DUOMENŲ STRUKTŪROS IR ALGORITMAI

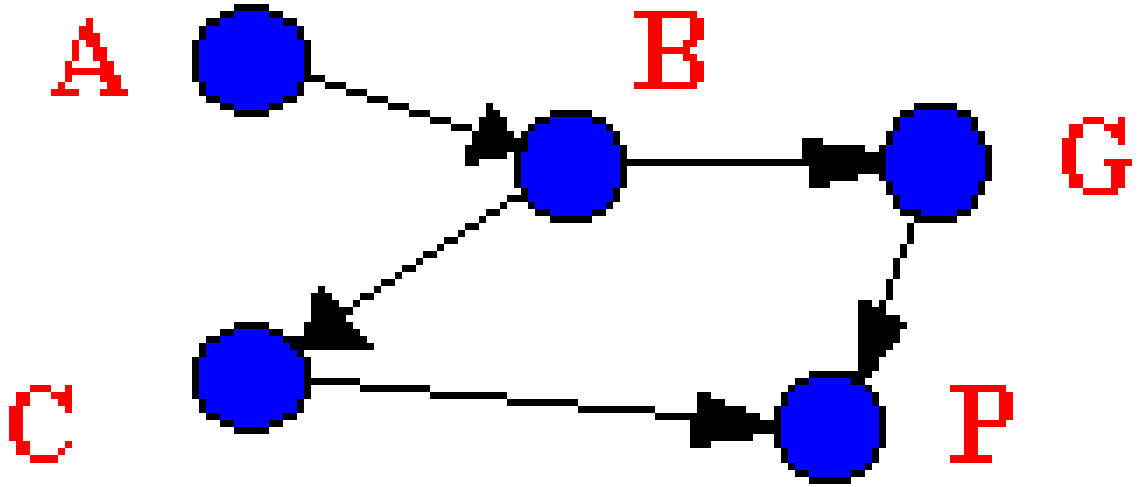
MARIUS GŽEGOŽEVSKIS

TURINYS

- ✓GRAFAI
- ✓TOPOLOGINIS RIKIAVIMAS
- ✓APRĖPTIES MEDŽIAI
- ✓DFS APRĖPTIES MEDIS
- ✓BFS APRĖPTIES MEDIS

Topologinis rikiavimas (*Topological Sorting*)

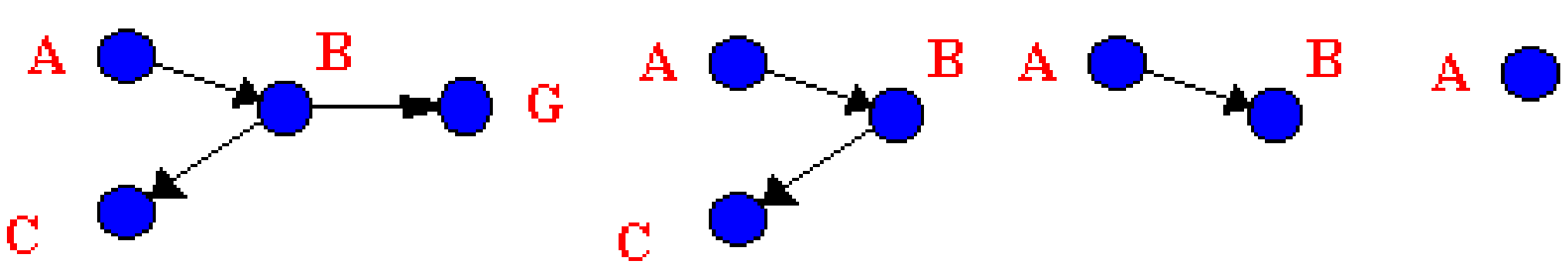
Taikomas orientuotam grafui be ciklų. Pvz.: briaunos išreiškia, koks dalykas prieš kokį dalyką turi būti išklausytas. Prieš dalyką **G** reikia išklausyti **B**. Du variantai: **ABCGP ABGCP**.



Kryptinis grafas be ciklų

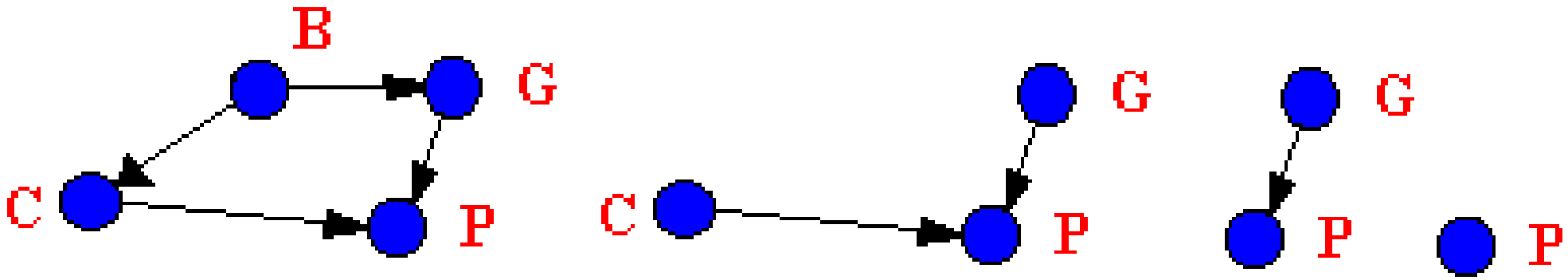
Paprastas algoritmas (*pa*):

1. Imti viršūnę iš kurios nėra išeinančių briaunų.
2. Įdėti ją į sąrašo pradžią ir iš grafo ją išmesti.



Topologinis rikiavimas (*pa*)

Modifikuotas paprastas algoritmas (*pam*):



Topologinis rūšiavimas (*pam*)

Šis būdas nėra labai efektyvus, jei bus naudojami kaimynystės sąrašai.

APRĖPTIES MEDIS (angl. Spanning tree)

Medis – jungus neorientuotas grafas be ciklų.

Grafo G ***aprėpties medis*** – grafo G pografis G’, turintis visas grafo G viršūnes ir pakankamai briaunų medžiui. (Grafui gali egzistuoti *keli* aprėpties medžiai.)

Jei jungiame neorientuotame grafe *išnaikinsime ciklus*, gausime aprėpties medį.

Neorientuoto grafo *savybės*:

Jungus neorientuotas grafas su N viršūnių turi ne mažiau kaip N-1 briauną.

