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 metodus. ***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***.

2.8.1. 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.

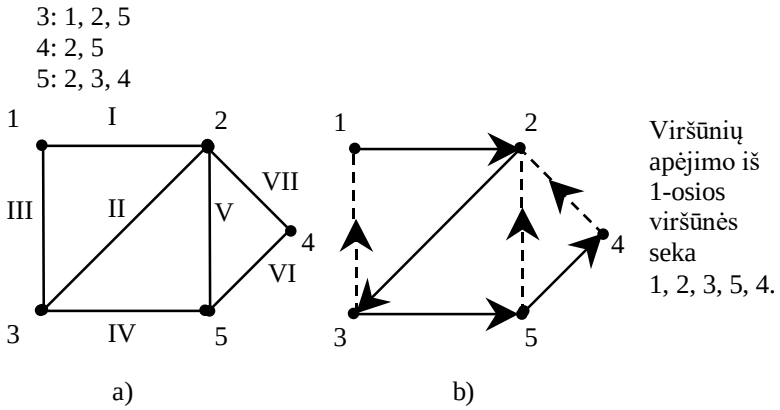
Paiešką baigiame, kai pradinė paieškos viršūnė v_0 tampa ***išsemta*** viršūne.

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

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



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-ąjai 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.

2.8.1.1. 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 ∈ N (v) do
        if naujas[u] then gylis (u);
    end;
```

Paieška gilyn grafe, kuris gali būti ir nejungusis, bus vykdoma taip:

```
    begin
        for v∈V do naujas [v] := true; {inicializacija}
        for v ∈ V do
            if naujas[v] then gylis (v);
    end;
```

2.8.1.2. 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) $(i, L[fst[i]])$ yra 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 {virsunė 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;
```

2.8.1.3. Paieška gilyn, nenaudojant rekursijos

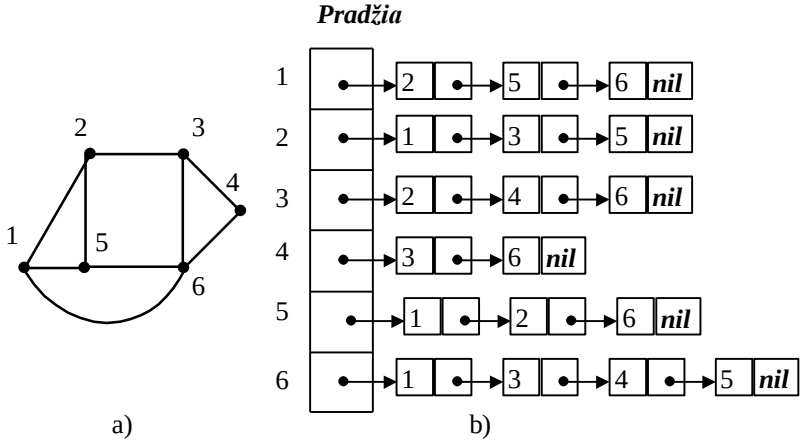
Tarkime, (n,m)-grafas – $G=(V,U)$ nusakytas gretimumo struktūra $N(v)$, $v \in V$, čia $N(v)$ – viršūnei *v* gretimų viršūnių aibė.

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 $\{E := \varnothing\}$

top := 0;

Elemento patalpinimas į steką $\{E \Leftarrow$ *naujas elementas*$\}$

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

Elemento pašalinimas iš steko $\{y \Leftarrow E\}$

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}
```

2.8.2. 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.

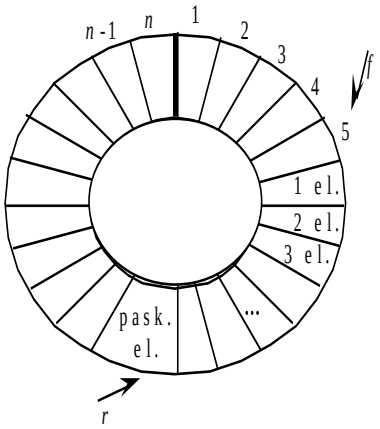
Aptarsime operacijas su eile. Prie operacijos riestiniuose skliaustuose rašysime simbolinį šios operacijos žymėjimą.

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* ⇐ *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* ⇐ *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 "***paiešką 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.

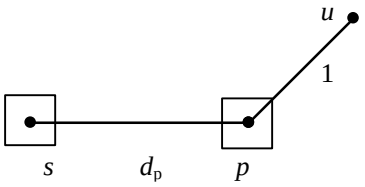
Pradžioje visos grafo viršūnės ***naujos***. Tą žymi masyvo *naujas* elementas:

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

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 $prec[u] = p$.

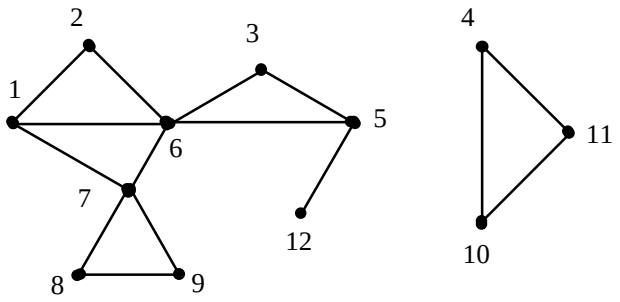
Aišku, kad pagal šias formules apskaičiuosime visus masyvo *d* ir *prec* elementus, atitinkančius paieškos platyn *k*-ojo žingsnio naujoms viršūnėms.

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).

2.10. Dvidalis grafas

Kaip buvo minėta aukščiau, (n,m)-grafas – $G=(V,U)$ yra dvidalis, jei jo viršūnių aibę *V* galima išskaidyti į du poaibius *A* ir *B* taip, kad visų grafo *G* briaunų galai priklausytų skirtingiems poaibiams. Todėl dvidalį grafą dažnai žymime $G=(A,B,U)$. Čia aptarsime du dvidalio grafo uždavinius:

- dvidalio grafo kriterijaus uždavinį,
- dalinio dvidalio grafo konstravimo uždavinį.

Dvidalio grafo kriterijaus uždavinys. Tarkime, kad duotas grafas $G=(V,U)$. Nustatyti, ar šis grafas yra dvidalis.

Šį uždavinį 1936 metais išsprendė D.Kionigas.

Teorema. (Kionigo teorema). Būtina ir pakankama sąlyga, kad grafas $G = (V,U)$ būtų dvidalis yra ta, kad jis neturėtų nelyginio ilgio ciklų.

Pateiksime šios teoremos įrodymą, nes šis įrodymas yra konstruktyvus, t.y. iš jo išplaikia algoritmas, leidžiąs nustatyti, ar grafas yra dvidalis.

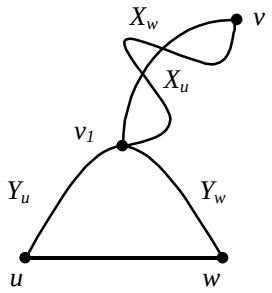
Būtinumas. Tarkime, *G* – dvidalis grafas, o μ – vienas iš jo ciklų, kurio ilgis yra *k*. Apeikime visas šio ciklo briaunas jų surašymo tvarka, pradedant viršūne *v*. Po *k* žingsnių grįšime į viršūnę *v*. Kadangi dvidalio grafo kiekvienos briaunos galai priklauso skirtingoms dalims(aibėms), tai *k* yra lyginis skaičius.

Pakankamumas. Nesiaurinant uždavinio, galime laikyti, kad grafas *G* yra jungusis, nes dvidalių grafų sąjunga yra dvidalis grafas.

Tarkime, kad (n,m)-grafas neturi nelyginio ilgio ciklų, o *v* – bet kuri grafo viršūnė. Viršūnių aibę *V* į dvi dalis *A* ir *B* išskaidysime taip: grafo viršūnes *u*, kurioms $d(v,u)$ yra lyginis skaičius, talpinsime į aibę *A*, o viršūnes, kurioms $d(v,u)$ yra nelyginis skaičius – į aibę *B*.

Belieka įrodyti, kad šių aibių indukuoti pografiai yra tuštieji.

Tarkime priešingai, kad egzistuoja dvi gretimos viršūnės *u* ir *w*,kurios priklauso vienai iš aibių. Aišku, kad nei viena iš šių viršūnių nesutaps su viršūne $v \in A$, nes visos viršūnei *v* gretimos viršūnės yra aibėje *B*.



2.10.1 pav. Kionigo teoremos iliustracija

Tarkime, kad *U* – trumpiausia (u,v)-grandinė, o *W* – trumpiausia (w,v)-grandinė. Tarkime, kad šių grandinių paskutinė (skaičiuojant nuo viršūnės *v*) bendra viršūnė yra v_1 (žr. 2.10.1 pav.). Simboliais X_u ir Y_u atitinkamai pažymėkime grandinės *U* subgrandines (v,v_1) ir (v_1,u), o simboliais X_w ir Y_w – grandinės *W* subgrandines (v,v_1) ir (v_1,w). Aišku, kad subgrandinių X_w ir X_u ilgiai yra lygūs. Vadinasi, Y_w ir Y_u ilgiai turi tą patį lyginumo charakterį.

Tačiau, tada, sujungus subgrandines Y_u, Y_w ir briauną (u,v) gausime elementarųjį nelyginio ilgio ciklą.

Išvada. Grafas yra dvidalis tada ir tiktai tada, kai jis neturi elementariųjų nelyginio ilgio ciklų.

Kionigo teoremos įrodymas nusako metodą, leidžiantį nustatyti, ar grafas yra dvidalis. Šis metodas susideda iš dviejų etapų.

1. Pasirenkama bet kuri grafo viršūnė *s*, ir 2.9 paragrafe aprašytos procedūros ***kelias*** pagalba randame atstumus nuo viršūnės *s* iki visų likusių viršūnių. Tada viršūnės, kurioms šis atstumas yra lyginis skaičius, talpinamos į aibę *A*, o likusios – į aibę *B*.
2. Tikriname kiekvieną grafo briauną (u,v), ar jos galai priklauso skirtingoms aibėms.

   Jei visoms grafo briaunoms (u,v) viršūnės *u* ir *v* priklauso skirtingoms aibėms (*A* ir *B*), tai grafas yra dvidalis, t.y. $G=(A,B,U)$.

   Jei egzistuoja bent viena briauna (u,v), kurios abi viršūnės *u* ir *v* priklauso arba aibei *A*, arba aibei *B*, tai grafas *G* nėra dvidalis.

Realizuoti 2-ąjį etapą galima įvairiais būdais. Labai patogus šio etapo realizavimo būdas būtų toks.

Apibrėžkime masyvą $ab[1..n]$, čia $ab[i]=0$, jei viršūnė *i* priklauso aibei *A* (jei procedūros *kelias* atstumų masyvo $d[1..n]$ elementas $d[i]$ yra lyginis skaičius), ir $ab[i]=1$, jei viršūnė *i* priklauso aibei *B* ($d[i]$ – nelyginis skaičius).

Tada briaunos (u,v) galai priklausys skirtingoms aibėms, jei $ab[u]\neq ab[v]$; priešingu atveju viršūnės *u* ir *v* priklausys vienai aibei.

Pastaba. Ar briaunos (u,v) galai priklauso skirtingoms aibėms, galima nustatyti tiesiogiai iš procedūros ***kelias*** masyvo $d[1..n]$:

if ((*d*[*n*] + *d*[*v*]) *mod* 2) = 0 *then "briaunos galai priklauso tai pačiai aibei, t.y. grafas nėra dvidalis"*.

Dalinio dvidalio grafo konstravimo uždavinys. Duotas grafas $G(V,U)$. Rasti šio grafo dalinį grafą, atmetant dalį pradinio grafo briaunų taip, kad atmetamų briaunų skaičius būtų nedidesnis nei $\lfloor m/2 \rfloor$, čia $m=|U|$.

Toks dalinis grafas konstruojamas taip:

$A:=\varnothing$; $B:=\varnothing$;

$A:=A\cup\{s \mid s$ - *bet kuri grafo viršūnė*$\}$;

for $(v\in V)$ *and* $(v\neq s)$ *do*

if $|A\cap N(v)| \lhd |B\cap N(v)|$ *then* $A := A\cup\{v\}$

else $B := B\cup\{v\}$;

čia $N(v)$ – viršūnės *v* aplinka, t.y. viršūnei *v* gretimų viršūnių aibė.

Tada $|A\cap N(v)|$ parodo atmetamų briaunų skaičių, jei viršūnę *v* talpiname į aibę *A*, o $|B\cap N(v)|$ – atmetamų briaunų skaičių, jei viršūnę talpintume į aibę *B*. Vadinasi, skaidant aibę *V* į dvi aibes *A* ir *B*, kiekvieną kartą viršūnę *v* talpinsime į tą aibę, kad atmetamų briaunų skaičius būtų mažiausias ir neviršytų $\lfloor d(v)/2 \rfloor$, čia $d(v)$ – *v*-osios viršūnės laipsnis.

Aišku, kad taip konstruojant dalinį dvidalį grafą, atmetamų briaunų skaičius bus nedidesnis nei $\lfloor m/2 \rfloor$.

2.11. Pagrindiniai grafų teorijos skaičiai

Pagrindiniai grafų teorijos skaičiai yra:

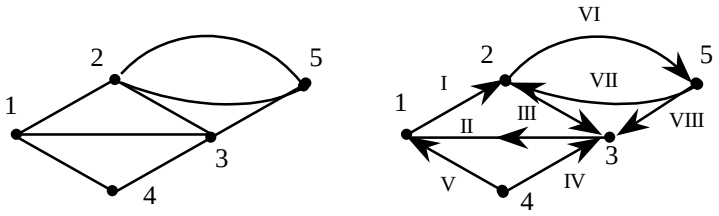
1) ***ciklomatinis*** skaičius,
2) ***chromatinis*** (spalvinis) skaičius,
3) ***nepriklausomumo*** (vidinio stabilumo) skaičius,
4) ***dominavimo*** (išorinio stabilumo) skaičius.

2.11.1. Ciklomatinis skaičius

Nagrinėsime neorientuotuosius grafus, kurie gali būti ir multigrafai. Grafo $G=(V,U)$ ***ciklomatinis skaičius*** $\nu(G)$ apibrėžiamas formule

$$\nu(G) = m - n + p\,,$$

čia *m* – briaunų skaičius, *n* – viršūnių skaičius, o *p* – jungiųjų komponenčių skaičius. Pavyzdžiui, grafui, pavaizduotam 2.11.1 pav., ciklomatinis skaičius $\nu(G) = 8-5+1=4$.



a) b)

2.11.1 pav. Grafo ciklomatinis skaičius

Kiekvienam grafo ciklui galima priskirti *m*-matį vektorių tokiu būdu:

1. sunumeruokime briaunas ir kiekvienai briaunai laisvai suteikime orientaciją (žr. 2.11.1 b) pav.);
2. jei apeinant ciklą, *k*-toji briauna (u,v) rodyklės kryptimi praeinama *s* kartų, o prieš rodyklę – *t* kartų, tai šiam ciklui atitinkančio vektoriaus *k*-oji komponentė lygi $s-t$.

Pavyzdžiui, 2.11.1 b) pav. ciklą $\mu_1=(1,2,5,3,1)$ atitiks vektorius $c_1=(\overset{1}{1}\ \overset{2}{1}\ \overset{3}{0}\ \overset{4}{0}\ \overset{5}{0}\ \overset{6}{1}\ \overset{7}{0}\ \overset{8}{1})$, o ciklą $\mu_2=(1,3,4,1)$ – vektorius $c_2=(\overset{1}{0}\ \overset{2}{-1}\ \overset{3}{0}\ \overset{4}{-1}\ \overset{5}{1}\ \overset{6}{0}\ \overset{7}{0}\ \overset{8}{0})$.

Nepriklausomiems vektoriams atitinkantys ciklai yra nepriklausomi.

Teorema. Multigrafo *G* ciklomatinis skaičius lygus didžiausiam nepriklausomų ciklų skaičiui.

Išvados.

- Grafas *G* neturi ciklų tada ir tik tada, kai $\nu(G)=0$.
- Grafas *G* turi vienintelį ciklą tada ir tiktai tada, kai $\nu(G)=1$.

Pavyzdžiui, 2.11.1 a) grafui nepriklausomų ciklų aibė (bazė) yra ciklai: $\mu_1=(1,2,3,1)$, $\mu_2=(1,3,4,1)$, $\mu_3=(2,5,2)$, $\mu_4=(2,5,3,2)$.

Aišku, kad nepriklausomų ciklų rinkinys yra nevienintelis. Pavyzdžiui, tam pačiam 2.11.1 a) grafui nepriklausomų ciklų bazę sudaro ir ciklai $\mu_1=(1,2,5,3,1)$, $\mu_2=(2,5,3,2)$, $\mu_3=(2,5,2)$, $\mu_4=(1,3,4,1)$. Šie ciklai yra tiesiškai nepriklausomi, nes kiekvienas iš jų turi bent po vieną unikalią briauną, nepriklausančią likusiems ciklams. Ciklas μ_1 turi unikalią briauną $(1,2)$, μ_2 – $(3,2)$, μ_3 – $(5,2)$ ir μ_4 – $(3,4)$ arba $(4,1)$.

Nepriklausomų ciklų apskaičiavimo uždavinys. Duotas jungusis neorientuotasis (n,m)-grafas (ne multigrafas) $G=(V,U)$. Rasti nepriklausomų ciklų bazę.

Įveskime ***atvirkštinės briaunos*** sąvoką. Tarkime, kad $H=(V,U_H)$ yra grafo *G* dalinis grafas, neturintis ciklų. Toks dalinis grafas vadinamas dengiančiu medžiu. (Apie medžius bus kalbama 2.13 paragrafe). Dengiantis medis turi $(n-1)$ briauną ir neturi ciklų.

Apibrėžimas. Grafo *G* briauna (u,v) yra ***atvirkštinė briauna*** **(*styga*)**, jei ji nepriklauso dengiančio medžio *H* briaunų aibei.

Kadangi $|U_H| = n-1$, tai atvirkštinių briaunų skaičius yra lygus $m-n+1$, t.y. kiekviena atvirkštinė briauna $e=(u,v)$ drauge su dengiančio medžio briaunomis, priklausančiomis grandinei, jungiančiai viršūnę *u* su viršūne *v*, sudaro vieną nepriklausomą grafo *G* ciklą. ***Šį ciklą žymėsime*** C_e.

Vadinasi, kiekvienas grafo *G* dengiantis medis apibrėš nepriklausomų ciklų aibę. Be to, neizomorfinių medžių nepriklausomų ciklų aibės bus skirtingos.

Pastaba. Taip apibrėžus nepriklausomus ciklus, visus kitus grafo ciklus galima išreikšti per nepriklausomų ciklų simetrinį skirtumą (sumą moduliu 2).

Apibrėžimas. Grafo $G=(V,U)$ briaunų aibė C vadinama ***pseudociklu***, jei grafo (V,C) visų viršūnių laipsniai yra lyginiai**.**

Tuščia aibė ir bet koks grafo ciklas yra pseudociklas.

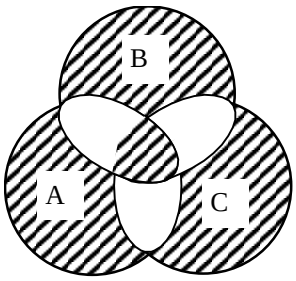
Priminsime aibių ***simetrinio skirtumo*** operaciją:

$$A \oplus B = (A \cup B) \setminus (A \cap B).$$

Šią operaciją taikysime didesniam aibių $A_1, A_2, ..., A_k$ skaičiui:

$$\bigoplus_{i=1}^{k} A_i = A_1 \oplus A_2 \oplus ... \oplus A_k .$$

Nesunku įsitikinti, kad aibių $A_1, A_2, ..., A_k$ simetrinį skirtumą, nepriklausomai nuo skliaustų, nurodančių veiksmų atlikimo tvarką, sudėjimo sudarys tik tie elementai, kurie priklauso nelyginiam skaičiui poaibių A_i (žr. 2.11.2 pav.)



2.11.2 pav. Aibių *A, B* ir *C* simetrinis skirtumas $A \oplus B \oplus C$

Lema. Bet kokio skaičiaus pseudociklų simetrinis skirtumas yra pseudociklas.

Įrodymas. Aišku, kad pakanka panagrinėti dviejų pseudociklų C_1 ir C_2 atvejį.

Simboliais $S_1(v)$, $S_2(v)$ ir $S(v)$ pažymėkime viršūnei *v* incidentiškų briaunų skaičių atitinkamai pseudocikluose C_1, C_2 ir $C = C_1 \oplus C_2$. Aišku, kad aibių $S_1(v)$ ir $S_2(v)$ elementų skaičius yra lyginis. Tada

$$|S(v)| = |S_1(v) \oplus S_2(v)| = |S_1(v)| + |S_2(v)| - 2|S_1(v) \cap S_2(v)|$$

taip pat yra lyginis skaičius, ir *C* yra pseudociklas.

Teorema. Tarkime, $G = (V,U)$ – jungusis neorientuotasis grafas, o (V,T) – jį dengiantis medis. Tada bet koks grafo *G* ciklas *C* vienareikšmiškai išreiškiamas per nepriklausomus ciklus C_e taip:

$$C = \bigoplus_{e \in C \setminus T} C_e .$$

Įrodymas. Simetrinis skirtumas $\bigoplus_{e \in C \setminus T} C_e$, remiantis lema, yra pseudociklas, susidedantis iš atvirkštinių briaunų (stygų) aibės $C \setminus T$ ir kažkokių medžio *T* briaunų. Tai išplaukia iš to fakto, kad kiekviena atvirkštinė briauna $e \in C \setminus T$ priklauso tik vienam nepriklausomam ciklui C_e.

Aibė $C \oplus \bigoplus_{e \in C \setminus T} C_e$, remiantis lema, taip pat nusakys pseudociklą, kurį gali sudaryti tik medžio *T* briaunos. (Pseudociklo *C* atvirkštinės briaunos *e* priklauso ir pseudociklui $\bigoplus_{e \in C \setminus T} C_e$, todėl jos nepriklausys aibei $C \oplus \bigoplus_{e \in C \setminus T} C_e$). Kadangi medis T neturi ciklų, tai šis pseudociklas yra tuščioji aibė. Vadinasi, $C = \bigoplus_{e \in C \setminus T} C_e$. šios formulės vienareikšmiškumas išplaukia iš to, kad ciklas C_e yra vienintelis nepriklausomas ciklas, turintis atvirkštinę briauną *e*.

Iš šio nagrinėjimo išplaukia, kad grafo nepriklausomus ciklus patogu ieškoti, naudojant paieškos gilyn metodą, aptartą 2.8.2 paragrafe.

Kaip paieškos gilyn metodu formaliai nustatyti ***atvirkštinę briauną (stygą)***?

Tam tikslui reikia žinoti viršūnių aplankymo eilės numerius. Įveskime masyvą $nr[1..n]$, čia $nr[i]$ yra *i*-osios viršūnės aplankymo eilės numeris.

Tarkime, kad paieškos gilyn metu iš viršūnės *u* atėjome į viršūnę *v*. Briauna (u,v) bus atvirkštinė briauna (styga) ir iššauks nepriklausomą ciklą, jei bus teisinga sąlyga: "***viršūnė v nenauja***" ***and*** "***į viršūnę u buvome atėję ne iš viršūnės v***" ***and*** "$nr[v] < nr[u]$, ***t.y. viršūnė v buvo aplankyta anksčiau nei viršūnė u***".

2.8.1.2 paragrafe aprašytam paieškos gilyn algoritme ši sąlyga bus užrašoma taip: $(prec[v] \neq 0)$ *and* $(prec[u] \neq v)$ *and* $(nr[v] < nr[u])$.

Žinant šią atvirkštinės briaunos sąlygą nesunku modifikuoti 2.8.1.2 paragrafe aprašytą paieškos gilyn metodą taip, kad jis apskaičiuotų nepriklausomus cilus. Žemiau ir pateikta ši nepriklausomų ciklų apskaičiavimo procedūra.

