2.14. Optimalių kelių ieškojimas

Aptarsime kelis optimalių kelių grafe ieškojimo uždavinius:

1) trumpiausio kelio radimo uždavinį,
2) plačiausios siauros vietos (didžiausios keliamosios galios) radimo uždavinį,
3) uždavinį apie stiprinimą,
4) ilgiausio kelio radimo uždavinį,
5) trumpiausių kelių tarp visų viršūnių porų apskaičiavimo uždavinį.

2.14.1. Trumpiausio kelio radimo uždavinys

Uždavinio formulavimas. Duotas svorinis grafas $G=(V,U,C)$, čia $C[1..m]$ – grafo briaunų svorių masyvas.

Rasti:

1) trumpiausius kelius nuo viršūnės *s* iki visų likusių grafo viršūnių,
2) trumpiausią kelią nuo viršūnės *s* iki viršūnės *t*.

Kadangi 2)-asis uždavinys yra 1)-ojo uždavinio atskirasis atvejas, tai čia nagrinėsime 1)-ąjį uždavinį. 2)-ojo uždavinio sprendimo algoritmas pilnai sutaps su 1)-ojo uždavinio sprendimo algoritmu, išskyrus algoritmo pabaigos sąlygą.

Pastaba. Trumpiausių kelių ieškojimo besvoriniame grafe uždavinys aptartas 2.9 paragrafe.

Yra keli 1)-ojo uždavinio sprendimo metodai, iš kurių efektyviausias yra Deikstros algoritmas (žr. Dijkstra E.W. A note on two problems in conection with graphs, Numer. Math., 1959,1,p.p. 259-271), kurį čia ir panagrinėsime.

Paprastai mus domina ne tiktai trumpiausio kelio ilgis, bet ir per kurias viršūnes šis kelias eina. Tam tikslui įvesime du masyvus:

$d[1..n]$ – kelių ilgių masyvas, čia $d[k]$ – trumpiausio kelio nuo viršūnės s iki viršūnės k ilgis, ir

$prec[1..n]$ – masyvas, parodantis per kurias viršūnes keliai eina;

$$prec[u]=\begin{cases} k, & \text{jei trumpiausias kelias į viršūnę } u \text{ ateina iš viršūnės } k, \\ u, & \text{jei kelias prasideda viršūnėje } u. \end{cases}$$

Deikstros algoritmas

begin

Paruošiamasis žingsnis

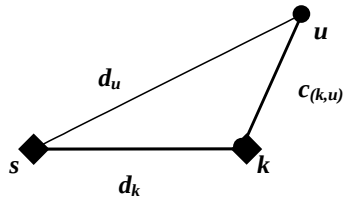
Visos svorinio grafo $G=(V,U,C)$ viršūnės – nenudažytos.

$$sum=\sum_{i=1}^{m} c_i + 1; \quad \{begalybė\}$$

```
d[i] := sum;   prec[i] := 0;   i = 1, n .
d[s] := 0;    {d(s,s)=0}
prec[s] := s; {visi keliai prasideda viršūnėje s}
```

Pagrindiniai skaičiavimai

```
while "yra nenudažytų viršūnių" do
  begin
    {Išrinkti nenudažytą viršūnę k, iki kurios trumpiausio kelio nuo
    viršūnės s ilgis yra nusistovėjęs. Tam tikslui, tarp masyvo d elementų,
    atitinkančių nenudažytas viršūnes, reikia rasti mažiausią elementą, t.y.
    d[k]=min(d[v] / v nenudažyta grafo viršūnė)}
    min:=sum;
    for i:=1 to n do
      if "viršūnė i nenudažyta" and (min>d[i])
        then
          begin
            min:= d[i]; k:=i ;
          end;
      if  min=sum
        then
          begin
            "grafas G – nejungusis";   stop;
          end
        else
          begin
            Nudažome viršūnę k;
            {Perskaičiuojame masyvų d ir prec elementus; jų
            perskaičiavimas pagrįstas 2.14.1 pav.}
            for u∈ N(k) do
              if "u nenudažyta viršūnė" and d[u]> d[k]+c(k,u)
              {c(k,u) – tai (k,u) briaunos ilgis}
                then {iki viršūnės u radome naują kelią,
                        trumpesnį už iki šiol žinomą iki šios
                        viršūnės trumpiausią kelio ilgį d[u]}
                  begin
                    d[u]:= d[k]+ c(k,u);
                    prec[u]:= k
                  end;
          end; {else}
  end; {while}
end;
```



2.14.1 pav. Deikstros algoritmas

Kodėl formulė $d[k] = \min_{v \in V} (d[v] / v - nenudažytos\ viršūnės)$ apskaičiuoja viršūnę, iki kurios trumpiausias kelias nuo viršūnės *s* – nusistovėjęs? Tarkime priešingai, kad iki viršūnės *k* yra dar trumpesnis kelias. Tuo atveju turi egzistuoti tokia nenudažyta viršūnė *l*, kad $d[l]+c(l,k) < d[k]$. Tačiau to negali būti, nes $d[k] = \min_{v \in V} (d[v] / v - nenudažytos\ viršūnės)$.

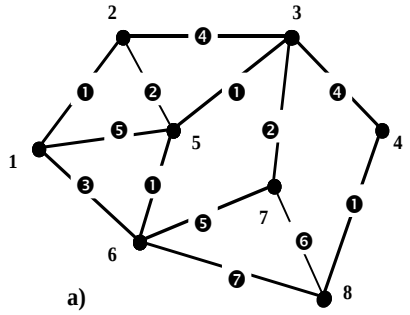
***Pastaba* 1.** 2)-ajam uždaviniui ciklo *while* antraštę reikėtų perrašyti taip: *while "viršūnė t nenudažyta" do*.

***Pastaba* 2.** Deikstros algoritmas teisingai skaičiuos net ir tuo atveju, jei dalis grafo briaunų svorių bus neigiami skaičiai, esant sąlygai, kad grafas neturėtų ciklo, kurį sudarytų neigiamo svorio briaunos. Šiuo atveju reiktų modifikuoti tą pateikto Deikstros algoritmo vietą, kur kalbama apie masyvų *d* ir *prec* elementų perskaičiavimą. Nudažius viršūnę *k*, turėtume nagrinėti visas viršūnei *k* gretimas viršūnes, tiek nenudažytas, tiek ir nudažytas. Jei rastume trumpesnį kelią iki nudažytos viršūnės, tai viršūnės spalvą nuvalytume, t.y. viršūnė taptų nenudažyta. Formaliai šį nagrinėjimą užrašytume taip:

```
for u∈ N(k) do
    if d[u] > d[k]+ c(k,u)
        then
            begin
                d[u]:= d[k]+c(k,u);
                prec[u]:= k;
                if "viršūnė u buvo nudažyta"
                    then viršūnės u spalvą nutriname;
            end;
```

Algoritmas bus vykdomas, jei bus bent viena nenudažyta grafo viršūnė, t.y. algoritmo pabaigos požymis yra: "visos grafo viršūnės yra nudažytos".

Pavyzdys. Panagrinėkime 2.14.2 pav. pavaizduoto svorinio grafo trumpiausių kelių nuo 1-osios viršūnės iki visų likusių apskaičiavimą Deikstros metodu. 2.14.2 pav. pavaizduotas masyvų *d* ir *prec* turinių kitimas. Šiam grafui parametro *sum* (begalybė) reikšmė yra 43.



	1	2	3	4	5	6	7	8
***d*:**	~~43~~	~~43~~	~~43~~	~~43~~	~~43~~	~~43~~	~~43~~	~~43~~
	0	1	~~5~~	8	~~5~~	3	~~8~~	~~10~~
			4		3		6	9

	1	2	3	4	5	6	7	8
***prec*:**	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~
	1	1	~~2~~	3	~~1~~	1	~~6~~	~~6~~
b)			5		2		3	4

2.14.2 pav. Trumpiausi keliai

Pavyzdžiui, trumpiausias kelias nuo 1-osios viršūnės iki 8-osios viršūnės yra 9, t.y. $d(1,8)=9$. Kelias eina per viršūnes: 8,4,3,5,2,1. Nesunku suvokti, kad masyvas *prec* nusako visų trumpiausių kelių, nuo viršūnės 1 iki visų likusių viršūnių, medį.

2.14.2. Didžiausios keliamosios galios (plačiausios siauros vietos) apskaičiavimo uždavinys

Uždavinio formulavimas. Duotas svorinis grafas $G=(V,U,C)$, čia, kaip ir anksčiau, $C[1..m]$ briaunų svorių masyvas. Tarkime, kad briauna vaizduoja kelią, jungiantį viršūnėms atitinkančias gyvenvietes. Tarkime, tame kelyje yra tiltas per upę, ir briaunos svoris reiškia to tilto keliamąją galią. Apskaičiuoti, kokius didžiausio svorio krovinius galima nuvežti iš viršūnės *s* iki

1) viršūnės *t*;
2) visų likusių grafo viršūnių.

Kadangi, 1)-asis uždavinys yra 2)-ojo uždavinio atskirasis atvejas, tai čia nagrinėsime antrąjį uždavinį. Pirmojo uždavinio sprendimo algoritmas skiriasi nuo antrojo uždavinio sprendimo algoritmo tik pabaigos sąlyga. Jei antrojo uždavinio sprendimo algoritmo pabaigos sąlyga yra: "*visos grafo viršūnės nudažytos*", tai pirmojo – "*viršūnė t nudažyta*".

Didžiausios keliamosios galios apskaičiavimo uždavinio sprendimo algoritmas yra analogiškas Deikstros algoritmui.

Apibrėžkime masyvus $d[1..n]$ ir $prec[1..n]$.

Masyvo *d* elementas $d[i]$- reiškia didžiausią svorį krovinio, kurį galima nugabenti iš viršūnės *s* iki viršūnės *i*. Pradžioje visi $d[i]=0$, $i=\overline{1,n}$.

Masyvo *prec* elementas čia, kaip ir 2.14.1 paragrafe, reiškia:

$$prec[i]=\begin{cases} k, & \text{jei kelias į viršūnę } i \text{ ateina iš viršūnės } k, \\ i, & \text{jei kelias prasideda viršūnėje } i. \end{cases}$$

Algoritmas

begin

Paruošiamasis žingsnis

Visos svorinio grafo G=(V,U,C) viršūnės – nenudažytos.