- ✓Reikia prisiminti, kad jungiame grafe egzistuoja kelias tarp bet kurių dviejų viršūnių.
- ✓Tarkime, pasirenkame viršūnę ir sujungiame ją briauna su kita viršūne: 2 viršūnės ir 1 briauna; jei pridedama dar vieną viršūnę, būtina ir papildoma briauna jai prijungti prie bet kurios kitos grafo viršūnės.

Neorientuoto grafo *savybės*:

Jungus neorientuotas grafas su N viršūnių ir N-1 briauna negali turėti ciklų.

- ✓Pagal ankstesnę savybę orientuotas grafas su N viršūnių privalo turėti ne mažiau kaip N-1 briauną, kad būtų jungus.
- ✓Jei jungus grafas turi ciklą, galima pašalinti bet kurią briauną iš ciklo ir grafas liks jungus.
- ✓Taigi, jei grafas su N viršūnių ir N-1 briauna turės ciklą, galima bus pašalinti vieną briauną ir gautas grafas su N-2 briauna turi likti jungus, kas prieštarauja ankstesnei savybei.

Neorientuoto grafo *savybės*:

Jungus neorientuotas grafas su N viršūnių ir daugiau kaip N-1 briauna turi bent vieną ciklą.

- ✓Paimkime jungų neorientuotą grafą su N viršūnių ir N-1 briauna. Pasirinkime bet kokią porą viršūnių X ir Y ir pridėkime naują briauną B, jungiančią viršūnes X ir Y.
- ✓Kadangi pradinis grafas buvo jungus, tai jame egzistavo kelias iš viršūnės X į viršūnę Y.
- ✓Pridėjus prie šio kelio naują briauną B, gaunamas ciklas, t.y. kelią iš viršūnės X į ją pačią.

Neorientuoto grafo *savybės*:

- ✓Taigi patikrinti, ar jungus neorientuotas grafas turi ciklų, galima paprasčiausiai suskaičiuojant jo viršūnes ir briaunas. Iš to išplaukia, kad grafo G su N viršūnių aprėpties medis turi N-1 briauną.
- ✓Du aprėpties medžio konstravimo algoritmai, remiasi anksčiau nagrinėtomis medžio apėjimo strategijomis: paieškos į gylį (DFS) ir paieškos į plotį (BFS).
- ✓Bendru atveju šie algoritmai sukonstruos skirtingus aprėpties medžius, kuriuos toliau sąlyginai vadinsime **DFS** ir **BFS aprėpties medžiais** atitinkamai.

DFS aprėpties medis

- ✓Vienas iš būdų sukonstruoti aprėpties medį jungiam neorientuotam grafui yra apeiti grafo viršūnes, naudojant paieškos į gylį algoritmą.
- ✓Apėjimo metu žymimos briaunos, kuriomis einama.
- ✓Baigus apėjimą, pažymėtos briaunos sudarys **paieškos į gylį aprėpties medį** (*DFS spanning tree*).

DFS aprėpties medis

Tereikia nežymiai modifikuoti paieškos į gylį algoritmą (iteracinį ar rekursinį), kad apėjimo metu žymėtų briaunas. Rekursinis algoritmas galėtų būti toks:

DFSmedis(V)

{ Suformuoja aprėpties medį, pradėdamas nuo viršūnės V ir naudodamas paieškos į gylį metodą

}

DFS aprėpties medis

```
Pažymėti viršūnę V kaip aplankytą
for kiekvienai neaplankytai viršūnei U kaimyninei su V do
    begin
        Pažymėti briauną iš V į U kaip aplankytą
    DFSmedis(U)
    end
```

BFS aprėpties medis

Kitas būdas sukonstruoti aprėpties medį **jungiam** neorientuotam grafui yra apeiti grafo viršūnes, naudojant paieškos į plotį algoritmą. Apėjimo metu žymimos briaunos, kuriomis einama.

Baigus apėjimą, pažymėtos briaunos sudarys **paieškos į plotį aprėpties medį** (*BFS spanning tree*).

Tam reikia modifikuoti paieškos į plotį algoritmą taip, kad jis pažymėtų briauną iš viršūnės W į viršūnę U, prieš pridėdamas viršūnę U į eilę.

```
{Q – eilė}
Add(Q, 'V') {įdedame viršūnę V į eilę}
Save('V','u') {Išsaugome briauną (V,'u')}
MarkV('V') {pažymi aplankytą viršūnę}
While (not IsEmpty(Q)) do
        begin
        w:= Get(Q) (Nuskaitomas ir išmetamas pirmas eilės elementas)
        for ∀ 'u', kuri yra neaplankyta kaimynė 'w' do
                begin
                Save(w,'u')
                MarkV('u')|
                Add(Q, 'u')
                end
        end
```

Minimalūs aprėpties medžiai (MAM) (Angl. *Minimum Spanning Trees*)

- ✓Tarkime, reikia suprojektuoti telefonų linijų sistemą, leidžiančią visiems miestams kalbėtis tarpusavyje. Akivaizdus sprendimas nutiesti telefonų linijas tarp bet kurių dviejų miestų. Tačiau sujungti kai kurias poras miestų gali būti neįmanoma, pavyzdžiui, dėl kalnų.
- ✓Ištyrus galimybes, rezultate buvo gautas jungus neorientuotas grafas su svoriais, kuriame briauna reiškia, kad nutiesti telefono liniją galima, bei briaunos svoris reiškia linijos nutiesimo kainą.
- ✓Akivaizdu, kad norima nutiesti linijų sistemą kuo pigiau.

Minimalūs aprėpties medžiai

- ✓Jei briaunos būtų be svorių (arba visų briaunų svoriai būtų vienodi), tereiktų surasti gautam grafui aprėpties medį. Bendra linijų nutiesimo kaina yra ***aprėpties medžio kaina***.
- ✓Kadangi gali egzistuoti ne vienas aprėpties medis ir jų kaina gali būti skirtinga, uždavinio sprendimas yra pasirinkti medį su mažiausia kaina.
- ✓Toks medis vadinamas ***minimaliu aprėpties medžiu***. Toks medis gali būti ne vienintelis, bet visų jų kainos yra lygios.

Kruskal'io Algoritmas. Pseudokodas

$\mathbf{E_{(1)}}$ – aibė briaunų priklausančių MAM.
$\mathbf{E_{(2)}}$ – aibė likusių briaunų.

V(x) – x yra viršūnė.