```
const c = 500;
type
    mas = array [1..c] of integer;
procedure ciklai(v, n, m : integer; L, lst : mas);
{ Procedūra ciklai, naudodama paiešką gilyn iš viršūnės v, kai grafas
nusakytas L ir lst masyvais, apskaičiuoja grafo nepriklausomus ciklus.
    Formalūs parametrai:
    v – pradinė paieškos viršūnė,
    n – grafo viršūniu skaičius,
    m – grafo briaunų (lanku) skaičius,
    L, lst – grafą nusakantys tiesioginių nuorodų masyvai. }
var i, k, u      : integer;
    sk, j        : integer;
    t, p         : boolean;
    fst, prec    : mas;
    nr           : 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 nenagrinetų briaunų, incidentiškų viršūnei
    k}
        begin
            t := false;
            p := true;
            prec [k] := k; { k – pradinė paieškos viršūnė }
            { Nagrinėti viršūnę k }
            nr[k] := 1;
            sk := 1; { Aplankytų viršūnių skaičius }
        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 {viršūnė u nauja}
                    begin
                        { Nagrinėti viršūnę u }
                        sk := sk + 1;
                        nr [u] := sk;
                        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
                    begin
                        p := false; { viršūnė u nenauja}
                        if (prec [k] <> u) and (nr [k] > nr [u]) then
                        { Briauna (k, u) yra atvirkštinė briauna. Spausdinti ciklą }
                            begin
                                writeln;
                                write (u : 3);
                                write (k : 3);
                                j := k;
                                while prec [j] <> u do
                                    begin
                                        j := prec [j];
                                        write (j : 3);
                                    end;
                                write(u : 3);
                                writeln;
                            end;
                    end;
            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 i viršūnę, iš kurios buvome atėję
                                    į viršūnę k } k := prec [k];
            end;
    end;
end;
```

Aptarkime dar vieną nepriklausomų ciklų apskaičiavimo, naudojant rekursiją, procedūrą.

Šią procedūrą patogu organizuoti naudojant steką (žr. 2.8.1.3 paragrafą).

Tarkime, kad grafas $G = (V, U)$ nusakytas gretimumo struktūra, t.y. $N(v)$ –aibė viršūnių, gretimų viršūnei *v*.

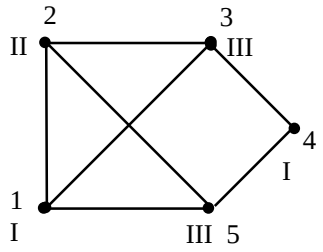
```
const c = 500;
type
    mas = array [0..c] of integer;
    matr = array [1..2, 1..c] of integer;
procedure ncikl (v : integer);
{ Grafo G jungiosios komponentės, turinčios savyje viršūnę v, nepriklausomų
ciklų apskaičiavimas.
Kintamieji num, top, stec, nr, L, lst – globalieji }
var u, i, top1 : integer;
begin
top := top +1;
stek [top] := v;
num := num + 1;
nr [v] := num;
for i := lst [v] + 1 to lst [v + 1] do
    begin
        u: = L [i];
        if nr [u] = 0 then ncikl (u)
            else
                if (nr [v] > nr [u]) and (u <> stek [top – 1]) then
```

```
                { Briauna (v,u) – atvirkštinė briauna. Įsiminti ciklą, einantį per
                viršūnes: u, stek [top], stek [top – 1], ..., stek [c], čia stek [c] = u.
                }
                    begin
                        writeln;
                        write (u : 3);
                        top1 := top;
                        while stek [top1] <> u do
                            begin
                                write (stek[top1]:3);
                                top1:=top1-1;
                            end;
                        write (u : 3);
                        writeln;
                    end;
    end;
top := top – 1; { Išsemta viršūnė v šalinama iš steko. }
end; { ncikl }
```

2.11.2. Chromatinis skaičius

(n,m)-grafo $G=(V,U)$ ***chromatinis skaičius*** yra mažiausias skaičius spalvų, kurioms grafo viršūnes galima nudažyti taip, kad bet kokios dvi gretimos viršūnės būtų nudažytos skirtinga spalva. Šį skaičių žymėsime $\gamma(G)$.

Pavyzdžiui, 2.11.3 pav. grafui chromatinis skaičius yra 3. Aišku, kad šio grafo negalima nudažyti su mažesniu spalvų skaičiumi, nes jis turi ilgio 3 ciklų.



2.11.3 pav. Grafo chromatinis skaičius

Aišku, kad K_n grafo chromatinis skaičius lygus *n*, o bet koks dvidalis grafas $G = (A, B, U)$ yra bichromatusis: aibės *A* viršūnės dažomos pirmąja spalva, o aibės *B* viršūnės – antrąja spalva.

Perfrazavus Kionigo teoremą, galime teigti, kad grafas yra bichromatusis tada ir tiktai tada, kai jis neturi nelyginio ilgio ciklų.

Chromatinio skaičiaus apskaičiavimo uždavinys yra diskrečiojo optimizavimo uždavinys. Formaliai jį galima užrašyti taip.

$\min\ z = p$, čia *p* – spalvų skaičius,

esant apribojimams:

a) kiekviena viršūnė turi būti dažoma viena spalva,

b) bet kokios gretimos viršūnės turi būti nudažytos skirtinga spalva.

Šiems apribojimams užrašyti įvesime kintamuosius

$$x_{ij} = \begin{cases} 1, \text{ jei } i\text{-toji viršūnė dažoma } j\text{-ąją spalva}, \\ 0, \text{ priešingu atveju}, \end{cases}$$

$$i = \overline{1, n}, \quad j = \overline{1, p}.$$

Tada a) apribojmas bus užrašomas taip:

$$\sum_{j=1}^{p} x_{ij} = 1, \quad i = \overline{1, n},$$

o b) apribojimas –

$$\sum_{i=1}^{n} x_{ij} \cdot a_{ik} \le 1, \quad j = \overline{1, p}, i = \overline{1, m},$$

čia a_{ik} – incidencijų matricos (žr. 2.7 paragrafą) elementas.

Pavyzdžiui, 2.11.3 pav. grafui optimalus šio uždavinio sprendinys yra:

$$X = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccc} \text{I} & \text{II} & \text{III} \end{array} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{array}.$$

Kaip buvo minėta aukščiau, šis uždavinys yra NP pilnasis [GD82] ir neturi efektyvių sprendimo algoritmų. todėl jis sprendžiamas naudojant įvairias euristikas.

Dažniausiai naudojami grafų dažymo algoritmai remiasi tokiomis euristikomis:

- pirma spalva, po to viršūnė,

- pirma viršūnė, po to spalva.

Detaliau aptarkime šiuos euristinius algoritmus.

Euristinis algoritmas, paremtas principu "pirma spalva, o po to viršūnė"

Šios euristikos idėja yra labai paprasta. Pagal kokį nors požymį sudaroma grafo viršūnių seka. Po to imama nauja spalva ir šia spalva iš eilės dažomos, jeigu galima, sekos viršūnės. Taip elgiamės iki nudažome visas grafo viršūnes.

Tuo būdu šis algoritmas susideda iš dviejų etapų.

1. *Grafo viršūnės išrikiuojamos jų laipsnių mažėjimo tvarka* $v_1, v_2, ..., v_n$, čia $v_i \in V$ ir $d(v_{i+1}) \le d(v_i)$, $i = \overline{1, n-1}$.
2. $p := 0$; {*spalvų skaičius*}

   while "yra nenudažytų viršūnių" do

   begin

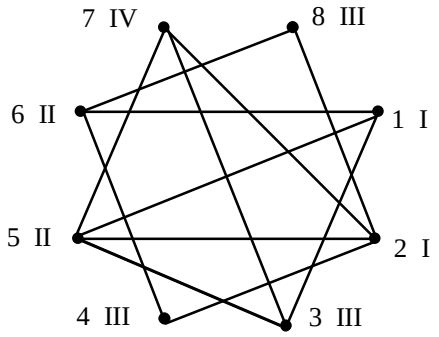
   $p := p + 1$;

   Pradedant pirmąja nenudažyta sekos viršūne, ***p*** *spalva nuosekliai viena po kitos dažomos, jei galima, nenudažytos sekos viršūnės.*

   end;

Pavyzdys. Panagrinėkime grafo, pavaizduoto 2.11.4 pav., dažymą, laikantis principo "pirma spalva, o po to viršūnė".

2.11.4 pav. pavaizduotam grafui seka viršūnių, išrikiuotų jų laipsnių mažėjimo tvarka, yra: 2, 5, 1, 3, 6, 7, 4, 8.



2.11.4 pav. Grafo dažymas laikantis principo "pirma spalva, o po to viršūnė"

Pirmąja spalva dažome 2-ąją viršūnę, po to pirmąja spalva galime dažyti tik 1-ąją viršūnę. Gausime:

2, 5, 1, 3, 6, 7, 4, 8
I I .

Imame antrąją spalvą ir dažome pirmąją nenudažytą sekos viršūnę – 5-ąją, po to antrąja spalva galime dažyti 6-ąją viršūnę. Gausime:

2, 5, 1, 3, 6, 7, 4, 8
I II I II .

Trečiąja spalva pirmiausia dažome 3-ąją viršūnę, po to – 4-ąją viršūnę ir, galiausiai, 8-ąją viršūnę. Gausime:

2, 5, 1, 3, 6, 7, 4, 8
I II I III II III III .

Kadangi dar yra nenudažytų viršūnių, tai imame IV-ąją spalvą ir dažome 7-ąją viršūnę, t.y. gausime:

2, 5, 1, 3, 6, 7, 4, 8
I II I III II IV III III .

Gautas sprendinys nėra optimalus. Šį grafą galima nudažyti trimis spalvomis (žr. euristiką "pirma viršūnė, o po to spalva").

Žemiau pateikta grafo dažymo, naudojant euristiką "pirma spalva, o po to viršūnė" procedūra.

```
const c = 500;
type
    mas  = array [1..c] of integer;
    matr = array [1..2, 1..c] of integer;
procedure grfdaz1 (n, m : integer; b : matr; var p: integer; var d : mas);
{ Procedūra grafdaz1 dažo grafo viršūnes, remdamasi euristika "pirma spalva,
o po to viršūnė".
Formalūs parametrai:
    n – grafo viršūnių skaičius,
    m – grafo briaunų (lankų) skaičius,
    b – grafo briaunų matrica ,
    p – spalvų skaičius,
    d [1..n] – viršūnių spalvų masyvas;
    jei d [i] = k, tai reiškia, kad i-oji viršūnė dažoma k-aja spalva. }
var i, k, z, sv, u,j ,x : integer;
    t : boolean;
    L, lst, v, s : mas;
begin
BLlst(n,m,b,L,lst);
{ Pradinių reikšmių suteikimas darbo masyvams ir kintamiesiems }
```

```
for i := 1 to n do
    begin
        v [i] := i; { Masyve v iš eilės surašomi viršūnių numeriai }
        s [i] := lst [i + 1] – lst [i]; {s [i] – i-tosios viršnės laipsnis }
        d[i] := 0;
    end;
{ Viršūnes masyve v išrikiuojame jų laipsnių mažėjimo tvarka }
for k := 1 to n – 1 do
for i: = 1 to n – k do
if s [i] < s [i + 1] then { Keičiame vietomis s [i] su s [i + 1] ir v [i] su v [i + 1] }
    begin
        z := s [i]; s [i] := s [i + 1]; s [i + 1] := z;
        z := v [i]; v [i] := v [i + 1]; v [i + 1] := z;
    end;
p := 0; { p – spalvų skaičius }
sv := 0; {sv - nudažytų viršūnių skaičius }
while sv < n do
    begin
        p := p + 1;
        for i:=1 to n do
            begin
                u := v [i];
                if d [u] = 0 then {Viršūnė u – nenudažyta.
                Ar ją galima dažyti p-qja spalva? }
                    begin
                        { Ar viršūnių, gretimų viršūnei u, tarpe yra
                        viršūnė,nudažyta p-qja spalva? }
                        j := lst [u] + 1; t := false;
                        { Jei t = false, tai viršūnę u galima dažyti p-qja spalva }
                        while (j <= lst [u + 1]) and not t do
                            begin
                                x := l [j];
                                if d[x] = p then t:=true
                                            else j:=j+1;
                            end;
                        if not t then
                            begin
                                d [u] := p;
```

```
                    sv := sv + 1;
                end;
            end;
        end;
    end;
end;
```

Euristinis algoritmas, paremtas principu "pirma viršūnė, o po to spalva"

Šios euristikos idėja yra ta, kad pirmiausia pagal kažkokį kriterijų išrenkama grafo viršūnė, o po to ji dažoma, jei galima, viena iš anksčiau panaudotų spalvų; jei išrinktos viršūnės negalima nudažyti nei viena iš anksčiau panaudotų spalvų, tai ji dažoma nauja spalva.

Viršūnės išrinkimo kriterijus

Apibrėžimas. Viršūnės *h* laipsnis – tai skaičius spalvų, kuriomis šios viršūnės negalima dažyti, t.y. šiai viršūnei negalimų spalvų skaičius.

Apibrėžimas. Viršūnės *k* laipsnis – tai šiai viršūnei gretimų nenudažytų viršūnių skaičius.

Pirmiausia pasirinksime viršūnę, kurios *h* laipsnis yra didžiausias.

Jei yra kelios viršūnės su tuo pačiu didžiausiu *h* laipsniu, tai iš jų išsirinksime viršūnę, kurios *k* laipsnis yra didžiausias.

Jei ir šiuo atveju bus daugiau nei viena viršūnė, imsime viršūnę, kurios numeris mažiausias.

Tuo būdu algoritmas aprašomas taip.

begin
$p := 0$;
while "yra nenudažytų viršūnių" do
begin
1) *pagal aukščiau aprašytą kriterijų parenkama nenudažyta viršūnė v*;
2) *jei galima, viršūnę v dažome viena iš anksčiau panaudotų spalvų* (*paprastai, pirmąja iš galimų*), *t.y. viena iš aibės* $\{1,2,\ldots,p\}$ *galimų spalvų*;
3) *jei viršūnės v negalima nudažyti nei viena iš anksčiau panaudotų spalvų, tai* $p := p+1$, *ir viršūnę v dažome p-ąją spalva*;
4) *visoms nenudažytoms viršūnėms perskaičiuojame h ir k laipsnius*;

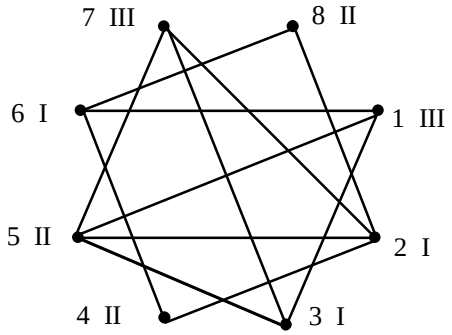
end;
end;

Pavyzdys. Panagrinėkime, kaip bus dažomos 2.11.4 pav. grafas. Skaičiavimai parodyti 2.11.1 lentelėje, o 2.11.5 pav. pavaizduotas nudažytas grafas.

2.11.1 lentelė. Grafo dažymas

NR	*h* – laipsnis	*k* – laipsnis	spalva	*
1	~~0~~ ~~1~~ 2	~~3~~ ~~2~~ 1	III	5
2	0	4	I	1
3	~~0~~ ~~1~~ 2	~~3~~ ~~2~~ 1	I	4
4	~~0~~ 1	~~2~~ 1	II	6
5	~~0~~ 1	~~4~~ 3	II	2
6	~~0~~ ~~1~~ 2	~~3~~ ~~2~~ 1	I	7
7	~~0~~ ~~1~~ 2	~~3~~ ~~2~~ 1	III	3
8	~~0~~ 1	~~2~~ ~~1~~ 0	II	8

čia * – viršūnės išrinkimo eilės numeris.



2.11.5 pav. Grafo dažymas laikantis principo “pirma viršūnė, o po to spalva”

Žemiau pateikta grafo dažymo, naudojant euristiką “pirma viršūnė, o po to spalva” procedūra.

