

DUOMENŲ STRUKTŪROS IR ALGORITMAI

MARIUS GŽEGOŽEVSKIS

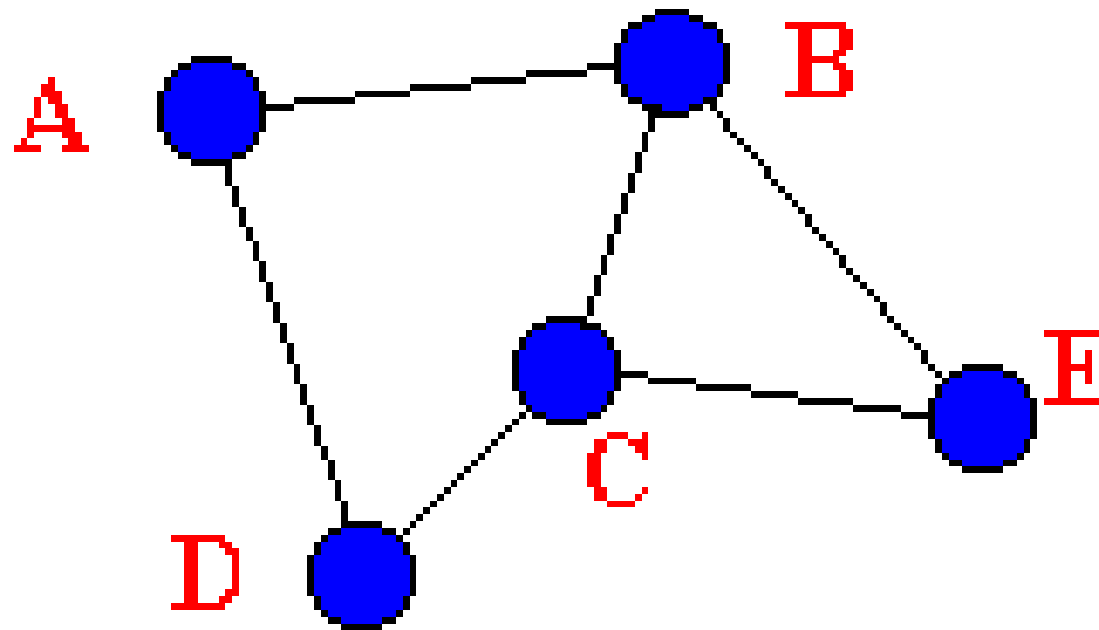
TURINYS

- ✓ GRAFAI
- ✓ GRAFAI KAIP ABSTRAKTŪS DUOMENŲ TIPAI (**ADT**)
- ✓ GRAFO REALIZACIJA
- ✓ DFS (PAIEŠKA Į GYLĮ)
- ✓ BFS (PAIEŠKA Į PLOTĮ).

GRAFAS

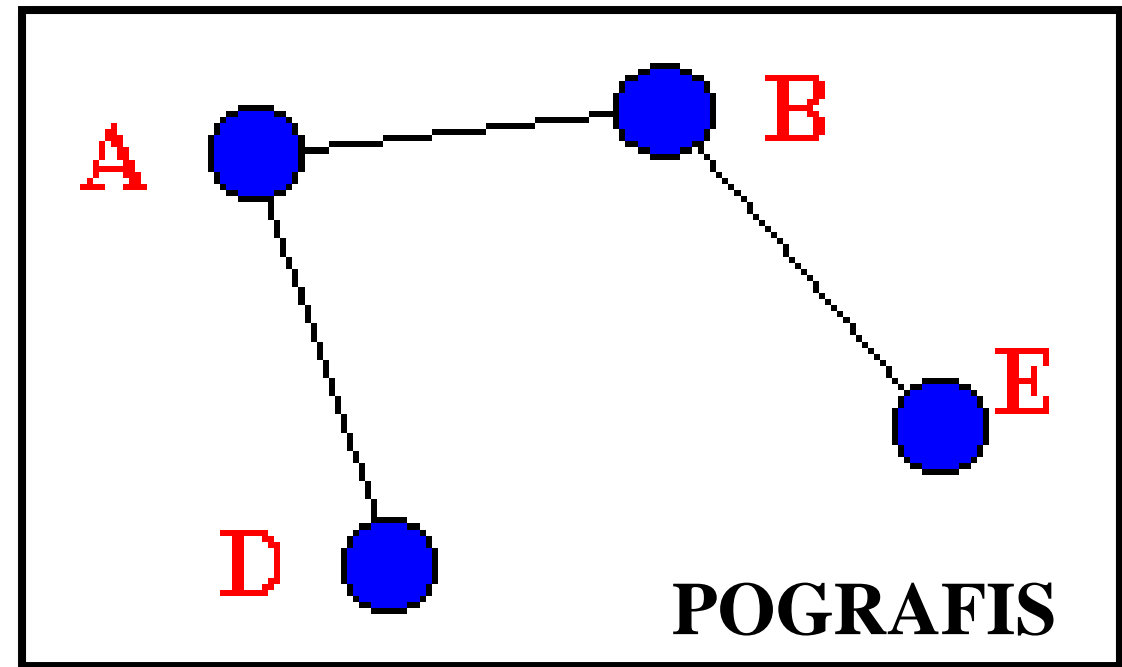
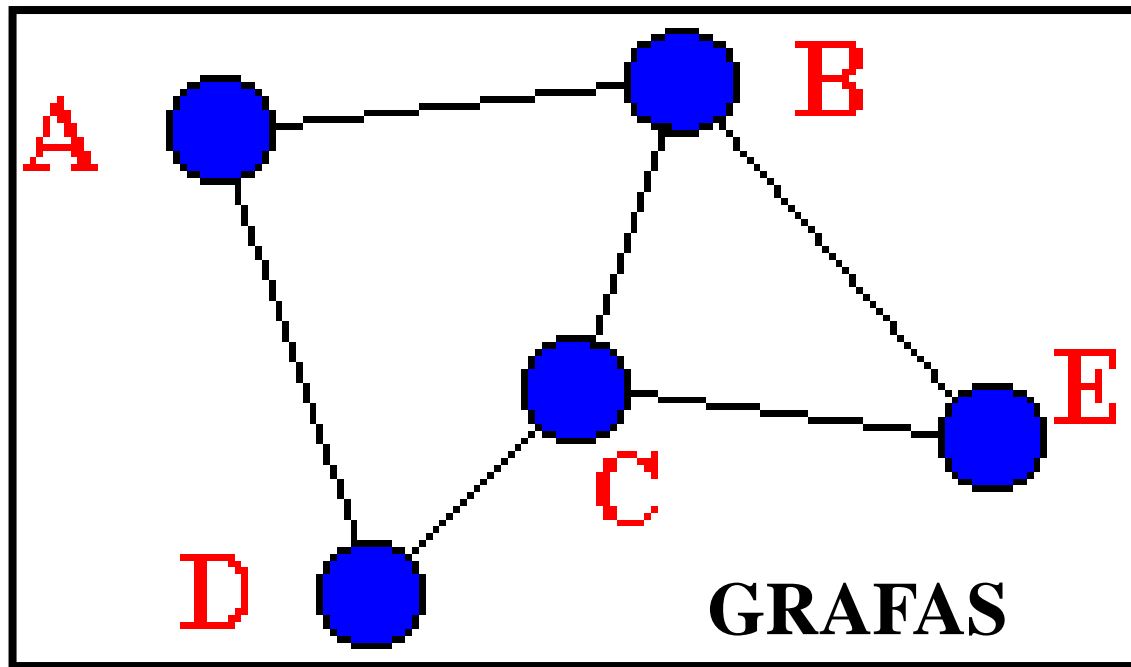
Grafas – aibių pora (V, L) [V – viršūnių aibė, L – briaunų aibė]

Briauna – atkarpa, jungianti dvi grafo viršūnes.



POGRAFIS

Pografis (*subgraph*) – poaibis grafo briaunų (*edge*) bei jų viršūnių (*vertex*).



