2.8 Grafo viršūnių peržiūros metodai

Daugelio grafų teorijos uždavinių sprendimo algoritmų pagrindą sudaro sisteminga grafo viršūnių peržiūra, t.y. toks grafo viršūnių apėjimas, kad kiekviena viršūnė nagrinėjama vienintelį kartą. Todėl labai svarbus uždavinys yra rasti gerus grafo viršūnių peržiūros meodus. ***Apskritai kalbant, viršūnių 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 viršūnių peržiūros metodai, tenkinantys pateiktus reikalavimus, yra ***paieškos gilyn metodas*** ir ***paieškos platyn metodas***.

Paieška gilyn

Paieškos gilyn (rus. poisk v glubinu; angl. Depth first search) metodą pirmasis 1972 m. pasiūlė R.Tarjanas[1]. Metodo idėja yra labai paprasta. Pradžioje visos grafo viršūnės yra ***naujos*** (***neaplankytos***). Tarkime, kad paieška pradedama iš viršūnės v_0. Viršūnė v_0 tampa ***nenauja***, 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 ***nenauja***) 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 baigiame, 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 gretimumo struktūroje išdėstytos jų numerių didėjimo tvarka:

1: 2, 3

2: 1, 3, 4, 5

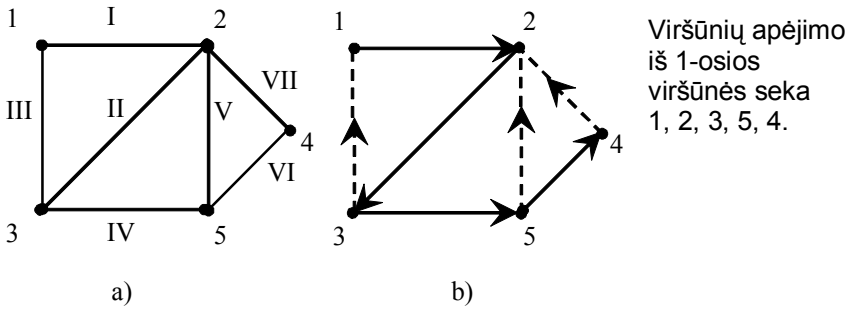
3: 1, 2, 5

4: 2, 5

5: 2, 3, 4

[1] Tarjan R. A. Depth first search and linear graph algorithms. SIAM J. Comput., 1972, 1, .p.p. 146 – 160.

Pastaba. Čia ir toliau, užrašant gretimumo struktūras, aibę žymintys riestiniai skliaustai – praleisti.



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

Paieškos gilyn metu pirmiausia iš 1-osios viršūnės einame į 2-ąją 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-ąją viršūnę. Kadangi pirmoji viršūnė nenauja (aplankyta), tai iš 2-osios viršūnės bandome eiti į kitą jai gretimą 3-iąją viršūnę. Trečioji viršūnė nauja, todėl ją aplankome ir peržiūrą tęsiame iš 3-osios viršūnės. Iš 3-iosios viršūnės bandome eiti į pirmąją jai gretimą viršūnę, – į 1-ąją viršūnę. Ši viršūnė nenauja. Tada bandome eiti į kitą 3-ajai viršūnei gretimą 2-ąją viršūnę. Ši viršūnė taip pat aplankyta, todėl bandome eiti į paskutiniąją 3-iajai viršūnei gretimą 5-ąją viršūnę. Ši viršūnė yra nauja, todėl peržiūrą dabar tęsime iš 5-osios viršūnės. Iš 5-osios viršūnės, po bandymų keliauti į 2-ąją ir 3-iąją viršūnes, ateisime į 4-ąją 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-ąją viršūnę atėjome iš 5-osios viršūnės. 5-ąjai viršūnei visos gretimos viršūnės yra nenaujos, 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-iąją viršūnę atėjome iš 2-osios viršūnės. 2-ajai viršūnei dar nenagrinėta gretima yra 5-toji viršūnė. Tačiau ši viršūnė nenauja, todėl 2-oji viršūnė tampa išsemta ir paiešką tęsiame iš 1-osios viršūnės. Iš 1-osios viršūnės bandome eiti į 3-iąją, 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.