$\mathbf{E_{(1)}} = 0$, $\mathbf{E_{(2)}} = E$

While ($\mathbf{E_{(1)}}$ turi mažiau ***n-1*** briaunų AND $\mathbf{E_{(2)}}=0$)

{

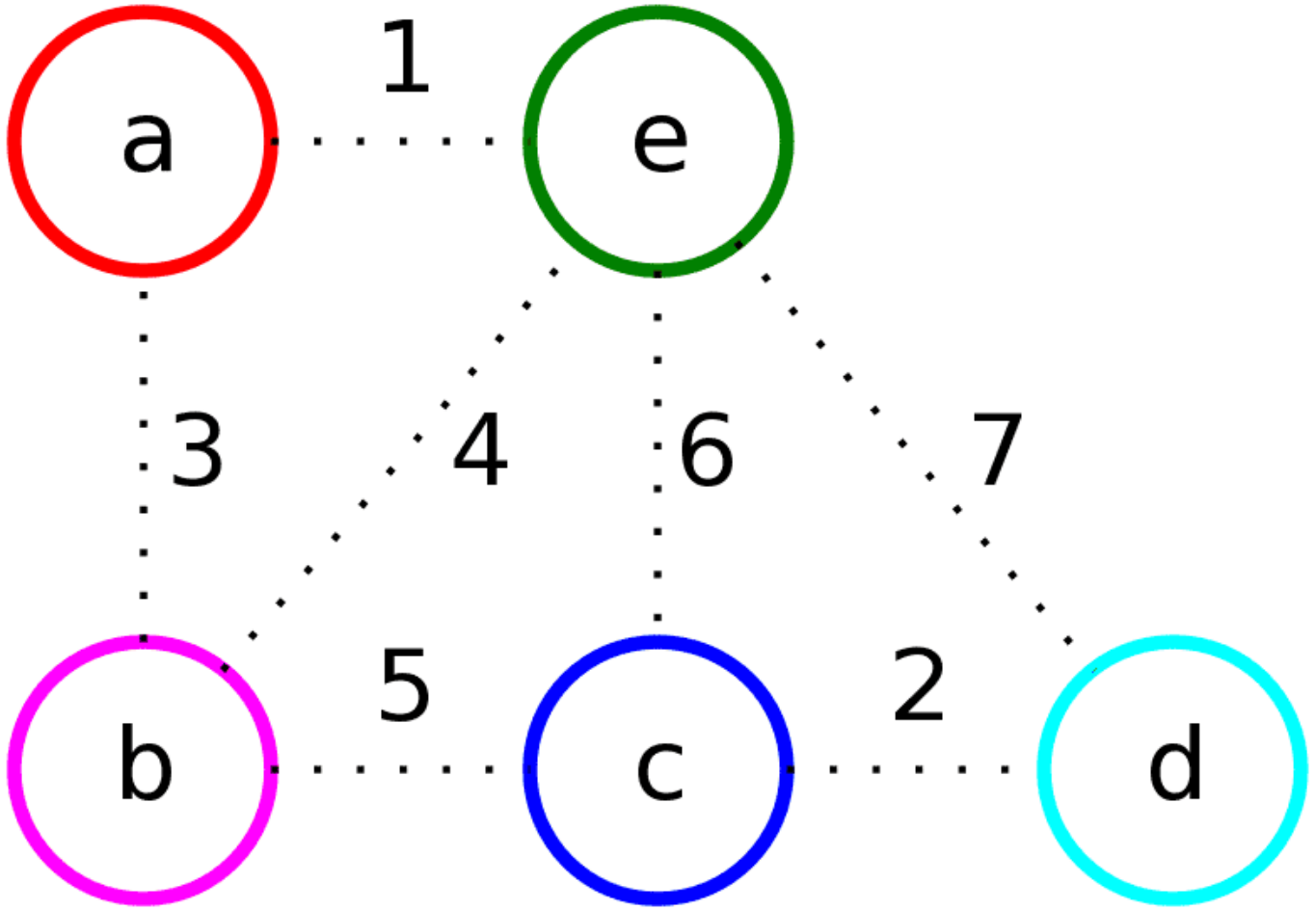
- Iš $\mathbf{E_{(2)}}$ išrinkti e(ij) su mažiausiu svoriu.
- $\mathbf{E_{(2)}}=\mathbf{E_{(2)}}$ - [e(ij)]
- If **V(i)**,**V(j)** tam pačiam medžiui {Apjungti medžius su **V(i)** ir **V(j)** į vieną.

}

Kruskal'io Algoritmas. Pseudokodas (Angliška versija)

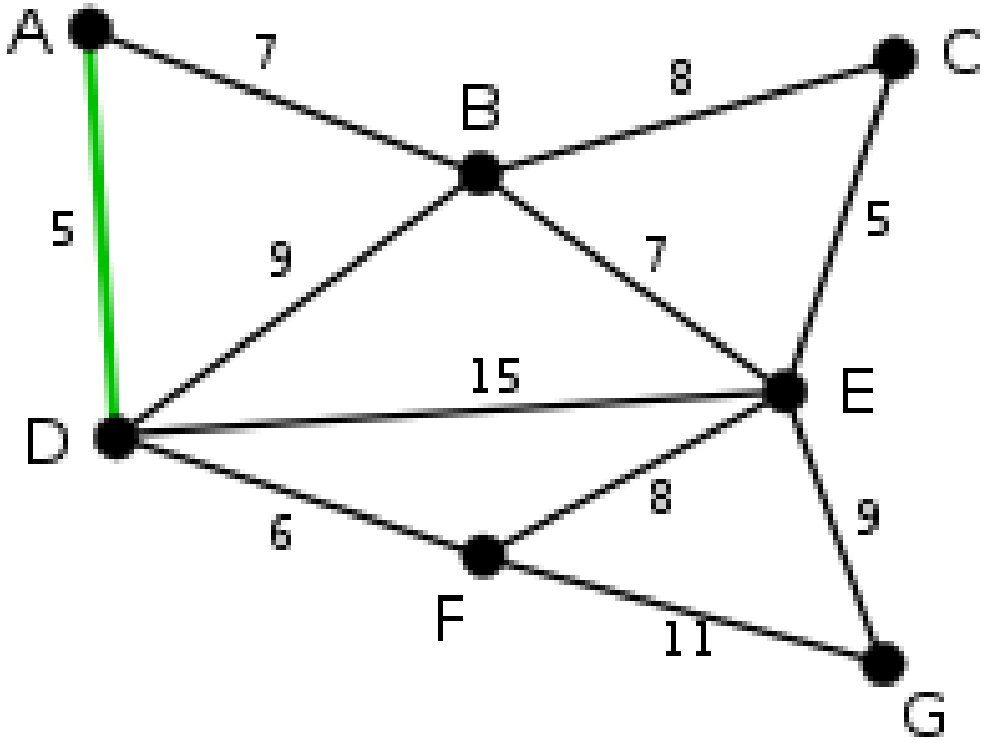
```
KRUSKAL(G):
1 A = ∅
2 foreach v ∈ G.V:
3    MAKE-SET(v)
4 foreach (u, v) ordered by weight(u, v), increasing:
5     if FIND-SET(u) ≠ FIND-SET(v):
6         A = A ∪ {(u, v)}
7         UNION(u, v)
8 return A
```

