1 skyrius

Algoritmai grafuose

1.1. Specialieji grafų algoritmai

Šiame skyriuje susipažinsime su grafo viršūnių peržiūros algoritmais ir jų taikymais, spręsdami 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,

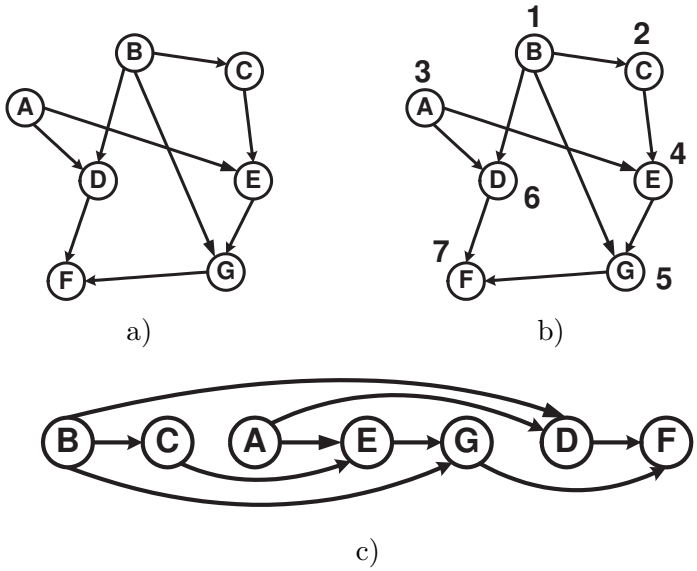
ir įvertinsime jų sudėtingumą bei parodysime, kaip šie metodai taikomi sprendžiant topologinio rūšiavimo ir trumpiausio kelio radimo uždavinius.

1.1.1. Topologinio rūšiavimo algoritmai

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

Pirmiausia pateiksime tikslesnį topologinio rūšiavimo uždavinio formulavimą. Turime orientuotąjį 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 pateiktas 1.1 paveiksle.



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

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 žengiame vieną žingsnį atgal ir ieškome naujo kelio iš tėvo viršūnės. Taip surandame visas viršūnes, kurias galima pasiekti iš pasirinktos pradinės viršūnės. Jei grafas nėra jungusis, 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 aplankoma pirmą kartą, ji tampa *nenauja* ir dažoma *pilka* spalva. Laiką, kada ji tapo nenauja, saugome masyvo elemente $d(v)$ (angl. *discovered*). Viršūnė nudažoma *juoda* spalva, kai išnagrinėjamos visos iš jos išeinančios briaunos, tokios viršūnės yra vadinamos

išsemtosiomis. 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

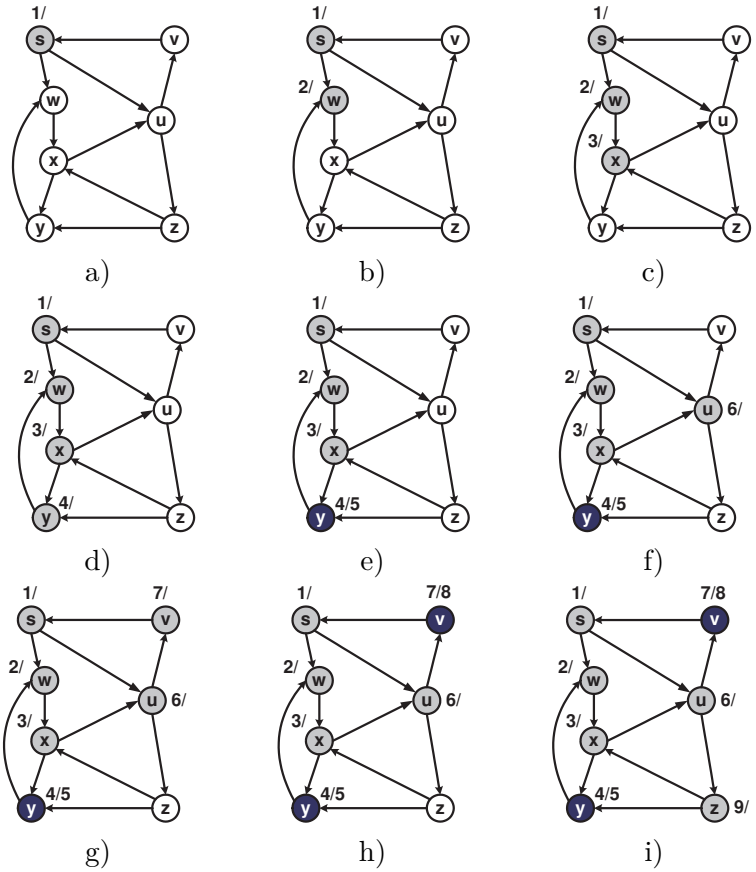
```
DepthFirstSearch (G)
begin
 (1)  for  ( v ∈ V ) do
 (2)       spalva(v) = balta;
 (3)       π(v) = NULL;
      end do
 (4)  t = 0;
 (5)  for  ( u ∈ V ) do
 (6)       if  ( spalva(u) == balta ) then
 (7)          DFS_Visit(u);
           end if
      end do
end DepthFirstSearch
```