GRAFŲ SAVYBĖS

Dvi viršūnės yra gretimos (*adjacent*), jei jos sujungtos briauna. Viršūnės V_i ir V_j yra kaimyninės, jeigu egzistuoja $B_k=(V_i, V_j)$. Pvz.: **A** kaimyninės viršūnės yra **B** ir **D**.

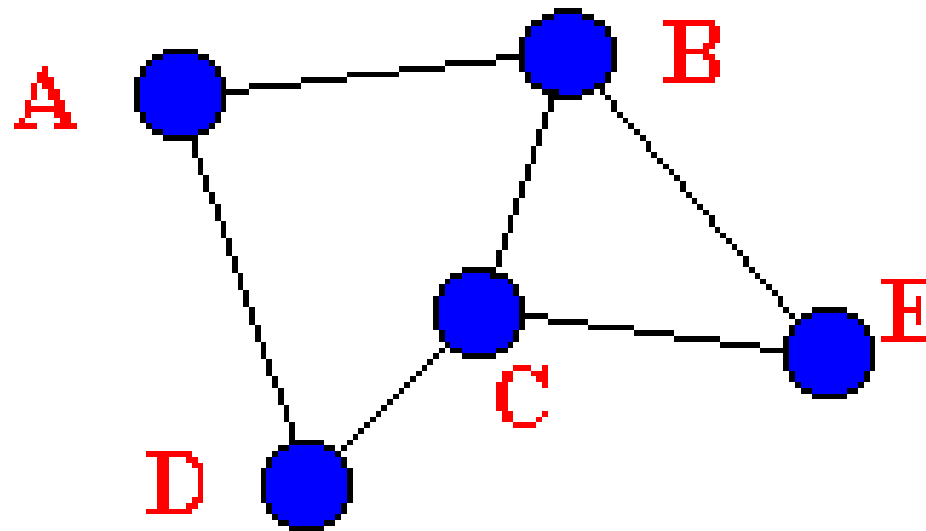
Plokščias grafas – grafas, kuri galima nupiešti plokštumoje (bent vienu būdu) taip, kad nė viena pora briaunų nesikirstų.

Kelias (*path*) tarp viršūnių – briaunų seka, prasidedanti vienoje viršūnėje ir besibaigianti kitoje viršūnėje.

GRAFŲ SAVYBĖS

Paprastas kelias (*simple path*) – kelias, per kiekvieną jam priklausančią viršūnę einantis tik po vieną kartą. Pvz.: Kelias – **ADCBC**E nėra paprastas kelias, nes per viršūnę C eina du kartus.

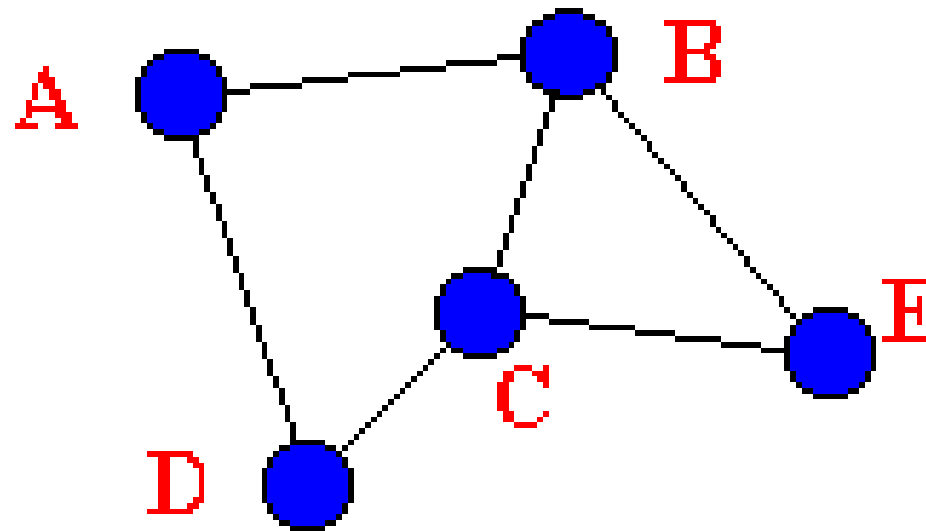
Ciklas (*cycle*) – paprastas kelias, kuris prasideda ir baigiasi toje pačioje viršūnėje. Pvz.: **ABCDA**.



JUNGUS GRAFAS

Jungus grafas (*connected*) – jei egzistuoja kelias tarp bet kurių viršūnių porų.

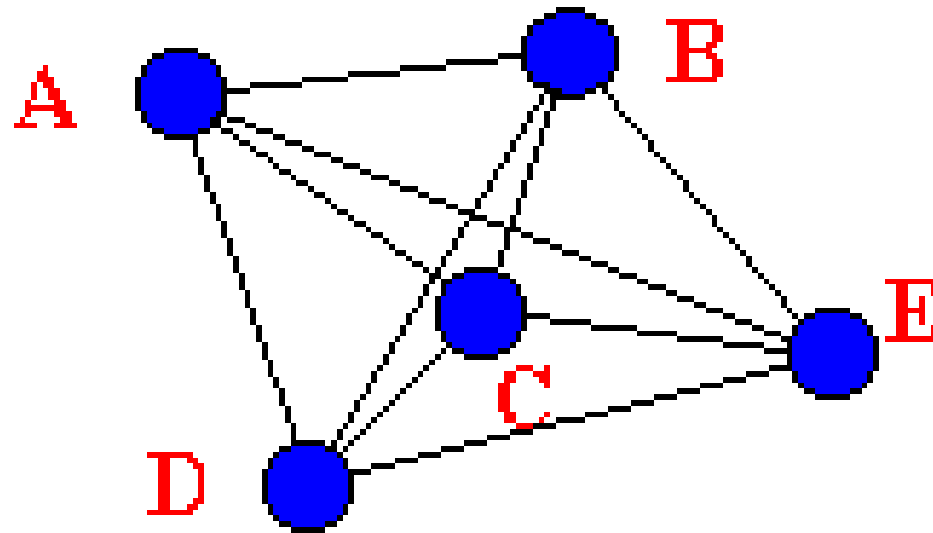
" V_i, V_j : \$kelias $V_i \rightarrow V_j$.



PILNAS GRAFAS

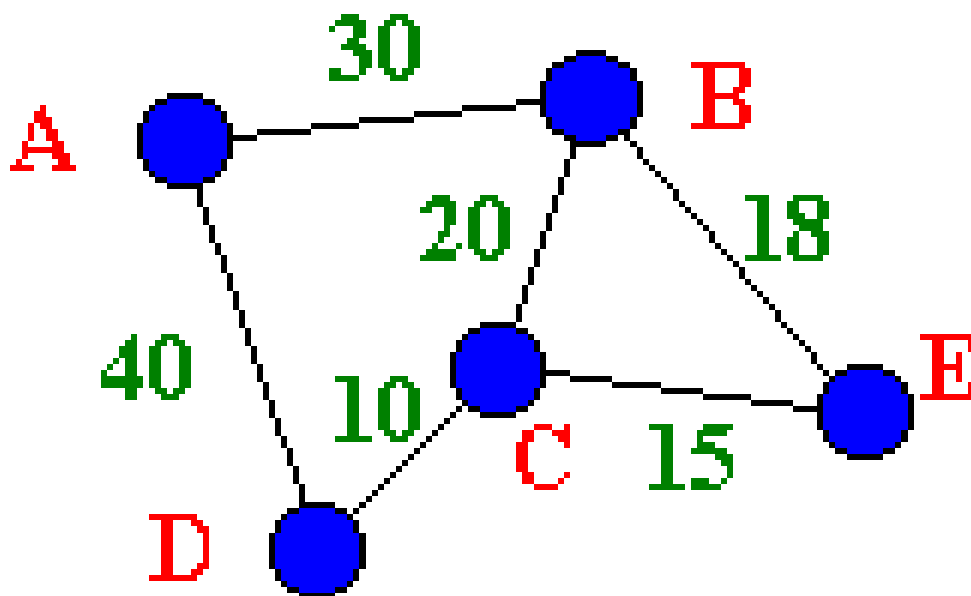
Pilnas grafas (*complete*) – jei yra briauna tarp kiekvienos viršūnių poros. Aišku, kad pilnas grafas taip pat yra ir jungus, tačiau jungus grafas nebūtinai yra pilnas.

