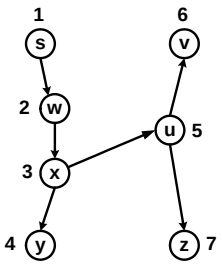
Paieškos kelius įsimename masyve π, jo elemento $\pi(v)$ reikšmė yra viršūnė u, iš kurios pirmą kartą aplankėme v, t.y. $\pi(v) = u$.

Paieškos gilyn algoritmas

DepthFirstSearch (G)
begin
(1) **for** ($v \in V$) {
(2) spalva(v) = *balta*;
(3) $\pi(v) =$ NULL;
}
(4) t = 0;
(5) **for** ($u \in V$)
(6) **if** (spalva(u) == *balta*)
(7) DFS_Visit(u);
end DepthFirstSearch

Pateikiame rekursyvų viršūnių aplankymo algoritmą.

DFS_Visit (u)
begin
(2) spalva(u) = *pilka*;
(3) t = t +1, d(u) = t;
(4) **for** ($v \in N(u)$)
(5) **if** (spalva(v) == *balta*) {
(6) $\pi(v) = u$;
(7) DFS_Visit(v);
}
(8) spalva(u) = *juoda*;
(9) t = t +1, f(u) = t;
end DFS_Visit



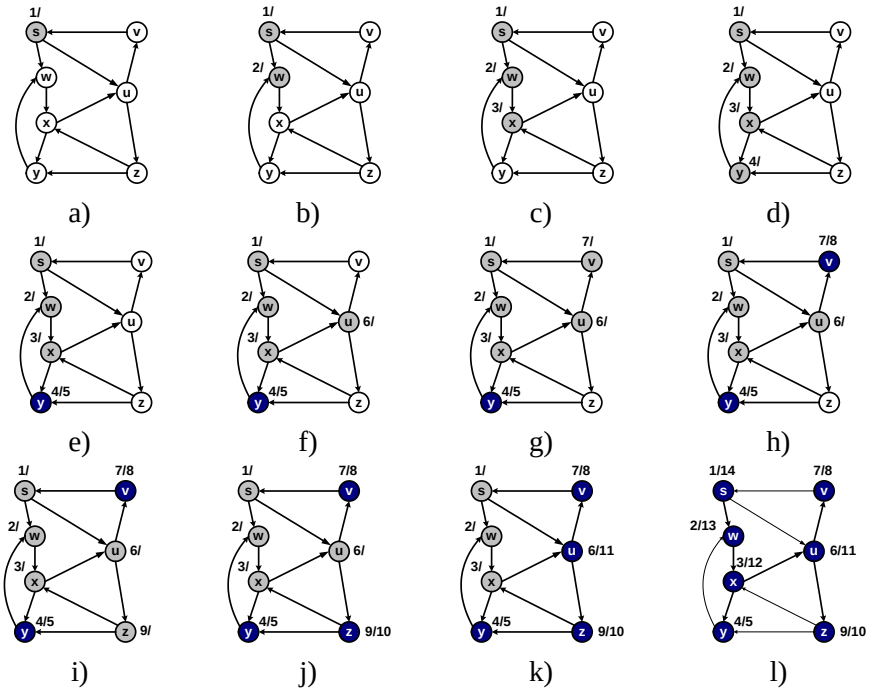
4.12 pav. Paieškos gilyn metodu aplankytų viršūnių eiliškumas ir keliai

4.7 pavyzdys. Grafo viršūnių lankymas paieškos gilyn metodu. Imkime grafą, pavaizduotą 4.13 paveikslo *a* dalyje. Jo viršūnes ieškome naudodami paieškos gilyn metodą. Viršūnių lankymo eiga po kiekvieno kreipinio į *DFS_Visit* funkciją yra pavaizduota paveikslo *a–l* dalyse. Viršūnese pateiktos $(d(v), f(v))$ reikšmės. 4.12 paveiksle pavaizduotas gautasis grafas $G_\pi = (V, E_\pi)$:

$$E_\pi = \{(\pi(v), v) : \ \ v \in V, \ \ \pi(v) \neq \text{NULL}\}\,.$$

Viršūnės numeris rodo jos suradimo eiliškumą.

Algoritmo sudėtingumo įvertinimas. Įvertinsime paieškos gilyn algoritmo sudėtingumą. Procedūroje *DepthFirstSearch* (2) ir (3) veiksmai atliekami $|V|$ kartų. (5) ciklas irgi kartojamas $|V|$ kartų ir kiekvienai viršūnei vieną kartą vykdome



4.13 pav. Grafo viršūnių aplankymas paieškos gilyn metodu

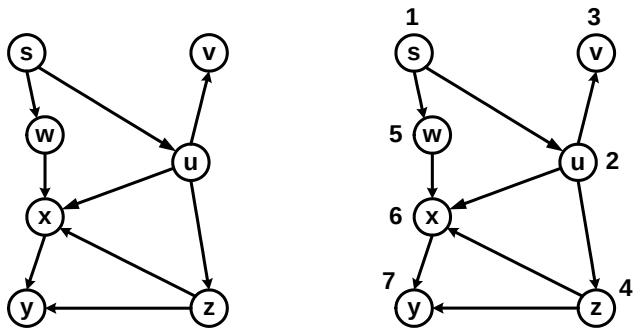
DFS_Visit procedūrą. Jos metu atliekame $\mathcal{O}(1)$ veiksmų (2), (3), (8) ir (9) algoritmo žingsniuose. (4) ciklas kartojamas $|N(v)|$ kartų, todėl bendra paieškos gilyn algoritmo apimtis yra $\mathcal{O}(|V| + |E|)$ veiksmų.