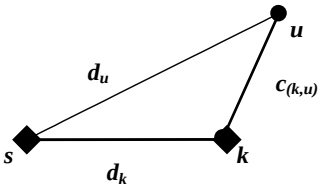
$sum := \sum_{i=1}^{m} c_i + 1$; {begalybė}

for i:=to n do
    begin
        d[*i*]:=0; *prec*[*i*]:=0
    end;
d[*s*]:= *sum; prec*[*s*]:= *s;*

Pagrindiniai skaičiavimai

while "yra nenudažytų viršūnių" do
    begin
        {*apskaičiuoti* $d[k] = \max_{v \in V}(d(v) / v - nenudažyta\ viršūnė)$}
        max:=0;
        for i:=1 *to n do*
            if "viršūnė i nenudažyta" and (*max*< *d*[*i*]) *then*
                begin
                    k:=*i;*
                    max:= *d*[*i*]
                end;
            if max=0 *then*
                begin
                    "grafas G - nejungusis"; stop
                end
                else
                    begin
                        Nudažome viršūnę k;
                        for u ∈ *N*(*k*) *do*
                            if " viršūnė u nenudažyta" and (*d*[*u*]< *min*(*d*[*k*],*c*(*k*,*u*)))
                                then
                                    begin
                                        {*d*[*u*]:= max (*d*[*u*],min (*d*[*k*],*c*(*k*,*u*))}
                                        d[*u*]:=*min* (*d*[*k*],*c*(*k*,*u*));
                                        prec[*u*]:= *k*
                                    end;
                    end;{*else*}
    end; {*while*}
end;

Masyvų *d* ir *prec* elementų perskaičiavimo pagrindimas parodytas 2.14.3 pav.



Kelio *s,k,u* keliamoji galia yra min(*d*[*k*],*c*(*k,u*))

2.14.3 pav. Didžiausias svoris

Kelio *s,k,u* didžiausia keliamoji galia yra min(*d*[*k*],*c*(*k,u*)). Vadinasi, jei šio naujojo kelio galia yra didesnė už visų žinomų kelių, jungiančių viršūnę s su viršūne u, keliamąją galią *d*[*u*], tai, *d*[*u*]:=min(*d*[*k*],*c*(*k,u*)), o *prec*[*u*]:=*k*, t.y. kelias į viršūnę *u* ateina iš viršūnės *k*.

2.14.3. Uždavinys apie stiprinimą

Uždavinio formulavimas. Tarkime, kad *G*=(*V,U,C*) yra orientuotasis grafas, neturintis ciklų ir turintis vieną viršūnę, neturinčią įeinančių lankų (minorantą) ir vieną viršūnę, neturinčią išeinančių lankų (mažorantą). Toks grafas vadinamas ***tinkliniu grafu*** Tarkime, kad lanko svoris reiškia stiprinimą. Kelio stiprinimas, tai visų jo lankų stiprinimų sandauga. Tarp viršūnių *s* ir *t* rasti kelią, kurio stiprinimas yra didžiausias.

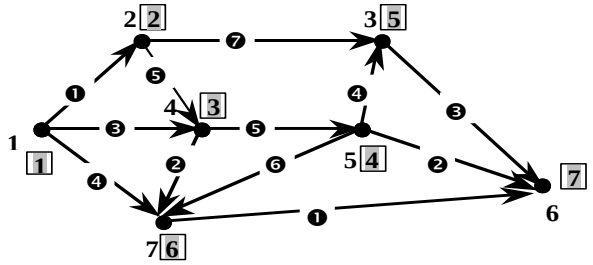
Pavyzdys. Tarkime, transportuojant prekę iš pagaminimo punkto į pardavimo punktą, kiekviename maršruto intervale prarandame dalį produkto. Koks turi būti prekės maršrutas, kad pervežimo nuostoliai būtų mažiausi.

Šį uždavinį galime suformuluoti, kaip kelio su didžiausiu stiprinimu uždavinį, jei kiekvienam lankui priskirsime dalį produkto, kuris be nuostolių pervežamas šiuo lanku.

Aišku, kad šio uždavinio sprendimo algoritmas yra visiškai toks pats, kaip ir didžiausios keliamosios galios uždavinio sprendimo algoritmas, išskyrus tai, kad pradžioje *d*[*s*]=1, o perskaičiuojant masyvų *d* ir *prec* elementus, kai nudažoma viršūnė *k* (žr. 2.14.3 pav.) elgsimės taip:

```
for u∈ N(k) do
    if d[u] < d[k]* c(k,u)
       then
          begin
             d[u]:=d[k]*c(k,u);
             prec[u]:= k;
             if viršūnė u nudažyta" then "viršūnės spalvą nutriname"
          end;
```

Pavyzdys. Grafui, pavaizduotam 2.14.4 pav. , tarp 1-osios ir 6-osios viršūnių raskime kelią, kurio stiprinimas būtų didžiausias.



2.14.4 pav. Didžiausias stiprinimas

Žemiau parodyta, kaip kito masyvų *d* ir *prec* turiniai. Žvaigždutės virš masyvo *d* elementų žymi kiek kartų šios viršūnės buvo dažytos: nubrauktos žvaigždutės žymi, kad viršūnė, kurią nusako elemento eilės numeris, pradžioje buvo nudažyta, o po to skaičiuojant jos spalva buvo nutrinta.

	1	2	3	4	5	6	7
***d*:**	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~
	1	1	~~60~~	~~3~~	~~15~~	~~4~~	~~4~~
			100	5	25	~~30~~	~~6~~
						~~90~~	~~90~~
						~~180~~	150
						300	

	1	2	3	4	5	6	7
***prec*:**	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~
	1	1	~~5~~	~~1~~	~~4~~	~~7~~	~~1~~
			5	2	4	~~5~~	~~4~~
						~~7~~	~~5~~
						~~3~~	5
						3	

Šis uždavinys būtų sprendžiamas žymiai paprasčiau, jei grafo viršūnės būtų sunumeruotos teisingai.

Apibrėžimas. Orientuotojo aciklinio grafo viršūnės ***sunumeruotos teisingai***, jei kiekvienam lankui (*i*,*j*) galioja sąlyga *i*<*j*.

Vienas iš paprasčiausių teisingo numeravimo metodų yra rangų metodas.

Apibrėžimas. Orientuotojo aciklinio grafo viršūnės *u* ***rangas***, tai ilgiausias kelias (pagal lankų skaičių) nuo minorantos iki viršūnės *u*.

Pavyzdžiui, 2.14.4 pav. tinklinio grafo viršūnių rangai surašyti 2.14.1 lentelėje.

2.14.1 lentelė. Viršūnių rangai

rangas	viršūnės Nr.
0	1
1	2
2	4
3	5
4	3,7
5	6

Aišku, kad to paties rango viršūnės nėra gretimos, todėl jos gali būti numeruojamos laisvai. Vadinasi, norint teisingai sunumeruoti grafą, reikia iš eilės einančiais natūraliaisiais skaičiais numeruoti viršūnes, pradedant nulinio rango viršūne. Laikantis šio principo, nagrinėjamam pavyzdžiui gausime tokį teisingo sunumeravimo masyvą *nr*(1,2,5,3,4,7,6); jei *nr*[*k*]=*l*, tai *k* viršūnės teisingas numeris yra *l* (žr. 2.14.4 pav.). Teisingi numeriai pažymėti kvadratėliuose.

Tarkime, kad grafas sunumeruotas teisingai. Tada didžiausio stiprinimo algoritmas bus toks.

```
begin
Paruošiamasis žingsnis
    for i:=1 to n do
        begin
            d[i]:=0;
            prec:=[i]:=0
        end;
    d[1]:=1; prec[1]:=1;
Pagrindiniai skaičiavimai
    for k:=1 to n-1 do
        for u∈N(k) do
            if d[u] < d[k] * c(k,u)  then
                begin
                    d[u]:= d[k]* c(k,u);
                    prec[u]:= k;
                end;
end;
```

Žemiau parodyta, kaip remiantis teisinga grafo numeracija ir šiuo algoritmu, kito masyvų *d* ir *prec* turiniai.

	¤ 1	¤ 2	¤ 3	¤ 4	¤ 5	¤ 6	¤ 7
***d*:**	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~
	1	1	~~3~~	25	~~7~~	~~4~~	~~50~~
			5		100	~~10~~	300
						150	

	1	2	3	4	5	6	7
***prec*:**	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~
	1	1	~~1~~	3	~~2~~	~~1~~	~~4~~
			2		4	~~3~~	5
						4	

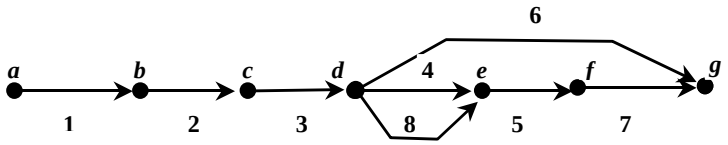
2.14.4. Ilgiausio kelio uždavinys

2.14.3 paragrafe apibrėžėme tinklinį grafą. Reikia pabrėžti, kad tinkliniai grafai yra patogi priemonė įvairiems projektams vaizduoti. Pavyzdžiui, norint pastatyti naują namą, turi būti atlikti tokie darbai:

1) sklypo išvalymas;
2) pamatų montavimas;
3) sienų mūrijimas;
4) elektros laidų pravedimas;
5) tinkavimas;
6) teritorijos sutvarkymas;
7) apdailos darbai;
8) stogo uždengimas.

Aišku, kad darbai turi eiliškumą, o taip pat reikalauja laiko sąnaudų bei materialinių resursų.

Namo statybos projektą galima pavaizduoti tinkliniu grafu (žr. 2.14.5 pav.). Grafo lankai vaizduoja darbus. Lanko svoris gali vaizduoti darbo trukmę arba šiam darbui atlikti reikalingų materialinių resursų kiekį. 2.14.5 pav. skaičiai prie lankų vaizduoja anksčiau išvardintų darbų numerius. Viršūnės vaizduoja darbų pradžios ir pabaigos momentus.



2.14.5 pav. Namo statybos grafas

Galimi įvairūs uždavinių formulavimai.

1. Tarkime, kad lanko svoris yra lankui atitinkančio darbo trukmė. Rasti trumpiausią laiką, per kurį, turint neribotus resursus, projektas gali būti įvykdytas.

Aišku, kad trumpiausias projekto įvykdymo laikas yra lygus ilgiausio kelio (kritinio kelio) nuo minorantos iki mažorantos ilgiui. Darbai, priklausą kritiniam keliui, vadinami kritiniais darbais. Bet kokio kritinio darbo suvėlinimas iššaukia viso projekto įvykdymo vėlinimą.