" $V_i, V_j: E = (V_i, V_j)$



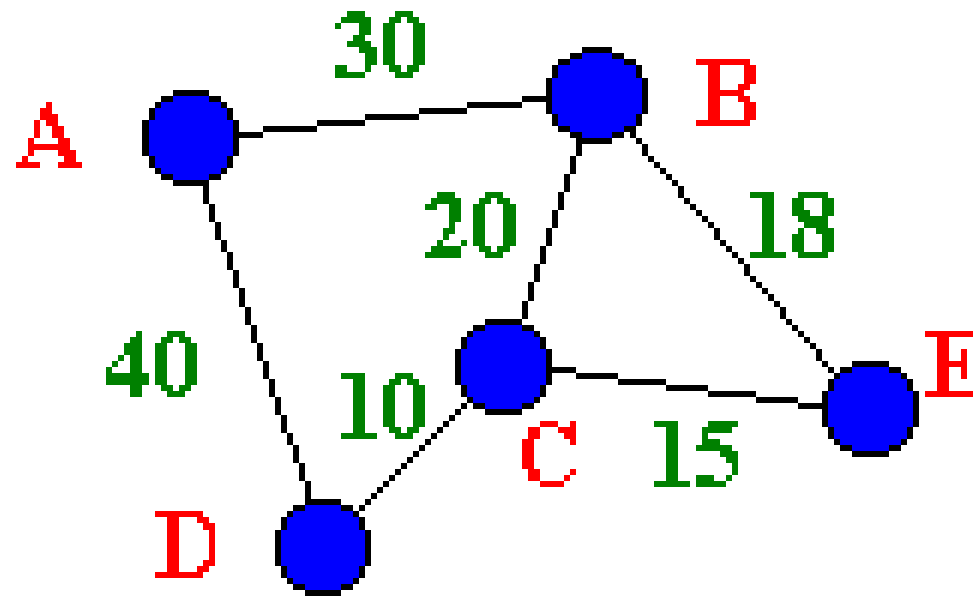
GRAFAS SU SVORIAIS

Grafas su svoriais (*weighted*) – grafas, kurio bet kokia briauna turi skaitinę reikšmę.



NEORIENTUOTAS GRAFAS

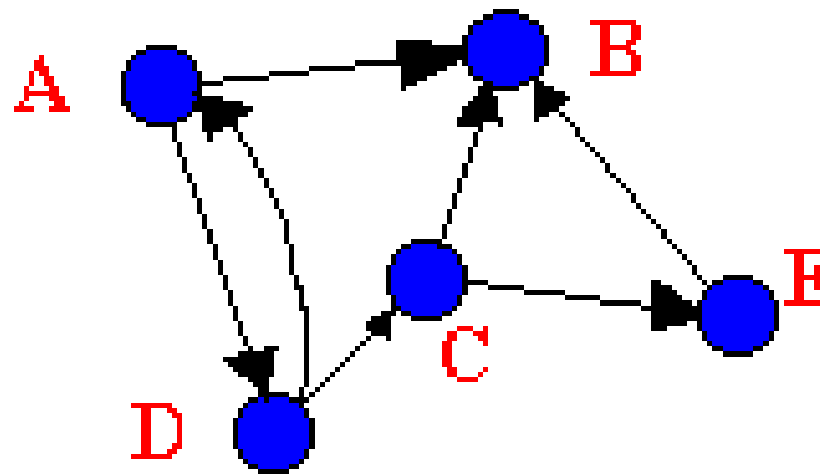
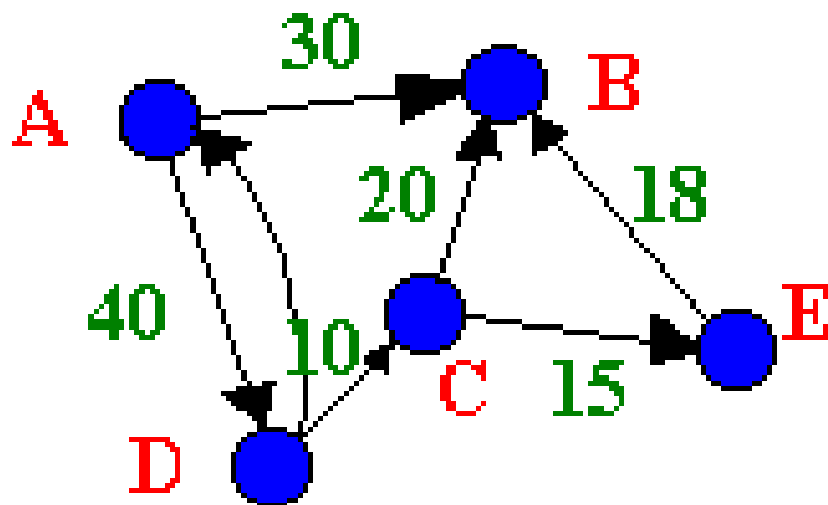
Iki šiol buvo kalbama apie grafus, kurių briaunos neturi krypties, t.y. briauna galima keliauti bet kuria kryptimi. Tai reiškia, kad tarp dviejų viršūnių gali būti tik viena briauna, jei grafas yra be svorių.



ORIENTUOTAS GRAFAS

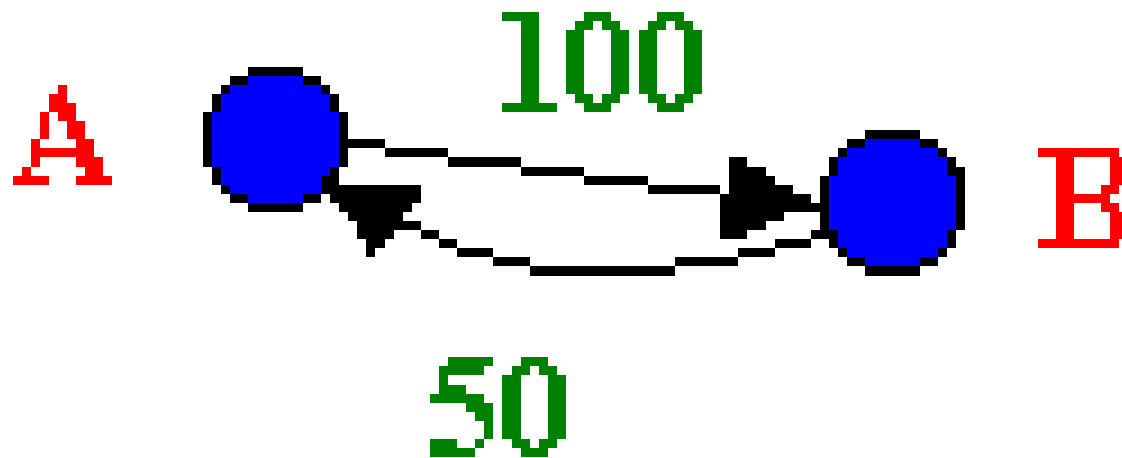
Orientuotas (kryptinis) **grafas** (*directed*) – grafas su lankais, t.y., visos briaunos turinčios kryptį.

Lankas – orientuoto grafo briauna, turinti kryptį.



PAVYZDYS

Knygų skolinimasis: **A** iš **B** pasiskolino 100 knygų, o **B** iš **A** pasiskolino 50 knygų.