Pateikiame rekursyvųjį viršūnių aplankymo algoritmą.

```
DFS_Visit (u)
begin
 (2)  spalva(u) = pilka;
 (3)  t = t +1,   d(u) = t;
 (4)  for  ( v ∈ N(u) ) do
 (5)       if  ( spalva(v) == balta ) then
 (6)          π(v) = u;
 (7)          DFS_Visit(v);
           end if
      end do
 (8)  spalva(u) = juoda;
 (9)  t = t +1,   f(u) = t;
end DFS_Visit
```

1.1 pavyzdys. Grafo viršūnių lankymas paieškos gilyn metodu. Imkime grafą, pavaizduotą 1.2 paveikslo *a* dalyje. Jo viršūnes ieškome taikydami paieškos gilyn metodą. Viršūnių lankymo eiga po kiekvieno kreipinio į *DFS_Visit* funkciją pavaizduota paveikslo *a*–*i* dalyse ir 1.3 paveikslo *j*–*l* dalyse. Viršūnėse pateiktos $(d(v), f(v))$ reikšmės. 1.3 paveikslo *m* dalyje taip pat pavaizduotas gautasis grafas $G_\pi = (V, E_\pi)$:

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



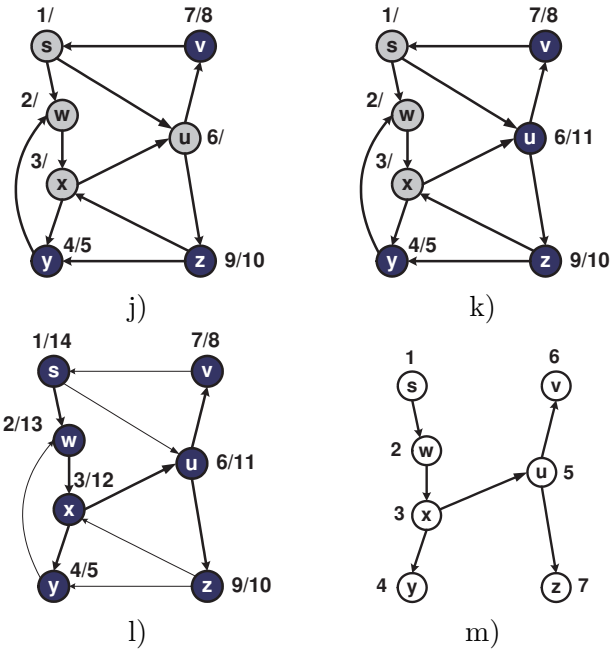
1.2 pav. Grafo viršūnių aplankymas paieškos gilyn metodu (algoritmo pradžia)

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

Topologinis grafo viršūnių rūšiavimas

Baigę paieškos gilyn algoritmą, randame grafą $G_\pi = (V, E_\pi)$. Norėdami gauti surūšiuotą viršūnių aibę, modifikuojame *DFS_Visit* procedūrą, jos



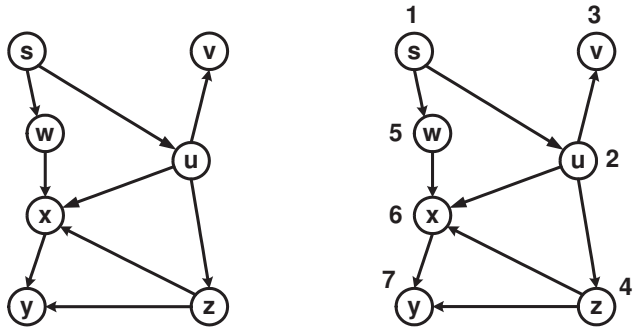
1.3 pav. Grafo viršūnių aplankymas paieškos gilyn metodu (algoritmo tęsinys) ir aplankytų viršūnių eiliškumas bei keliai

pabaigoje viršūnę u įterpiame į tiesinio sąrašo pradžią:

DFS_VisitSort (u)
begin
(2) spalva(u) = *pilka*;
(3) t = t +1, d(u) = t;
(4) **for** ($v \in N(u)$) **do**
(5) **if** (spalva(v) == *balta*) **then**
(6) $\pi(v) = u$;
(7) DFS_VisitSort(v);
end if
end do
(8) spalva(u) = *juoda*;
(9) t = t +1, f(u) = t;
(10) List.InsertHead (u);
end DFS_VisitSort

1.2 pavyzdys. Grafo viršūnių topologinis rūšiavimas. Nagrinėkime grafą, pateiktą 1.4 *a* paveiksle. Jo topologiškai surūšiuotų

viršūnių sąrašas pavaizduotas 1.4 *b* paveiksle.