```
const c = 500;
type
    mas = array [1..c] of integer;
    matr = array [1..2, 1..c] of integer;
procedure grfdaz2(n, m : integer; b : matr; var p : integer; var d : mas);
{ Procedūra grafdaz2 dažo grafo viršūnes, remdamasi euristika “pirma
viršūnė, o po to spalva”.
Formalūs parametrai:
    n – grafo viršūnių skaičius,
```

```
    m – grafo briaunų (lankų) skaičius,
    b – grafo briaunų matrica ,
    p – spalvų skaičius,
    d [1..n] – viršūnių spalvų masyvas;
    jei d [i] = k, tai reiškia, kad i-oji viršūnė dažoma k-ąja spalva. }
var i, sv, max, maxka, v, u, j, x, sp : integer;
    t,st : boolean;
    L,lst,h,ka : mas;
begin
BLlst (n, m, b, L, lst);
{ Pradinių reikšmių suteikimas darbo masyvams ir kintamiesiems }
for i := 1 to n do
    begin
        ka [i] := lst [i + 1] – lst [i]; { ka [i] – i-tosios viršūnės laipsnis }
        d[i] := 0;
        h[i] := 0;
    end;
p := 0; { p – spalvų skaičius }
sv := 0; {sv – nudažytų viršūnių skaičius }
while sv < n do
    begin
        { Viršūnės išrinkimas }
        max := –1;
        for i := 1 to n do
        if (d [i] = 0) and (max < h [i]) then max := h [i];
        maxka := –1;
        for i:=1 to n do
        if (d [i] = 0) and (h[i] = max) and (maxka < ka [i]) then
            begin
                maxka := ka [i];
                v:=i;
            end;
        { Kuria spalva dažyti viršūnę v ? }
        sv := sv + 1; { Dažome viršūnę v }
        t := false; { t = false, jei viršūnei v spalva neparinkta }
        i := 1; { Bandome viršūnę v dažyti spalva i }
        while (i <= p) and (not t) do
        begin
            { Tikriname, ar viršūnei v gretimų viršūnių tarpe yra viršūnė, kuri
            nudažyta i-ąja spalva }
            j:=lst [v] + 1;
```

```
        st := false; { Jei st = false, tai reiškia, kad viršūnei v gretimų
        viršūnių tarpe spalvos i nėra }
        while (j <= lst [v + 1]) and (not st) do
            begin
                u := L [j];
                sp:=d[u];
                if i = sp then st := true
                            else j:=j+1;
            end;
        if st then i := i + 1 { Bandome naują spalvą }
        else t:=true; { Viršūnė v gali būti dažoma i-qja spalva }
    end;
if t then d [v] := i { Viršūnę v dažome i-qja spalva }
    else begin
        p := p + 1; { Įvedame naują spalvą }
        d [v] := p; { Viršūnę v dažome nauja spalva }
    end;
{ h ir ka laipsnių perskaičiavimas nenudažytoms viršūnėms, gretimoms
viršūnei v }
for i:=lst[v]+1 to lst[v+1] do
    begin
        u := L [i];
        if d [u] = 0 { Viršūnė u , gretima viršūnei v, – nenudažyta } then
            begin
                ka[u] := ka [u] – 1;{ Viršūnei u gretimų nenudažytų viršūnių
                skaičius sumažėjo }
                j := lst [u] + 1;
                t := false; { Nei viena iš viršūnei u gretimų viršūnių
                nenudažyta d [v] spalva }
                while (j <= lst [u + 1]) and ( not t) do
                    begin
                        x:=L[j];
                        if (x <> v) and (d [x] = d [v]) then t := true
                                else j := j + 1;
                    end;
                if not t { Viršūnei u gretimų nudažytų viršūnių x tarpe nėra
                spalvos, lygios d[v] }
```

```
                then h [u] := h [u] + 1;
            end;
        end;
    end;
end;
```

Uždavinys, analogiškas grafo viršūnių dažymo uždaviniui, yra briaunų dažymo uždavinys.

Mažiausias skaičius spalvų, kuriomis grafo briaunas galima nudažyti taip, kad bet kurios dvi gretimos briaunos būtų nudažytos skirtinga spalva, vadinamas grafo ***chromatine klase***.

Aišku, kad (n,m)-grafo G chromatinė klasė yra lygi šiam grafui atitinkančio briauninio grafo (žr. 2.5 paragrafą) chromatiniam skaičiui.

Vadinasi, aukščiau aptarti chromatinio skaičiaus apskaičiavimo algoritmai tinka ir grafo chromatinei klasei apskaičiuoti. Tik šiuo atveju reikia nagrinėti grafui G atitinkanti briauninį grafą.

Reikia pabrėžti, kad eilė praktinių uždavinių yra formuluojami kaip grafo dažymo uždaviniai. Šiems uždaviniams būdinga tai, kad kai kurie objektai dėl vienokių ar kitokių priežasčių negali būti jungiami į grupes.

Pirmas pavyzdys. Tarkime, kad per galimai trumpiausią laiką reikia perskaityti keletą paskaitų. Kiekvienos paskaitos trukmė yra 1 valanda ir kai kurias paskaitas skaito tas pats lektorius. Sudarykime grafą G, kurio viršūnės vaizduoja paskaitas, o dvi viršūnės yra gretimos, jei jų negalima skaityti vienu metu, t.y. jei jas skaito tas pats lektorius. Aišku, kad bet koks teisingas grafo nudažymas apibrėžia paskaitų skaitymo tvarkaraštį, t.y. paskaitos, atitinkančios viršūnėms, kurios nudažytos ta pačia spalva, gali būti skaitomos vienu metu. Vadinasi, grfo chromatinis skaičius bus lygus mažiausiam valandų skaičiui, kuris reikalingas perskaityti paskaitų ciklą.

Antras pavyzdys. Duota darbų $V = \{v_1, v_2, ..., v_n\}$ aibė ir mechanizmų $S = \{s_1, s_2, ..., s_m\}$ aibė. Visų darbų trukmės yra lygios, o darbams atlikti naudojami aibės S mechanizmai. Aišku, kad tas pats mechanizmas vienu metu negali būti naudojamas keliems darbams atlikti. Mechanizmus reikia paskirstyti taip, kad bendras visų darbų atlikimo laikas būtų trumpiausias.

Sudarykime grafą G, kurio viršūnių aibė yra darbų aibė V. Viršūnės v_i ir v_j $(i \neq j)$ yra gretimos, jei šių darbų atlikimui reikia vieno ir to paties mechanizmo. Aišku, kad teisingai nudažius grafą, darbai, nudažyti ta pačia spalva, gali būti vykdomi tuo pačiu metu.

Chromatinio skaičiaus įverčiai [EM90]. Kadangi chromatinio skaičiaus apskaičiavimo uždavinys yra sunkus, tai svarbu žinoti chromatinio skaičiaus įverčius.

Simboliais $\delta(G)$ ir $\Delta(G)$ atitinkamai pažymėkime grafo G mažiausią ir didžiausią viršūnių laipsnį, t.y. $\delta(G)=\min_{v\in V} d(v)$, o $\Delta(G)=\max_{v\in V} d(v)$. Tada jungiajam grafui G galima nurodyti tokius chromatinio skaičiaus įverčius.

1. Grafo G chromatinis skaičius tenkina nelygybę $\gamma(G)\le 1+\Delta(G)$.
2. ***Brukso teorema*** (1941). Jei G – jungusis ir nepilnasis grafas, kuriam $\Delta(G)\ge 3$, tai $\gamma(G)\le\Delta(G)$.
3. $\gamma(G)\le \min\limits_{1\le i\le n}\max(d(i)+1,i)$ (Welsh, Powell).
4. $\gamma(G)\ge \dfrac{n}{n-\dfrac{1}{2m}\sum\limits_{i=1}^{n} d^2(i)}$ (Elphick).
5. ***Teorema*** (A.P.Eršovas, G.I.Kožuchinas, 1962).

$$-\left[-n\Big/\left[\frac{n^2-2m}{n}\right]\cdot\left(1-\left\{\frac{n^2-2m}{n}\right\}\Big/\left(1+\left[\frac{n^2-2m}{n}\right]\right)\right)\right]\le$$

$$\le\gamma(G)\le\left[\frac{3+\sqrt{9+8(m-n)}}{2}\right],$$

čia simbolis $[\cdot]$ žymi skaičiaus sveikąją dalį, o $\{\cdot\}$ – skaičiaus trupmeninę dalį.

Šio paragrafo pabaigoje paminėsime G.Hadvigerio 1943 m. iškeltą hipotezę.

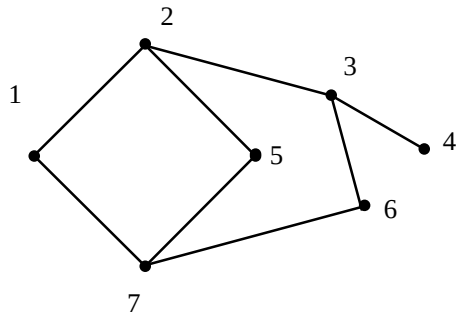
Hadvigerio hipotezė. Bet koks jungusis n-chromatinis grafas gali būti sutrauktas į K_n grafą.

Ši hipotezė įrodyta, kai $n\le 5$. Plačiau apie tai žr. lit. [EM90].

2.11.3. Nepriklausomumo skaičius

Apibrėžimas. (n,m)-grafo $G=(V,U)$ viršūnių poaibis $A(A\subseteq V)$ yra ***nepriklausomoji aibė*** (kitur literatūroje – ***vidiniai stabilioji aibė***), jei aibę A sudarančios viršūnės tarpusavyje nėra gretimos.

Pavyzdžiui, 2.11.6 pav. grafui aibės $A=\{1,5,4\}$, $B=\{2,7,4\}$ yra nepriklausomosios.



2.11.6 pav. Nepriklausomumo skaičius

Akivaizdu, jei *A* yra nepriklausomoji aibė, tai bet koks šios aibės poaibis taip pat bus nepriklausomasis. Vadinasi, kiekvienam grafui galima sudaryti nepriklausomųjų aibių šeimą *T*.

Apibrėžimas. Nepriklausomoji aibė, kuri nėra nei vienos kitos nepriklausomosios aibės tikrinis poaibis, vadinama ***maksimaliąja*** nepriklausomąja aibe.

Apibrėžimas. Nepriklausomoji aibė, turinti didžiausią elementų skaičių, vadinama ***didžiausiąja*** nepriklausomąja aibe.

Apibrėžimas. Skaičius $\alpha(G) = \max_{A \in T} |A|$, t.y. didžiausios nepriklausomosios aibės elementų skaičius, vadinamas grafo nepriklausomumo skaičiumi.

Pavyzdžiui, 2.11.3.1 pav. grafui didžiausia nepriklausomoji aibė yra $A = \{1,4,5,6\}$, o nepriklausomumo skaičius $\alpha(G) = 4$.

Nepriklausomumo skaičiaus radimo uždavinys yra diskrečiojo programavimo uždavinys, ir, kaip ir chromatinio skaičiaus ieškojimo uždavinys, neturi efektyvių sprendimo algoritmų. Kitaip tariant, visų tikslų šio uždavinio sprendimo algoritmų veiksmų skaičius išreiškiamas eksponentiškai, grubiai kalbant, nuo grafo viršūnių skaičiaus.

Pavyzdžiui, grafo nepriklausomumo skaičiaus ieškojimo uždavinį galime formalizuoti taip.

Įveskime kintamuosius

$$x_i = \begin{cases} 1, \text{ jei } i\text{-oji viršūnė priklauso didžiausiai nepriklausomajai aibei,} \\ 0, \text{ priešingu atveju,} \end{cases}, \; i = \overline{1,n}.$$

Tada reikia maksimizuoti tikslo funkciją:

$$z = \sum_{i=1}^{n} x_i \to \max ,$$

esant apribojimams

$$\sum_{i=1}^{n} a_{ij} \cdot x_i \le 1, \quad j = \overline{1,m},$$

čia a_{ij} – grafo incidencijų matricos elementas; šis apribojimas reiškia, kad nepriklausomosios aibės viršūnės tarpusavyje nėra gretimos.

$$x_i \in \{0,1\}, \quad i = \overline{1,n} .$$

Kadangi šis uždavinys neturi tikslių efektyvių sprendimo algoritmų, tai jis sprendžiamas įvairiais euristiniais algoritmais.

Viena iš populiariausių euristikų yra "godus" algoritmas, kai, formuojant nepriklausomąją aibę, kiekviename žingsnyje į šią aibę įtraukiame mažiausio laipsnio viršūnę. Šį euristinį algoritmą galima užrašyti taip.

begin
$A := \varnothing$;
while "grafas turi viršūnių" do
begin
1) Rasti mažiausio laipsnio viršūnę v.
2) $A := A \cup \{v\}$;
3) Iš grafo pašalinti viršūnę v ir jai gretimas viršūnes, t.y. pašalinti viršūnių $\{v\} \cup N(v)$ *aibę.*
end;
end;

Aišku, kad šis algoritmas visada apskaičiuos maksimaliąją, tačiau ne visada didžiausiąją nepriklausomąją aibę.

Žemiau pateikta šį algoritmą realizuojanti procedūra.

```
const c = 100;
type
   mas  = array [1..c] of integer;
procedure neprk (n, m : integer; L,l st : mas; var alfa : integer; var a : mas);
{ Procedūra neprk apskaičiuoja grafo, nusakyto masyvais L ir lst,
nepriklausomumo skaičių alfa ir didžiausią nepriklausomą aibę a.
Formalūs parametrai:
   n – grafo viršūnių skaičius,
   m – grafo briaunų (lankų) skaičius,
   L, lst – grafą nusakantys tiesioginių nuorodų masyvai,
   alfa – grafo nepriklausomumo skaičius,
```

```
    a – didžiausia nepriklausoma aibė. }
var s, i, k, u, v, min : integer;
    p : boolean;
    r, d : mas;
begin
alfa := 0;
{ Visos viršūnės – nenudažytos }
for i := 1 to n do d [i] := 1;
p := true; { Nepriklausoma aibė – neapskaičiuota }
while p do
    begin
        { Viršūnių laipsnių apskaičiavimas }
        for i := 1 to n do
        if d [i] = 1 then
            begin
                s := 0;
                for k := lst [i] + 1 to lst [i + 1] do
                    begin
                        u :=l [k];
                        s := s + d [u];
                    end;
                r [i] := s;
            end;
        { Rasti mažiausio laipsnio viršūnę }
        min := n;
        for i := 1 to n do
        if (d [i] = 1) and (min > r[i]) then
            begin
                min := r [i];
                k := i;
            end;
        if min = n then p := false
                    else
            begin
                { Viršūnę k talpiname į aibę a }
                alfa := alfa + 1;
                a [alfa] := k;
                { Šaliname viršūnę k ir visas jai gretimas viršūnes }
                d [k] := 0;
                for i := lst [k] + 1 to lst [k + 1] do
                    begin
```