2. Kaip organizuoti darbus, kad, esant ribotiems resursams, projektas būtų įvykdytas galimai trumpiausiu laiku.

3. Kaip organizuoti darbus ir koks reikalingas minimalus resursų kiekis, kad projektas būtų įvykdytas per nurodytą laiką (aišku, laikas netrumpesnis nei kritinio kelio ilgis).

Čia aptarsime 1-ojo uždavinio sprendimą.

Kritinio kelio tinkliniame grafe radimo algoritmas

Apibrėžkime du masyvus $d[1..n]$ ir $prec[1..n]$, kurių prasmė yra ta pati, kaip ir Deikstros algoritme (žr. 2.14.1 paragrafą), išskyrus tai, kad $d[k]$ reiškia ilgiausio kelio nuo minorantos iki viršūnės k ilgį. Tarkime, tinklinis grafas

sunumeruotas teisingai (žr. 2.14.3 paragrafą). Tada kritinio kelio radimo algoritmas gali būti užrašytas taip.

```
begin
Paruošiamasis žingsnis
    for i:=1 to n do
        begin
            d[i]:=0;
            prec[i]:=0
        end;
    prec[1]:=1;
Pagrindiniai skaičiavimai
    for k:=1 to n-1 do
        for u∈N(k) do
            if d[u] < d[k] + c(k,u) then
                begin
                    d[u]:= d[k]+c(k,u);
                    prec[u]:= k;
                end;
end;
```

Pavyzdžiui, 2.14.4 pav. Tinklinio grafo kritinio kelio skaičiavimas iliustruojamas žemiau pateiktais masyvais d ir prec, t.y. parodant šių masyvų turinių kitimą.

	1	2	3	4	5	6	7
***d*:**	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~
		1	~~3~~	11	~~8~~	~~4~~	~~13~~
			6		15	~~8~~	18
						17	

	1	2	3	4	5	6	7
***prec*:**	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~
		1	~~1~~	3	~~2~~	~~1~~	~~4~~
			2		4	~~3~~	5
						4	

Šio tinklinio grafo kritinio kelio ilgis yra 18, ir šis grafas turi du kritinius kelius, nes, nagrinėjant 6-tajai viršūnei gretimas viršūnes, gauname, kad $d[6]+c(6,7)=18$. Tačiau, kelią, kurio ilgis lygus 18, iki 7-osios viršūnės jau buvome atradę iš 5-osios viršūnės. Tuo būdu masyvas *prec* žymi tik vieną kritinį kelią: 7,5,4,3,2,1.

Darbai, nepriklausantys kritiniams keliams, turi laiko rezervą, t.y. jei šio darbo atlikimo vėlinimas neviršija laiko rezervo, tai bendras projekto vykdymo laikas nepailgėja. Tuo tarpu darbų, priklausančių kritiniams keliams, laiko rezervai lygūs nuliui.

Pastaba. Darbų (operacijų) laiko rezervų apskaičiavimo algoritmai išnagrinėti literatūroje [Ma81].

Kadangi tinklinio grafo uždaviniai sprendžiami paprasčiau, kai grafas sunumeruotas teisingai, tai dabar aptarsime rangų apskaičiavimo ir teisingo numeravimo algoritmus.

Tinklinio grafo viršūnių rangų apskaičiavimas

Metodo idėja yra labai paprasta. Minoranta yra nulinio rango viršūnė. *k*-ojo rango viršūnės yra viršūnės, kurioms įėjimo puslaipsnis tampa lygus nuliui, kai iš grafo pašalinamos viršūnės, kurių rangas mažesnis už *k*.

Šį metodą patogu realizuoti naudojant paiešką į plotį: į eilę talpiname tas viršūnei *u* (viršūnė *u* – pirmoji eilės viršūnė) gretimas viršūnes, kurioms perskaičiuotas įėjimo puslaipsnis lygus nuliui.

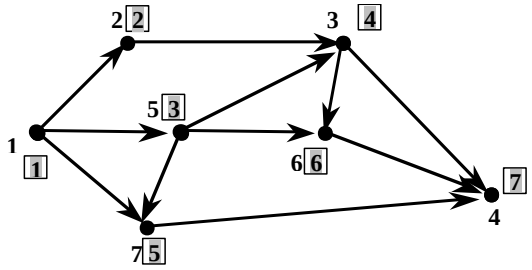
Tarkime, kad tinklinis grafas nusakytas briaunų matrica *B*[1..2,1..*m*], čia (*b*[1,*j*],*b*[2,*j*]) – *j*-tasis lankas ir *b*[1,*j*] – lanko pradžia, o *b*[2,*j*] – pabaiga. Tada viršūnių rangų apskaičiavimo procedūra gali būti užrašyta taip.

```
const c = 200;
type mas= array[1..c] of integer;
    matr= array[1..2,1..c] of integer;
procedure rangas ( n, m ,mn: integer; b:matr; var r: mas;var L1,lst1:mas);
    {procedūra rangas apskaičiuoja tinklinio grafo viršūnių rangus}
    {Formalūs parametrai
        n – tinklinio grafo viršūnių skaičius,
        m – tinklinio grafo lankų skaičius,
        mn – minorantos numeris,
        b – lankų matrica, b[1,j] – lanko pradžia, b[2,j] – pabaiga,
        r[1..n] – rangų masyvas, jei r[u]=k, tai reiškia, kad viršūnės u rangas
                  yra k;
        masyvų L1 ir lst1 prasmės paaiškintos žemiau.}
        {Pradžioje apskaičiuosime masyvus L1[1..m], lst1[1..n+1] ir L2[1..m] ir
    lst2[1..n+1].
        Masyvo L1[k], k = \overline{lst1[u]+1, lst1[u+1]}, elementai, žymi viršūnes, į
    kurias eina lankai iš viršūnės u.
        Masyvo L2[k], k = \overline{lst2[u]+1, lst2[u+1]}, elementai žymi viršūnes, iš
    kurių ateina lankai į viršūnę u.}
        var  i, j ,k ,u ,v, p, f, n1, s, x: integer;
            S1, S2, L2, lst2, fst1, fst2, eilė, d :mas;
    begin
     {Viršūnių laipsnių apskaičiavimas}
        for i:=1 to n do
            begin
                S1[i]:=0; S2[i]:=0;
            end;
        for j:=1 to m  do
            begin
                k:=b[1,j];
```

```
            S1[k]:=S1[k]+1;
            k:=b[2,j];
            S2[k]:=S2[k]+1;
        end;
{Masyvų lst1 ir lst2 formavimas}
    lst1[1]:=0; lst2[1]:=0;
    for i:=1 to n do
        begin
            lst1[i+1]:= lst1[i]+ S1[i];
            lst2[i+1]:= lst2[i]+ S2[i];
        end;
{Masyvų L1 ir L2 formavimas }
    for i:=1 to n do
        begin
            fst1[i]:= lst1[i]+ 1;
            fst2[i]:= lst2[i]+ 1;
        end;
    for j:=1 to m do
        begin
            k:=b[1,j];
            L1[fst1[k]]:=b[2,j];
            fst1[k]:=fst1[k]+1;
            k:=b[2,j];
            L2[fst2[k]]:=b[1,j];
            fst2[k]:=fst2[k]+1
        end;
{Rangų apskaičiavimas}
    for i:=1 to n do d[i]:=1; {d[i]=1, jei i- oji viršūnė – nenudažyta;
                                d[i]=0, jei i-oji viršūnė – nudažyta}
    n1:=n+1;
    p:=1; f:=1;  {eilė:=∅}
    r[mn]:=0;  d[mn]:=0;
    {eilė ⇐ mn}
    if  p=n1  then p:=1
         else p:=p+1;
    if  p= f  then
                begin
                    writeln ('eilės persipildymas');
                    exit;
                end
            else eilė[p]:=mn;
```

```
    while   f <> p   { eilė ≠ ∅ }  do
        begin
            {u ⇐ eilė}
            if  p=f  then
                begin
                    writeln (‘eilė tuščia’);
                    exit;
                end
            else
                begin
                    if  f = n1 then  f:=1
                        else  f:=f+1;
                    u:=eilė[f];
                end;
            for  i:=lst1[u]+1  to lst1[u+1]  do
                begin
                    v:=L1[i];
                    {Ar viršūnės  v įėjimo puslaipsnis lygus nuliui}
                    s:=0;
                    for j:= lst2[v]+1  to  lst2[v+1]  do
                        begin
                            x:= L2[j];
                            s:=s+d[x];
                        end;
                    if s=0  {Viršūnės v puslaipsnis lygus nuliui}
                        then
                            begin
                                r[v]:= r[u]+1; d[v]:=0;
                            end;
                    {eilė ⇐ v }
                    if  p= n1  then  p:=1
                        else  p:=p+1;
                    if  p = f  then
                        begin
                            writeln (‘eilės persipildymas);       exit;
                        end
                    else  eilė[p]:=v;
                end;{for i}
            end;  {while}
    end;  {rangas}
```

Pavyzdys. Panagrinėkime 2.14.6 pav. pavaizduotą tinklinį grafą, kurį nusako briaunų matrica $B = \begin{pmatrix} 1 & 1 & 1 & 2 & 3 & 3 & 5 & 5 & 5 & 6 & 7 \\ 2 & 5 & 7 & 3 & 6 & 4 & 3 & 7 & 6 & 4 & 4 \end{pmatrix}$.



2.14.6 pav. Tinklinio grafo viršūnių rangai

Pirmiausia bus apskaičiuoti masyvai:

$$L1: \overset{1}{2},\ \overset{2}{5},\ \overset{3}{7},\ \overset{4}{3},\ \overset{5}{6},\ \overset{6}{4},\ \overset{7}{3},\ \overset{8}{7},\ \overset{9}{6},\ \overset{10}{4},\ \overset{11}{4},$$

$$lst1: \overset{1}{0},\ \overset{2}{3},\ \overset{3}{4},\ \overset{4}{6},\ \overset{5}{6},\ \overset{6}{9},\ \overset{7}{10},\ \overset{8}{11},$$

$$L2: \overset{1}{1},\ \overset{2}{2},\ \overset{3}{5},\ \overset{4}{3},\ \overset{5}{6},\ \overset{6}{7},\ \overset{7}{1},\ \overset{8}{3},\ \overset{9}{5},\ \overset{10}{1},\ \overset{11}{5},$$

$$lst2: \overset{1}{0},\ \overset{2}{0},\ \overset{3}{1},\ \overset{4}{3},\ \overset{5}{6},\ \overset{6}{7},\ \overset{7}{9},\ \overset{8}{11}.$$

Į eilę viršūnės bus talpinamos tokia tvarka: 1, 2, 5, 3, 7, 6, 4 , o rangų masyvas bus toks:

$$r: \overset{1}{0},\ \overset{2}{1},\ \overset{3}{2},\ \overset{4}{4},\ \overset{5}{1},\ \overset{6}{3},\ \overset{7}{2}.$$

Tinklinio grafo teisingo pernumeravimo algoritmas

Tarkime, kad tinklinis grafas nusakytas masyvais *L*1[1..*m*], *lst*1[1..*n*+1] ir žinomas viršūnių rangų masyvas *r*[1..*n*], t.y. teisingo pernumeravimo algoritmo įėjimo parametrai sutampa su procedūros "***rangas"*** išėjimo parametrais.