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*** (***neaplankyta***) 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;

Paieškos gilyn su mažiausia atminties apimtimi organizavimas

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

n – grafo viršūnių skaičius,

m – grafo briaunų (lankų) skaičius,

L [1.. 2*m*] (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 nenagrinėtos viršūnės, gretimos viršūnei *i*, $i = \overline{1,n}$; tiksliau briauna (jei tokia yra) $(i, L[\,fst[i]])$ – paieškos gilyn metu dar nenagrinėta 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:

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

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

Kintamasis k žymi viršūnę, iš kurios tęsiame 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*:

$$t = \begin{cases} true, \text{ jei paieška gilyn baigta}; \\ false, \text{ jei paieška gilyn nebaigta}. \end{cases}$$

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 nuorodu 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 {virsune u nauja}
            begin { Nagrinėti viršūnę u }
               writeln (u);
               prec [u] := k;{į 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 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;
```

Paieška gilyn, nenaudojant rekursijos

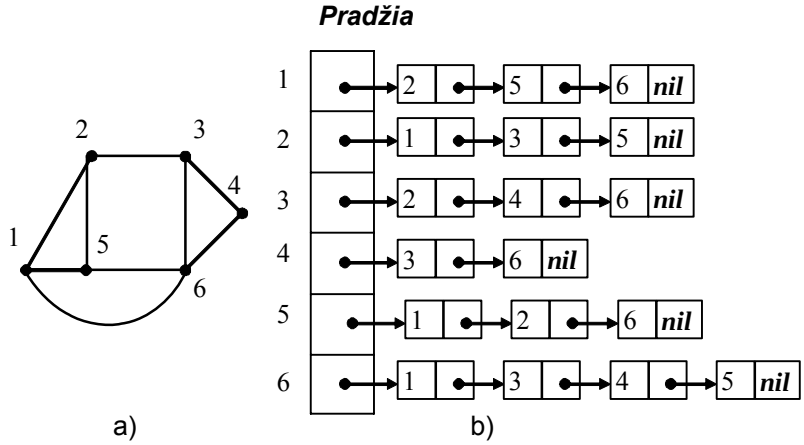
Tarkime, (n,m)-grafas – $G=(V,U)$ nusakytas gretimumo struktūra $N(v)$, $v \in V$, čia $N(v)$ – viršūnei *v* gretimų viršūnių aibė.

Aibes $N(v)$ vaizduosime ***vienryšiais 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 angliškų 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*:= ∅}

top := 0*;*

Elemento patalpinimas į steką {*E* ⇐ *naujas elementas*}

top := *top* + 1*;*
if top <= *n then E* [*top*] := *naujas elementas*
else "persipildymas";

Elemento pašalinimas iš steko {*y* ⇐ *E*}

if top = 0 *then "stekas tuščias"*
else begin y := *E* [*top*]*; top* := *top* – 1*; end;*

Viršutinio steko elemento nuskaitymas, neperskaičiuojant steko viršūnės {u := 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*]
{*Paieška gilyn grafe iš viršūnės v. Nerekursyvinė procedūros gylis versija* (*žr.* 2.*8.*1.1 *paragrafą*).
Tarkime, kad paieškos proceso pradžioje P[*v*] = *rodyklė* (*ląstelės adresas*) *į pirmąjį N*(*v*) *elementą kiekvienai viršūnei v* ∈ *V.*
Masyvai P ir Naujas – globalieji.}

```
begin
   STEK := ∅; STEK ⇐ v; nagrinėti v;
   Naujas [v] := false;
   while STEK ≠ ∅ 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;
               nagrinėti t;
               Naujas[t] := false;
            end
                  else {viršūnė t išsemta}
            {Viršūnę t šalinti iš steko, t.y. šalinti viršutinį steko elementą.}
```