```
                    v := L [i];
                    d [v] := 0;
                end;
            end;
    end;
end;
```

Eilė praktinių uždavinių yra formuluojami kaip grafo nepriklausomumo skaičiaus ieškojimo uždaviniai.

Pirmas pavyzdys (Gausas). Ar galima šachmatų lentoje sustatyti 8 valdoves taip, kad jos nekirstų viena kitos?

Pastaba. Analogišką uždavinį galima formuluoti vietoj valdovių imant kitas šachmatų figūras, t.y. kokį didžiausią skaičių tų pačių šachmatų figūrų galima šachmatų lentoje sustatyti taip, kad jos nekirstų viena kitos.

Sudarykime 64 viršūnių grafą. Šio grafo viršūnės vaizduoja šachmatų lentos langelius. Viršūnes u ir v jungsime briauna, jei iš šachmatų lentos langelio u galime patekti į langelį v valdovės ėjimu.

Tada, jei šio grafo nepriklausomumo skaičius yra lygus 8, tai šis uždavinys turi sprendinį, ir valdoves reikia statyti į didžiausios nepriklausomosios aibės viršūnes atitinkančius šachmatų lentos langelius.

1854 m. Berlyno šachmatų žurnale buvo paskelbta 40 šio uždavinio sprendinių. Gausas galvojo, kad yra 76 šio uždavino sprendiniai. Iš tikro jų yra 92 ir juos galima sutraukti į 12 diagramų:
(7, 2, 6, 3, 1, 4, 8, 5), (6, 1, 5, 2, 8, 3, 7, 4), (5, 8, 4, 1, 7, 2, 6, 3), (3, 5, 8, 4, 1, 7, 2, 6), (4, 6, 1, 5, 2, 8, 3, 7), (5, 7, 2, 6, 3, 1, 4, 8), (1, 6, 8, 3, 7, 4, 2, 5), (5, 7, 2, 6, 3, 1, 8, 4), (4, 8, 1, 5, 7, 2, 6, 3), (5, 1, 4, 6, 8, 2, 7, 3), (4, 2, 7, 5, 1, 8, 6, 3), (3, 5, 2, 8, 1, 7, 4, 6).

Kaip šifruoti diagramas? Pavyzdžiui, pirmoji diagrama reiškia, kad, sunumeravus šachmatų lentos eilutes iš apačios į viršų skaičiais nuo 1 iki 8, valdoves statysime taip: 8-ojoje (viršutinėje) eilutėje valdovę statysime 7-ajame stulpelyje, 7-ojoje eilutėje valdovę statysime 2-ajame stulpelyje ir t.t. Kitos diagramos iššifruojamos taip pat.

Kiekviena diagrama, išskyrus paskutiniąją (3, 5, 2, 8, 1, 7, 4, 6), duoda 8 sprendinius, kurie gaunami pasukus šachmatų lentą 90°, 180° ir 270° bei paėmus kiekvieno varianto veidrodinį atspindį pagrindinės įstrižainės atžvilgiu.

Paskutinioji diagrama duoda tik 4 sprendinius, nes, pasukus lentą 180°, digrama transformuojasi pati į save.

Antras pavyzdys. Tai maksimaliosios klikos radimo uždavinys.

Apibrėžimas. (n,m)-grafo $G=(V,U)$ viršūnių poaibis vadinams klika, jei bet kokios dvi šio poaibio viršūnės yra gretimos, t.y. jei šio poaibio indukuotas poaibis yra pilnasis.

Klika yra maksimalioji, jei ji nėra didesnės klikos pografis.

Didžiausioji klika – tai klika, kurios viršūnių skaičius tarp visų klikų yra didžiausias.

Didžiausiosios klikos viršūnių skaičius vadinamas grafo tankiu (arba klikiniu skaičiumi) ir žymimas $\varphi(G)$, t.y. $\varphi(G) = \max_{A \in T} |A|$, čia T – maksimaliųjų klikų šeima.

Aišku, kad grafo G didžiausioji klika sutampa su grafo G papildomojo grafo $\overline{G}$ didžiausiąja nepriklausomąja aibe, t.y. $\varphi(G) = \alpha(\overline{G})$.

Pastaba. Visų maksimaliųjų klikų radimo algoritmas išnagrinėtas lit. [RND80, 389 – 396 p.p.].

Reikia pabrėžti, kad grafo maksimaliųjų klikų skaičius gali augti eksponentiškai, priklausomai nuo viršūnių skaičiaus.

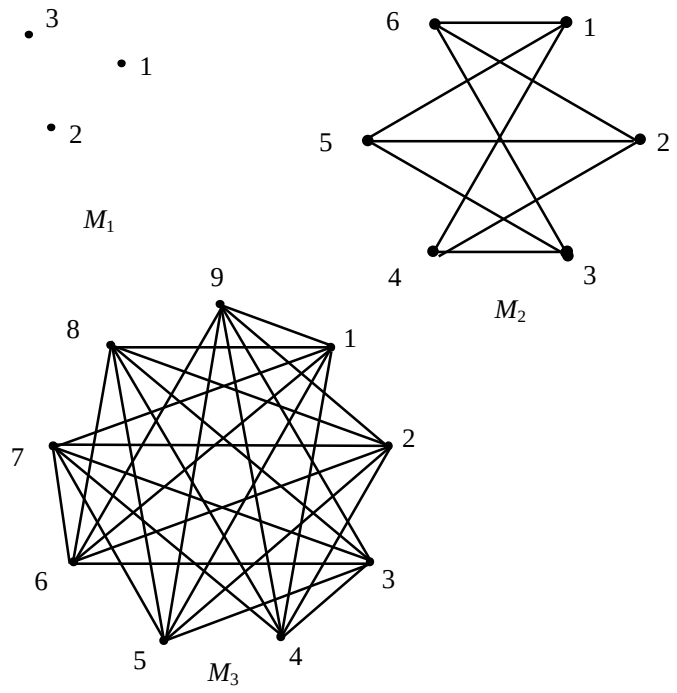
Pavyzdžiui, panagrinėkime Muno ir Mozerio gautą M_n. Šis grafas turi 3n viršūnių, kurios išskaidytos triadomis:

$$\{1, 2, 3\}, \{4, 5, 6\},...,\{3n\text{-}2, 3n\text{-}1, 3n\};$$

M_n grafe kiekvienos triados viršūnės tarpusavyje nėra gretimos, tačiau kiekvienos triados viršūnės yra gretimos visoms likusioms viršūnėms. 2.11.7 pav. pavaizduoti M_1, M_2 ir M_3 grafai.

Naudojant indukciją, galime įrodyti, kad M_n turi 3^n maksimaliųjų klikų ir kiekviena klika turi n viršūnių.

Tai akivaizdu, kai $n = 1$ ir $n = 2$. Tarkime, kad M_{n-1} turi $3^{n\text{-}1}$ maksimaliųjų klikų, turinčių po n-1 viršūnę. Tada kiekviena iš trijų viršūnių, pridedamų formuojant M_n grafą, duoda kliką su kiekviena M_{n-1} grafo klika. Vadinasi, M_n maksimaliųjų klikų skaičius yra $3 \cdot 3^{n-1} = 3^n$, ir kiekviena klika turi n viršūnių.



2.11.7 pav. Muno ir Mozerio grafai

Trečias pavyzdys (15-os mergaičių uždavinys). Pensione mokosi 15-ka megaičių, kurias pažymėkime taip:a, b, c, d, e, f, g, h, i, j, k, l, m, n, o. Mergaitės po tris kiekvieną dieną eina pasivaikščioti. Ar galima taip sudaryti savaitės pasivaikščiojimų tvarkaraštį, kad bet kurios dvi mergaitės į trejetą patektų ne daugiau kaip vieną kartą?

Keli, pasinaudodamas simetrija, rado šio uždavinio sprendinį, kuris pateiktas mergaičių pasivaikščiojimo tvarkaraščio lentelėje (žiūr 2.11.2 lentelę).

Norėdami išspręsti šį uždavinį, pirmiausia spręskime pagalbinį uždavinį: ar galima 15 mergaičių suskirstyti į 35 trejetus, kad bet kurios dvi mergaitės nepatektų į trejetą daugiau kaip vieną kartą?

2.11.2 lentelė. Mergaičių pasivaikščiojimo tvarkaraštis

Pirma-dienis	Antra-dienis	Trečia-dienis	Ketvir-tadienis	Penkta-dienis	Šesta-dienis	Sekma-dienis
a f k	a b l	a l m	a d o	a g n	a h j	a c i
b g l	c n o	b c f	b i k	b d j	b m n	b h o
c m h	d f l	d e h	c j l	c e k	c d g	d k m
d i n	g k h	g i o	e g m	f m o	e f i	e l n
e j o	i j m	j k n	f h n	h i l	k l o	f g j

Šiam uždaviniui spręsti sudarykime 455 viršūnių grafą. Kiekviena viršūnė vaizduoja C_{15}^3 trejetą. Dvi viršūnės *u* ir *v* jungiamos briauna, jei *u* ir *v* trejetuose yra tos pačios dvi mergaitės. Raskime šio grafo nepriklausomumo skaičių ir šį skaičių atitinkančią nepriklausomąją aibę. Kadangi kiekviena mergaitė negali būti daugiau kaip 7-iuose trejetuose, tai didžiausioji nepriklausomoji aibė turės $15 * 7 * 1/3 = 35$ trejetus.

Turėdami šio pagalbinio uždavinio sprendinį, sudarykime grafą, kurio viršūnės vaizduoja sprendinio trejetus. Dvi viršūnės jungiamos briauna, jei ta pati mergaitė priklauso abiem trejetams. Jei šio grafo chromatinis skaičius lygus 7, tai trejetai, nudažyti ta pačia spalva, yra mergaičių pasivaikščiojimų vienos dienos tvarkaraštis. Reikia pastebėti, kad ne kiekvienas pagalbinio uždavinio sprendinys duoda 15-os mergaičių uždavinio sprendinį.

Nepriklausomumo skaičiaus įverčiai

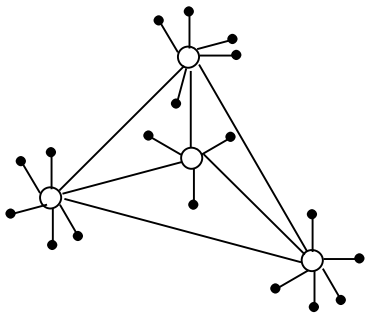
1. $\alpha(G) \geq \sum_{v \in V} (1 + d(v))^{-1}$ (Wei).
2. $\alpha(G) \geq \frac{n}{1 + \bar{d}}$ (Myers, Liu), čia $\bar{d} = \frac{2m}{n}$ – vidutinis viršūnės laipsnis.
3. $\alpha(G) \geq \left\lceil \frac{2pn - 2m}{p(p+1)} \right\rceil$, $p = 1 + \left\lfloor \frac{2m}{n} \right\rfloor$ (Jeršovas, Kožuchinas).
4. $\alpha(G) \leq p^0 + \min\{p^-, p^+\}$, čia p^+, p^- ir p^0 atitinkamai grafo gretimumo matricos teigiamų, neigiamų ir nulinių tikrinių reikšmių skaičius (D.Cvetkovičius, 1973).
5. $\alpha(G) \geq \frac{n}{\max\limits_{v \in V} d(v)}$ (Brooks).

Primename, kad šiose formulėse n – viršūnių skaičius, m – briaunų skaičius, o $d(v)$ – v-osios viršūnės laipsnis.

Dar kartą apie grafo dažymą. Chromatinis skaičius $\gamma(G)$ ir nepriklausomumo skaičius $\alpha(G)$ surišti nelygybe: $\alpha(G)\cdot\gamma(G)\geq n$.

Aišku, kad grafo viršūnes galima išskaidyti į $\gamma(G)$ nepriklausomųjų poaibių: poaibį sudaro viršūnės, nudažytos ta pačia spalva. Kadangi kiekvieno šio poaibio elementų skaičius neviršija $\alpha(G)$, tai $\alpha(G)\cdot\gamma(G)\geq n$. Kyla klausimas, ar tarp šių skaičių nėra glaudesnio ryšio? Gal chromatinį skaičių galima rasti tokiu būdu. Pirmiausia apskaičiuojame grafo G didžiausiąją nepriklausomąją aibę A_1 ir šios aibės viršūnes nudažome pirmąja spalva. Po to randame pografio, kurį indukuoja aibė V/A_1, didžiausiąją nepriklausomąją aibę A_2 ir šias viršūnes dažome antrąja spalva ir t.t., kol nudažome visas grafo viršūnes.

Reikia pabrėžti, kad šis metodas gali apskaičiuoti neoptimalų chromatinį skaičių (žr. 2.11.8 pav.). Šiam grafui $\gamma(G)=4$. Tamsiais taškais pažymėtos didžiausios nepriklausomosios aibės viršūnės. Jei visas jas nudažysime viena spalva, tai likusio grafo viršūnėms nudažyti dar teks panaudoti keturias naujas spalvas. Vadinasi, aukščiau pateiktas metodas duos neoptimalų chromatinį skaičių.

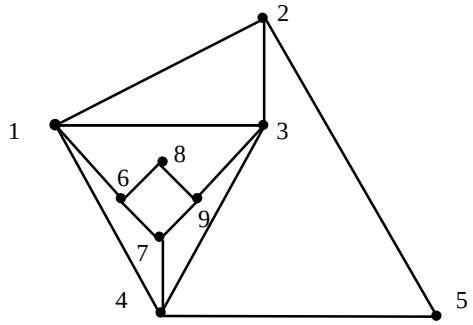


2.11.8 pav. Ryšys tarp $\alpha(G)$ ir $\gamma(G)$

2.11.4. Dominavimo skaičius

Apibrėžimas. (n,m)-grafo $G=(V,U)$ viršūnių poaibis A $(A\subseteq V)$ yra ***dominuojančioji aibė*** (kitur literatūroje ***išoriškai stabilioji aibė***), jei kiekviena grafo viršūnė, nepriklausanti poaibiui A, yra gretima bent vienai poaibio A viršūnei, t.y. jei visos likusios grafo viršūnės, nutolusios nuo bent vienos dominuojančios aibės viršūnės atstumu, lygiu vienetui.

Pavyzdžiui, 2.11.9 pav. grafui aibės $\{1,2,7\}$, $\{2,7\}$, $\{1,3,5\}$.



2.11.9 pav. Dominavimo skaičius

Akivaizdu, kad jei A yra dominuojančioji aibė ir $A \subseteq B$, tai B taip pat dominuojančioji aibė.

Apibrėžimas. Dominuojančioji aibė, kurios bet kuris tikrinis poaibis nėra dominuojančioji aibė, vadinama ***minimaliąja*** dominuojančiąja aibe.

Apibrėžimas. Dominuojančioji aibė, turinti mažiausią elementų skaičių, vadinama ***mažiausiąja*** dominuojančiąja aibe.

Apibrėžimas. Skaičius $\beta(G) = \min_{A \in T}(A)$, čia T – grafo G dominuojančiųjų aibių šeima., t.y. mažiausios dominuojančiosios aibės elementų skaičius, vadinamas grafo dominavimo skaičiumi.

2.11.9 pav. grafo dominavimo skaičius yra 2, o mažiausia dominuojančioji aibė – $\{2,7\}$.

Mažiausios dominuojančiosios aibės, o tuo pačiu ir dominavimo skaičiaus radimo uždavinys yra tipiškas padengimo uždavinys.

Padengimo uždavinį galima formuluoti taip. Duota aibė $S = \{s_1, s_2, ..., s_n\}$ ir aibės S poaibių šeima $A = \{A_1, A_2, ..., A_m\}$. Rasti tokį mažiausią poaibių A_i rinkinį (padengimą), kad kiekvienas aibės S elementas s_k $k = \overline{1,n}$ priklausytų (būtų padengtas) bent vienam rinkinio poaibiui A_i.

Formalai šį uždavinį galime užrašyti taip.

Įveskime kintamuosius $x_i = \begin{cases} 1, \text{ jei } A_i \text{ įeina į padengimą}, \\ 0, \text{ priešingu atveju}, \end{cases}$ $i = \overline{1,m}$.

Poaibių A_i šeimą vaizduokime $(n \times m)$ formato matrica $A = [a_{ij}]$, $i = \overline{1,n}$, $j = \overline{1,m}$, čia

$$a_{ij} = \begin{cases} 1, \text{ jei } s_i \in A_j, \\ 0, \text{ priešingu atveju,} \end{cases}$$

t.y. *j*-asis matricos *A* stulpelis vaizduoja poaibį A_j. Tada padengimo uždavinys ekvivalentiškas tokiam optimizavimo uždaviniui:

$$\min \; z = \sum_{i=1}^{m} x_i$$

esant apribojimams:

- "kiekvienas elementas priklauso bent vienam į padengimą įeinančiam poaibiui",
- $\sum_{j=1}^{m} a_{ij} \cdot x_j \geq 1, \quad i = \overline{1,n}$,
- $x_i \in \{0,1\}, \quad i = \overline{1,n}$.

Dominavimo skaičiaus ieškojimo atveju aibė *S* yra grafo viršūnių aibė *V*. Kiekviena grafo viršūnė $v \in V$ apibrėžia viršūnių poaibį $A_v = \{v\} \cup N(v)$, $v \in V$. Reikia rasti tokį mažiausią poaibių A_v skaičių (viršūnių *v* aibę), kad kiekviena grafo viršūnė priklausytų bent vienam rinkinio poaibiui A_v.

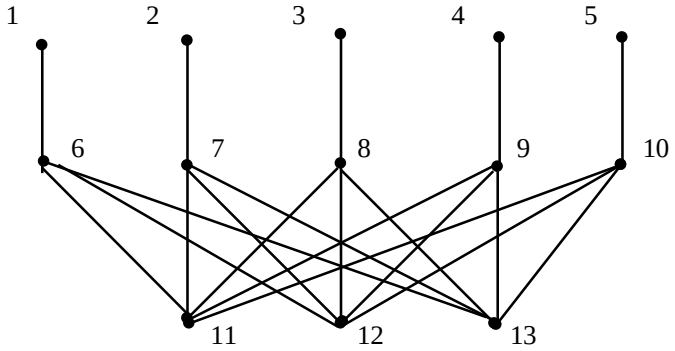
Kadangi visi tikslūs padengimo uždavinio sprendimo metodai yra neefektyvūs, tai paprastai šis uždavinys sprendžiamas euristiniais algoritmais. Viena iš populiariausių euristikų yra "godaus" algoritmo euristika: "kol yra nepadengtų viršūnių, kiekviename žingsnyje į dominuojančiąją aibę (padengimą) įtrauksime tokią aibę A_v, kuriai priklauso didžiausias skaičius nepadengtų viršūnių, t.y. tokią viršūnę, kurios laipsnis yra didžiausias".

Formalai šį algoritmą galime užrašyti taip.

begin
  $A := \varnothing$; {*A – mažiausioji dominuojančioji aibė*}
  while "grafas turi viršūnių" do
  begin
    1. *Rasti didžiausio laipsnio viršūnę v.*
    2. $A := A \cup \{v\}$
    3. *Iš grafo pašalinti viršūnių* $\{v\} \cup N(v)$ *aibę.*
  end;
end;

Nesunku pastebėti, kad šis algoritmas yra analogiškas didžiausios nepriklausomosios aibės apskaičiavimo algoritmui. Aišku, kad šis algoritmas visada apskaičiuos minimaliąją dominuojančiąją aibę, tačiau ji ne visada bus mažiausioji. Pavyzdžiui, 2.11.10 pav. grafui mažiausioji dominuojančioji aibė

yra $\{6,7,8,9,10\}$, o pagal algoritmą apskaičiuota minimalioji dominuojančioji aibė yra $\{11,12,13,1,2,3,4,5\}$.



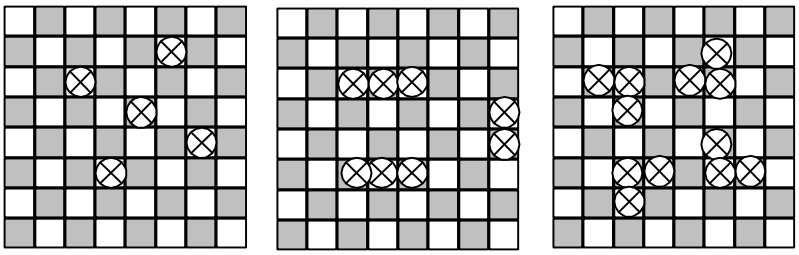
2.11.10 pav. "Godus" algoritmas neapskaičiuoja minimaliosios dominuojančiosios aibės

Aptarsime keletą pavyzdžių, kuriuos sprendžiant reikia apskaičiuoti mažiausiąją dominuojančiąją aibę.

Pirmas pavyzdys. ***Uždavinys apie sargybinius***. Tarkime, kad grafas *G* (pvz. žr. 2.11.9 pav.) yra *N*-miesto kalėjimo planas. Grafo viršūnės vaizduoja kameras, kuriose įkalinti pavojingi nusikaltėliai. Viršūnės *u* ir *v* jungiamos briauna, jei jas jungia tiesus koridorius. Reikia rasti mažiausią skaičių sargybinių, kad jie galėtų sekti visų kamerų duris.

Antras pavyzdys. ***Penkių valdovių uždavinys***. Kiek mažiausiai šachmatų lentoje reikia pastatyti valdovių (galima klausti, kiek rikių, kiek žirgų ir pan.), kad kiekvienas šachmatų lentos langelis būtų kertamas.

pav. a) parodytas mažiausias valdovių skaičius, b) – mažiausias rikių skaičius, c) – mažiausias žirgų skaičius.



a) valdovės $\beta(9)=5$ b) rikiai $\beta(9)=8$ c) žirgai $\beta(9)=12$

2.11.11 pav. Šachmatų uždavinys

Aišku, kad sprendžiant šį uždavinį nagrinėsime 64 viršūnių grafą. Kiekviena viršūnė vaizduoja šachmatų lentos langelį. Viršūnė u jungiama briauna su viršūne v, jei iš langelio u galima patekti į langelį v nurodytos figūros (valdovės, žirgo ir pan.) ėjimu. Tada šiame grafe reikės rasti mažiausiąją dominuojančiąją aibę.

Trečias pavyzdys. Prancūzijos mugėse mėgstamas toks žaidimas: ant stalo padėtas didelis baltas diskas (tarkime, šio disko spindulys lygus 1 m); šį diską reikia pilnai uždengti šešiais mažesniais (spindulio $\rho < 1\,\text{m}$) raudonais diskais, kuriuos žaidėjas vieną po kito deda ant balto disko. Kartą padėtas raudonasis diskas nebejudinamas. Koks turi būti mažiausias raudonojo disko spindulys ρ, kad baltasis diskas būtų pilnai padengtas?

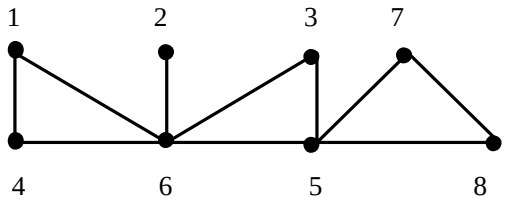
Šis uždavinys ekvivalentus uždaviniui: rasti mažiausią dominuojančiąją aibę begaliniame grafe (V,U), čia V – baltojo disko taškų aibė, o viršūnės u ir v jungiamos briauna, jei atstumas tarp šių viršūnių $d(v,u) < \rho$.

Ketvirtas pavyzdys. Duota aibė gyvenviečių, sujungtų kelių tinklu. Kai kuriose gyvenvietėse reikia pastatyti buitines įmones taip, kad nuo bet kurios gyvenvietės buitinė įmonė būtų ne toliai nei duotas atstumas c. Kokiose gyvenvietėse statyti įmones, kad jų skaičius būtų mažiausias?

Sudarykime grafą, kurio viršūnės vaizduoja gyvenvietes. Dvi viršūnės jungiamos briauna, jei atstumas tarp jų yra mažesnis už c. Tada mažiausios dominuojančiosios aibės viršūnės ir nurodys gyvenvietes, kuriose turi būti statomos buitinės įmonės.

Apibrėžimas. Grafo viršūnių poaibis, kuris vienu metu yra ir nepriklausomas ir dominuojantis, vadinamas ***grafo branduoliu***.

Aišku, kad grafo nepriklausomoji aibė yra maksimali (nebūtinai didžiausia) tada ir tiktai tada, kada ji yra dominuojanti. Pavyzdžiui, 2.11.4.4 pav. grafo aibės $\{1,2,3,7\}$, $\{1,2,3,8\}$, $\{2,3,5,7\}$, $\{2,3,5,8\}$, $\{4,7\}$, $\{4,8\}$ yra grafo branduoliai.



2.11.12 pav. Grafo branduolys

Reikia pabrėžti, kad tiek nepriklausomosios aibės, tiek ir dominuojančiosios aibės pateikti generavimo algoritmai įgalina apskaičiuoti neorientuotojo grafo branduolį.

Apibrėžimas. Sakykime, kad viršūnė ir briauna dengia viena kitą, jei jos incidentiškos. Pavyzdžiui, briauna (u,v) dengia viršūnes *u* ir *v*, o viršūnė *u* arba viršūnė *v* dengia briauną (u,v).

Apibrėžimas. Grafo viršūnių poaibis V' vadinamas ***viršūniniu denginiu***, jei kiekviena grafo briauna yra incidentiška bent vienai aibės V' viršūnei.

Denginys vadinamas ***minimaliuoju***, jei bet koks jo tikrinis poaibis nėra denginys; denginys vadinams ***mažiausiuoju***, jei jo elementų skaičius yra mažiausias tarp visų denginių. Šis elementų skaičius žymimas $\beta_0(G)$ ir vadinamas ***viršūninio denginio skaičiumi***.

Pavyzdžiui, 2.11.12 pav. grafui aibės $X_1 = \{4,5,6,8\}$, $X_2 = \{4,5,6,7\}$ ir $X_3 = \{1,2,3,5,6,8\}$ yra denginiai. X_1 ir X_2 – mažiausi denginiai. Vadinasi, $\beta_0(G) = 4$.

Tarp denginio ir nepriklausomosios aibės yra glaudus ryšys, kurį nusako žemiau pateikta teorema.

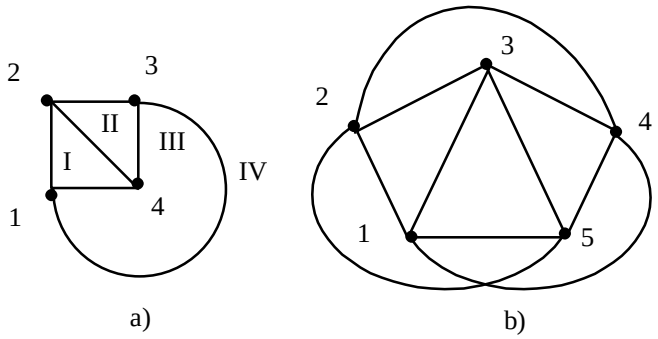
Teorema. Grafo $G = (V,U)$ viršūnių poaibis *A* yra mažiausiasis (minimalusis) denginys tada ir tiktai tada, kai aibė $\overline{A} = V \setminus A$ yra didžiausioji (maksimalioji) grafo *G* nepriklausomoji aibė. Vadinasi, $\alpha(G) + \beta_0(G) = n$.

Tuo būdu, visi nepriklausomumo skaičiaus įverčiai gali būti naudojami ir denginio įverčiui.

2.12. Plokštieji grafai

Apibrėžimas. (n,m)- grafas (bendru atveju multigrafas) $G = (V,U)$ vadinamas plokščiuoju, jei jį galima plokštumoje pavaizduoti taip, kad briaunos kirstųsi tik viršūnėse.

2.12.1 a) pav. pavaizduotas plokštusis grafas K_4, o 2.12.1 b) pav. – neplokštusis grafas K_5. Kad K_5 nėra plokštusis grafas, įrodysime vėliau.



2.12.1 pav. Plokštieji grafai: a) plokštusis; b) neplokštusis

Reikia pažymėti, kad nustatymas fakto, ar grafas yra plokštusis ir grafo pavaizdavimas plokštumoje, kad jo briaunos kirstųsi tik viršūnėse, yra praktiškai svarbus uždavinys: elektrinės schemos (spausdintas montažas) yra plokštieji grafai.

Aptarsime keletą plokščiojo grafo sąvokų.

Grafo siena – tai plokštumos dalis, apribota ciklu, kurioje nėra nei viršūnės, nei briaunos.

Ciklas, ribojantis grafo sieną, vadinamas ***minimaliuoju ciklu***.

Dvi grafo sienos yra ***gretimos***, jei jos turi bent vieną bendrą briauną.

Pavyzdžiui, K_4 grafas (žr. 2.12.1 a) pav.) turi keturias sienas. Jos pažymėtos romėniškais skaičiais, o jų minimalieji ciklai atitinkamai yra: $1,2,4,1$; $2,3,4,2$; $1,4,3,1$ ir $1,2,3,1$. Ciklas $1,2,3,1$ riboja begalinę IV sieną, kuri yra šio ciklo išorėje. Kiekvienas plokštusis grafas turi vieną begalinę sieną. Todėl plokščiojo grafo baigtinės sienos dar vadinamos ***vidinėmis***, o begalinė siena – ***išorine***.

Teorema. (n,m) - plokščiojo grafo baigtinių sienų minimalūs ciklai yra tiesiškai nepriklausomi ir sudaro bazę.

Išvada. (Oilerio formulė). Plokščiojo grafo sienų skaičius *f*, briaunų skaičius *m* ir viršūnių skaičius *n* susieti formule

$$f - m + n = 2,$$

kuri vadinama Oilerio formule.

Oilerio formulės teisingumas išplaukia iš to, kad plokščiojo grafo baigtinių sienų minimalūs ciklai yra nepriklausomi ir sudaro bazę. Kitaip tariant, plokščiojo grafo ciklomatinis skaičius $\nu(G) = f - 1$. Iš čia