Grafai kaip abstraktūs duomenų tipai (ADT)

Grafas gali būti traktuojamas kaip abstraktus duomenų tipas. Įterpimo ir ištrynimo operacijos šiek tiek skiriasi nuo kitų ADT. ADT grafus galima apibrėžti tiek su reikšmėmis, tiek ir be jų. Pagrindinės operacijos su grafais kaip ADT:

Grafai kaip abstraktūs duomenų tipai (ADT)

1. Sukurti tuščią grafą.
2. Įdėti/išmesti viršūnę.
3. Įdėti/išmesti briauną tarp viršūnių V_1 ir V_2 .
4. Sužinoti (rasti), ar yra kelias tarp viršūnių V_1 ir V_2 .
5. Sužinoti (V_1, V_2) svorį.
6. Pakeisti (V_1, V_2) svorį.

GRAFŲ REALIZAVIMAS

Du pagrindiniai grafų realizavimo būdai:

1. kaimynystės matrica;
2. kaimynystės sąrašai.

Abiem atvejais patogiausia įsivaizduoti, kad viršūnės numeruojamos 1, 2 ir taip toliau iki N .

KAIMYNYSTĖS MATRICA

Kaimynystės matrica grafui be svorių su N viršūnių yra N iš N loginis masyvas A toks, kad $A[i, j]$ yra teisingas tada ir tik tada, kai egzistuoja briauna iš viršūnės ‘ i ’ į viršūnę ‘ j ’.

Pagal susitarimą $A[i, i]$ yra klaidingas. Įsidėmėtina, kad kaimynystės matrica **neorientuotam grafui** yra simetriška, tai yra $A[i, j] = A[j, i]$.

KAIMYNYSTĖS MATRICA

	A	B	C	D
A	F	T	F	F
B	T	F	T	T
C	F	T	F	T
D	F	T	T	F

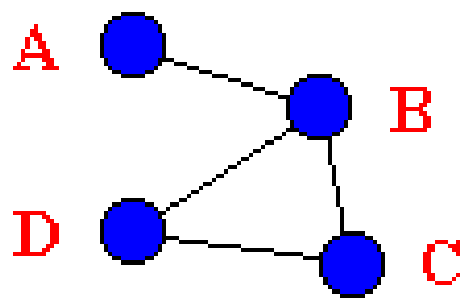


Diagram illustrating an undirected graph with 4 nodes (A, B, C, D) and 5 edges. The edges are: (A, B), (A, D), (B, C), (B, D), and (C, D). The nodes are labeled A, B, C, and D in red text.

**NEORIENTUOTAS
GRAFAS** (SIMETRIŠKA
MATRICA)

	A	B	C	D
A	F	F	F	F
B	T	F	T	F
C	F	F	F	T
D	F	T	F	F

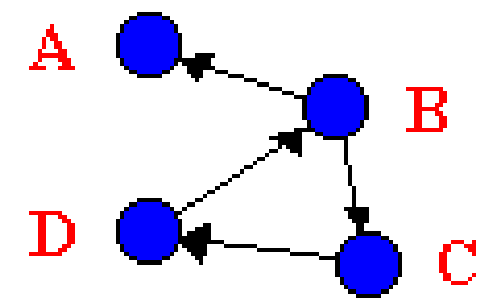


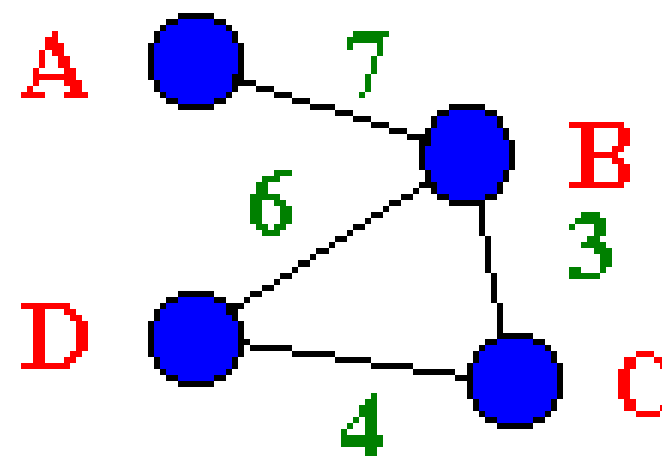
Diagram illustrating a directed graph with 4 nodes (A, B, C, D) and 5 edges. The edges are: (B, A), (B, C), (C, B), (C, D), and (D, A). The nodes are labeled A, B, C, and D in red text.

ORIENTUOTAS GRAFAS
(DAŽNIAUSIAI
NESIMETRIŠKA MATRICA)

GRAFO SU SVORIAIS MATRICA

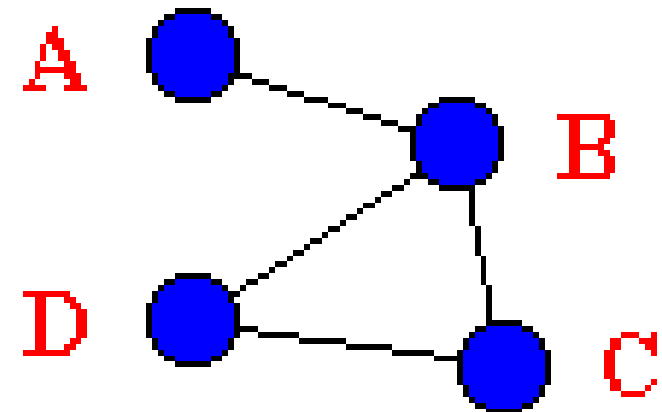
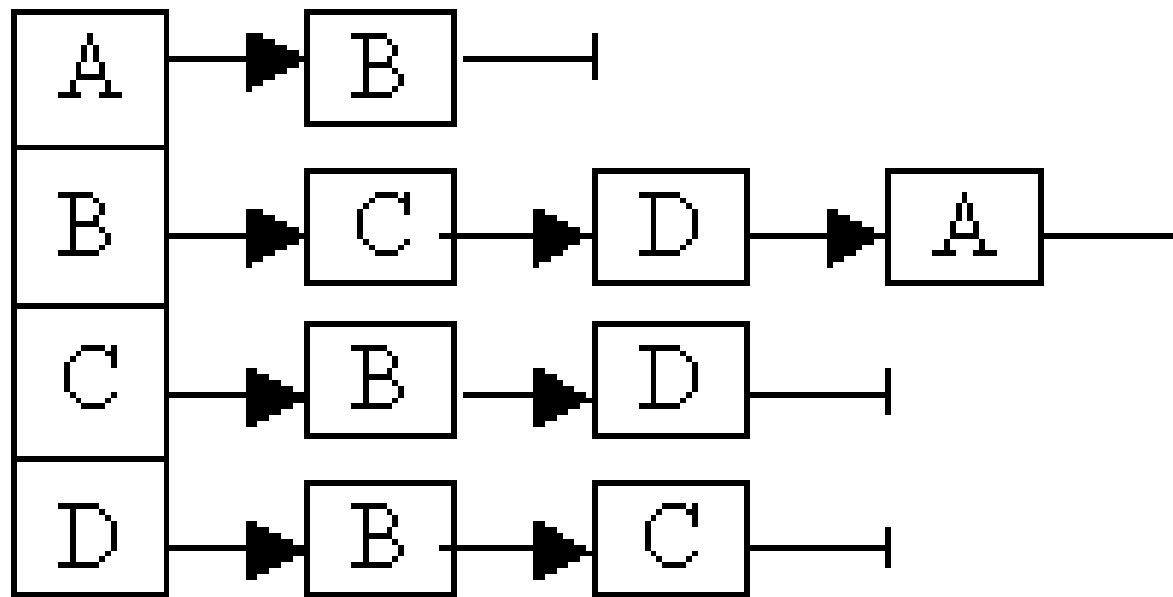
Kai turim grafą su svoriais, yra patogiu, kad $A[i, j]$ būtų briaunos iš viršūnės 'i' į viršūnę 'j' svoris. Tada $A[i, j]$ žymėti ∞ , kai nėra briaunos iš viršūnės 'i' į viršūnę 'j'. Be to, įstrižainės $A[i, i]$ reikšmės lygios 0.