Algoritmo išėjimo parametrai bus masyvai *L* [1..*m*], *lst* [1..*n*+1],čia L[*i*], $i = \overline{lst[u]+1, lst[u+1]}$, teisingai sunumeruoto tinklinio grafo viršūnės, į kurias lankai eina iš viršūnės *u*.

Įvesime masyvus: *nr*[1..*n*], čia *nr*[*u*] – naujas viršūnės *u* numeris ir *nrs*[1..*n*], čia *nrs*[*u*] yra viršūnės *u* senas numeris.

```
procedure numeravimas (n, m : integer; L1, lst1: mas; r :mas;
                                var nr:mas; var L, lst:mas);
    var i, k, t : integer;
        nrs : mas;
    begin
        {Masyvo nr[1..n] formavimas}
        k:=0; {rango numeris};
        t:=0; {viršūnės eilės numeris};
        while t < n {kol nesunumeruotos visos viršūnės} do
            begin
                for i:=1 to n do
                    if r[i] = k then
                        begin
                            t:= t+1;
                            nr[i]:= t;
                            nrs[t]:=i;
                        end;
                    k:=k+1;
            end; {while}
        {Masyvų L ir lst formavimas}
        lst[1]:=0;
        for i:=1 to n do
            begin
                {Naujam numeriui i randame seną numerį k}
                k:= nrs[i];
                {Apskaičiuojame masyvų L ir lst elementus}
                lst[i+1]:= lst[i]+lst1[k+1]-lst1[k];
                for  t:= lst1[k]+1 to lst1[k+1] do
                    L[lst[i]+t-lst1[k]]:= nr[L1[t]];
            end;
    end; {numeravimas}
```

Pavyzdys. Panagrinėkime 2.14.6 pav. Pavaizduoto tinklinio grafo viršūnių pernumeravimą. Po procedūros "***rangas"*** gausime masyvus:

	1	2	3	4	5	6	7
r:	0,	1,	2,	4,	1,	3,	2 ,

	1	2	3	4	5	6	7	8	9	10	11
*L*1:	2,	5,	7,	3,	6,	4,	3,	7,	6,	4,	4 ,

	1	2	3	4	5	6	7	8
*lst*1:	0,	3,	4,	6,	6,	9,	10,	11 ,

Šie masyvai yra procedūros "***numeravimas"*** įėjimo parametrai.

Pirmiausia apskaičiuosime masyvus *nr* ir *nrs.*

	1	2	3	4	5	6	7
nr:	1	2	4	7	3	6	5 ,

	1	2	3	4	5	6	7
nrs:	1	2	5	3	7	6	4 .

Masyvo *nr* elementai riša senuosius numerius su naujaisiais; pavyzdžiui, *nr*[5]=3 reiškia, kad 5-osios viršūnės naujasis numeris yra 3. Masyvo *nrs* elementai riša naujuosius numerius su senaisiais; pavyzdžiui, *nrs*[4]=3 reiškia, kad naujojo numerio 4 senasis numeris yra 3. Po masyvų *nr* ir *nrs* apskaičiavimo, bus apskaičiuoti masyvai *L* ir *lst*, kurie ir nusakys teisingai sunumeruotą tinklinį grafą (2.14.6 pav. teisingi numeriai parašyti kvadratėliuose):

	1	2	3	4	5	6	7	8
lst:	0,	3,	4,	7,	9,	10,	11,	11

ir

	1	2	3	4	5	6	7	8	9	10	11
L:	2,	3,	5,	4,	4,	5,	6,	6,	7,	7,	7 .

2.14.5. Trumpiausių kelių tarp visų viršūnių porų apskaičiavimas

Uždavinio formulavimas. Duotas svorinis (*n*,*m*)-grafas *G*=(*V*,*U*,*C*), čia *C* – briaunų svorių aibė. Rasti trumpiausius kelius tarp visų viršūnių porų.

Šio uždavinio sprendimas yra n-osios eilės kvadratinė matrica *D*=[d_{ij}] $i=\overline{1,n}$, $j=\overline{1,n}$, kurios elementas d_{ij} reiškia ilgį trumpiausio kelio tarp viršūnių *i* ir *j* poros. Jei be kelio ilgio mus domina ir per kurias viršūnes šis kelias eina, tai šalia matricos *D* reikia apskaičiuoti *n*-osios eilės matricą *P*, kurios elementai p_{ij} rodo, kuria kryptimi (į kurią viršūnę arba iš kurios viršūnės) iš viršūnės einama.

Deikstros metodas

Aišku, kad šį uždavinį galima spręsti Deikstros algoritmo pagalba. 2.14.1 paragrafe išnagrinėtas Deikstros algoritmas leidžia apskaičiuoti trumpiausius kelius nuo viršūnės *s* iki visų likusių viršūnių, t.y. Deikstros algoritmo pagalba randami masyvai *d*[1..*n*] ir *prec*[1..*n*]. Masyvo *d* elementas d_k reiškia ilgį trumpiausio kelio nuo viršūnės *s* iki viršūnės *k*, o masyvo *prec* elementai žymi trumpiausių kelių nuo viršūnės *s* iki likusių grafo viršūnių medį (žr. 2.14.1 paragrafą).

Aišku, kad masyvas *d* yra matricos *D* *s*-toji eilutė, o masyvas *prec* – matricos *P* *s*-toji eilutė. Vadinasi norint apskaičiuoti *D* ir *P* matricas, į Deikstros algoritmą reikės kreiptis, kai *s* kinta nuo 1 iki *n*, ir po kiekvieno

kreipinio masyvų *d* ir *prec* elementus reikės patalpinti į atitinkamas matricų *D* ir *P* eilutes. Aišku, kad d_{ij} reikš ilgį trumpiausio kelio, tarp *i* ir *j* viršūnių, o per kurias viršūnes šis kelias eina sužinosime taip: tarkime, kad $p_{ij}= t_1$, $p_{it_1}= t_2, \dots ,$ $p_{it_{k-1}}= t_k$, $p_{it_k}= i$; tada trumpiausias kelias eis per viršūnes: $i, t_k, t_{k-1}, \dots ,t_2, t_1, j$.

Floido metodas

Floido metodo idėja yra labai paprasta. Tarkime, kad svorinio (*n*,*m*)-grafo *G*=(*V*,*U*,*C*) viršūnės sunumeruotos iš eilės einančiais natūraliaisiais skaičiais nuo 1 iki *n*.

Pastaba. Kalbėdami apie žymėtuosius grafus įsivaizdavome, kad grafo viršūnės būtent taip ir sunumeruotos.

Tada matrica *D* gaunama nuosekliai apskaičiuojant matricas $D^0, D^1, \dots, D^m, \dots, D^n$. Matricos $D^m = [d_{ij}^m]$ $i = \overline{1,n}$, $j = \overline{1,n}$ elementas reiškia ilgį trumpiausio kelio tarp *i* ir *j* viršūnių, kai tarpinėmis šio kelio viršūnėmis gali būti tik viršūnės su numeriais nuo 1 iki *m*. Priminsime, kad tarpine kelio viršūne vadinsime bet kurią jo viršūnę, nesutampančią su pradine ir galine šio kelio viršūne. Kalbamuoju atveju tai viršūnės, nesutampančios su viršūnėmis *i* ir *j*.

Jei tarp viršūnių *i* ir *j* nurodyto tipo kelio nėra, tai $d_{ij}^m = \infty$.

Pirmiausia apibrėšime matricą. D^0 Šios matricos elementas d_{ij}^0, tai ilgis trumpiausio kelio tarp viršūnių *i* ir *j*, kuris neturi tarpinių viršūnių, t.y. $d_{ij}^0 = c(i, j)$, čia c (i, j) – briaunos (lanko) (i,j) ilgis. Aišku, kad $d_{ii}^0 = 0, \quad i = \overline{1,n},$ o $d_{ij}^0 = \infty, \quad jei \quad (i, j) \notin U,$ (briaunos (lanko) (i,j) nėra).

Vadinasi $$d_{ij}^0 = \begin{cases} c(i, j), & jei \quad (i, j) \in U, \\ \infty, & jei \quad (i, j) \notin U, \\ 0, & jei \quad i = j, \end{cases}$$ visiems $i = \overline{1,n}, \quad j = \overline{1,n}.$

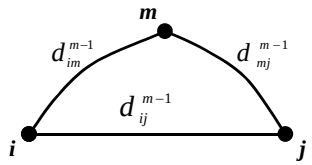
Kaip iš matricos D^{m-1} apskaičiuoti matricą D^m, *m*=1, 2, … , *n* ?

Panagrinėkime 2.14.7 pav. Šiame paveiksle

d_{ij}^{m-1} žymi ilgį trumpiausio kelio tarp viršūnių *i* ir *j*, kai tarpinėmis šio kelio viršūnėmis yra viršūnės, kurių numeriai yra tarp 1 ir *m*-1;

d_{im}^{m-1} žymi ilgį trumpiausio kelio tarp viršūnių *i* ir *m*, kai tarpinėmis kelio viršūnėmis yra viršūnės su numeriais tarp 1 ir *m*-1;

d_{mj}^{m-1} žymi ilgį trumpiausio kelio tarp viršūnių *m* ir *j*, kai tarpinėmis kelio viršūnėmis yra viršūnės su numeriais tarp 1 ir *m*-1.



2.14.7 pav. Floido metodas

Kadangi d_{ij}^{m} žymi ilgį trumpiausio kelio nuo *i* iki *j*, kai to kelio tarpinėmis viršūnėmis yra viršūnės su numeriais nuo 1 iki *m*, tai aišku, kad šio kelio ilgis yra $\min\left(d_{ij}^{m-1}, d_{im}^{m-1} + d_{mj}^{m-1}\right)$.