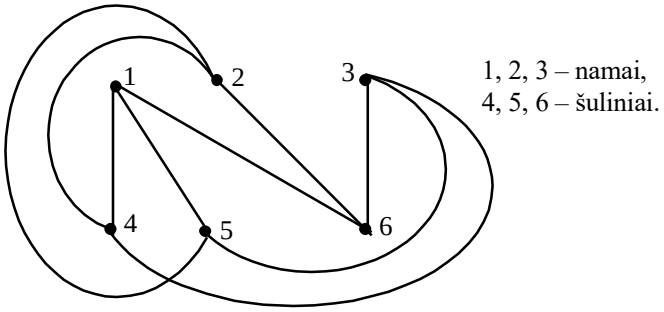
$$m - n + 1 = f - 1.$$

Tada

$$f - m + n = 2.$$

Remdamiesi Oilerio formule įrodysime, kad pilnasis dvidalis grafas $K_{3,3}$ ir penkių viršūnių pilnasis grafas K_5 nėra plokštieji.

$K_{3,3}$ grafas nėra plokštusis. Tarkime priešingai, kad $K_{3,3}$ grafas yra plokštusis. ($K_{3,3}$ grafas pavaizduotas 2.12.2. pav.).



2.12.2 pav. $K_{3,3}$ grafas nėra plokštusis

Tada, remdamiesi Oilerio formule, galime apskaičiuoti šio grafo sienų skaičių:

$$f = m - n + 2 = 9 - 6 + 2 = 5.$$

Sudarysime pagalbinį incidencijų sienos – briaunos grafą. Šis grafas yra dvidalis (A,B,U). Aibės A viršūnės vaizduoja $K_{3,3}$ grafo sienas; aibės B viršūnės vaizduoja $K_{3,3}$ grafo briaunas. Viršūnė $a \in A$ jungiama briauna su viršūne $b \in B$, jei $K_{3,3}$ briauna b yra incidentiška $K_{3,3}$ sienai a.

Įvertinkime šio pagalbinio grafo briaunų skaičių m^*.

Aišku, kad $m^* \le 2m$, čia m – $K_{3,3}$ grafo briaunų skaičius (kiekviena grafo $K_{3,3}$ briauna incidentiška nedaugiau kaip dviem sienom). Kita vertus, $m^* \ge 4f$, nes $K_{3,3}$ grafo sienų minimalių ciklų ilgiai nemažesni nei 4. Tarus, kad minimalaus ciklo ilgis yra 3, turėtume jungti arba du namus, arba du šulinius, o taip daryti negalima, nes grafas yra dvidalis ir nemultigrafas. Vadinasi, $4f \le m^* \le 2m$. Iš šios nelygybės gausime, kad $20 \le m^* \le 18$. Gautas prieštaravimas rodo, kad prielaida buvo klaidinga.

K_5 grafas nėra plokštusis. Įrodymas, kad K_5 grafas nėra plokštusis grafas, yra analogiškas, kaip ir įrodymas, kad $K_{3,3}$ grafas nėra plokštusis.

Tarkime, kad K_5 grafas yra plokštusis grafas. Tada pagal Oilero formulę apskaičiuosime jo sienų skaičių:

$$f = 10 - 5 + 2 = 7.$$

Kaip ir $K_{3,3}$ atveju, sudarykime pagalbinį incidencijų sienos – briaunos grafą ir įvertinkime jo briaunų skaičių m^*. Nesunku suprasti, kad m^* tenkina nelygybę

$$3f \le m^* \le 2m,$$

t.y. kiekviena grafo K_5 briauna incidentiška ne daugiau kaip dviem sienom, o trumpiausio minimalaus ciklo ilgis yra nemažesnis nei 3 (K_5 nėra multigrafas). Vadinasi, $21 \le m^* \le 20$. Gautas prieštaravimas rodo, kad K_5 nėra plokštusis grafas.

Aptarsime porą plokščiojo grafo savybių. Viena iš jų gali būti dalinis kriterijus, nustatant, ar grafas yra plokštusis, t.y. būtina plokščiojo grafo sąlyga, o antroji – įrodant, kad bet kokį plokštųjį grafą galima nudažyti 5-mis spalvomis.

Pirmoji savybė. Jei G – jungusis plokštusis (n,m)-grafas yra nemultigrafas, tai, kai $n \ge 3$, $m \le 3n - 6$.

Įrodymas. Kadangi grafas G nėra multigrafas, tai kiekvienos sienos minimalaus ciklo ilgis yra nemažesnis nei 3. Pasinaudodami incidencijų sienos – briaunos grafu, gausime, kad $3f \le 2m$. Iš šios nelygybės išplaukia, kad $f \le (2/3)m$. Remdamiesi Oilerio formule, gausime:

$$f - m + n = 2,$$

$$m = f + n - 2,$$

$$m \le \frac{2}{3}m + n - 2,$$

$$m \le 3n - 6.$$

Antroji savybė. Bet kuriame plokščiajame grafe G, kuris nėra multigrafas, yra bent viena viršūnė, kurios laipsnis nedidesnis nei 5.

Įrodymas. Sudarykime grafo G incidencijų sienos – briaunos grafą. Tada šio grafo briaunų skaičius m^* tenkina nelygybę:

$$3f \le m^* \le 2m.$$

Iš šios nelygybės gauname, kad $f \le (2/3)m$.

Sudarykime plokščiojo grafo G incidencijų viršūnės – briaunos grafą (A,B,U). Viršūnių aibė A vaizduoja grafo G viršūnes, o viršūnių aibė B – grafo G briaunas. Viršūnė $a \in A$ jungiama briauna su viršūne $b \in B$, jei grafo

G viršūnė *a* incidentiška grafo *G* briaunai *b*. Tarkime, kad plokščiojo grafo *G* visų viršūnių laipsniai nemažesni nei 6. Tada pagalbinio incidencijų viršūnės – briuanos grafo briaunų skaičius m^{**} tenkins nelygybę:

$$6n \le m^{**} \le 2m .$$

Iš šios nelygybės gausime, kad $n \le \frac{1}{3}m$. Remdamiesi Oilerio formule, gausime:

$$2 = f - m + n \le \frac{2}{3}m - m + \frac{1}{3}m = 0 .$$

Gautas prieštaravimas rodo, jog prielaida, kad visų grafo *G* viršūnių laipsniai yra nemažesni nei 6, yra klaidinga. Vadinasi, plokštusis grafas turi bent vieną viršūnę, kurios laipsnis nedidesnis už 5.

Kaip nustatyti, ar duotas grafas yra plokštusis? Atsakymą į šį klausimą pirmasis davė lenkų matematikas G.Kuratovskis 1930 m. (G.Kuratovski. Sur le probleme des courbes gauches en topologie. Fund. Math., 15-16 (1930), 271 p.)[2]. Čia pateikiame 1937 m. K.Vagnerio teoremą, kuri yra analogiška Kuratovskio – Pontriagino teoremai.

Teorema (G.Vagneris, 1937). Grafas *G* yra plokštusis tada ir tiktai tada, kada jis neturi pografių, kuriuos galima sutraukti į $K_{3,3}$ arba K_5 grafus.

Teorema (Koršunov A.D. 1985) [EM90] Beveik nėra plokščiųjų grafų.

Reikia pabrėžti, kad minėtos teoremos padeda tik nustatyti faktą, ar grafas yra plokštusis. Tačiau atsakymas, kad grafas yra plokštusis, yra nekonstruktyvus: lieka neaišku, kaip grafą pavaizduoti plokštumoje, kad jo briaunos kirstųsi tik viršūnėse? Atsakymas į šį klausimą kaip tik ir turi pagrindinę praktinę vertę. Todėl labai svarbu turėti algoritmą, kuris leistų pavaizduoti grafą plokštumoje taip , kad briaunos kirstųsi tik viršūnėse, jei grafas yra plokštusis, arba duotų atsakymą “ne”, jei, vaizduojant grafą plokštumoje, paaiškėtų, kad jis nėra plokštusis. Toks algoritmas detaliai išdėstytas literatūroje [RND80, 402 – 424 p.p.].

Plokščiojo grafo dažymas. Kaip buvo minėta aukščiau, plokščiojo grafo keturių spalvų hipotezę, t.y. hipotezę, kad bet kokio plokščiojo grafo viršūnes galima nudažyti 4-iomis spalvomis taip, kad gretimos viršūnės būtų nudažytos skirtinga spalva, iškėlė Augustas de Morganas savo laiške, rašytame 1852 m. spalio 23 d. serui Viljamui Rovanui Hamiltonui.

---

[2] Nepriklausomai tokią pat teoremą 1927 m. suformulavo rusų matematikas L.S.Pontriaginas. Tačiau ši teorema nebuvo publikuota. Todėl teorema, nusakanti būtinas ir pakankamas sąlygas, kad grafas būtų plokštusis, dažnai vadinama Kuratovskio – Pontriagino teorema.

Pirmasis šią hipotezę 1880 m. bandė įrodyti A.Kempė. Tačiau 1890 m. R.Hivudas Kempės įrodyme rado klaidą[3]. Tais pačiais metais R.Hivudas suformulavo ir įrodė teoremą.

Teorema. Kiekvienas plokštusis grafas gali būti nudažytas 5-iomis spalvomis.

Įrodymas. Teorema įrodoma matematinės indukcijos metodu.

Teorema teisinga, jei plokščiojo grafo viršūnių skaičius nedidesnis už penkis.

Tarkime, kad ši teorema teisinga, kai plokščiojo grafo viršūnių skaičius yra $n > 5$.

Įrodysime, kad ši teorema teisinga, kai plokščiojo grafo viršūnių skaičius lygus $n+1$.

Tarkime, kad kai plokščiojo grafo viršūnių skaičius yra $(n+1)$. Kaip įrodėme aukščiau, jame egzistuoja bent viena viršūnė v_0, kurios laipsnis nedidesnis nei 5.

Panagrinėkime du atvejus.

1. $|N(v_0)| \leq 4$. Aišku, kad grafas $G - v_0$ yra plokštusis, ir jį galima nudažyti 5-iomis spalvomis. Viršūnę v_0 nudažysime viena iš 5-ių spalvų, kuri nepanaudota viršūnės v_0 gretimų viršūnių dažyme.
2. Tarkime, kad $N(v_0) = 5$. Tada aibėje $N(v_0)$ egzistuoja dvi negretimos viršūnės v_1 ir v_2, nes, priešingu atveju, aibės $N(v_0)$ indikuotas pografis būtų K_5, o tai reiškia, kad nagrinėjamas grafas nėra plokštusis.

Tarkime, G' yra grafas, gautas apjungus grafo $G - v_0$ viršūnes v_1 ir v_2. Šią apjungtąją viršūnę pažymėkime raide v. Grafą G' galima nudažyti 5-iomis spalvomis. O tai reiškia, kad grafą $G - v_0$ galima nudažyti 5-iomis spalvomis taip, kad viršūnės v_1 ir v_2 būtų nudažytos ta pačia spalva, t.y. viršūnės v spalva. Tada viršūnę v_0 dažysime viena iš spalvų, nepanaudotų viršūnių $N(v_0)$ dažyme.

2.13. Medžiai

Nagrinėsime neorientuotuosius jungiuosius grafus. Jei vienas jungiųjų grafų ribinis atvejis yra pilnieji grafai, t.y. jungieji grafai, turintys didžiausią briaunų skaičių, tai kitas ribinis atvejis yra jungieji grafai, turintys mažiausią briaunų skaičių. Tie grafai vadinami medžiais.

---

[3] Kaip buvo minėta aukščiau, keturių spalvų hipotezę K.Apelis, W.Hakenas ir J.Kochas įrodė 1976 m.

Istoriškai medžiai nepriklausomai buvo atrasti kelis kartus. 19 amžiuje Kirchgofas naudojo medžius tirdamas elektrines grandines. Vėliau matematikas Keli iš naujo apibrėžė medžius, ištyrė jų savybes ir juos naudojo norėdamas apskaičiuoti sočiųjų angliavandenilių izomerų skaičių. Po jo matematikas Žordanas medžius tyrė kaip grynai matematinį objektą.

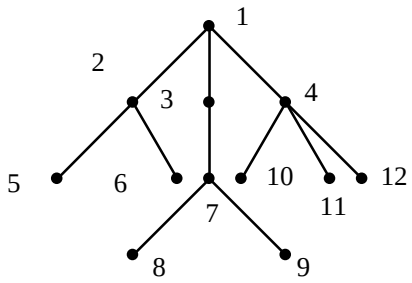
Apibrėžimas. Jungusis grafas, neturintis ciklų, vadinamas medžiu.

Medis turi 6 ekvivalenčias savybes, kiekviena iš kurių gali būti medžio apibrėžimas. Šias savybes nusako

Teorema. Jei (n,m)-grafas G yra medis, tai

1) G – jungusis ir neturi ciklų,
2) G – jungusis ir $m = n-1$,
3) G – neturi ciklų ir $m = n-1$,
4) G neturi ciklų, tačiau įvedus naują briauną, jungiančią bet kokias dvi negretimas medžio viršūnes, atsiranda vienintelis ciklas,
5) Grafas G yra jungusis, tačiau praranda šią savybę, pašalinus bet kurią briauną,
6) Bet kuri viršūnių pora sujungta grandine ir tiktai viena.

2.13.1 pav. pavaizduotas medis. Nesunku įsitikinti šių savybių teisingumu.



2.13.1 pav. Medis

2.13.1 pavaizduoto medžio 1-oji viršūnė vadinama ***medžio šaknimi***. Viršūnės, kurios nutolę nuo šaknies atstumu k (k – natūralusis skaičius), vadinamos ***k-tojo lygio viršūnėmis***. Medžio viršūnės, kurių laipsnis yra lygus 1, vadinamos ***kabančiomis viršūnėmis***. 2.13.1 pav. grafo 2-oji, 3-ioji ir 4-oji viršūnės yra pirmo lygio viršūnės. 5-oji, 6-oji, 8-oji, 9-oji, 10-oji, 11-oji ir 12-oji viršūnės yra kabančios viršūnės.

2.13.1. Dengiančiojo medžio apskaičiavimo uždavinys

Apibrėžimas. (n,m)-grafo G dalinis grafas medis vadinamas ***dengiančiuoju medžiu***.

Teorema. Būtina ir pakankama sąlyga, kad (n,m)-grafas G turėtų dalinį grafą medį yra ta, kad grafas G būtų jungusis.

Būtinumas. Jei grafo dalinis grafas yra medis, tai grafas G yra jungusis, nes medis yra jungusis grafas.

Pakankamumas. Grafas G yra jungusis. Ar grafas turi briauną, kurią pašalinus jungumas neišyra? Jei tokios briaunos nėra, tai pagal 5-ąją savybę grafas G (dalinis grafas) yra medis. Jei tokia briauna yra, tai ją pašaliname ir klausimą kartojame iš naujo.

Šios teoremos įrodymas yra konstruktyvus, t.y. iš jo išplaukia dalinio grafo medžio konstravimo algoritmas. Tačiau reikia pabrėžti, kad šis algoritmas yra neracionalus: kiekviename žingsnyje turime ieškoti briaunos, kurią pašalinus grafas nesuirtų.

Žymiai racionalesni dengiančiojo medžio radimo algoritmai yra pagrįsti ***paieška platyn*** arba ***paieška gilyn***. Tarkime, kad paieškos platyn ar gilyn metu iš viršūnės u atėjome į naują (neaplankytą) viršūnę v. Tada briauna (u,v) yra medžio briauna.

Reikia pastebėti, kad paiešką gilyn ar platyn galima baigti, kai į medį įtrauksime $(n-1)$ briauną, t.y. paiešką galima nutraukti nelaukiant jos natūralios pabaigos.

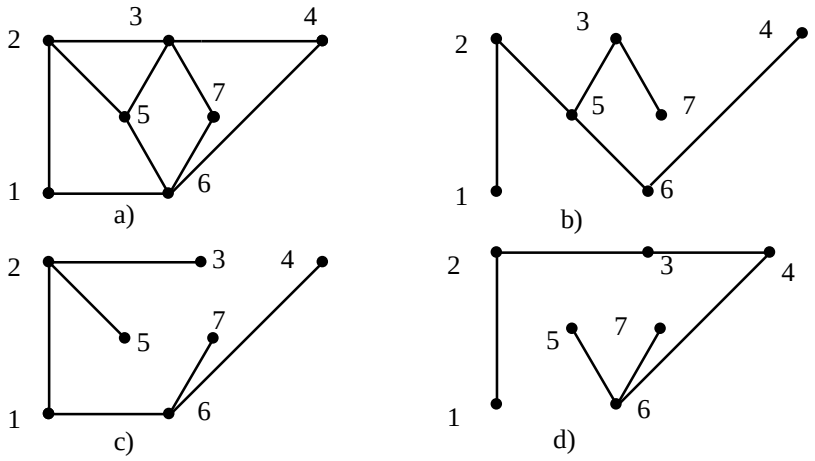
Pavyzdys. Panagrinėkime dengiančio medžio konstravimą visais trim metodais. 3.13.2 a) pav. pavaizduotam grafui.

2.13.2 b) paveiksle pavaizduotas dengiantis medis, apskaičiuotas metodu, išplaukiančiu iš teoremos įrodymo, kai iš pradinio grafo buvo pašalintos briaunos: $(1,6)$, $(2,3)$, $(3,4)$, $(6,7)$. Aišku, kad šių briaunų skaičius yra lygus $m-n+1$, t.y. lygus grafo ciklomatiniam skaičiui. Kaip buvo minėta aukščiau, nagrinėjant nepriklausomų ciklų išskyrimo uždavinį, šios briaunos vadinamos ***atvirkštinėmis briaunomis*** arba ***stygomis***, ir kiekviena iš jų iššaukia nepriklausomą ciklą, kuris sudarytas iš šios briaunos ir medžio briaunų.

2.13.2 c) paveiksle pavaizduotas dengiantis medis, gautas paieškos platyn iš 1-osios viršūnės metodu, laikant, kad kiekvienai viršūnei gretimos viršūnės išdėstytos jų numerių didėjimo tvarka.

Paieškos platyn metu iš 1-osios viršūnės aplankome naujas 2-ąją ir 6-ąją viršūnes. Tuo pačiu briaunos $(1,2)$ ir $(1,6)$ yra medžio briaunos. Po to, paieškos platyn antrojo žingsnio metu iš 2-osios viršūnės aplankome naujas 3-iąją ir 5-ąją viršūnes, o iš 6-osios viršūnės –naujas 4-ąją ir 7-ąją viršūnes. Tuo

būdu briaunos $(2,3)$, $(2,5)$, $(6,4)$ ir $(6,7)$ yra medžio briaunos. Kadangi į medį įtraukėme $(n-1)$, t.y. šešias briaunas, paiešką nutraukiame.



2.13.2 pav. Dalinio grafo – medžio konstravimas

2.13.2 d) paveiksle pavaizduotas dengiantis medis, gautas paieškos gilyn iš 1-osios viršūnės metodu, laikantis taisyklės, kad pirmiausia eisime į gretimą viršūnę, kurios numeris yra mažiausias.

Iš pirmosios viršūnės ateiname į naują 2-ąją viršūnę. Vadinasi, $(1,2)$ yra medžio briauna. Iš 2-osios viršūnės, po bandymo eiti į 1-ąją viršūnę, ateiname į naują 3-iąją viršūnę. Tuo būdu $(2,3)$ yra medžio briauna. Iš 3-iosios viršūnės ateiname į naują 4-ąją viršūnę ir gausime dar vieną medžio briauną $(3,4)$. Iš 4-osios atėjome į naują 6-ąją viršūnę, o iš pastarosios į naujas 5-ąją ir 7-ąją viršūnes. Tuo būdu medis pasipildys briaunomis $(4,6)$, $(6,5)$ ir $(6,7)$. Kadangi į medį įtraukėme $(n-1)$, t.y. šešias briaunas, paiešką gilyn nutraukiame.

2.8.1 ir 2.8.2 paragrafuose detaliai aptarėme paieškos gilyn ir paieškos platyn organizavimo algoritmus. Šie algoritmai su nedidele modifikacija pilnai tinka aptartiems medžio konstravimo metodams realizuoti.

Dengiantys medžiai yra naudojami sprendžiant įvairius uždavinius, pavyzdžiui, konstruojant algoritmą, kurio dėka nustatome, ar grafas yra plokštusis (žr. [RND80]).

Čia aptarsime vieną svarbų praktinį uždavinį, susijusį su ***hidrauliniu vandentiekio tinklo skaičiavimu***.

Tarkime, kad 2.13.2 a) paveiksle pavaizduotas grafas yra vandentiekio (dujotiekio) tinklas. Šio grafo viršūnės vaizduoja sutelktinius vandens vartotojus, o briaunos – vandentiekio vamzdžius (žr. [APS83]). Tarkime, kad žinant vandentiekio vamzdžių skersmenis bei vandens šaltinių debitą [l/s] reikia kiekviename tarpe (briaunoje) apskaičiuoti vandens debitą, vandens tekėjimo greitį bei slėgio kritimą.

Procesai, vykstantys vandentiekio tinkle, aprašomi Kirchgofo dėsniais.

Pirmasis Kirchgofo dėsnis. Kiek vandens atiteka į mazgą (viršūnę), tiek ir išteka, atmetus jo suvartojimą mazge.

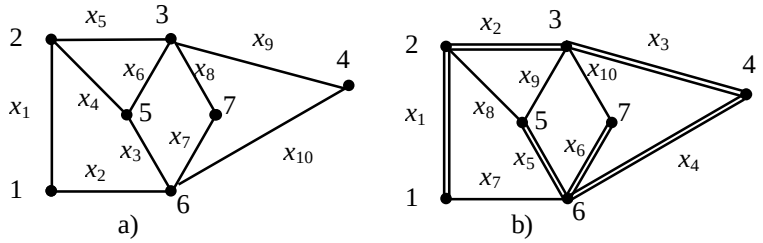
Antrasis Kirchgofo dėsnis. Slėgių kritimų suma žiede (nepriklausomame cikle) lygi nuliui.

Reikia pabrėžti, kad pirmasis Kirchgofo dėsnis užrašomas tiesine lygtimi, o antrasis – netiesine.

Tuo būdu, vandentiekio tinklo procesai aprašomi $(n-1)$ tiesine lygtimi (n – (viršūnių) mazgų skaičius), ir $m-n+1$ netiesine lygtimi.

Ši sistema sprendžiama taip. Pirmiausia apskaičiuojamas atskiras tiesinės dalies sprendinys. Tam tikslui $m-n+1$ kintamasis parenkamas laisvai (remiantis inžineriniais samprotavimais), o likę $(n-1)$ kintamųjų – apskaičiuojami iš $(n-1)$ tiesinių lygčių sistemos. Po to šis sprendinys statomas į netiesines lygtis ir, jei tikslumas netenkinamas, sprendinys iteracinės procedūros pagalba koreguojamas, taip, kad jis visą laiką tenkintų tiesines lygtis, o netiesinių lygčių netiktis artėtų į nulį.

Laisvai pažymėjus kintamuosius (debitus) ir surašius tiesines lygtis, sistema bus reta ir nenuliniai koeficientai matricoje bus išsidėstę netvarkingai.



2.13.3 pav. Sistemos nežinomųjų ir lygčių numeracija

2.13.3 a) pav. pavaizduotam tinklui tiesinės sistemos matrica bus tokia (varnelės žymi nenulinius koeficientus):

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	Mazgo nr.
√	√									1
√			√	√						2
				√	√		√	√		3
								√	√	4
		√	√		√					5
	√	√				√			√	6

Kintamųjų x_7, x_8, x_9 ir x_{10} reikšmes parinkus laisvai, teks spręsti šešių lygčių sistemą, kurios matrica yra kairiau dvigubos linijos. Norėdami išspręsti šią sistemą, pavyzdžiui, Gauso metodu, pirmiausia ją turėsime perskaičiuoti į trikampį pavidalą, o po to, atvirkštinio etapo metu, nuosekliai apskaičiuosime x_6, x_5, ..., x_1 reikšmes. Realių vandentiekio tinklų mazgų (viršūnių) skaičius yra šimtų eilės, o tarpų (briaunų) – tūkstančių eilės. Todėl laisvas nežinomųjų ir lygčių sunumeravimas sprendimo eigoje iššauks dideles atminties ir laiko sąnaudas.

Kelkime klausimą: “Gal galima nurodyti tokią kintamųjų sunumeravimo eilę ir tokią lygčių surašymo tvarką, kad tiesinės sistemos dalies matrica būtų pseudotrikampė?” Atsakymas į šį klausimą teigiamas. Paieškos platyn arba gilyn metodu raskime dengiantį medį ir surašykime grafo briaunas ta tvarka, kaip jos buvo jungiamos į medį. Taip pat surašykime grafo viršūnes ta tvarka, kuria jos buvo dengiamos konstruojant medį. Pavyzdžiui, paieškos gilyn metodu į medį briaunos buvo jungiamos tokia tvarka $(1,2)$, $(2,3)$, $(3,4)$, $(4,6)$, $(6,5)$, $(6,7)$, o viršūnės dengiamos taip: 1, 2, 3, 4, 6, 5 ir 7. Medžio briaunoms, jų surašymo tvarka, nuosekliai priskirkime kintamuosius, pradedant kintamuoju x_1. Kintamieji briaunoms, nepriklausančioms medžiui, priskiriami laisvai. Toks kintamųjų priskyrimas pavaizduotas 2.13.3 b) pav. Jame dviguba linija pažymėtos medžio briaunos. Tada, jei lygtis nuosekliai rašysime mazgams (viršūnėms), pradedant antruoju, ta tvarka, kokia jie buvo dengiami konstruojant medį, sistemos tiesinės dalies matrica bus pseudotrikampė. 2.13.3 b) pav.grafui ši mazgų tvarka būtų: 2, 3, 4, 6, 5 ir 7.

Pastaba. Remiantis medžio savybėmis, nesunku suvokti, kad taip sunumeravus kintamuosius ir lygtis, sistemos tiesinės dalies matrica bus pseudotrikampė.

Žemiau pateikta 3.13.3 b) grafo sistemos tiesinės dalies matrica. Matome, kad ji yra pseudotrikampė.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	Mazgo n.r
√	√						√			2
	√	√						√	√	3
		√	√							4
			√	√	√	√				6
				√			√	√		5
					√				√	7

Dabar, parinkus kintamųjų x_7, x_8, x_9 ir x_{10} reikšmes, iš karto gausime tiesinę lygčių sistemą su trikampe matrica, kurią pilnai nusako grafo gretimumo struktūra. Vadinasi, šiuo atveju tiek sistemos saugojimui, tiek ir sprendimui atminties ir laiko sąnaudos bus minimalios.

Dengiančio medžio konstravimo algoritmai

Aptarkime dengiančio medžio konstravimo algoritmus, pagrįstus paieška gilyn ir paieška platyn.

Pirmasis algoritmas (be rekursijos), pagrįstas paieška gilyn.

Duota: *n* – grafo viršūnių skaičius,
m – grafo briaunų skaičius,
L[1.. 2*m*] – grafo briaunų masyvas,
lst[1.. *n* + 1] – grafo briaunų adresų masyvas.

Rasti: Dengiantį medį *T*, kuris nusakytas briaunų matrica *B*[1..2, 1..*n*-1].

Procedūros vidiniai darbo masyvai ir kintamieji sutampa su paieškos gilyn (žr. 2.8.1.2 paragrafą) procedūros vidiniais darbo masyvais ir kintamaisiais.