	A	B	C	D
A	0	7	∞	∞
B	7	0	3	6
C	∞	3	0	4
D	∞	6	4	0



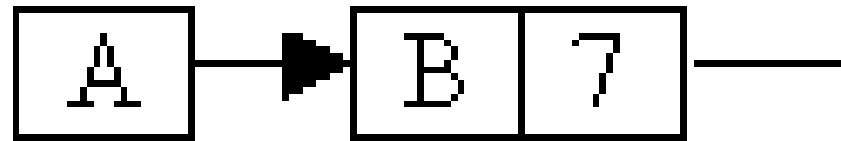
KAIMYNYSTĖS SĄRAŠAS

Kaimynystės sąrašas grafo iš N viršūnių, kurios numeruojamos $1, 2, \dots, N$, susideda iš N sujungtų sąrašų.



KAIMYNYSTĖS SĄRAŠAS GRAFUI SU SVORIAIS

Jei grafas yra su svoriais, tai jie saugomi kartu su viršūnė.



Dvi dažniausiai naudojamos grafų operacijos yra:

1. Duotos dvi viršūnės 'i' ir 'j'; rasti, ar yra briauna iš 'i' į 'j'.
2. Rasti visas viršūnes, kurios yra kaimynės duotajai viršūnei V_i

Jei grafas yra netoli pilno grafo, tai masyvas yra efektyvesnis už sąrašą. Jei briaunų mažai, tai lieka daug nepanaudotos vietos matricoje, kas yra minusas taupant, tuomet geriau sąrašas.

GRAFO APĖJIMAS (angl. Traversal)

Egzistuoja du apėjimo algoritmai: į gylį (**DFS** – *Depth First Search*) ir į plotį (**BFS** – *Breadth First Search*).

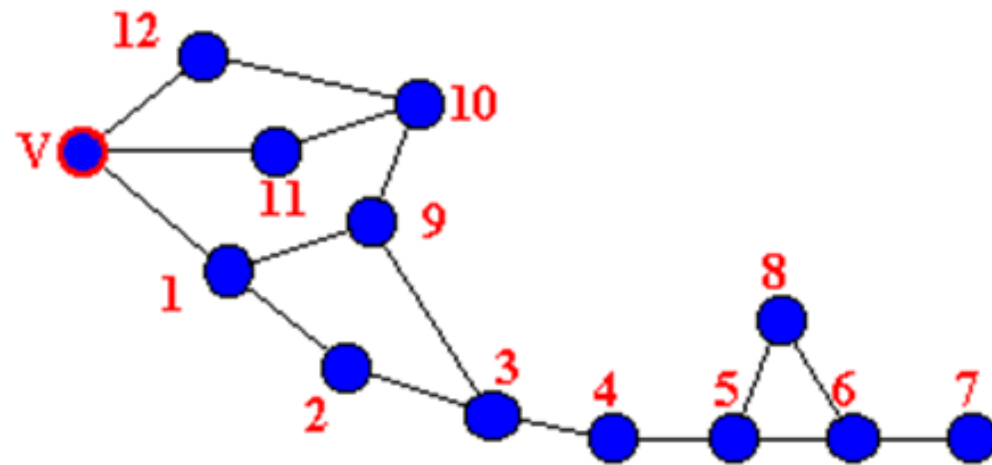
DFS (paieška į gylį). Iš duotos viršūnės ‘v’ einama gilyn ir gilyn, kol pasiekiamas viršūnė, iš kurios giliau eiti jau nebeįmanoma.

Tai yra, aplankius viršūnę ‘v’, DFS algoritmas aplanko dar neaplankytą kaimyninę viršūnę. Jis tęsiasi iš ‘u’ tiek toli, kiek galima iki grįžimo į ‘v’, kad aplankytų kitą neaplankytą viršūnę kaimyninę ‘v’. Galima ir rekursyvi DFS realizacija.

DFS pseudokodas:

```
{S – stekas (stack)}  
Push(S, v)  
MarkV(v) {pažymi aplankytą viršūnę}  
  
While (not IsEmpty(S)) do  
  If (neegzistuoja neaplankyta viršūnė kaimynė steko viršuje esančiai viršūnei) then Pop(S)  
    else  
      begin  
        imame 'u' – S viršūnės kaimynė ('u' ∈ V)  
        Push(S, u)  
        MarkV(u)  
      end  
    end  
end
```

Grafo apėjimo DFS būdu (1 iš 6)



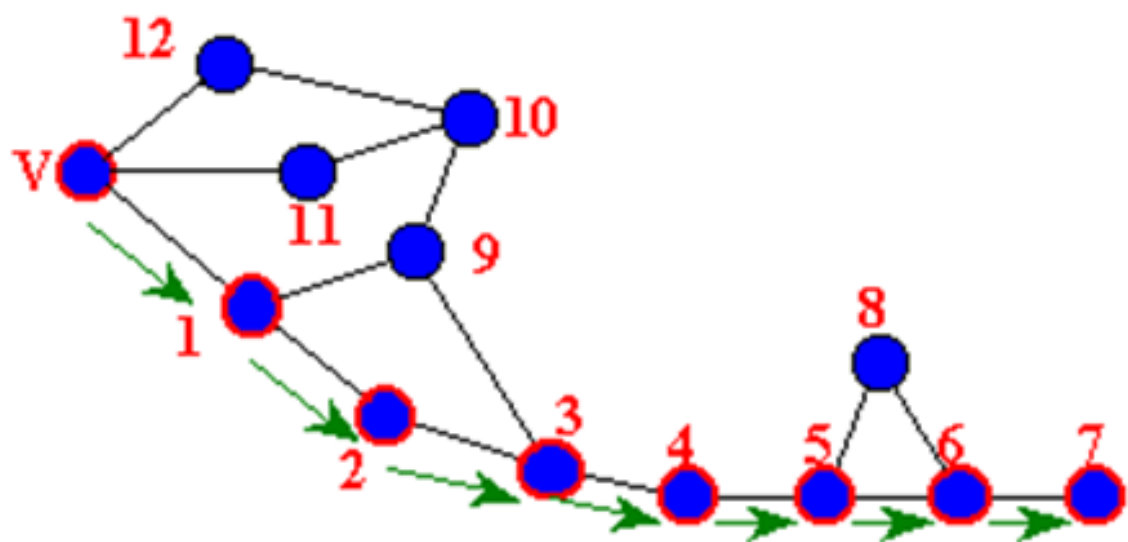
Grafas bus pradedamas apeidinėti nuo viršūnės 'V'.

Į steką įdedame viršūnę ,nuo kurios pradėsime apėjimą į gylį:

Viršūnė: 'V'

Stekas: 'V'

Grafo apėjimo DFS būdu (2 iš 6)



Maksimalus ėjimas į gylį nuo viršūnės V:

Viršūnės

Stekas

1

V 1

2

V 1 2

3

V 1 2 3

4

V 1 2 3 4

5

V 1 2 3 4 5

6

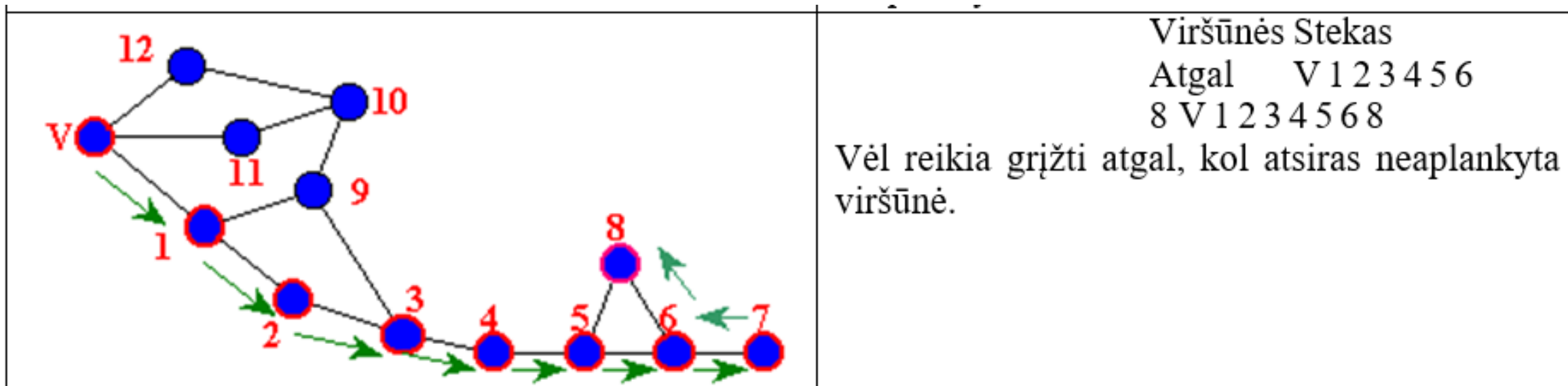
V 1 2 3 4 5 6

7

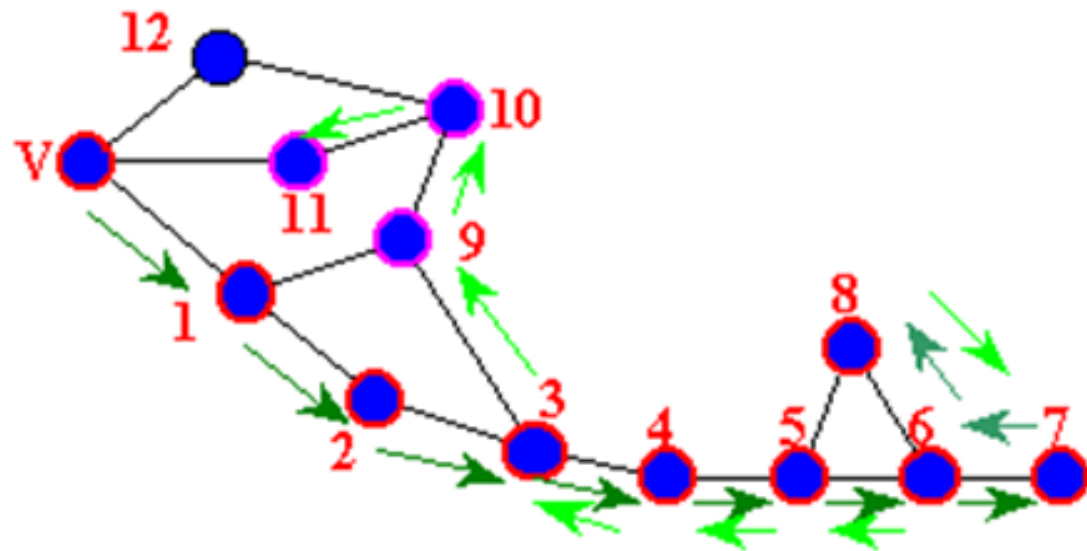
V 1 2 3 4 5 6 7

Dabar reikia grįžti atgal, kol vėl atsirastų neaplankyta viršūnė.

Grafo apėjimo DFS būdu (3 iš 6)



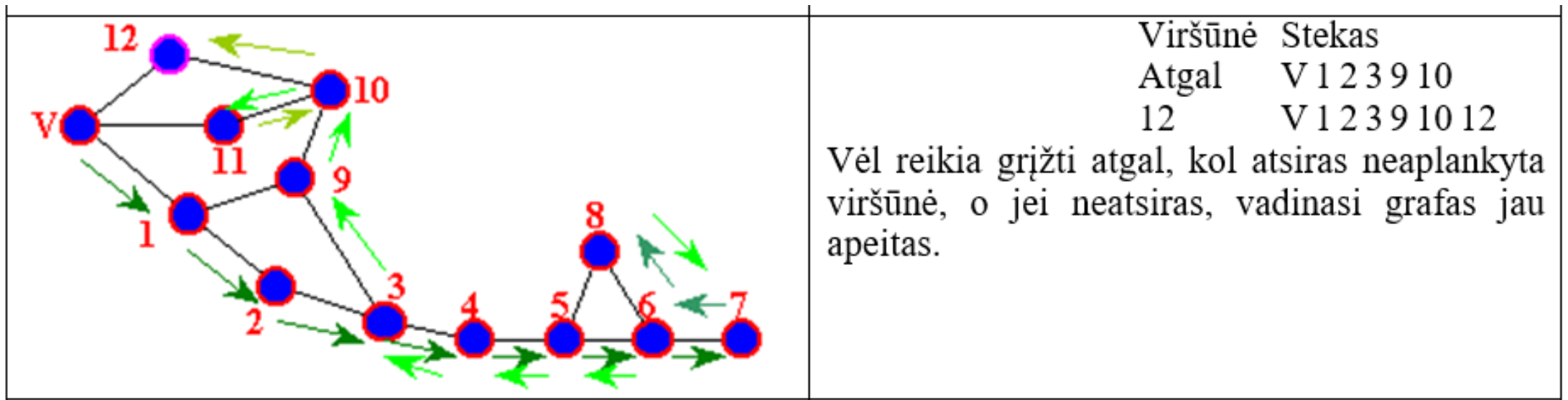
Grafo apėjimo DFS būdu (4 iš 6)



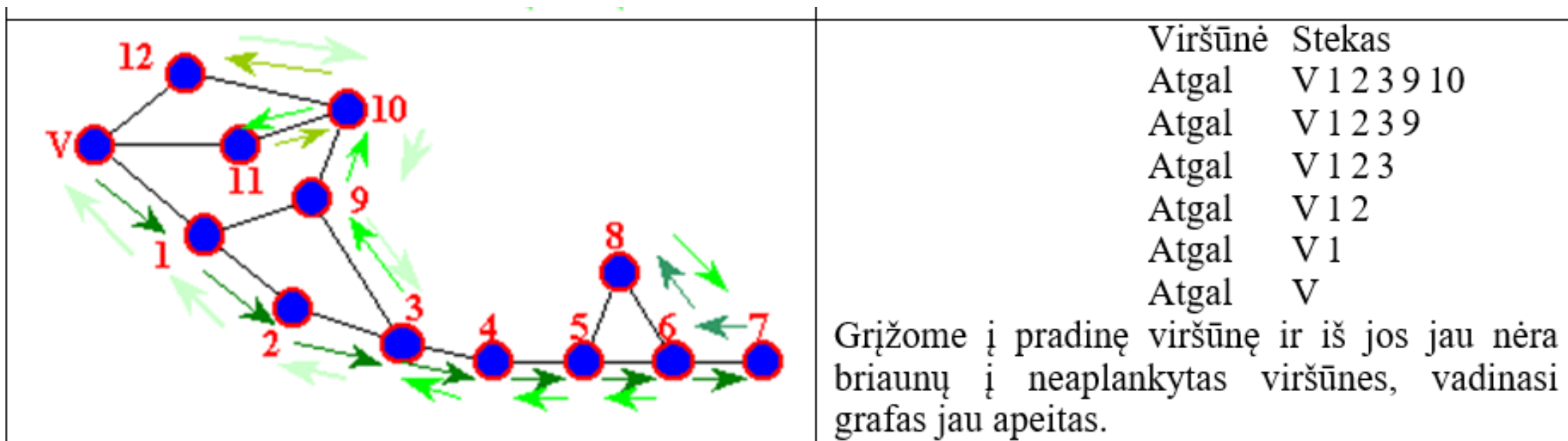
Viršūnė	Stekas
Atgal	V 1 2 3 4 5 6
Atgal	V 1 2 3 4 5
Atgal	V 1 2 3 4
Atgal	V 1 2 3
9	V 1 2 3 9
10	V 1 2 3 9 10
11	V 1 2 3 9 10 11

Vėl reikia grįžti atgal, kol atsiras neaplankyta viršūnė.