Edge	ab	ae	bc	be	cd	ed	ec
Weight	3	1	5	4	2	7	6



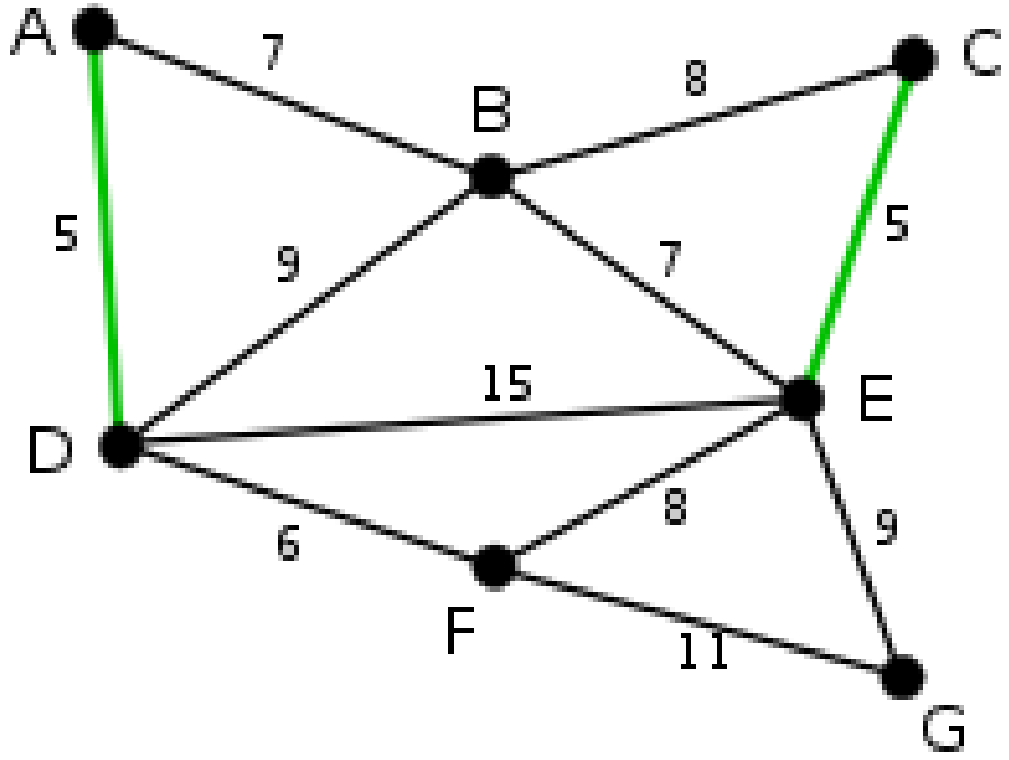
Kruskal'io Algoritmas

AD ir **CE** briaunos yra su mažiausiu svoriu 5 ir yra parenkama briauna **AD**, kurią ir pažymime.



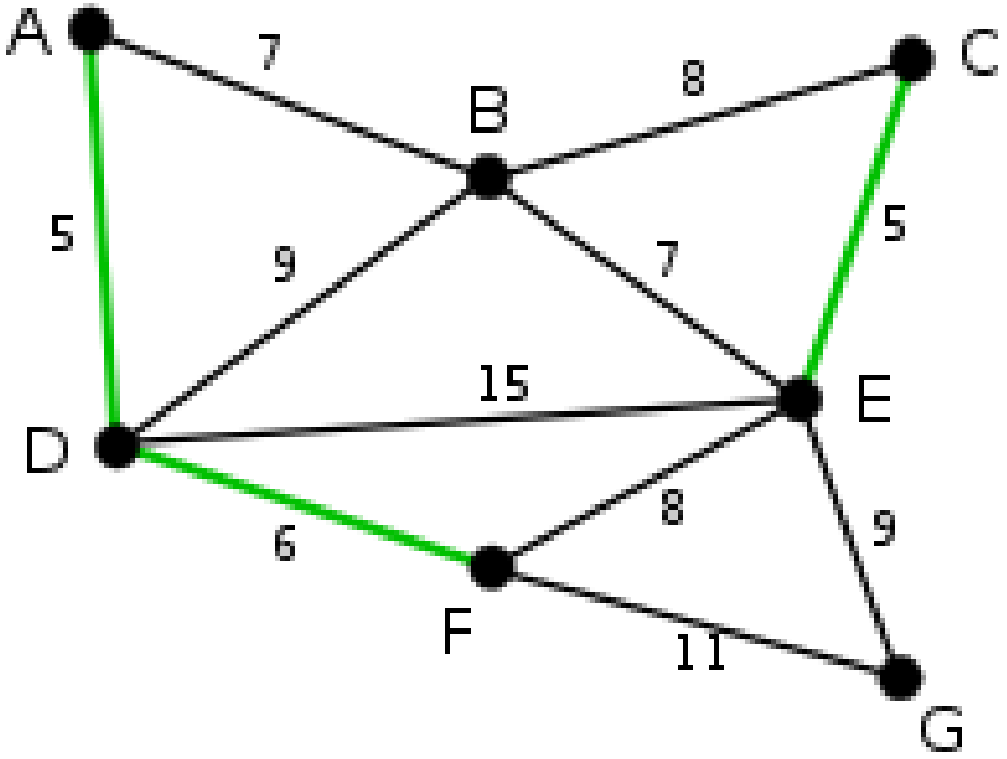
Kruskal'io Algoritmas

CE yra trumpiausia briauna su svoriu 5, kuri nesudaro ciklo, taigi **CE** yra pažymima kaip 2 briauna.