```
const c = 500;
type
   mas = array [1..c] of integer;
   matr = array [1..2, 1..c] of integer;
procedure medis1 (v, n, m : integer; L, lst : mas; var b : matr);
{ Procedūra medis1 konstruoja dengiantį medį paieškos gilyn iš viršūnės v
metodu, 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 nuorodų masyvai,
```

```
    b – dengiančio medžio briaunų matrica. }
var i, k, u, is : 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 }
        is := 0; { Medžio briaunų skaičius }
    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 { (k, u) – medžio briauna }
                    begin
                        is := is + 1;
                        b [1, is] := k;
                        b [2, is] := u;
                        if is = n – 1 then { Medis sukonstruotas } t := true;
                        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 viršųnei 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;
```

Antrasis algoritmas (su rekursija), pagrįstas paieška gilyn.

Duota: Jungusis (n,m)-grafas $G=(V,U)$, nusakytas gretimumo struktūra: $N(v)$ – aibė viršūnių, gretimų viršūnei v.

Rasti: Dengiantį medį (V,T).

Įveskime masyvą *naujas* [1..*n*];

$$naujas[v]=\begin{cases} true, \text{ jei } v \text{ - nauja,} \\ false, \text{ priešingu atveju,} \end{cases} \quad v \in V .$$