Vadinasi,

$$d_{ij}^{m} = \begin{cases} d_{ij}^{m-1}, & jei \quad d_{ij}^{m-1} \le d_{im}^{m-1} + d_{mj}^{m-1}, \\ d_{im}^{m-1} + d_{mj}^{m-1} & priešingu\,atveju, \end{cases} \text{ visiems } i = \overline{1,n}, \quad j = \overline{1,n}.$$

Kaip minėjome anksčiau, paprastai mus domina ne tik trumpiausio kelio tarp viršūnių *i* ir *j* ilgis, bet ir per kokias viršūnes šis kelias eina. Tam tikslui apibrėšime matricas $P^0, P^1, \ldots, P^m, \ldots, P^n$. Matricos P^m elementas p_{ij}^{m} reiškia numerį viršūnės, į kurią tiesiogiai trumpiausias kelias veda iš viršūnės *i* į viršūnę *j*.

Aišku, kad

$$p_{ij}^{0} = \begin{cases} j, & jei \quad (i,j) \in U, \\ 0, & jei \quad (i,j) \notin U, \end{cases} \qquad \text{visiems} \quad i = \overline{1,n}, \quad j = \overline{1,n}.$$

Kaip iš matricos P^{m-1} apskaičiuoti matricą P^m? Iš 2.14.7 pav. nesunku suvokti, kad

$$p_{ij}^{m} = \begin{cases} p_{ij}^{m-1}, & jei \quad d_{ij}^{m-1} \le d_{im}^{m-1} + d_{mj}^{m-1}, \\ p_{im}^{m-1} & priešingu\,atveju, \end{cases} \text{ visiems } i = \overline{1,n}, \quad j = \overline{1,n}.$$

Kaip minėjome anksčiau, matricos D^n ir P^n atitinkamai žymi ilgius trumpiausių kelių tarp visų viršūnių porų ir trumpiausių kelių medžius.

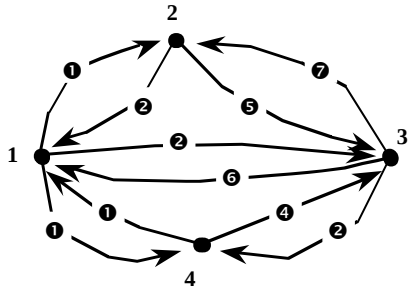
Kaip sužinoti, per kurias viršūnes eina trumpiausias kelias, jungiantis *i* ir *j* viršūnes?

Tarkime, kad $p_{ij} = t_1$, $p_{t_1 j} = t_2$, … , $p_{t_{k-1} j} = t_k$, ir $p_{t_k j} = j$; tada trumpiausias kelias eina per viršūnes: $i, t_1, t_2, \ldots, t_k, j$.

Pavyzdys. Panagrinėkime orientuotąjį grafą, pavaizduotą 2.14.8 pav.

Floido metodu raskime atstumus tarp visų viršūnių porų.

Žemiau pateiktos matricos D^m ir P^m, $m = \overline{1,4}$, kurios apskaičiuotos pagal anksčiau pateiktas formules.



2.14.8 pav. Floido metodo iliustracija

$$D^0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 2 & 1 \\ 2 & 2 & 0 & 7 & - \\ 3 & 6 & 5 & 0 & 2 \\ 4 & 1 & - & 4 & 4 \end{array}, \quad P^0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 2 & 3 & 4 \\ 2 & 1 & 0 & 3 & 0 \\ 3 & 1 & 2 & 0 & 4 \\ 4 & 1 & 0 & 3 & 0 \end{array}.$$

$$D^1 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 2 & 1 \\ 2 & 2 & 0 & 4 & 3 \\ 3 & 6 & 5 & 0 & 2 \\ 4 & 1 & 2 & 3 & 0 \end{array}, \quad P^1 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 2 & 3 & 4 \\ 2 & 1 & 0 & 1 & 1 \\ 3 & 1 & 2 & 0 & 4 \\ 4 & 1 & 1 & 1 & 0 \end{array}.$$

$$D^2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 2 & 1 \\ 2 & 2 & 0 & 4 & 3 \\ 3 & 6 & 5 & 0 & 2 \\ 4 & 1 & 2 & 3 & 0 \end{array}, \quad P^2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 2 & 3 & 4 \\ 2 & 1 & 0 & 1 & 1 \\ 3 & 1 & 2 & 0 & 4 \\ 4 & 1 & 1 & 1 & 0 \end{array}.$$

$$D^3 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 2 & 1 \\ 2 & 2 & 0 & 4 & 3 \\ 3 & 6 & 5 & 0 & 2 \\ 4 & 1 & 2 & 3 & 0 \end{array}, \quad P^3 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 2 & 3 & 4 \\ 2 & 1 & 0 & 1 & 1 \\ 3 & 1 & 2 & 0 & 4 \\ 4 & 1 & 1 & 1 & 0 \end{array}.$$

$$D^4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 2 & 1 \\ 2 & 2 & 0 & 4 & 3 \\ 3 & 3 & 4 & 0 & 2 \\ 4 & 1 & 2 & 3 & 0 \end{array}, \qquad P^4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 2 & 3 & 4 \\ 2 & 1 & 0 & 1 & 1 \\ 3 & 4 & 4 & 0 & 4 \\ 4 & 1 & 1 & 1 & 0 \end{array}.$$

Pavyzdžiui, d_{31}= 3 ir trumpiausias kelias iš 3-osios į 1-ąją viršūnę eina: 3, 4, 1. d_{24}= 3, o kelias eina: 2, 1, 4.

2.15. Maršrutai

Šiame paragrafe nagrinėsime Oilerio ir Hamiltono maršrutus: grandines ir ciklus. Šie klausimai iškyla ne tiktai sprendžiant įvairius galvosūkius, bet ir praktiniuose uždaviniuose.

Pavyzdžiui, buitinio aptarnavimo mašina turi aplankyti kokio tai rajono visas gatves. Koks turi būti jos maršrutas? Tai Oilerio maršruto uždavinys.

Su Hamiltono maršrutais susijęs garsus komivojažerio arba keliaujančio pirklio uždavinys, kuris formuluojamas taip. Turime n miestų, o matrica $C=[c_{ij}]$ $i=\overline{1,n}, \quad j=\overline{1,n}$ yra atstumų matrica: c_{ij} – atstumas tarp miestų i ir j. Pirklys, išėjęs iš 1-ojo miesto turi apeiti visus miestus po vieną kartą ir grįžti į 1-ąjį miestą. Koks turi būti pirklio maršrutas, kad jo ilgis būtų trumpiausias?

2.15.1. Oilerio maršrutai

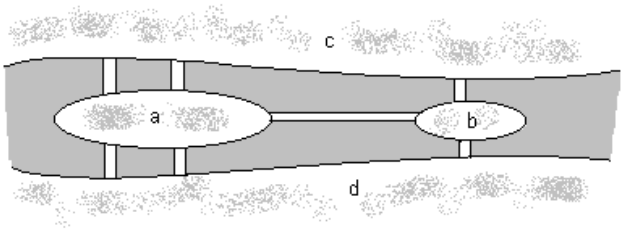
Visa, kas bus kalbama šiame paragrafe, tinka ir multigrafams.

Apibrėžimas. Maršrutas (kelias), apeinantis visas grafo briaunas po vieną kartą, vadinamas ***Oilerio maršrutu***. Jei pradinė ir galinė maršruto viršūnės nesutampa, tai toks maršrutas vadinamas ***Oilerio grandine***, priešingu atveju – ***Oilerio ciklu***.

Grafas, turintis Oilerio maršrutą, vadinamas Oilerio grafu.

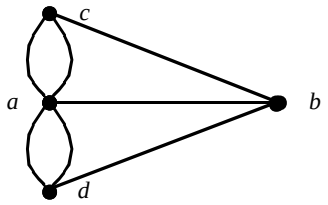
Reikia pastebėti, kad Oilerio maršruto radimo uždavinys yra pirmasis grafų teorijos uždavinys. Tai uždavinys apie Karaliaučiaus tiltus, kurį 1736 m. sėkmingai išsprendė Oileris.

Per Karaliaučių teka upė Prėglis, kurioje yra dvi salos. Šios salos su krantais ir tarpusavyje yra sujungtos tiltais (žr. 2.15.1 pav.).



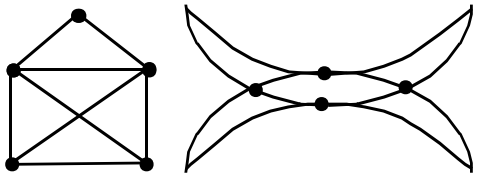
2.15.1 pav. Karaliaučiaus tiltai

Ar galima, išėjus iš namų, apeiti visus tiltus po vieną kartą ir vėl grįžti į namus. Kitaip tariant, ar multigrafas, pavaizduotas 2.15.2 pav. turi Oilerio maršrutą?



2.15.2 pav. Karaliaučiaus tiltų grafas

Su Oilerio maršrutu yra susiję įvairūs galvosūkiai. Pavyzdžiui, ar galima, neatitraukus rankos nuo popieriaus, nupiešti figūras pavaizduotas 2.15.3 a) ir b) pav.?



2.15.3 pav. Oilerio maršrutai

Kaip minėjome, ar grafas turi Oilerio maršrutą, išsprendė Oileris įrodydamas teoremą.

Teorema (Oilerio teorema, 1736). Būtina ir pakankama sąlyga, kad grafas *G* turėtų Oilerio maršrutą yra:

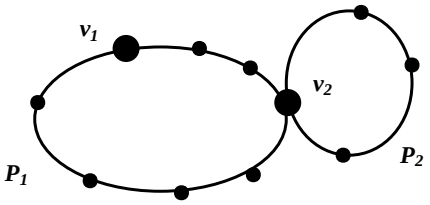
1) G turi būti jungusis,
2) visų jo viršūnių laipsniai turi būti arba lyginiai, arba G turi turėti tik dvi nelyginio laipsnio viršūnes.

Pirmuoju atveju grafas turi Oilerio ciklą, o antruoju – grandinę, prasidedančią ir besibaigiančią nelyginio laipsnio viršūnėse.

Pateiksime šios teoremos įrodymą, kadangi jis yra konstruktyvus: iš įrodymo išplaukia Oilerio maršruto konstravimo algoritmas.

Būtinumas. Tarkime G – Oilerio grafas. Oilerio ciklas eidamas per kiekvieną viršūnę , patenka į viršūnę viena briauna, o išeina – kita briauna. Tai reiškia, kad kiekviena grafo G viršūnė incidentiška lyginiam Oilerio ciklo briaunų skaičiui. Kadangi Oilerio ciklas apima visas grafo briaunas, tai reiškia, kad grafas yra jungusis ir, kad kiekvienos grafo viršūnės laipsnis yra lyginis.