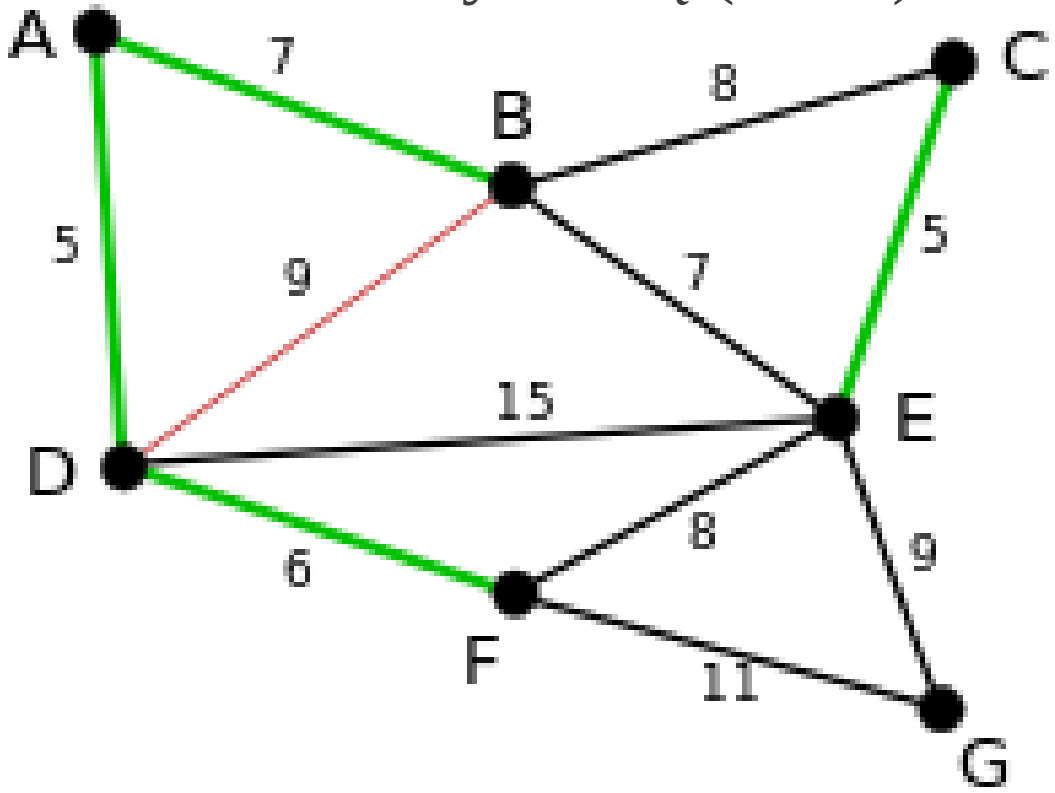
Kruskal'io Algoritmas

Sekanti briauna **DF** su svoriu 6 yra pažymima taikant tą pačią metodiką, kaip ir prieš tai.



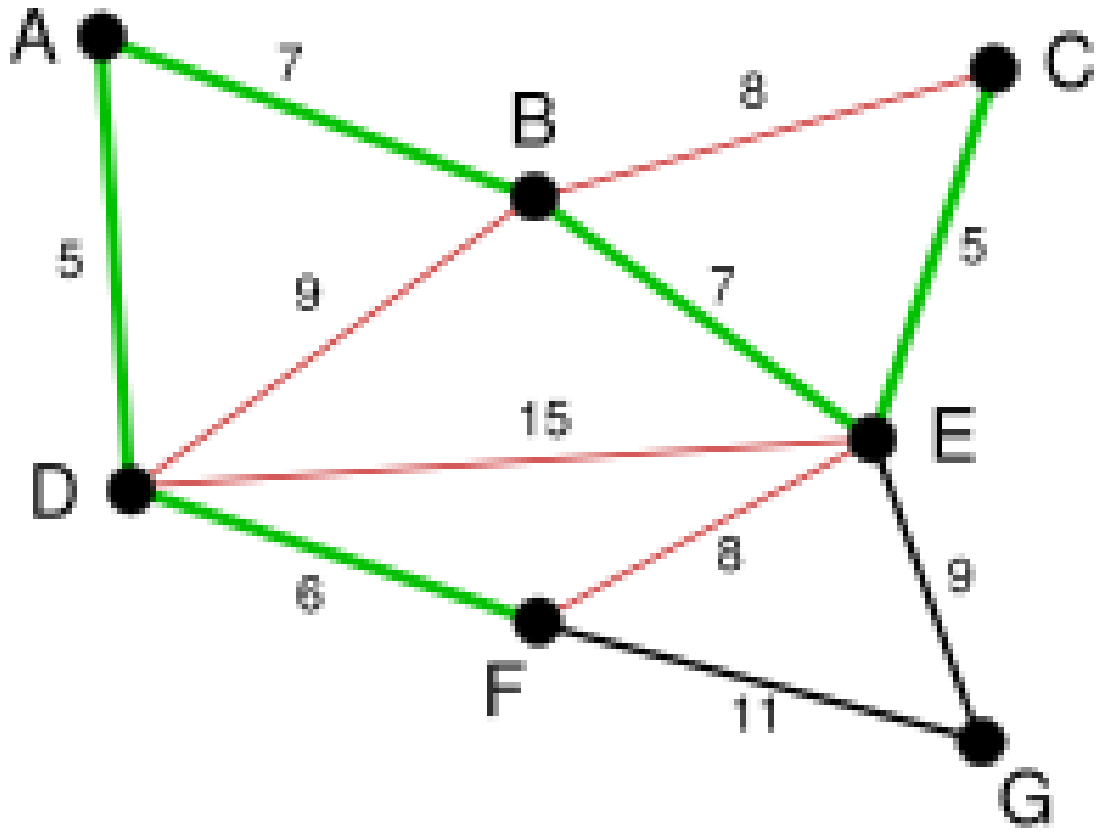
Kruskal'io Algoritmas

Sekančios trumpiausios briaunos **AB** ir **BE**, kurių svoris yra 7. **AB** yra pasirenkama ir pažymima. Briauna **BD** pažymėta raudonai, kadangi jau egzistuoja kelias pažymėtas žaliai tarp B ir D, taigi jeigu mes pasirinksime **BD** sudarys ciklą (**ABD**).