Grafo viršūnių topologinis rūšiavimas

Pabaigę paieškos gilyn algoritmą, randame grafą $G_\pi = (V, E_\pi)$. Norėdami gauti surūšiuotą viršūnių aibę modifikuojame *DFS_Visit* procedūrą, jos pabaigoje viršūnę u įterpiame į tiesinio sąrašo pradžią:

(8) spalva(u) = *juoda*;
(9) t = t +1, f(u) = t;
(10) List.InsertHead (u);
end DFS_Visit

4.8 pavyzdys. Grafo viršūnių topologinis rūšiavimas. Nagrinėkime grafą, pateiktą 4.14*a* paveiksle . Jo topologiškai surūšiuotų viršūnių sąrašas pavaizduotas 4.14*b* paveiksle.



a) b)

4.14 pav. Grafo viršūnių topologinis rūšiavimas: a) pradinis grafas, b) surūšiuotos viršūnės

4.4.2. Trumpiausio kelio radimas labirinte

Turime grafą $G = (V, E)$. Grafo briaunos nėra įvertintos, todėl laikysime, kad visų briaunų ilgiai yra lygus vienetui. Reikia rasti trumpiausią kelią nuo duotosios viršūnės $u \in V$ iki likusių grafo viršūnių. Kelio ilgis sutampa su tarpinių briaunų skaičiumi. Įdomus šio uždavinio atvejis yra trumpiausio kelio paieška labirinte, kai žinome įėjimo viršūnę ir reikia rasti kelią, vedantį link išėjimo.

Aišku ir tokį uždavinį galime spręsti Deikstros metodu, tačiau naujasis uždavinys yra paprastesnis, nes visų briaunų ilgiai yra vienodi. Todėl galime tikėtis sukurti efektyvesnius tokio uždavinio sprendimo metodus.

Paieškos platyn metodas

Šios paieškos strategija yra tokia: pirmiausia nagrinėjame viršūnes, gretimas pradinei viršūnei u, po to kaimynų gretimas viršūnes ir taip toliau, kol surandame visas viršūnes, pasiekiamas iš viršūnės u. Grafas gali būti orientuotas arba neorientuotas. Tokia strategija yra vadinama *paieškos platyn* metodu (angl. *breadth first search*).

Panašiai kaip ir paieškos gilyn metode viršūnė gali būti nudažyta viena iš trijų spalvų: *baltos* – jei ji dar nesurasta, *pilkos* – viršūnė jau aplankyta, bet dar ne visi jos kaimynai yra patikrinti, ir *juodos* – kai patikrintos visos gretimos viršūnės. Visos pilkos viršūnės sudaro paieškos frontą, o juodos viršūnės yra apgaubtos šio fronto. Jeigu briauna $(v, w) \in E$ ir v yra juodos spalvos viršūnė, tai w gali būti tik juodos arba pilkos spalvos. Taigi neaplankytų (baltų) viršūnių užtenka ieškoti tik pilkos spalvos viršūnių aplinkose $N(v)$.

Pilkos spalvos viršūnes saugome *eilėje* Q (priminsime, kad eilėje galioja FIFO

principas: iš sąrašo pirmiausia išimamas tas elementas, kuris anksčiausiai pateko į eilę).

Paieškos platyn algoritmas

```
BreadthFirstSearch (G)
begin
 (1) for ( v ∈ V ) {
        (2) spalva (v) = balta;
        (3) π(v) = NULL,  d(v) = ∞;
     }
 (4) d(u) = 0,  Q.InsertRear (u);
 (5) while ( Q ≠ ∅ ) {
           (6) v = Q.TakeHead();
           (7) for ( w ∈ N(v) )
                  (8) if ( spalva (w) == balta ) {
                         (9) spalva (w) = pilka;
                         (10) π(w) = v,  d(w) = d(v) + 1;
                         (11) Q.InsertRear (w);
                      }
           (12) spalva (v) = juoda;
    }
end BreadthFirstSearch
```

Trumpiausią kelią nuo viršūnės u iki viršūnės v randame panaudodami masyvo π reikšmes. Pateikiame algoritmą, kuris šį kelią spausdina atvirkščia tvarka nuo paskutinės viršūnės v iki pradinės viršūnės u.

```
(1) w = v;
(2) while ( w ≠ NULL) {
        (3) print (w);
        (4) w = π(w);
    }
```