```
procedure Medis2 (v);
    {Paieška gilyn sujungta su medžio briaunų radimu;
    kintamieji naujas ir T – globalieji}
        begin
            naujas [v] := false;
                for u ∈ N (v) do
                    if naujas [u] then {(v, u) – medžio briauna}
                        begin
                            T := T ∪ {(v, u)};
                            Medis2 (u);
                        end;
        end;
begin {pagrindinė programa}
    for v ∈ V do naujas [v] := true;
    T := ∅; {T – medžio briaunų aibė} r := bet kuri grafo viršūnė;
    Medis2 (r);
end;
```

Trečiasis algoritmas, pagrįstas paieška platyn.

Duota: Jungusis (n,m)-grafas $G=(V,U)$, nusakytas gretimumo struktūra: $N(v)$ – aibė viršūnių, gretimų viršūnei *v*.

Rasti: Dengiantį medį (V,T).

Kaip ir antrajame algoritme, įveskime masyvą *naujas* [1..*n*].

Tada medžio ieškojimo procedūra bus užrašoma taip:

```
procedure Medis3 (v);
{Paieška platyn sujungta su medžio briaunų radimu.
Kintamieji naujas ir T – globalieji}
   begin
      Eilė := ∅;
      Eilė ⇐ v;
      naujas [v] := false;
      while Eilė ≠ ∅ do
         begin
            p ⇐ Eilė;
               for u ∈ N (p) do
               if naujas [u] then {(p, u) – medžio briauna}
                  begin
                     T := T ∪ {(p, u)};
                     Eilė ⇐ u;
                  naujas [u] := false;
                  end;
         end;
   end;
```

Pagrindinė programa sutampa su antrojo algoritmo pagrindine programa.

2.13.2. Trumpiausio dengiančio medžio svoriniame grafe apskaičiavimo uždavinys

Apibrėžimas. Jei (n,m)-grafo $G=(V,U)$ kiekvienai briaunai (u,v) yra priskirtas svoris – realusis skaičius, tai grafas *G* vadinamas svoriniu grafu.

Uždavinio formulavimas. Duotas svorinis jungusis (n,m)-grafas $G=(V,U)$. Rasti trumpiausią dengiantį medį, t.y. dengiantį medį, kurio briaunų svorių suma būtų mažiausia tarp visų galimų dengiančių medžių.

Pastaba. Tolesniame dėstyme vietoje žodžių "briaunos svoris" naudosime žodžius "briaunos ilgis".

Nors tai optimizavimo uždavinys, tačiau šio uždavinio tikslūs sprendimo algoritmai yra efektyvūs. Daugiausia naudojami du šio uždavinio sprendimo metodai: Kraskalo (Kruskal J.B.)[4] ir Primo (Prim R.C.)[5].

Kraskalo algoritmas reikalauja $O(m\log m)$ operacijų, o Primo (Deikstros) – $O(n^2)$ operacijų. 1975 m. buvo sukurtas trumpiausio dengiančio medžio radimo algoritmas, reikalaujantis $O(m\log\log n)$ operacijų (žr. Yao A.C.C. An $O(m\log\log n)$ Algorithm for Finding Minimum Spanning Trees, Info. Proc. Let., 4, 1975, p.p. 21 – 23; arba Cheriton D., Tarjan R.E. Finding Minimum Spanning Trees, SIAM J. Comput., 5, 1976, p.p. 724 – 742).

Žemiau aptarsime Kraskalo ir Primo metodus ir algoritmus.

Kraskalo metodas. Kraskalo metodą nusako teorema.

Teorema. Tarkime, kad $G=(V,U)$ jungusis pilnasis grafas ir visų jo briaunų ilgiai (svoriai) skirtingi. Tada egzistuoja vienintelis trumpiausias dengiantis medis, kuris konstruojamas taip: "iš likusių grafo *G* briaunų randame trumpiausią briauną ir ją įtraukiame į medį, jei ši briauna neiššaukia ciklo su anksčiau paimtom medžio briaunom".

Pastaba. Likusios grafo briaunos – tai grafo *G* briaunos, atmetus medžio briaunas ir briaunas, kurias bandėme įtraukti į medį.

Įrodymas. Tarkime, kad $H=(V,U_H)$ yra dengiantis medis, sukonstruotas teoremoje nurodytu metodu. Aišku, kad medžio *H* formavimas bus baigtas, kai bet kuri iš likusių grafo *G* briaunų iššauks ciklą su anksčiau paimtom medžio briaunom, t.y. su briaunom, priklausančiom aibei U_H. (žr. 4-ąją medžio savybę). Tarkime, kad šis medis nėra trumpiausias. Tada egzistuoja trumpiausias dengianti medis $T=(V,U_T)$ ir $U_T\neq U_H$. Kadangi $U_T\neq U_H$, tai tarkime, kad u_k – pirmoji iš aibės U_H briaunų, nepriklausanti aibei U_T, t.y. abiejų aibių U_H ir U_T pirmieji $(k-1)$ elementai sutampa, o $u_k\in U_H$, bet $u_k\notin U_T$.

Panagrinėkime briaunų aibę $U_T\cup\{u_k\}$. Aišku (žr. 4-ąją medžio savybę), ši aibė turės vienintelį ciklą, kuriame bus bent viena briauna $u_0\notin U_H$; priešingu atveju medyje (V, U_H) yra ciklas. Aptarkime medį, kurį apibrėžia briaunų aibė $W=U_T\cup\{u_k\}\setminus\{u_0\}$. Grafas (V,W) tikrai yra medis, nes jis turi

[4] Kruskal J.B. On the shortest spanning subtree of a graph and the travelling salesman problem. Proc. Amer. Math. Soc., 1956, 7, p.p. 48 – 50.

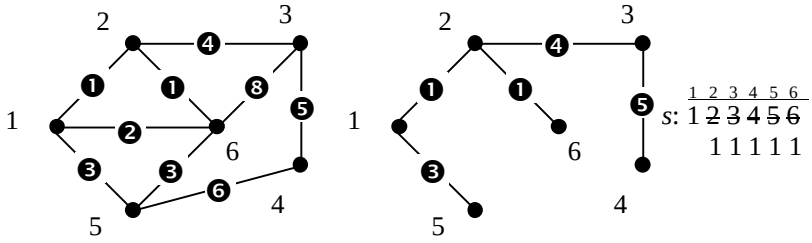
[5] Prim R.C. Shortest Connection Networks and Some Generalizations, Bell Syst. Tech. J., 36, 1957, p.p. 1389 – 1401. Dijkstra E. Two problems in Connexon with Graphs, Num. Math. 1, 1959, 259 – 271.

$(n-1)$ briauną ir neturi ciklų. Aišku, kad $l(u_0)$ (u_0-osios briaunos ilgis) yra didesnis nei $l(u_k)$ (u_k-osios briaunos ilgis). Iš tikro, kadangi briaunos $u_1, u_2, ..., u_{k-1}$ ir u_0 priklauso aibei U_T, tai jos nesudaro ciklo. Formuojant medį $H = (V, U_H)$ teoremoje nurodytu metodu, *k*-ajame žingsnyje iš visų likusių briaunų išrinkome trumpiausią briauną, kuri nesudarė ciklo su anksčiau paimtom briaunom $u_1, u_2, ..., u_{k-1}$. Kadangi briauna u_0 neiššaukia ciklo su briaunomis $u_1, u_2, ..., u_{k-1}$ ir šios briaunos neišrinkome, tai reiškia, kad $l(u_0) > l(u_k)$. Vadinasi, medžio (V, W) briaunų ilgių suma yra mažesnė (ilgesnę briauną u_0 pakeitėme trumpesne briauna u_k) nei, pagal prielaidą, paties trumpiausio dengiančio medžio $T = (V, U_T)$ briaunų ilgių suma. Vadinasi, prielaida yra klaidinga ir medis $H = (V, U_H)$ yra trumpiausias dengiantis medis.

Pastaba. Ši teorema galioja ir tuo atveju, jei (n, m) - grafas $G = (V, U)$ yra jungusis, bet nepilnasis ir kai kurių jo briaunų ilgiai yra lygūs. Šiuo atveju galime įsivaizduoti, kad grafas yra pilnasis, priimant, kad nesančių briaunų ilgiai yra begalybės.

Jei kai kurių briaunų ilgiai yra lygūs, tai bendru atveju gali egzistuoti keli ekvivalentūs sprendiniai. Tada to paties ilgio briaunas sunumeruosime, ir jas bandysime jungti į medį jų numerių didėjimo tvarka. Pavyzdžiui, tarkime, kad $l(u_1) = l(u_2) = l(u_3) = l$, tada, jei konstruojant medį, trumpiausios briaunos ilgis bus lygus l, tai pirmiausia bandysim į medį įtraukti briauną u_1, po to – briauną u_2, o po to – briauną u_3.

Pavyzdys. Kraskalo metodu sukonstruokime trumpiausią dengiantį medį 2.13.4 pav. a) pavaizduotam grafui.



2.13.4 pav. Kraskalo metodas

2.13.4 b) pavaizduotas trumpiausias dengiantis medis, apskaičiuotas Kraskalo metodu, kai briaunas į medį bandėme traukti tokia tvarka: $(1,2)$,

$(2,6)$, $\underline{(1,6)}$, $(1,5)$, $\underline{(5,6)}$, $(2,3)$ ir $(3,4)$. Pabrauktos briaunos į medį nebuvo įtrauktos, nes iššaukdavo ciklą su anksčiau paimtom medžio briaunom.

Aptarkime, kaip formaliai nustatyti, "***ar trumpiausia briauna iš likusių sudaro ciklą su anksčiau paimtom medžio briaunom?***".

Tam tikslui įveskime masyvą $s[1..n]$, čia s_i yra jungiosios komponentės, kuriai priklauso viršūnė i, numeris. Aišku, kad pradžioje, kai trumpiausią medį sudaro tuščiasis grafas, kiekviena viršūnė priklauso skirtingai jungiajai komponentei, t.y., pradžioje $s_i = i$, $i = \overline{1,n}$. Tarkime, kad konstruojant medį, trumpiausia briauna iš likusių yra briauna (u,v), tada:

if s [*u*] ≠ *s* [*v*] *then* {*briauna* (*u, v*) *neiššaukia ciklo su anksčiau paimtom medžio briaunom*}
begin
briauną (*u, v*) *įtraukti į medį.*
{*perskaičiuoti jungiųjų komponenčių masyvo s elementus*}
l := *s* [*u*]; *k* := *s* [*v*];
for i := 1 *to n do*
if s [*i*] = *k then s* [*i*] := *l*;
end;

2. 13. 4 c) pav. parodyta, kaip skaičiuojant kito jungiųjų komponenčių masyvo *s* turinys:

Kraskalo metodo algoritmas

Duota: *n* – grafo viršūnių skaičius,
m – grafo briaunų skaičius,
b [1..2, 1..*m*] – jungiojo grafo briaunų matrica,
c [1..*m*] –briaunų ilgių masyvas:
c [*j*] yra briaunos (*b* [1, *j*], *b* [2, *j*]) ilgis.
Rasti: Trumpiausią dengiantį medį, t.y. medžio briaunų matricą *t* [1..2, 1..*n*-1] ir jų ilgių masyvą *d* [1..*n*-1].
Vidiniai darbo masyvai:
s [1..*n*] – viršūnių jungiųjų komponenčių masyvas,
p [1..*m*] – požymių masyvas;

$$p[i] = \begin{cases} 0, & \text{jei } i \text{ - oji briauna nenagrinėta,} \\ 1, & \text{priešingu atveju.} \end{cases}$$

Žodžiai "briauna nagrinėta" reiškia, kad ji yra arba medžio briauna, arba ją bandėme įtraukti į medį.

```
const cc = 500;
type
    mas = array [1..cc] of integer;
    mas1 = array [1..cc] of real;
    matr = array [1..2, 1..cc] of integer;
procedure kraskalas (n, m : integer; b : matr; c : mas1; var t : matr;
                     var d : mas1);
{ Procedūra kraskalas apskaičiuoja jungiojo svorinio grafo, nusakyto briaunų
matrica, trumpiausią dengiantį medį.
Formalūs parametrai:
    n – grafo viršūnių skaičius,
    m – grafo briaunų (lankų) skaičius,
    b – grafo briaunų matrica,
    c – briaunų ilgių masyvas,
    t [1..n-1] – trumpiausio dengiančio medžio briaunų matrica,
    d[1..n-1] – trumpiausio dengiančio medžio briaunų ilgių masyvas.}
var i, j, k, l, u, v, x, y : integer;
    sum, min : real;
    s, p : mas;
begin
for i := 1 to n do s [i] := i;
sum := 1;
for i := 1 to m do
    begin
        p [i] := 0;
        sum :=sum + c [i];
    end;
k := 0; { Į medį įtrauktų briaunų skaičius }
while k < n – 1 do
    begin
        { Rasti trumpiausią briauną iš likusių }
        min := sum;
        for i := 1 to m do
        if (p [i] = 0) and (min > c [i]) then
            begin
                min := c [i];
                l := i;
            end;
        if min = sum then begin
            writeln ('Grafas – nejungusis');
            exit;
```

```
        end;
        { Ar galima l-ąją briauną įtraukti į medį? }
        p [l] := 1; { l-oji briauna – nagrinėta }
        u := b [1, l];
        v := b [2, l];
        if s [u] <> s [v] then
            begin
                { Briauną (u, v) įtraukiame į medį }
                k := k + 1;
                t [1, k] := u; t [2 ,k] := v;
                d [k] := c [l];
                { Perskaičiuojame jungiąsias komponentes }
                x := s [u]; y := s [v];
                for i := 1 to n do
                if s [i] = y then s [i] := x;
            end;
    end;
end;
```

Primo metodas. Jei Kraskalas trumpiausią dengiantį medį apskaičiuoja taikydamas "godų" algoritmą, tai Primas šį uždavinį sprendžia "artimiausio kaimyno" metodu.

Primo algoritmas

Nulinis žingsnis

a) (n,m) - *jungiojo grafo* $G=(V,U)$ *visos viršūnės nenudažytos.*
b) *Trumpiausias dengiantis medis T yra tuščiasis grafas, turintis n viršūnių.*
c) *Pasirenkame bet kurią grafo G viršūnę ir ją nudažome.*

Bendrasis žingsnis

while "grafas G turi nenudažytų viršūnių" do
begin

a) *Randame trumpiausią briauną tarp nudažytų ir nenudažytų grafo G viršūnių. Tarkime, kad tokia briauna yra briauna* (u,v)*, čia u – nudažyta, o v – nenudažyta viršūnė.*
b) *Briauną* (u,v) *įjungiame į medį T.*
c) *Nudažome grafo G viršūnę v.*

end;

Kaip rasti ***trumpiausią briauną tarp nudažytų ir nenudažytų viršūnių***? Tam tikslui įveskime tris pagalbinius masyvus.

$d[1..n]$, $d[i] = \begin{cases} 0, & \text{jei grafo } G \ i\text{-oji viršūnė nenudažyta,} \\ 1, & \text{priešingu atveju.} \end{cases}$

$t[1..n]$ – trumpiausių briaunų tarp nudažytų ir nenudažytų viršūnių masyvas.

Tarkime, kad kažkuriame bendrojo žingsnio etape trumpiausia briauna, jungianti nenudažytą viršūnę k su nudažytomis viršūnėmis yra briauna (k, s), čia s – nudažyta viršūnė, gretima viršūnei k. Tada $t[k] = l(k, s)$, t.y. lygi briaunos (k, s) ilgiui.

Aišku, kad pradžioje (nuliniame žingsnyje) visi $t[k] =$ "begalybei". ("Begalybė" šiam uždaviniui imamas skaičius $sum = 1 + \sum_{u \in U} l(u)$, čia $l(u)$ – grafo u-osios briaunos ilgis).

$nr[1..n]$ – jei $t[k] = l(k, s)$, tai $nr[k] = s$. Kitaip tariant, $nr[k]$ žymi numerį dažytos viršūnės, kuri riboja trumpiausią briauną tarp nedažytos viršūnės k ir dažytų viršūnių.

Aišku, kad pradžioje visi $nr[k] = 0$, $k = \overline{1, n}$.

Tarkime, kad trumpiausios briaunos tarp nudažytų ir nenudažytų viršūnių ilgis kažkuriame bendrojo žingsnio etape yra $t_k = \min_{1 \le i \le n} (t_i \ / \ i\text{-}$ nenudažyta viršūnė$)$, o trumpiausia briauna – $(k, nr[k])$.

Įtraukus briauną $(k, nr[k])$ į medį, viršūnė k nudažoma, o masyvai d, t, ir nr perskaičiuojami taip:

```
d [k] := 1;
for u ∈ N (k) do
    if (d [u] = 0) {viršūnė u, gretima viršūnei k, yra nenudažyta} and
                   (t [u] > l (k, u) {briaunos (k,u) ilgis mažesnis už ilgį
                   trumpiausios briaunos, jungiančios nenudažytą viršūnę u su
                   kitomis nudažytomis viršūnėmis} then
                        begin
                            t [u] := l (k, u);
                            nr [u] :=k;
                        end;
```

Formaliai Primo algoritmą galima užrašyti taip.

Tarkime, kad (n,m)-grafas G yra jungusis ir nusakytas masyvais $L[1..2m]$ ir $lst[1..n+1]$. Grafo G briaunų ilgiai apibrėžti masyvu $C[1..2m]$, t.y. jei viršūnei k gretima viršūnė u yra patalpinta masyve L adresu l, tai briaunos (k,u) ilgis patalpintas masyve C tuo pačiu adresu l.

Apskaičiuoto trumpiausio medžio briaunos bus talpinamos matricoje $B[1..2,1..n-1]$, o jų ilgiai – masyve $R[1..n-1]$, t.y. briaunos $(b[1,l],b[2,l])$ ilgis bus lygus $R[l]$.