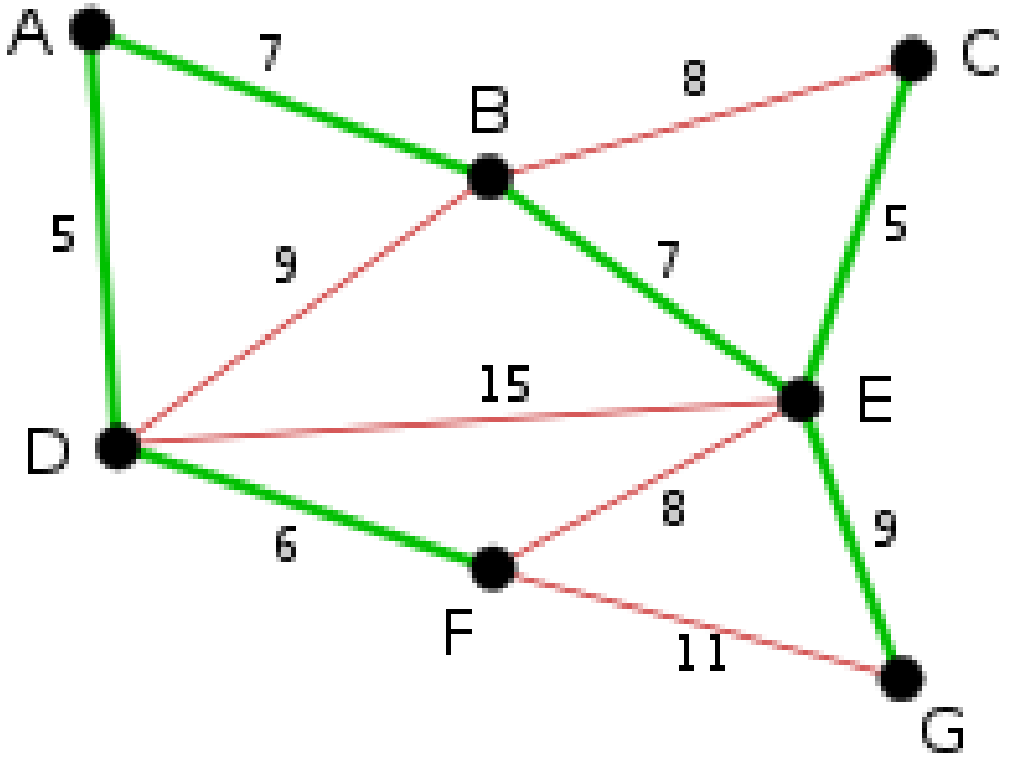
Kruskal'io Algoritmas

Procesas yra tęsiamas, žymima sekanti trumpiausia briauna **BE** su svoriu 7. Daug kitų briaunų yra pažymimos raudonai: **BC** kadangi sudarytų ciklą **BCE**, **DE** suformuoja ciklą **DEBA** ir **FE** sudaro ciklą **FEBAD**.



Kruskal'io Algoritmas

Galiausiai procesass užbaigiamas ties briauna **EG**, kurios svoris 9 ir minimalus aprėpties medis yra surastas.



Trumpiausi keliai (Angl. *Shortest Paths*)

- ✓Tarkime, kad orientuotas grafas su svoriais vaizduoja lėktuvų maršrutus: viršūnės – miestai, briaunos – egzistuojantys skrydžiai, o svoriai – atstumai (neneigiami).
- ✓Dažnai orientuotam grafui su svoriais reikia sužinoti trumpiausią kelią tarp kažkurių dviejų viršūnių.
- ✓Trumpiausias kelias yra kelias, kurio svoris (jį sudarančių briaunų svorių suma) yra minimalus. Nors tradiciškai vartojama sąvoka trumpiausias kelias, bet svoriai nebūtinai turi būti atstumai, jie gali išreikšti ir, pavyzdžiui, kainą, skrydžio laiką.

Trumpiausi keliai (Angl. *Shortest Paths*)

Trumpiausio kelio grafe radimo algoritmas priskiriamas E. Dijkstra. Patogumui pažymėkime pradinę viršūnę, iš kurios ieškome kelio, ‘1’, o visas likusias grafo viršūnes - nuo ‘2’ iki N.

Pastebėkime, kad algoritmas randa trumpiausius kelius tarp pradinės viršūnės ir visų kitų grafo viršūnių.

Trumpiausi keliai (Angl. *Shortest Paths*)

- ✓Algoritmas naudoja pasirinktų viršūnių aibę S ir masyvą W, kur W[v] yra svoris trumpiausio (pigiausio) kelio iš viršūnės 1 į viršūnę v, einančio per aibės S viršūnes. Jei v priklauso S, tai kelias eina tik per aibės S viršūnes.
- ✓Jei v nepriklauso S, tai yra vienintelė viršūnė, priklausanti keliui, bet nepriklausanti S, t.y. kelias baigiasi briauna, jungiančia viršūnę iš S ir viršūnę v.
- ✓Pradžioje aibėje S yra tik viršūnė 1 ir masyve W yra tik svoriai kelių, sudarytų iš vienos briaunos tarp viršūnės 1 ir kitų viršūnių, kitaip sakant masyvas W yra pirma kaimynystės matricos A eilutė.

Trumpiausi keliai (Angl. *Shortest Paths*)

Po inicializavimo žingsnio pasirenkamos viršūnės, nepriklausančios aibei S, ir atitinkamai koreguojamas masyvas W. Pasirenkama viršūnė v, nepriklausanti aibei S, tokia, kad W[v] yra minimalus.

Pridėjus viršūnę v į aibę S, reikia patikrinti reikšmes W[u] visoms viršūnėms u, nepriklausančioms S, tam, kad užtikrinti, jog šios reikšmės iš tikrųjų yra minimalios.

Kitais žodžiais, reikia patikrinti, ar galima sumažinti W[u] – kelią 1 => u, einantį per naujai pasirinktą viršūnę v. Tam kelią nuo viršūnės 1 iki viršūnės u galima suskaidyti į 2 dalis ir rasti jų svorius:

Trumpiausio kelio radimo pseudokodas

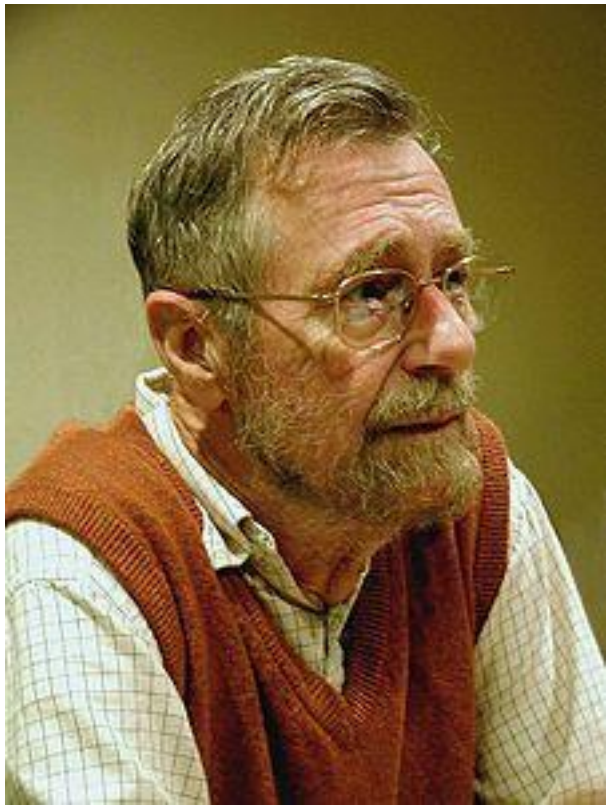
W[v] = svoris trumpiausio kelio nuo 1 iki v
A[v, u] = svoris briaunos iš v į u
Tada reikia palyginti dabartinį W[u] su W[v]+ A[v, u] ir pakeisti W[u], jei nauja reikšmė mažesnė. Trumpiausio kelio radimo algoritmo pseudokodas:

```
ShortestPath(G, W)
{ Randa trumpiausius kelius tarp viršūnes 1 ir visų kitų viršūnių orientuotame grafe G su N
  viršūnių }
```

```
{ žingsnis 1: Inicializavimas }
S := [1];
for v := 1 to N do
W[v] := A[1, v]
 { žingsniai 2 … N }
 { Ciklo invariantas:
 v ∈ S, W[v] yra trumpiausias kelias iš 1 į v, einantis tik per viršūnes iš S iki v;
 v ∉ S, W[v] yra trumpiausias iš visų kelių iš 1 į v, einantis tik per S viršūnes
 }
                for Step := 2 to N do
                begin
                    Rasti mažiausią W[v], kur v nepriklauso S
                    S := S + [v];

                    { Patikrinti W[u] visoms u, nepriklausnačioms S }
                    for visoms u nepriklausan,ioms S do
                        if W[u] > W[v] + A[v, u]
                            then W[u] := W[v] + A[v, u]
                end { for }
```

Dijkstros algoritmas. Autorius: Edsger Wybe Dijkstra



"Computer Science is no more about computers than astronomy is about telescopes."

BIOGRAFIJA. Edsger Wybe Dijkstra

- May 11, 1930 – August 6, 2002

- Received the 1972 A. M. Turing Award, widely considered the most prestigious award in computer science.

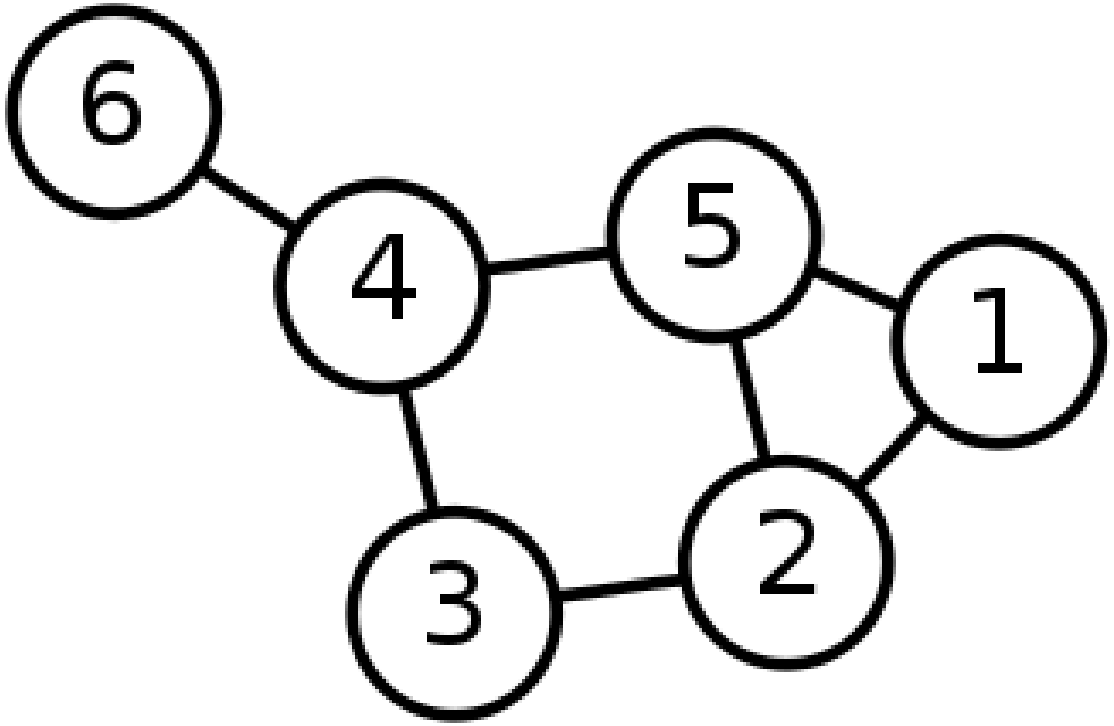
- The Schlumberger Centennial Chair of Computer Sciences at The University of Texas at Austin from 1984 until 2000

- Made a strong case against use of the GOTO statement in programming languages and helped lead to its deprecation.

- Known for his many essays on programming.

Single-Source Shortest Path Problem

Single-Source Trumpiausio kelio problema – surasti trumpiausią kelią iš nurodytos arba source viršunės iki kitų egzistuojančių viršunių.



Dijkstros algoritmas

Dijkstra's algorithm – trumpiausio kelio radimo algoritmas grafų teorijoje.

Veiksmingas dviejų tipų grafams tiek su kryptimi tiek be. Tačiau visos briaunos privalo būti teigiamos.

Įvestis: Svorinis grafas G={E,V} ir fiksuota viršūnė (angl. source) $v \in V$, kur visų briaunų svoriai yra neneigiami.