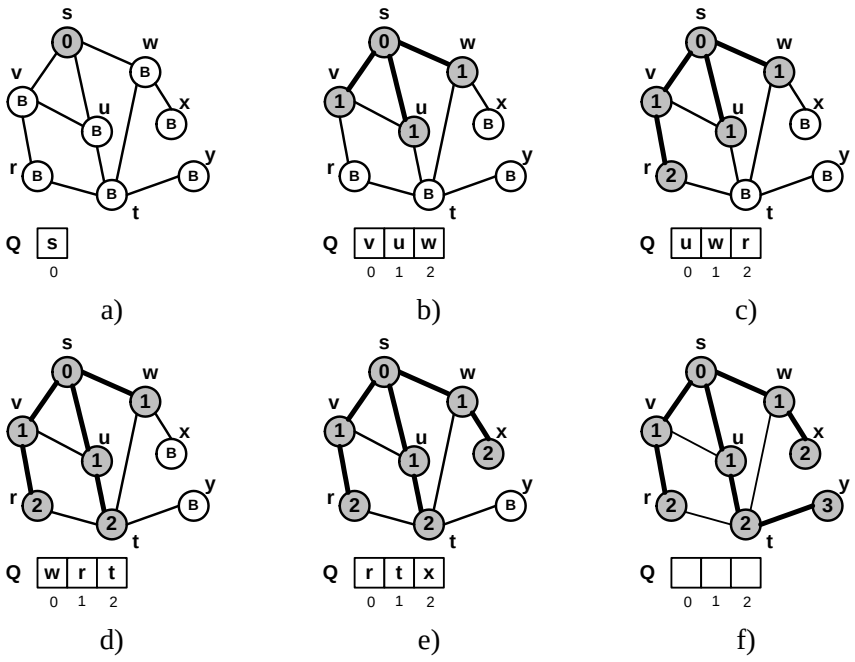
Jeigu norime atspausdinti trumpiausią kelią nuo pradinės viršūnės u iki viršūnės v, tai algoritmo (3) žingsnyje viršūnę w užrašome į tiesinio sąrašo pradžią, o po to atspausdiname gautąjį sąrašą. Taip pat galime sudaryti spausdinimo procedūrą, naudojant rekursiją.

4.9 pavyzdys. Trumpiausio kelio radimas. Imkime grafą, pavaizduotą 4.15 paveikslo *a* dalyje. Jo viršūnes aplankome paieškos platyn metodu,

pradinė viršūnė yra s. Paveikslo *a-f* dalyse pavaizduotas trumpiausio kelio formavimas po kiekvieno paieškos algoritmo (5) ciklo žingsnio. Paveikslo *f* dalyje pavaizduotas grafas $G_\pi = (V, E_\pi)$

$$E_\pi = \{(\pi(v), v) : \ \ v \in V, \ \ \pi(v) \neq \mathrm{NULL}\},$$

kuris ir apibrėžia trumpiausius kelius nuo pradinės viršūnės s iki likusių grafo G viršūnių. Viršūnės numeris rodo jos atstumą nuo s.



4.15 pav. Trumpiausio kelio radimas paieškos platyn metodu

Algoritmo sudėtingumo įvertinimas. Masyvų pradinių reikšmių skaičiavimo apimtis yra $\mathcal{O}(|V|)$ veiksmų. Kiekviena grafo viršūnė yra talpinama į eilę ne daugiau nei vieną kartą, todėl ir išimta iš eilės ji gali būti tik vieną kartą. Elemento talpinimo į eilės galą ir šalinimo iš eilės pradžios sudėtingumas yra $\mathcal{O}(1)$. Kiekvienos viršūnės kaimynus nagrinėjame tik vieną kartą, kai viršūnę šaliname iš eilės, todėl (8) – (11) žingsniai yra atliekami $|E|$ kartų orientuotame grafe ir $2\,|E|$ kartus neorientuotame grafe. Taigi paieškos platyn algoritmo apimtis yra $\mathcal{O}(|V| + |E|)$.

Algoritmo teisingumo analizė. Turime grafą $G = (V, E)$, kurio visų briaunų ilgiai yra lygūs vienetui. Pažymėkime $\delta(u, v)$ ilgį trumpiausio kelio nuo viršūnės u iki kitos viršūnės v, be to tarsime, kad $\delta(u, v) = \infty$, jei v yra nepasiekiama iš u.

Tada paieškos platyn algoritmo teisingumas seka iš tokios lemos:

4.2 lema. *Kiekvienam natūriniam skaičiui k egzistuoja toks paieškos platyn algoritmo vykdymo momentas, kai teisingi šie teiginiai*

1. *Visos viršūnės iki kurių atstumas yra mažesnis už k yra juodos spalvos, lygus k – pilkos spalvos ir didesnis už k – baltos spalvos.*
2. *Eilėje Q yra saugomos visos pilkos viršūnės.*
3. *Masyvo d elemento reikšmė $d(v)$ yra lygi trumpiausio kelio ilgiui, jei v yra juoda arba pilka viršūnė.*
4. *Jeigu v yra pilka arba juoda viršūnė, tai $\delta(u, \pi(v)) = \delta(u, v) - 1$, o briauna $(\pi(v), v) \in E$.*