Pakankamumas. Tarkime, kad visų grafo G viršūnių laipsniai yra lyginiai. Pradėkime kelionę iš viršūnės v_1 ir, eidami grafo briaunomis, laikysimės taisyklės: "niekada neiti ta pačia briauna". Tai tolygu taisyklei: "pereitą briauną – ištriname". Atėję į bet kurią viršūnę $v \neq v_1$, iš jos visada galėsime išvykti, nes viršūnės v laipsnis yra lyginis. Jei patekome į viršūnę, iš kurios negalime išvykti, tai reiškia, kad esame pradinėje kelio viršūnėje v_1. Vadinasi mūsų kelias sudarė ciklą P_1. Pašalinę šį ciklą iš grafo G, gausime grafą G', kurio kiekvienos viršūnės laipsnis yra arba lyginis, arba lygus nuliui. Jei visų viršūnių laipsniai yra lygūs nuliui, tai reiškia, kad P_1 yra Oilerio ciklas. Priešingu atveju nagrinėsime grafą G'. Kadangi grafas G yra jungusis grafas, tai grafai P_1 ir G' turi bent vieną bendrą viršūnę v_2. Pradėję kelionę grafe G' iš viršūnės v_2 ir laikydamiesi tos pačios taisyklės, sukonstruosime ciklą P_2. Iš ciklų P_1 ir P_2 sukonstruosime bendrą ciklą taip: iš viršūnės v_1 ciklo P_1 briaunomis nukeliausime į viršūnę v_2, po to apeisime ciklą P_2 ir po to iš viršūnės v_2 ciklo P_1 briaunomis ateisime į viršūnę v_1 (žr. 2.15.4 pav.).



2.15.4 pav. Oilerio ciklo konstravimas

Pašalinkime šį ciklą iš grafo G. Jei po ciklo pašalinimo grafas yra tuščiasis, tai šis ciklas yra Oilerio ciklas. Priešingu atveju, šiame grafe atliksime analogiškus veiksmus. Ir taip elgsimės iki sukonstruosime ciklą, einantį per visas grafo G briaunas.

Teorema (R.Reidas, 1962). Beveik nėra Oilerio grafų. [EM90].

Primename, kad sąvoka "beveik nėra grafų" apibrėžiama taip. Simboliu $GP(n)$ pažymėkime skaičių n-viršūninių grafų, turinčių Oilerio maršrutą, t.y. Oilerio grafų skaičių. Simboliu $G(n)$ pažymėkime visų n-viršūninių grafų skaičių. Tada $\lim\limits_{n\to\infty}\dfrac{GP(n)}{G(n)}=0$, ir sakome, kad "beveik nėra Oilerio grafų".

Oilerio ciklo konstravimo algoritmas

Duota: Jungusis grafas G, kurio visų viršūnių laipsniai yra arba lyginiai, arba G turi dvi nelyginio laipsnio viršūnes. Tarkime, kad grafas G nusakytas gretimumo struktūra: $N(v)$ – tai aibė viršūnių gretimų viršūnei v.

Rasti: Sukonstruoti Oilerio maršrutą.

Įrodyme pateiktą Oilerio maršruto konstravimą patogu realizuoti naudojant du stekus: *STEK* ir *Oiler*.

Priminsime, kad stekas, tai tiesinis sąrašas, kuriame naujo elemento patalpinimas ir pašalinimas atliekamas viename sąrašo gale. Kitaip dar stekas vadinamas sąrašu LIFO (nuo angliškų žodžių "last in first out" (paskutinis į steką – pirmas iš jo)).

Steką patogu organizuoti vienmačiu masyvu *STEK* [1..n]. Jį charakterizuoja vienas parametras – rodyklė "*top*", kuri reiškia paskutinio elemento sąraše adresą.

Stekas tuščias {*žymime* ***STEK*:= ∅**}
top: =0;
Elemento patalpinimas į steką {*žymime* ***STEK*⇐ *v***}.
top: = *top* +1;
if top > *n then "steko persipildymas"*
else STEK[*top*]: = *v;*
Elemento pašalinimas iš steko {*žymime* ***v*⇐ *STEK***}.
if top = 0 *then "stekas tuščias"*
else begin
v: = *STEK*[*top*]*;*
top: =*top* -1
end;
Elemento nuskaitymas iš steko {*žymime* ***v*: = *top*(*STEK*)**}.
if top=0 *then "stekas tuščias"*
else v: =*STEK*[*top*]*;*

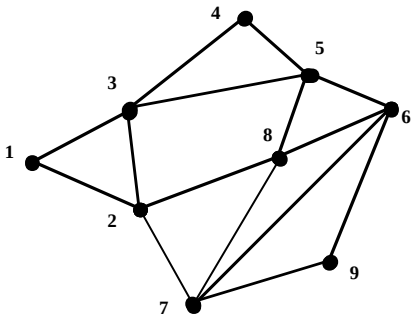
Algoritmo veikimo principas. Tarkime v_1 yra viršūnė, iš kurios pradėsime brėžti Oilerio ciklą. Šią viršūnę patalpinkime į steką *STEK*. Iš viršūnės v_1 keliausime per grafą, talpindami kelio viršūnes į steką *STEK* ir ištrindami kelio briaunas. Jei atėjome į viršūnę, iš kurios negalime išeiti, tai

reiškia, kad esame pradinėje kelio viršūnėje (jei grafo G visų viršūnių laipsniai buvo lyginiai) arba kitoje nelyginio laipsnio viršūnėje (jei v_1 laipsnis buvo nelyginis). Šią viršūnę patalpinsime į steką *Oiler* ir bandysime keliauti iš priešpaskutinės kelio viršūnės. Kelionė galima, nes iš grafo G pašalinome ciklą, einantį per viršūnę v_1, ir likusių viršūnių laipsniai yra arba lyginiai arba lygūs nuliui.

Taip elgsimės iki stekas *STEK* taps tuščias. Tada viršūnės patalpintos steke *Oiler* ir nusakys Oilerio maršrutą.

```
begin
    STEK:= ∅ ;  Oiler:= ∅ ;
    v: = bet kuri grafo viršūnė, jei visų grafo viršūnių laipsniai yra
         lyginiai arba nelyginio laipsnio viršūnė, jei grafas turi dvi
         nelyginio laipsnio viršūnes.
    STEK⇐v;
    while STEK ≠ ∅  do
        begin
            v:= top(STEK); {v = viršutinis steko elementas; rodyklė top
                                neperskaičiuojama}
            if  N(v) ≠ ∅
                then
                    begin
                            u:= pirmoji aibės N(v) viršūnė;
                            STEK ⇐u;
                            {pašalinti (v,u) briauną}
                            N(v):= N(v) \ {u} ;
                            N(u):= N(u) \ {v} ;
                    end
                else { N(v) = ∅ }
                    begin
                        v⇐ STEK ;
                        Oiler ⇐ v;
                    end;
        end;
end;
```

Pavyzdys. Remdamiesi algoritmu, sukonstruokime Oilerio ciklą grafui, pavaizduotam 2.15.5 pav.



Grafas užrašytas gretimumo struktūra
1: 2, 3;
2: 1, 3, 7, 8;
3: 1, 2, 4, 5;
4: 3, 5;
5: 3, 4, 6, 8;
6: 5, 7, 8, 9;
7: 2, 6, 8, 9;
8: 2, 5, 6, 7;
9: 6, 7.

2.15.5 pav. Oilerio ciklo pavyzdys

Kadangi grafo visų viršūnių laipsniai yra lyginiai, tai kelionę (Oilerio ciklo konstravimą) pradėsime iš 1-osios viršūnės. Tada stekų: *STEK* ir *Oiler* turiniai bus tokie:

STEK: 1, 2, 3, ~~1~~, 4, 5, ~~3~~, 6, 7, 2, 8, ~~5~~, 6, 9, 7, ~~8~~

Oiler: 1, 3, 5, 8, 7, 9, 6, 8, 2, 7, 6, 5, 4, 3, 2, 1.

Nubraukti elementai steke STEK žymi momentus, kai keliones metu atėjome į viršūnes, iš kurių nebuvo galima išeiti.

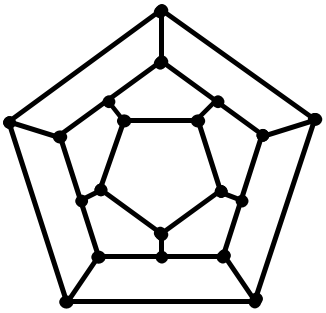
Pastaba. Praktikoje tenka spręsti tokį uždavinį. Duotas svorinis grafas *G*, kuris nėra Oilerio grafas. Kokias šio grafo briaunas pakartoti, kad jis taptų Oilerio grafu ir Oilerio ciklo ilgis būtų trumpiausias. Čia šio uždavinio nenagrinėsime, pažymėdami, kad jo sprendimas pateiktas literatūroje [K78].

2.15.2. Hamiltono maršrutai

Apibrėžimas. Maršrutas (kelias) apeinantis visas grafo viršūnes po vieną kartą vadinamas Hamiltono maršrutu. Jei pradinė ir galinė maršruto viršūnės sutampa, tai šis maršrutas vadinamas Hamiltono ciklu; priešingu atveju – Hamiltono grandine.

Grafas turintis Hamiltono maršrutą vadinamas Hamiltono grafu.

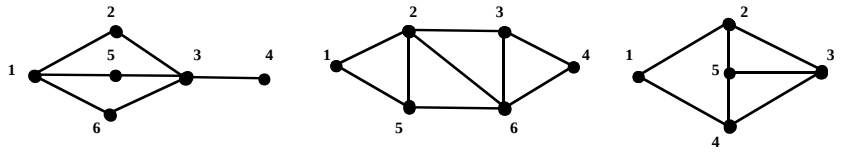
Hamiltono grafo sąvoka susijusi su žymaus airių matematiko W. Hamiltono vardu , kuris 1859 m. pasiūlė žaidimą, pavadintą “Kelionė aplink pasaulį”. Kiekvienai iš 20-ies dodekaedro viršūnių priskiriamas koks nors žymaus pasaulio miesto vardas. Reikia, einant iš vieno miesto į kitą, aplankyti visus miestus po vieną kartą ir grįžti į pradinį miestą. (žr. 2.15.6 pav.)