```
            t ⇐ STEK;
        end; {while}
end; {gylis2}
```

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***, ***neperžiūrėtos***). 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 nagrinėtoms viršūnėms, 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 nagrinėtoms viršūnėms, t.y. viršūnės, kurios nuo viršūnės v_0 nutolę atstumu, lygiu *k*. Paieška platyn baigiama, 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, kuriame 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 dviem parametrais: *r* ir *f*, – *r* žymi eilės galą, o *f* – eilės pradžią (žr. 2.8.3 pav.), tiksliau, *f* rodo į prieš pirmąjį eilės elementą esančią ląstelę.

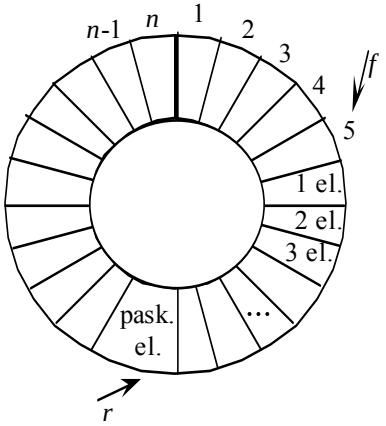
Aptarsime operacijas su eile. Prie operacijos riestiniuose skliaustuose rašysime simbolinį šios operacijos žymėjimą.

Eilei priskiriama tuščia aibė { *Eilė* := ∅}

r := 1; *f* := 1;

Ar Eilė tuščia? { *Eilė* = ∅}

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
      Eilė := ∅;
      Eilė ⇐ v;
      naujas [v] := false;
      while Eilė ≠ ∅ do
      begin
         p ⇐ Eilė;
         aplankyti (nagrinėti) p;
         for u ∈ N(p) do
         if naujas [u] then
                     begin
                        Eilė ⇐ 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 jungiąsias 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 "***paieška 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

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

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.

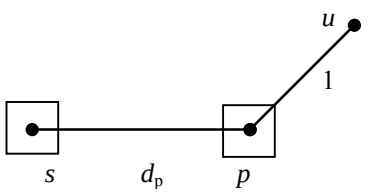
Pradžioje visos grafo viršūnės ***naujos***. Tą žymi masyvo *naujas* elementas:

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

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 $d_i = m+1$, $i = \overline{1,n}$. Vadinasi, jei grafas yra nejungusis, tai, įvykdžius žemiau pateiktą procedūrą, masyvo *d* elementai, atitinkantys grafo viršūnes, nepriklausančias viršūnės *s* jungiąjai 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 kelias nuo viršūnės *s* iki viršūnės *u***

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

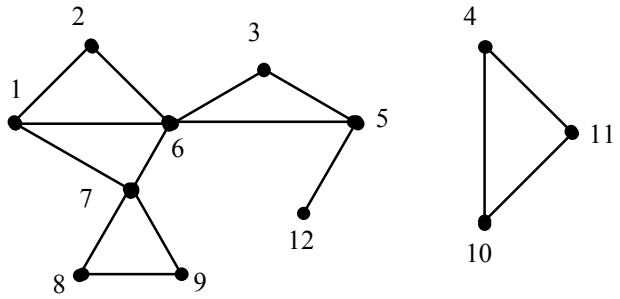
Aišku, kad pagal šias formules apskaičiuosime visus masyvo *d* ir *prec* elementus, atitinkančius paieškos platyn *k*-ojo žingsnio naujoms viršūnėms.

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 viršūnės s iki likusių grafo
viršūnių besvoriniame grafe, kai grafas nusakytas L ir lst masyvais.
Formalūs parametrai:
   s – pradinė kelio viršūnė,
   n – grafo viršūnių 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 viršūnę v ateina iš viršūnės 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 = ∅ }
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 ← 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 ← 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 keliai**

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

	1	2	3	4	5	6	7	8	9	10	11	12
d:	0	1	2	16	2	1	1	2	2	16	16	3,

	1	2	3	4	5	6	7	8	9	10	11	12
prec:	1	1	6	0	6	1	1	7	7	0	0	5.

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 viršūnes.

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

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