```
const cc = 100;
type
    mas = array [1..cc] of integer;
    mas1 = array [1..cc] of real;
    matr = array [1..2, 1..cc] of integer;
var i, j, n, m : integer;
    c, r : mas1;
    b, bb : matr;
procedure BLClst (n, m : integer; b : matr; c : mas1 var L, lst : mas,
                  var cl : mas1);
{ Procedūra BLClst perveda svorinio neorientuotojo grafo briaunų  matricą į
tiesioginių nuorodų masyvus L ir lst ir perskaičiuoja briaunų ilgio masyvą.
Formalūs parametrai:
    n – grafo viršūnių skaičius,
    m – grafo briaunų (lankų) skaičius,
    b – grafo briaunų matrica;
        (b [1, j], b [2, j]) – j-toji grafo briauna;
    c – grafo briaunų ilgių masyvas;
    c[j] – j-osios briaunos ilgis.
    L, lst – grafą nusakantys tiesioginių nuorodų masyvai;
    cl – grafo briaunų ilgiai, kai grafas užrašomas L ir lst masyvais.
Vidiniai kintamieji:
    d[1..n] – viršūnių laipsnių masyvas;
        d[i] – i-osios viršūnės laipsnis.
    fst[1..n] – adresų masyvas;
        pradžioje fst [i] = lst [i] + 1, i = 1, 2, ..., n. fst[i] – tai adresas masyve L,
        kur turi būti talpinamas numeris pirmos viršūnės, gretimos viršūnei i. }
var i, j, k : integer;
    fst, d : mas;
begin
{ Viršūnių laipsnių apskaičiavimas. }
for i := 1 to n do d [i] := 0;
```

```
for i := 1 to 2 do     {*}
    for j := 1 to m do
        begin
            k := b [i, j];
            d [k] := d [k] + 1;
        end;
{ Masyvo lst formavimas }
lst [1] := 0;
for i: = 1 to n do lst [i + 1] := lst [i] + d [i];
{ Masyvo L formavimas }
for i: = 1 to n do fst [i]:=lst [i] + 1;
for j := 1 to m do
    begin
        k := b[1 ,j];
        L [fst [k]] := b [2, j];
        cl [fst [k]] := c [j];
        fst [k] := fst [k] + 1;
        k := b [2, j];            {**}
        L [fst [k]] := b [1, j];     {**}
        cl [fst [k]] := c [j];
        fst [k] := fst [k] + 1;      {**}
    end;
end;
procedure primo (n, m : integer; bb : matr; c : mas1; var b : matr; var r :
mas1);
{ Procedūra primo apskaičiuoja jungiojo svorinio grafo,  nusakyto briaunų
matrica, trumpiausią dengiantį medį.
Formalūs parametrai:
    n – grafo virsšūnių skaicius,
    m – grafo briaunų (lankų) skaičius,
    bb – grafo briaunų matrica,
    c – briaunų ilgių masyvas,
    b [1..n-1] – trumpiausio dengiančio medžio briaunų matrica,
    r [1..n-1] – trumpiausio dengiančio medžio briaunų ilgių masyvas.}
var i, k, j, u, v, next : integer;
    sum, min : real;
    L, lst, d, nr : mas;
    cl, t : mas1;
begin
BLClst (n, m, bb, c, L, lst, cl);
sum := 1;
```

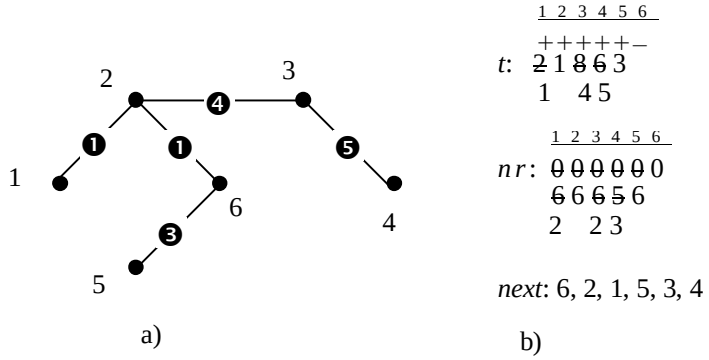
```
for i := 1 to m do sum := sum + c [i];
for i := 1 to n do
    begin
        d i] = 0;
        t [i] := sum;
        nr [i] := 0;
    end;
next := n;
d [next] := 1;
k := 1; { Nudažytų viršūnių skaičius }
j : =0; { Į medį įtrauktų briaunų skaičius }
while k <= n – 1 do
    begin
        { Perskaičiuojame masyvus t ir nr }
        for i := lst [next] + 1 to lst [next + 1] do
            begin
                u := L [i];
                if (d [u] = 0) and (t[u] > cl[i]) then
                    begin
                        t [u] := cl [i];
                        nr [u] := next;
                    end;
            end;
    { Randame trumpiausią briauną tarp nudažytų ir nenudažytų grafo G
    viršūnių }
        min := sum;
        for i := 1 to n do
        if (d [i] = 0) and (min > t [i]) then
            begin
                min := t [i];
                v := i;
            end;
        if min = sum then begin
            writeln('Grafas – nejungusis');
            exit;
        end;
        { Briauną (v, nr [v]) įtraukiame į medžio briaunų aibę }
        j := j + 1;
        b [1, j] := v; b [2, j] := nr [v];
        r [j] := min;
        next := v;
```

```
        d [v] := 1;
        k := k + 1;
    end;
end;
begin
    writeln;
    writeln ('įveskite viršūnių skaičių');
    read (n);
    writeln ('įveskite briaunų skaičių');
    read (m);
    writeln ('įveskite briaunų matricą');
    for i :=1 to 2 do
        begin
            for j := 1 to m do read (bb [i, j]);
            readln;
        end;
    writeln ('įveskite briaunų ilgius');
    for i := 1 to m do read (c [i]);
    primo (n, m, bb, c, b, r);
    writeln;
    writeln ('Matr. b');
    for i := 1 to 2 do
        begin
            for j := 1 to n – 1 do write (b [i, j] : 2);
            writeln;
        end;
    writeln;
    writeln ('Br. ilgiai');
    for i :=1 to n – 1 do write (r [i] : 4 : 1);
end.
```

2.13.5 a) pav. pavaizduotas 2.13.4 a) pav. grafo trumpiausias dengiantis medis, kai pradžioje $next = 6$. Briaunų įtraukimo į medį tvarka buvo $(6,2)$, $(2,1)$, $(6,5)$, $(2,3)$ ir $(3,4)$.

2.13.5 b) pav. parodyta, kaip skaičiuojant kito masyvų *t*, *nr* ir kintamojo *next* turiniai.

Pastaba. Yra kitas, mažiau racionalus, nei aukščiau pateiktas trumpiausios briaunos tarp nudažytų ir nenudažytų viršūnių radimo būdas. Grafo briaunų aibę išrikiuokime jų ilgių didėjimo tvarka. Tada trumpiausia briauna tarp nudažytų ir nenudažytų viršūnių bus pirmoji sekos briauna, kuriai tik viena viršūnė nudažyta.



2.13.5 pav. Primo metodas

Keli uždavinys. Kaip buvo minėta anksčiau, Keli medžius naudojo norėdamas apskaičiuoti galimą cheminių junginių izomerų skaičių.

Keli uždavinys formuluojamas taip: ***plokštumoje duota n taškų. Keliais skirtingais būdais juos galima sujungti, kad gautasis grafas būtų medis?***

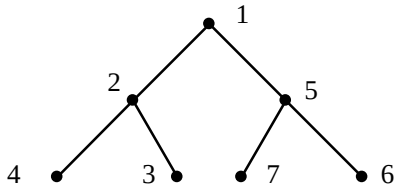
Tam, kad šį uždavinį galėtume išspręsti, pirmiausia aptarsime kompaktišką medžio užrašą – medžio kodą.

Tarkime, $G=(V,U)$ yra medis. Šio medžio briaunas užrašykime laikydamiesi taisyklių.

1. Rasti kabančią briauną, kurios kabančios viršūnės numeris yra pats mažiausias.
2. Užrašyti šią briauną. Rašant briauną, pirmiausia rašyti kabančios viršūnės numerį.
3. Ištrinti kabančią briauną drauge su kabančia viršūne.

Aišku, kad užrašydami medžio briaunas, taip elgsimės, kol medis turės bent vieną briauną.

Pavyzdžiui, 2.13.6 pav. medžio briaunų seka, sudaryta laikantis išvardintų taisyklių, bus: $(3,2)$, $(4,2)$, $(2,1)$, $(1,5)$, $(6,5)$ ir $(5,7)$.



2.13.6 pav. Medžio kodas

Jei, laikantis šių taisyklių, medžio $G=(V,U)$ briaunas surašysime iš eilės, tai gausime tokią briaunų seką: (a_1,b_1), (a_2,b_2), ..., (a_{n-2},b_{n-2}), (a_{n-1},b_{n-1}).

Remdamiesi šia seka, sudarykime aibę $B=\{b_1,b_2,...,b_{n-2}\}$. Ši aibė vienareikšmiškai nusako medį, t.y. kiekvienam medžiui atitiks vienintelė aibė B ir atvirkščiai, kiekviena aibė vieninteliu būdu nusakys medį. Šią aibę pavadinsime ***medžio kodu***.

Nagrinėtam pavyzdžiui medžio kodas bus aibė $B=\{2,2,1,5,5\}$.

Kaip, turint aibę B, rasti medžio briaunas?

Tam tikslui elgsimės taip.

1. Sudarome aibę $A=\{1,2,3,...,n\}$.
2. Aibėje A ieškome pirmo elemento, nepriklausančio aibei B. Tarkime, tas elementas yra a_1. Tada (a_1,b_1) yra medžio briauna, čia b_1 yra pirmasis aibės B elementas.
3. a_1 šaliname iš aibės A, o b_1 šaliname iš aibės B, t.y. $A:=A\setminus\{a_1\}$, $B:=B\setminus\{b_1\}$.

Aišku, kad taip elgsimės tol, kol aibė B turės bent vieną elementą. Kai aibė B taps tuščia, tai aibėje A liks du elementai, kurie ir apibrėš paskutinę medžio briauną. Rašydami šią briauną pirmiausia rašysime mažesnį elementą.

Laikantis šių taisyklių, gausime tą pačią medžio briaunų seką, kuria remdamiesi ir sudarėme medžio kodą.

Nagrinėtam pavyzdžiui turėsime: $A=\{1,2,3,4,5,6,7\}$ ir $B=\{2,2,1,5,5\}$.

Pirmasis aibės A elementas, nepriklausantis aibei B, yra 3. Vadinasi, $(3,2)$ yra medžio briauna. Iš aibės A pašalinę elementą 3, o iš aibės B – elementą 2, gausime: $A=\{1,2,4,5,6,7\}$ ir $B=\{2,1,5,5\}$. Dabar pirmasis aibės A elementas, nepriklausantis aibei B, yra 4. Vadinasi, $(4,2)$ yra medžio briauna. Po atitinkamų elementų pašalinimo iš aibių A ir B, gausime: $A=\{1,2,5,6,7\}$, o $B=\{1,5,5\}$. Šios aibės nusakys briauną $(2,1)$, ir po perskaičiavimo aibės bus: $A=\{1,5,6,7\}$ ir $B=\{5,5\}$. Iš jų gausime $(1,5)$, ir po perskaičiavimo $A=\{5,6,7\}$, o $B=\{5\}$. Iš šių aibių gausime briauną $(6,5)$, o po aibių A ir B perskaičiavimo gausime: $A=\{5,7\}$, o $B=\varnothing$. Vadinasi, paskutinioji briauna yra $(5,7)$. Kaip matome, gavome tą pačią medžio briaunų seką.

Iš šio nagrinėjimo darome išvadą, kad medžio kodas yra $(n-2)$-jų elementų iš aibės $A=\{1,2,3,...,n\}$ rinkinys su pasikartojančiais elementais. Rinkinys nuo rinkinio skiriasi arba pačiais elementais, arba jų tvarka, t.y.

turime gretinius iš *n* elementų po $n-2$ su pasikartojimais: $\overline{A_n^{n-2}}$. Šių junginių skaičius lygus n^{n-2}. Vadinasi, skirtingų Keli medžių yra n^{n-2}.

Pavyzdžiui, kai $n=4$, skirtingų Keli medžių bus $4^2=16$, kuriuos nusakys kodai:

1,1 ; 1,2 ; 1,3 ; 1,4 ; 2,1 ; 2,2 ; 2,3 ; 2,4 ;

3,1 ; 3,2 ; 3,3 ; 3,4 ; 4,1 ; 4,2 ; 4,3 ; 4,4 .

Pavyzdžiui, kodas – aibė $B=\{2,3\}$ duos medį: $(1,2)$; $(2,3)$; $(3,4)$, o kodas – aibė $B=\{3,2\}$ duos medį: $(1,3)$; $(3,2)$; $(2,4)$.

Šteinerio uždavinys grafe. Duotas jungusis svorinis grafas $G=(V,U)$ ir viršūnių aibės *V* poaibis *A*. Rasti jungųjį pografį *T*, tenkinantį sąlygas:

1) poaibis *A* yra pografio *T* viršūnių poaibis,
2) pografio *T* briaunų suma turi būti mažiausia tarp visų pografių, tenkinančių 1) sąlygą.

Aišku, kad Šteinerio uždavinio sprendinys yra medis. Jis vadinamas Šteinerio medžiu. Aišku taip pat, kad Šteinerio medžio radimo uždavinys ekvivalentus uždaviniui: tarp visų grafo *G* indukuotų pografių, kuriems aibė *A* yra jų viršūnių poaibis, rasti minimalų medį

Nėra žinomi efektyvūs šio uždavinio sprendimo metodai. Aišku, kad tikslų sprendinį duos pilnas grafo *G* indukuotų pografių, kuriems aibė *A* yra jų viršūnių poaibis, perrinkimas. Tačiau šis metodas reikalautų perrinkti 2^{n-k} variantų, čia $|V|=n$, o $|A|=k$. Todėl sprendžiant šį uždavinį naudojamos įvairios euristikos. Aptarkime vieną iš euristikų, kuri iš esmės yra artimiausio kaimyno metodas.

Tarkime, duotas svorinis grafas $G=(V,U)$ ir jo viršūnių poaibis *A* (poaibio *A* viršūnės vadinamos Šteinerio viršūnėmis). Tarkime, kad į Šteinerio tinklą jau įtrauktos Šteinerio viršūnės $a_1,a_2,...,a_t$ ir viršūnės $v_1,v_2,...,v_l$, priklausančios aibei $V\setminus A$. Tada į tinklą įtrauksime tą Šteinerio viršūnę a_k, kuriai $d(a_k,u)=\min(d(a_k,v)/v\in\{a_1,a_2,...,a_t\}$ arba $v\in\{v_1,v_2,...,v_l\})$, t.y. tą viršūnę, kuri nuo tinklo viršūnių yra arčiausiai.

begin

Visos grafo G viršūnės – nenudažytos. Išrenkame Šteinerio viršūnę, nuo kurios atstumų iki likusių Šteinerio viršūnių suma yra mažiausia, t.y. viršūnę a, kuriai $\sum_{v\in A} d(a,v)\to\min$ *. Šteinerio viršūnę a nudažome.*

while "yra nenudažytų Šteinerio viršūnių" do

begin

Tarp nenudažytų Šteinerio viršūnių randame tokią viršūnę, nuo kurios atstumas iki nudažytų grafo G viršūnių yra mažiausias. Tarkime, kad tai Šteinerio viršūnė "next", o jai artimiausia nudažyta grafo viršūnė yra v. (*Aišku, kad nudažytos grafo viršūnės priklauso Šteinerio tinklui*).

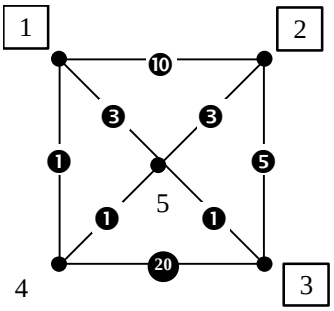
Tada į Šteinerio tinklą įtraukiame trumpiausią grandinę, jungiančią viršūnę "next" su viršūne v.

Visas nenudažytas šios grandinės viršūnes nudažome.

end;

end;

Pavyzdys. Panagrinėkime 2.13.7 pav. pavaizduotą grafą. Šio grafo Šteinerio viršūnės yra $\{1,2,3\}$. 2.13.7 pav. jos pažymėtos kvadratukais.



2.13.7 pav. Šteinerio uždavinys

Trumpiausių kelių tarp Šteinerio viršūnių matrica $C = \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & 5 & 3 \\ 2 & 5 & 0 & 4 \\ 3 & 3 & 4 & 0 \end{array}$.

Kadangi 3-iajai viršūnei atstumų iki likusių Šteinerio viršūnių suma yra mažiausia, tai išrenkame 3-iąją viršūnę ir ją nudažome. Tada trumpiausios grandinės, jungiančios nenudažytas Šteinerio viršūnes su nudažytomis grafo *G* viršūnėmis, ilgis yra 3, o grandinė yra $1,4,5,3$. Šią grandinę įjungiame į Šteinerio tinklą, o 1-ąją, 4-ąją ir 5-ąją viršūnes nudažome. Likusioji nenudažyta 2-oji Šteinerio viršūnė yra arčiausiai nudažytos 5-osios viršūnės. Todėl į Šteinerio tinklą įjungiame trumpiausią grandinę – briauną $(2,5)$ ir nudažome Šteinerio viršūnę – 2-ąją viršūnę. Kadangi visos Šteinerio viršūnės nudažytos, tai Šteinerio tinklas sudarytas, ir jį sudarys briaunos $(1,4)$; $(4,5)$; $(5,3)$ ir $(2,5)$.