2.15.6 pav. Kelionė aplink pasaulį

Skirtingai nei Oilerio grafo atveju nėra žinoma nei vienos paprastos būtinos ir pakankamos sąlygos, kuri pasakytų ar grafas yra Hamiltono grafas. Negana to, nėra žinoma nei vieno algoritmo, nustatančio, ar grafas yra Hamiltono grafas ir kurio veiksmų skaičius būtų išreiškiamas polinomu nuo viršūnių skaičiaus *n*. Kitaip tariant, Hamiltono grafo nustatymo uždavinys priklauso *NP* uždavinių klasei [GD82].

2.15.7 pav. pateikti trys grafai, iš kurių grafas a) nėra Hamiltono grafas, grafas b) turi Hamiltono grandinę, o grafas c) – Hamiltono ciklą.



a) Nėra b) 1, 5, 2, 6, 3, 4 c) 1, 2, 3, 5, 4, 1

2.15.7 pav. Hamiltono maršrutai

Aišku, jei grafas yra pilnasis, tai jis yra Hamiltono grafas.

Pateiksime trejetą teoremų, nusakančių pakankamas sąlygas, kad grafas būtų Hamiltono grafas.

Teorema (V. Hvatalas, 1972) [EM90]. Grafas su viršūnių laipsnių seka $d_1 \le d_2 \le \cdots \le d_n$ yra Hamiltono grafas, jei bet kuriam sveikajam *k*, tenkinančiam sąlygą $1 \le k < \lfloor n/2 \rfloor$ teisinga implikacija $(d_k \le k) \Rightarrow (d_{n-k} \ge n-k)$.

Teorema (O. Ore, 1960) [EM90]. Jei *n* viršūnių ($n \ge 3$) grafo *G* bet kuriai negretimų viršūnių porai *u* ir *v* teisinga nelygybė: $d(u) + d(v) \ge n$, tai G – Hamiltono grafas, čia $d(u)$, $d(v)$ – viršūnių *u* ir *v* laipsniai.

Teorema (G. Dirakas, 1952) [EM90]. Jei n viršūnių grafo G kiekvienos viršūnės laipsnis nemažesnis nei $n/2$, tai G - Hamiltono grafas.

Teorema (Prepielica V.A., 1969) [EM90].Beveik visi grafai yra Hamiltono grafai.

Aptarkime Hamiltono maršrutų radimo uždavinį:

1) nustatyti ar duotasis grafas G yra Hamiltono grafas ir
2) duotajam grafui G rasti visus Hamiltono maršrutus.

Aišku, kad 1)-asis uždavinys yra 2)-ojo uždavinio atskirasis atvejas, todėl čia nagrinėsime 2)-ąjį uždavinį.

Aišku, kad visus Hamiltono maršrutus rasime išnagrinėję visas maršrutų galimybes, kurių skaičius yra $n!$. Kitaip tariant, kiekvienas n – elementų kėlinys gali būti Hamiltono maršrutas, t.y. kiekvienam kėliniui reikės patikrinti ar jis nusako Hamiltono maršrutą. Šio algoritmo sudėtingumas yra $O(n \bullet n!)$. Kadangi faktorialas auga greičiau nei eksponentinė funkcija, tai šis kelias yra labai neracionalus.

Aptarkime racionalesnį Hamiltono maršrutų apskaičiavimo algoritmą, nors ir šio algoritmo veiksmų skaičius apribotas eksponente. Šis algoritmas pagrįstas “paieška gilyn su grįžimu” (angl. backtracking).

Šio metodo esmė yra labai paprasta: jei paieškos gilyn metu viršūnė u tampa išsemta, tai pamirštame, kad šioje viršūnėje buvome, t.y. viršūnė u tampa nauja. Šis metodas įgalina perrinkti visus galimus grafo maršrutus.

Žemiau pateikiame šį metodą realizuojantį algoritmą.

Tarkime, kad grafas $G=(V,U)$ yra jungusis ir nusakytas masyvais L ir *lst* (žr. 2.7 paragrafą). Paiešką pradėsime iš 1-osios viršūnės. Paieškos gilyn metu aplankytas viršūnes talpinsime į masyvą $X[1..n+1]$. Kitaip tariant, masyvo X elementai apibrėš aplankytųjų viršūnių maršrutą. Jei paieškos gilyn metu masyve X yra n elementų, ir iš viršūnės x_n atėjome į 1-ąją viršūnę, tai reiškia, kad radome Hamiltono ciklą. Jei viršūnei x_n gretimų viršūnių tarpe nėra 1-osios viršūnės, tai masyvo X elementai nusakys Hamiltono kelią. Žemiau pateiktoje procedūroje pastaroji sąlyga netikrinama.

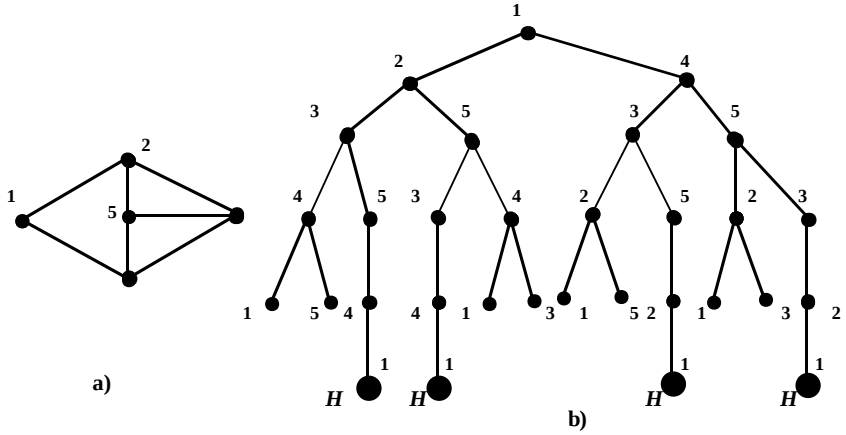
```
const c = 500;
type mas = array[1..c] of integer;
    matr = array[1..2,1..c] of integer;
var X, fst, prec: mas;
procedure hamil (n,m:integer; L,lst:mas; var alfa:boolean);
    { Procedūra hamil apskaičiuoja Hamiltono ciklus paieškos gilyn iš
    pirmosios viršūnės metodu, kai grafas nusakytas L ir lst masyvais. Rasti
    ciklai atspausdinami.
```

```
        Formalūs parametrai:
        n   - grafo viršūnių skaičius,
        m   - grafo briaunų (lankų) skaičius,
        L, lst - grafą nusakantys tiesioginių nuorodų masyvai. }
    var i, k, u, v   : integer;
        j  : integer;
        t,p : boolean;
        fst, prec  : mas;
        x : mas;
begin
    {Inicializacija}
        alfa:=false;
        v:=1;
        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 }
                    j:=1;
                    x[j]:=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 {viršūnė u nauja}
                             begin
                                 j:=j+1;
```

```
                    x[j]:=u;
                    prec[u]:=k; {į viršūnė u atejome iš viršūnės k}
                    if fst[u] <= lst[u+1] then
                        {viršūnė u neišsemta}
                        k:=u
                    else {viršūnė u išsemta}
                        p:=false;
                end
            else
                begin
                    { write(u:3);}
                    p:=false; {viršūnė u nenauja}
                    if (j = n) and (u = 1)
                        then { Radome Hamiltono ciklą }
                            begin
                                alfa:=true;
                                x[n+1]:=1;
                                writeln('Ciklas');
                                for i:=1 to n+1 do
                                    write(x[i]: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 į viršūnę, iš kurios buvome atėję į
                        viršūnę k  }
                    begin
                        u:=prec[k];
                        prec[k]:=0;
```

```
                    fst[k]:=lst[k]+1;
                    k:=u;
                    j:=j-1;
                  end;
            end;
      end;
end;
```

Pavyzdys. Panagrinėsime 2.15.7 c) pav. grafą. Aprašytos procedūros pagalba gausime tokius Hamiltono ciklus: 1, 2, 3, 5, 4, 1; 1, 2, 5, 3, 4, 1; 1, 4, 3, 5, 2, 1; 1, 4, 5, 3, 2, 1. 2.15.8 a) pav. pavaizduotas 2.15.7 c) pav. grafas, 2.15.8 b) pav. pavaizduotas paieškų medis, t.y. paieškos gilyn su grįžimu perrinkti maršrutai. Raide "H" pažymėti Hamiltono ciklai.



2.15.8 pav. Hamiltono ciklų ieškojimas

Reikia pažymėti, nors pateiktos procedūros veiksmų skaičius yra žymiai mažesnis nei pilno perrinkimo veiksmų skaičius, kuris lygus $O(n \bullet n!)$, tačiau, nepalankiausiu atveju, ir šios procedūros veiksmų skaičius auga eksponentiškai nuo viršūnių skaičiaus. Tai bus teisinga ir tuo atveju, jei algoritmą nutrauksime, radę pirmąjį Hamiltono ciklą. Jei grafas neturi Hamiltono ciklo, tai ir šiuo atveju reikės peržiūrėti visus galimus kelius.

Keliaujančio pirklio uždavinys

Kaip buvome minėję, su Hamiltono ciklu yra susijęs keliaujančio pirklio uždavinys. Priminsime šio uždavinio formulavimą.

Turime n miestų ir žinoma atstumų tarp šių miestų matrica $C=[c_{ij}]\quad i=\overline{1,n},\quad j=\overline{1,n}$, čia c_{ij} – atstumas tarp miestų i ir j. Pirklys, išėjęs

iš 1-ojo miesto, turi apeiti visus miestus po vieną kartą ir grįžti į pirmąjį miestą. Koks turi būti pirklio maršrutas, kad jo ilgis būtų trumpiausias.

Šis uždavinys taip pat priklauso NP uždavinių klasei ir visų tikslių šio uždavinio sprendimo algoritmų sudėtingumas išreiškiamas eksponente nuo kintamųjų skaičiaus n. Todėl, praktiškai sprendžiant šį uždavinį, naudojami įvairūs euristiniai algoritmai. Dažniausiai naudojamos euristikos yra:

1) artimiausio kaimyno metodas ir
2) miestų įterpimo metodas.

Artimiausio kaimyno metodas. Šis metodas pagrįstas taisykle: "jei pirklys yra mieste k, tai pirklys toliau keliaus į artimiausią miestui k neaplankytą miestą l; aišku, - $l \neq 1$, jei yra kita galimybė". Po šio veiksmo iš matricos C pašalinama k-oji eilutė (iš miesto k pirklys daugiau neišeis) ir l-asis stulpelis (į l-ąjį miestą pirklys niekada nebeužeis).