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

1.1.2. Trumpiausio kelio radimas labirinte

Turime grafą $G = (V, E)$. Grafo briaunos nėra įvertintos, todėl teigsime, kad visų briaunų ilgiai lygūs 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 išėjimo link.

Aišku, ir tokį uždavinį galime spręsti Dijkstros 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, paskui kaimynų gretimas viršūnes ir taip toliau, kol surandame visas viršūnes, pasiekiamas iš viršūnės u. Grafas gali būti orientuotas arba neorientuotas. Tokia strategija 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 ) do
 (2)       spalva (v) = balta;
 (3)       π(v) = NULL,   d(v) = ∞;
      end do
 (4)  d(u) = 0,   Q.InsertRear (u);
 (5)  while  ( Q ≠ ∅ ) do
 (6)       v = Q.TakeHead();
 (7)       for  ( w ∈ N(v) ) do
 (8)            if  ( spalva (w) == balta ) then
 (9)               spalva (w) = pilka;
 (10)              π(w) = v,   d(w) := d(v) + 1;
 (11)              Q.InsertRear (w);
                end if
           end do
 (12)      spalva (v) = juoda;
      end do
end BreadthFirstSearch
```

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

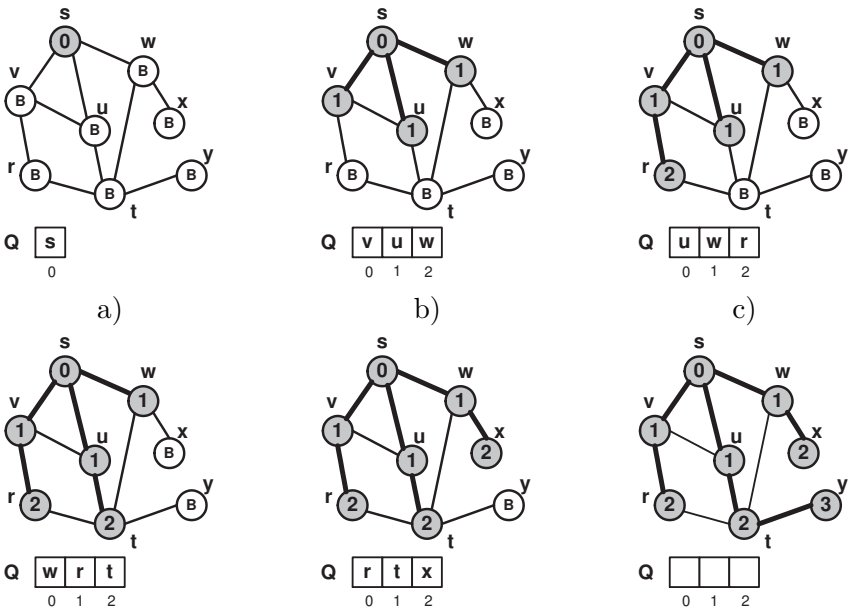
```
PrintPath ()
begin
 (1)  w = v;
 (2)  while  ( w ≠ NULL) do
 (3)          print (w);
 (4)          w = π(w);
      end do
end PrintPath
```

Jeigu norime išspausdinti trumpiausią kelią nuo pradinės viršūnės u iki viršūnės v, tai algoritmo (3) žingsniu viršūnę w užrašome į tiesinio sąrašo pradžią, o paskui išspausdiname gautąjį sąrašą. Taip pat galime sudaryti spausdinimo procedūrą, naudodami rekursiją.

1.3 pavyzdys. Trumpiausio kelio radimas. Imkime grafą, pavaizduotą 1.5 paveikslo *a* dalyje. Jo viršūnes aplankome paieškos platyn metodu, kai pradinė viršūnė yra s. Paveikslo *a–f* dalyse pavaizduota, kaip formuojamas trumpiausias kelias po kiekvieno paieškos algoritmo (5) ciklo žingsnio. Paveikslo *f* dalyje pavaizduotas grafas $G_\pi = (V, E_\pi)$

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

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



1.5 pav. Trumpiausio kelio radimas paieškos platyn metodu

Algoritmo sudėtingumo įvertinimas. Masyvų pradinių reikšmių skaičiavimo apimtis yra $\mathcal{O}(|V|)$ veiksmų. Kiekviena grafo viršūnė eilėje būna ne daugiau nei vieną kartą, todėl ir išimta iš eilės ji gali būti tik vieną kartą.

Elemento pateikimo eilės gale 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 orientuotajame grafe ir $2|E|$ kartus neorientuotajame 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. Trumpiausio kelio nuo viršūnės u iki kitos viršūnės v ilgį pažymėsime $\delta(u, v)$, be to, tarsime, kad $\delta(u, v) = \infty$, jei v yra nepasiekiama iš u. Kol kas dar nežinome, ar visada pilkoms ir juodoms viršūnėms galioja lygybė $d(v) = \delta(u, v)$. Paieškos platyn algoritmo teisingumas seka iš tokios lemos (žr. [**?**]).

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

1. *Visos viršūnės, atstumas iki kurių yra mažesnis už k, yra juodos spalvos, atstumas 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 $(\pi(v), v) \in E$.*