Grafo apėjimo DFS būdu (5 iš 6)



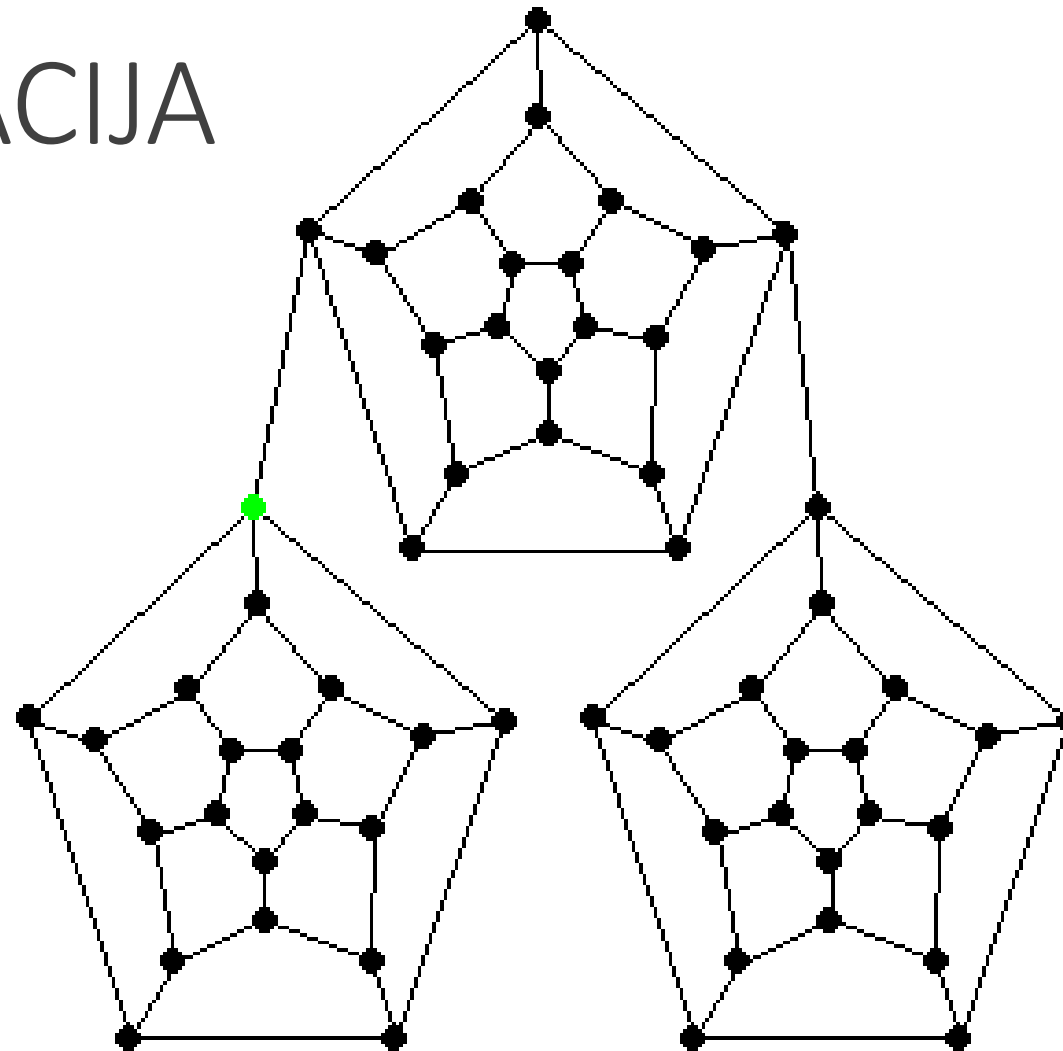
Grafo apėjimo DFS būdu (6 iš 6)



DFS APIBENDRINIMAS

DFS paieškos algoritmas tiksliai nenusako tvarkos, kuria turi būti aplankomos kaimyninės viršūnės. Viena galimybė yra aplankyti viršūnes, kaimynines 'v', abėcėlės arba numerių didėjimo tvarka.

DFS ANIMACIJA



www.combinatorica.com

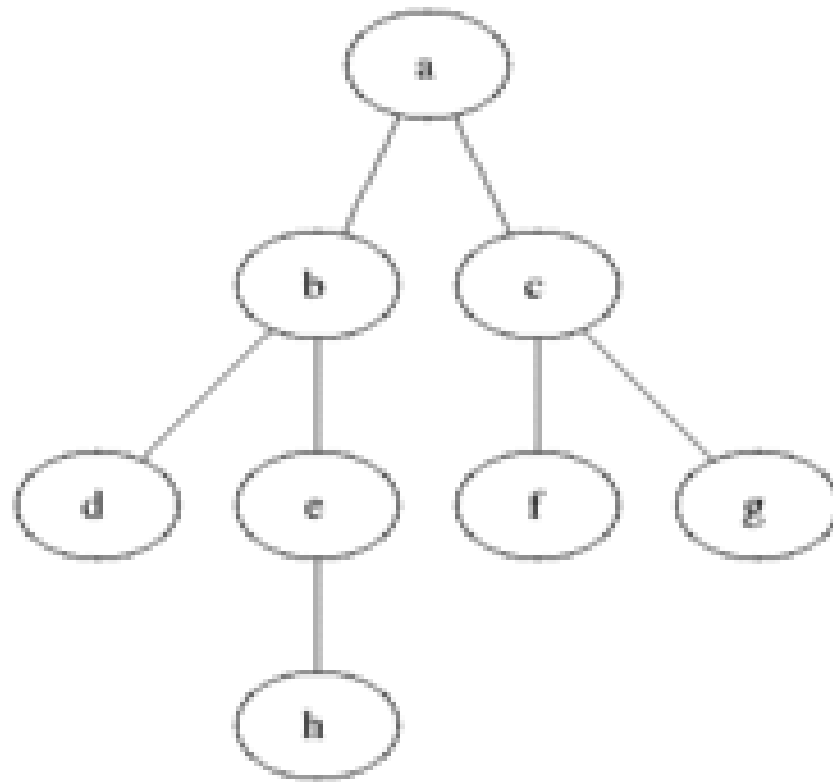
BFS (paieška į plotį)

BFS (paieška į plotį). Aplankius viršūnę v , BFS aplanko visas viršūnes kaimyninės ' v '. Tokiu būdu apėjimas neprasidės iš jokios kitos viršūnės kaimyninės ' v ', kol nėra aplankytos visos galimos kaimyninės viršūnės. Pastebėkime, kad BFS atveju nėra rekursyvios realizacijos.