Pavyzdys. Taikydami artimiausio kaimyno metodą, apskaičiuokime pirklio maršrutą, esant tokiai atstumų matricai:

$$C = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 10 & 7 & 4 & 3 \\ 2 & 10 & 0 & 7 & 2 & 6 \\ 3 & 7 & 7 & 0 & 5 & 4 \\ 4 & 4 & 2 & 5 & 0 & 8 \\ 5 & 3 & 6 & 4 & 8 & 0 \end{array}.$$

Iš pirmojo miesto eisime į 5-ąjį miestą, nes $c_{15} = \min\limits_{\substack{1 \le j \le 5 \\ j \neq 1}} (c_{1j} / c_{1j} \neq 0)$. Iš atstumų matricos pašalinę 1-ąją eilutę ir 5-ąjį stulpelį, gausime:

$$C = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 2 & 10 & 0 & 7 & 2 \\ 3 & 7 & 7 & 0 & 5 \\ 4 & 4 & 2 & 5 & 0 \\ 5 & 3 & 6 & 4 & 8 \end{array}.$$

Dabar, iš 5-ojo miesto eisime į 3-ąjį, nes $c_{53} = \min\limits_{j \neq 1}(c_{5j} / c_{5j} \neq 0)$. Iš perskaičiuotos atstumų matricos pašaliname 5-ąją eilutę ir 3-ąjį stulpelį. Gauname:

$$C = \begin{array}{c|ccc} & 1 & 2 & 4 \\ \hline 2 & 10 & 0 & 2 \\ 3 & 7 & 7 & 5 \\ 4 & 4 & 2 & 0 \end{array}.$$

Toliau iš 3-ojo miesto keliausime į 4-ąjį miestą, nes $c_{34} = \min_{j \neq 1}(c_{3j} / c_{3j} \neq 0)$. Pašalinus 3-ąją eilutę ir 4-ąjį stulpelį, gausime:

$$C = \begin{array}{c|cc} & 1 & 2 \\ \hline 2 & 10 & 0 \\ 4 & 4 & 2 \end{array}.$$

Tada iš 4-ojo miesto keliausime į 2-ąjį miestą, o iš 2-ojo į 1-ąjį.

Tuo būdu, artimiausio kaimyno metodu apskaičiuotas maršrutas yra: 1, 5, 3, 4, 2, 1, ir jo ilgis yra $c_{15}+c_{53}+c_{34}+c_{42}+c_{21}= 3+4+5+2+10=24$.

Įterpimo metodas. Metodo idėja yra labai paprasta. Pirklio maršrutą konstruosime nuosekliai, pradėdami nuo ciklo v_1,v_2,v_1, ir kiekviename žingsnyje šį ciklą praplėsime įterpdami po vieną naują miestą, t.y. po antrojo žingsnio turėsime ciklą, jungiantį tris miestus, po trečiojo – keturis miestus ir t.t., kol po (n-1)-ojo žingsnio turėsime ciklą, einantį per visus miestus. Praplėsdami ciklą, t.y. įterpdami naują miestą, elgsimės taip, kad praplėsto ciklo ilgio padidėjimas būtų minimalus. Be to pradinį ciklą v_1,v_2,v_1 parinksime taip, kad jo ilgis būtų mažiausias iš visų galimų to ilgio ciklų.

Formaliai aprašysime įterpimo metodą.

Pradinio ciklo parinkimas.

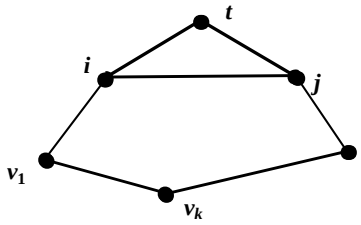
1. Apskaičiuoti $p = \min\limits_{\substack{1 \le i \le n \\ 1 \le j \le n \\ i \neq j}} (c_{ij} + c_{ji})$.

Pastaba. Jei atstumų matrica yra simetrinė, tai pakanka rasti $\min\limits_{\substack{1 \le i \le n \\ 1 \le j \le n \\ i \neq j}} c_{ij}$.

2. Tarkime, kad $p = c_{kl} + c_{lk}$. Tada pradinis ciklas yra k,l,k (žr. 2.15.9 pav.).

2.15.9 pav. Pirklio pradinis ciklas

Ciklo praplėtimas. Tarkime po (k-1)-ojo žingsnio turime ciklą $v_1,v_2,v_3,\ldots,i,j,\ \ldots,v_k,v_1$. Norėdami praplėsti ciklą, bandysime įterpti vieną iš likusių miestų tarp dviejų gretimų ciklo miestų i ir j. Iš visų galimų (i,j) porų pasirinksime tokią porą ir tokį įterpiamą miestą, kad ciklo pailgėjimas būtų minimalus, t.y. reikia apskaičiuoti $d = \min\limits_{\forall (i,j):(i,j) \in ciklui} \min\limits_{t \in V - ciklas} (c_{it} + c_{tj} - c_{ij})$ (žr. 2.15.10 pav.).



2.15.10 pav. Miesto įterpimas

Tarkime, kad mažiausia d reikšmė yra miestų (i,j) porai, o įterpiamas miestas - t. Tada gausime ciklą $v_1,v_2,\ldots, i,t,j,\ldots,v_k,v_1$.

Pavyzdys. Sukonstruokime pirklio maršrutą, esant tai pačiai atstumų matricai C (žr. artimiausio kaimyno metodą).

$$C = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 10 & 7 & 4 & 3 \\ 2 & 10 & 0 & 7 & 2 & 6 \\ 3 & 7 & 7 & 0 & 5 & 4 \\ 4 & 4 & 2 & 5 & 0 & 8 \\ 5 & 3 & 6 & 4 & 8 & 0 \end{array}.$$

Pradinis ciklas bus: $2,4,2$ (žr. 2.15.11 a) pav.). Simboliu $d_{i,l,j}$ pažymėkime ciklo pailgėjimą, kai tarp miestų i ir j įterpiamas miestas l. Tada

$$d_{2,1,4} = c_{21} + c_{14} - c_{24} = 10 + 4 - 2 = 12 \ ,$$

$$d_{2,3,4} = c_{23} + c_{34} - c_{24} = 7 + 5 - 2 = 10 \ ,$$

$$d_{2,5,4} = c_{25} + c_{54} - c_{24} = 6 + 8 - 2 = 12 \ .$$

Kadangi matrica C yra simetrinė, tai, terpiant miestą tarp 4-ojo ir 2-ojo, gausime tuos pačius atstumus. Vadinasi, 3-ąjį miestą įterpsime tarp 2-ojo ir 4-ojo miestų. Gausime ciklą $2,3,4,2$ (žr. 2.15.11 b) pav.).

Apskaičiuosime

$$d_{2,1,3} = c_{21} + c_{13} - c_{23} = 10 + 7 - 7 = 10 \ ,$$

$$d_{2,5,3} = c_{25} + c_{53} - c_{23} = 6 + 4 - 7 = 3 \ ,$$

$$d_{3,1,4} = c_{31} + c_{14} - c_{34} = 7 + 4 - 5 = 6 \ ,$$

$$d_{3,5,4} = c_{35} + c_{54} - c_{34} = 4 + 8 - 5 = 7 \ ,$$

$$d_{4,1,2} = c_{41} + c_{12} - c_{42} = 4 + 10 - 2 = 12 \ ,$$

$$d_{4,5,2} = c_{45} + c_{52} - c_{42} = 8 + 6 - 2 = 12 \ .$$

Iš šių skaičių mažiausias yra 3. Vadinasi, pailgėjęs ciklas yra: $2,5,3,4,2$ (žr. 2.15.11 c) pav.).

Apskaičiuosime

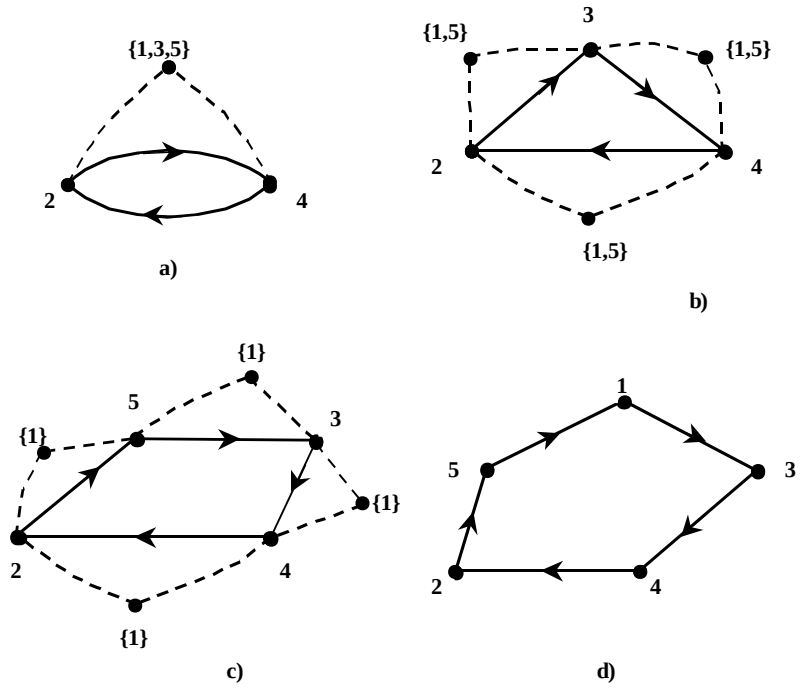
$$d_{2,1,5} = c_{21} + c_{15} - c_{25} = 10 + 3 - 6 = 7,$$

$$d_{5,1,3} = c_{51} + c_{13} - c_{53} = 3 + 7 - 4 = 6,$$

$$d_{3,1,4} = c_{31} + c_{14} - c_{34} = 7 + 4 - 5 = 6,$$

$$d_{4,1,2} = c_{41} + c_{12} - c_{42} = 4 + 10 - 2 = 12.$$

Yra du lygūs mažiausi skaičiai. Pasirenkame pirmąjį $d_{5,1,3}$. Tada pirklio maršrutas yra: $2,5,1,3,4,2$ ir jo ilgis yra $c_{25} + c_{51} + c_{13} + c_{34} + c_{42} = 6 + 3 + 7 + 5 + 2 = 23$ (žr. 2.15.11 d) pav.). Šis pirklio maršrutas yra trumpesnis už maršrutą, gautą artimiausio kaimyno metodu. Tas paaiškinama tuo, kad įterpimo metode buvo atliktas didesnis perrinkimas.



2.15.11 pav. Pirklio maršruto konstravimas įterpimo metodu