Įrodymas. Lemą įrodysime taikydami matematinės indukcijos metodą. Pirmiausia patikrinsime, kad visi teiginiai yra teisingi, kai $k = 0$. Tikrai, tada tik pradinė viršūnė u yra pilkos spalvos viršūnė, visos kitos viršūnės yra baltos, o atstumas iki jų yra ne mažesnis už vienetą. Viršūnę u saugome eilėje Q, o $d(u) = 0$. Be to, $\pi(u) = \text{NULL}$, todėl ir (4) lemos teiginys yra teisingas.

Dabar tarkime, kad lema yra teisinga, kai $k = j$. Įrodysime, kad tada egzistuoja toks paieškos platyn algoritmo vykdymo momentas, kai visi lemos teiginiai yra teisingi ir kai $k = j + 1$. Pažymėkime $Q(j), W(j)$ pilkų ir baltų viršūnių aibes, kurias gauname vykdydami $k = j$ algoritmą.

Vykdydami algoritmą iš Q, išimame pilkas viršūnes, jų baltas kaimynes dažome pilka spalva ir dedame į eilės Q galą. Išimtas viršūnes nudažome juoda spalva. Remiantis eilės duomenų struktūros savybėmis, iš Q pirmiausia bus išimtos anksčiausiai ten patekusios viršūnės. Nagrinėkime momentą,

kai jau išimtos visos pilkos viršūnės, ten patekusios prieš pradedant $(j+1)$ etapą, ir tik jos, t. y. aibės $Q(j)$ viršūnės.

Juodos spalvos viršūnėmis tapo visos $Q(j)$ viršūnės iki šiol buvusios pilkomis. Jų $d(v)$ reikšmė nepasikeitė, todėl, remiantis lemos (3) teiginiu, $d(v) = j$. Taigi dalis lemos (1) teiginio yra teisinga – jei atstumas iki viršūnės yra mažesnis už $(j+1)$, tai viršūnė yra juoda.

Dabar parodysime, kad jei $d(w) = j+1$, tai w yra pilkos spalvos viršūnė. Pilkomis viršūnėmis tapo tos baltos viršūnės $w \in W(j)$, kurios buvo gretimos eilėje $Q(j)$ saugotoms viršūnėms. Be to, remiantis indukcine prielaida, $d(v) = j$, jei $v \in Q(j)$. Todėl gauname, kad naujoms pilkoms viršūnėms teisinga lygybė

$$d(w) = d(v) + 1 = j + 1.$$

Lieka įsitikinti, kad šis atstumas apskaičiuotas teisingai. Tiesiogine briauna (v, w) sujungtų viršūnių atstumams galioja nelygybė

$$\delta(u, w) \leqslant \delta(u, v) + 1\,,$$

nes galbūt trumpiausias kelias nuo u iki w neina per v. Remdamiesi šia nelygybe ir įverčiais

$$\delta(u, v) = j, \text{ jei } v \in Q(j)\,,$$
$$\delta(u, w) \geqslant j + 1, \text{ jei } w \in W(j),$$

gauname, kad į eilę Q patekusių viršūnių atstumai yra $\delta(u, w) = j + 1$. Visos šios viršūnės yra pilkos, jų atstumai įvertinti teisingomis reikšmėmis $d(w) = j + 1$.

Dabar įrodysime, kad pilkomis tapo visos viršūnės, kurių atstumai iki u yra lygūs $(j+1)$. Tarkime, kad $\delta(u, w) = j + 1$, tada būtinai egzistuoja tokia gretima viršūnė v, kad $\delta(u, v) = j$. Iš tiesų, trumpiausiame kelyje nuo u iki w imkime paskutinę briauną (v, w). Tada turime j ilgio kelią, vedantį į v, o trumpesnio kelio ir negali būti, atsižvelgiant į indukcinę prielaidą. Vadinasi, v buvo pilka viršūnė ir priklausė eilei $Q(j)$, todėl kažkuriuo algoritmo žingsniu w bus gretima kuriai nors eilėje $Q(j)$ saugotai viršūnei (nebūtinai v) ir ji bus pateikta aibėje $Q(j+1)$.

Iš pateiktos analizės ir paieškos platyn algoritmo išeina, kad taip pat galioja (3) ir (4) lemos teiginiai. □