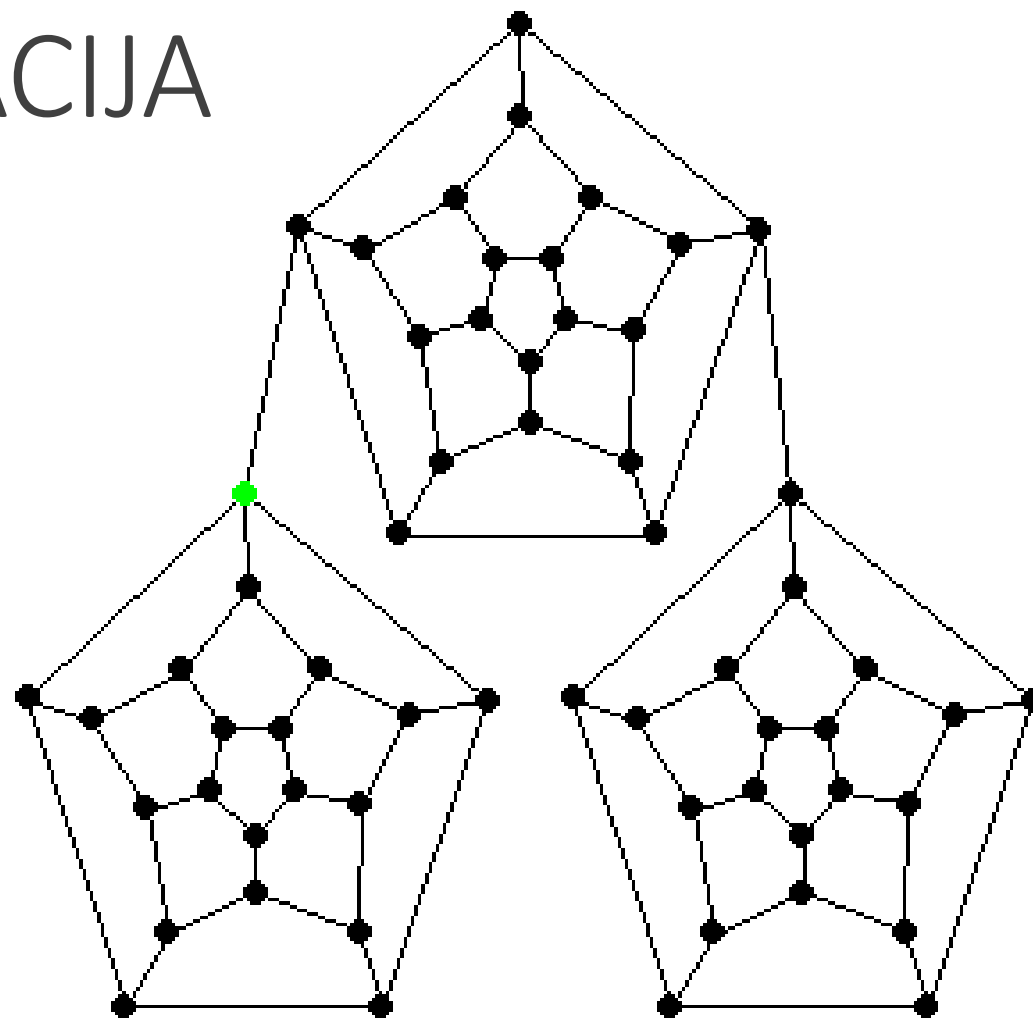
BFS PSEUDOKODAS

```
{Q – eilė}
Add(Q, V) {įdedame viršūnę V į eilę}
MarkV(V) {pažymi aplankytą viršūnę}
While (not IsEmpty(Q)) do
begin
    w := Get(Q) (Nuskaitomas ir išmetamas pirmas eilės elementas)
    for  $\forall u$  kaimynei w do
    begin
        MarkV(u)
        Add(Q, u)
    End
end
```

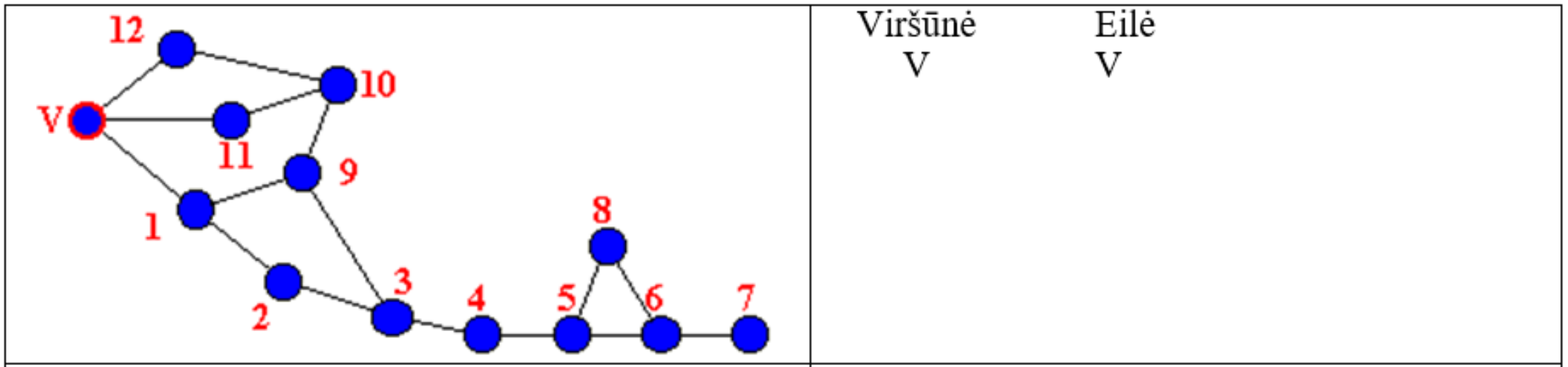

BFS ANIMACIJA



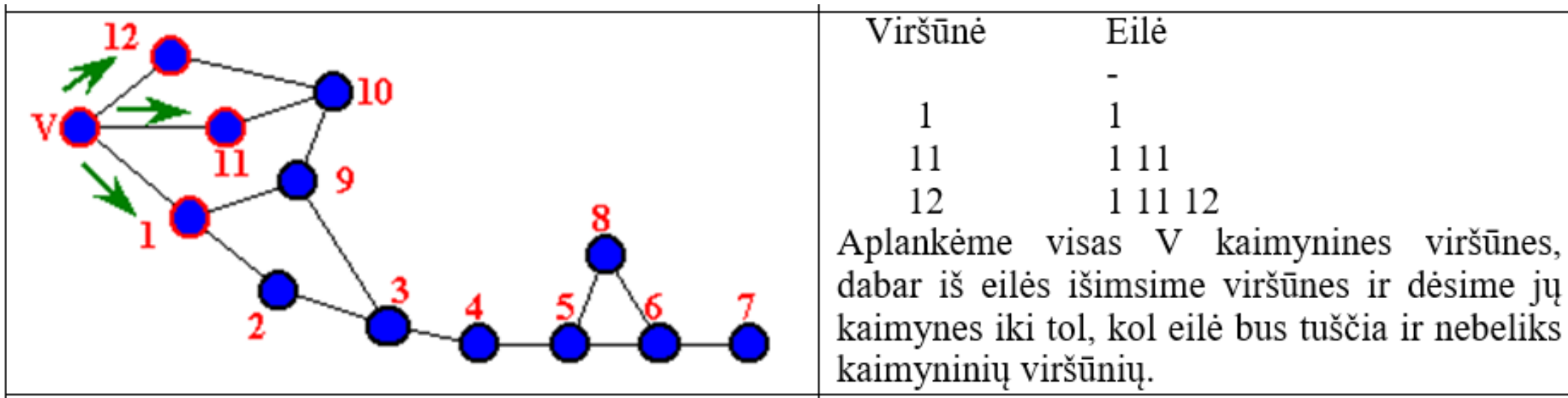
BFS ANIMACIJA

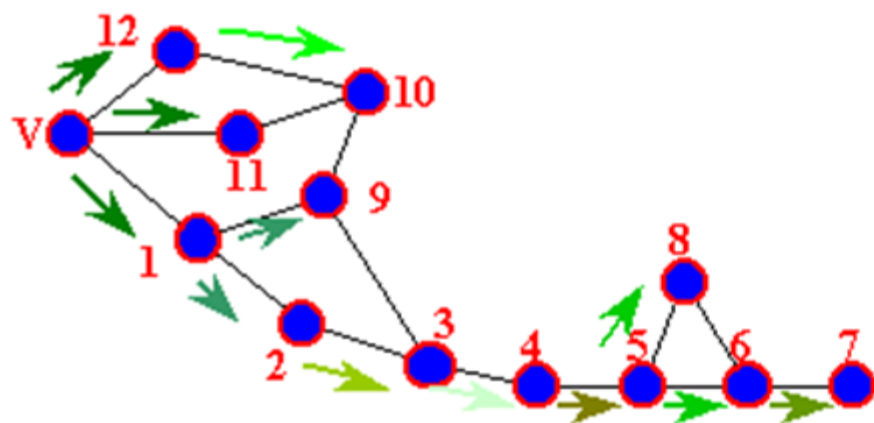


Grafo apējimo BFS būdu (1 iš 3)



Grafo apėjimo BFS būdu (2 iš 3)





Viršūnė

Eilė

	11 12
2	11 12 2
9	11 12 2 9
	12 2 9
10	12 2 9 10
	2 9 10
	9 10
3	9 10 3
	10 3
	3
	-
4	4
	-
5	5
	-
6	6
8	6 8
	8
7	8 7
	7
	-

Kaimyninių viršūnių nebeliko ir eilė tuščia, vadinasi grafas yra visas apeitas.