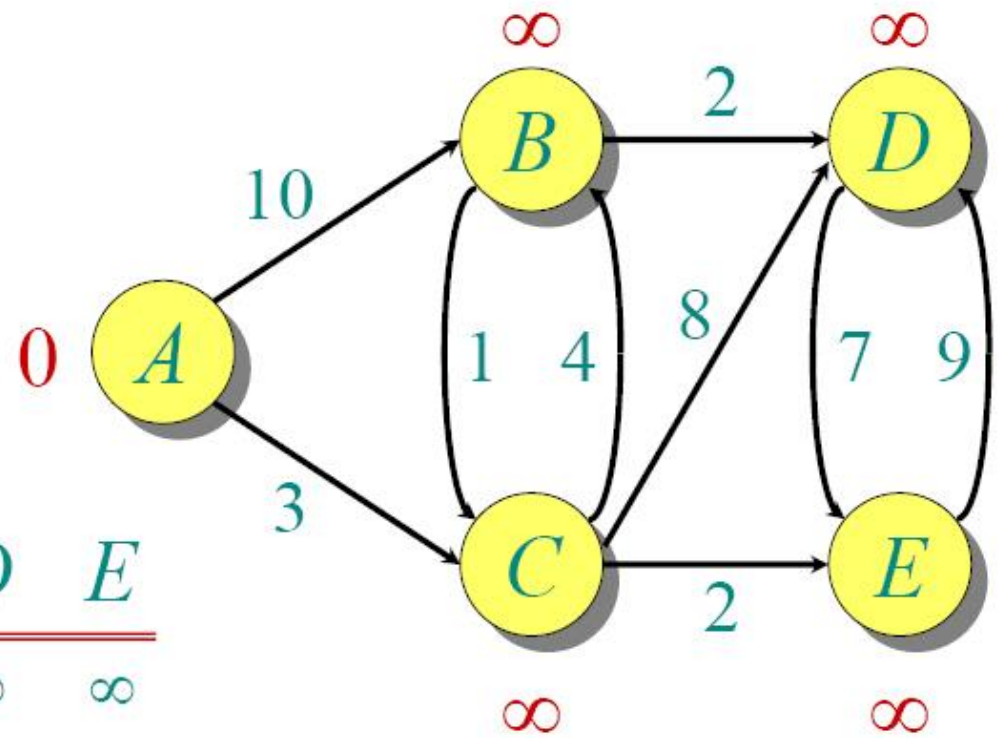
Išvestis: Ilgis trumpiausio kelio iš duotosios viršūnės $v \in V$ iki kitų egzistuojančių viršūnių.

Dijkstros pseudokodas

```
dist[s] ←o                          (Atstumas iki einamosios viršūnės priskiriamas  o)
for  all v ∈ V–{s}
       do  dist[v] ←∞                (Priskirti visus likusius atstumus begalybei)
S←∅                                  (S, priskiriam aplankytų viršunių tuščiai aibei)
Q←V                                  (Q, eilėje  patalpina visas viršūnes)
while Q ≠∅                           (Kol eilė netuščia)
do   u ← mindistance(Q,dist)         (Pasirinkti elementą Q su minimaliu atstumu)
     S←S∪{u}                         (Įdėti u į aplankytų viršunių sąrašą)
      for all v ∈ neighbors[u]
            do  if   dist[v] > dist[u] + w(u, v)    (jeigu naujas trupmpiausias kelias rastas)
                  then     d[v] ←d[u] + w(u, v) (priskiriam naują reikšmę trumpiausiui keliui)
                     return dist
```

Dijkstra Animacija

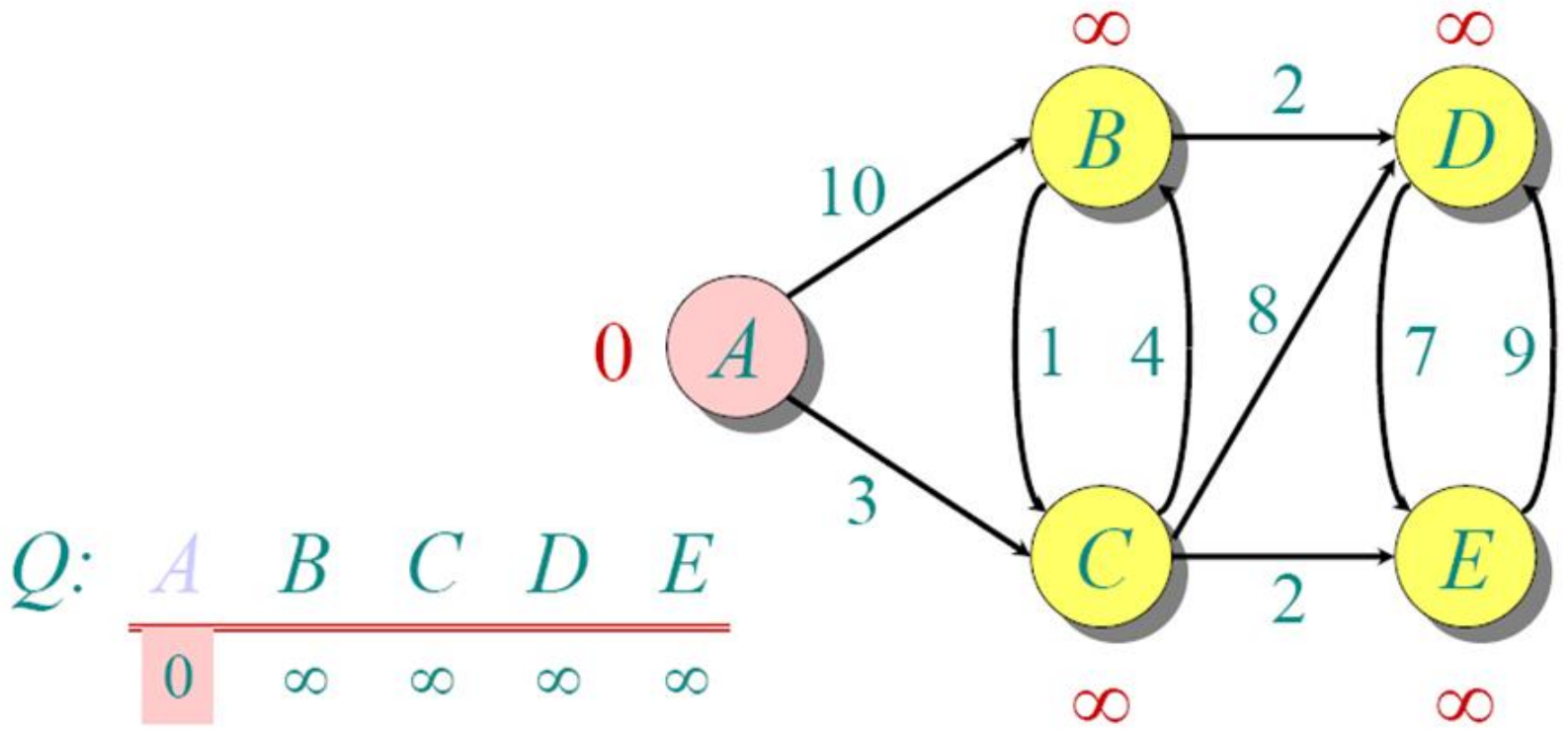
Initialize:



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

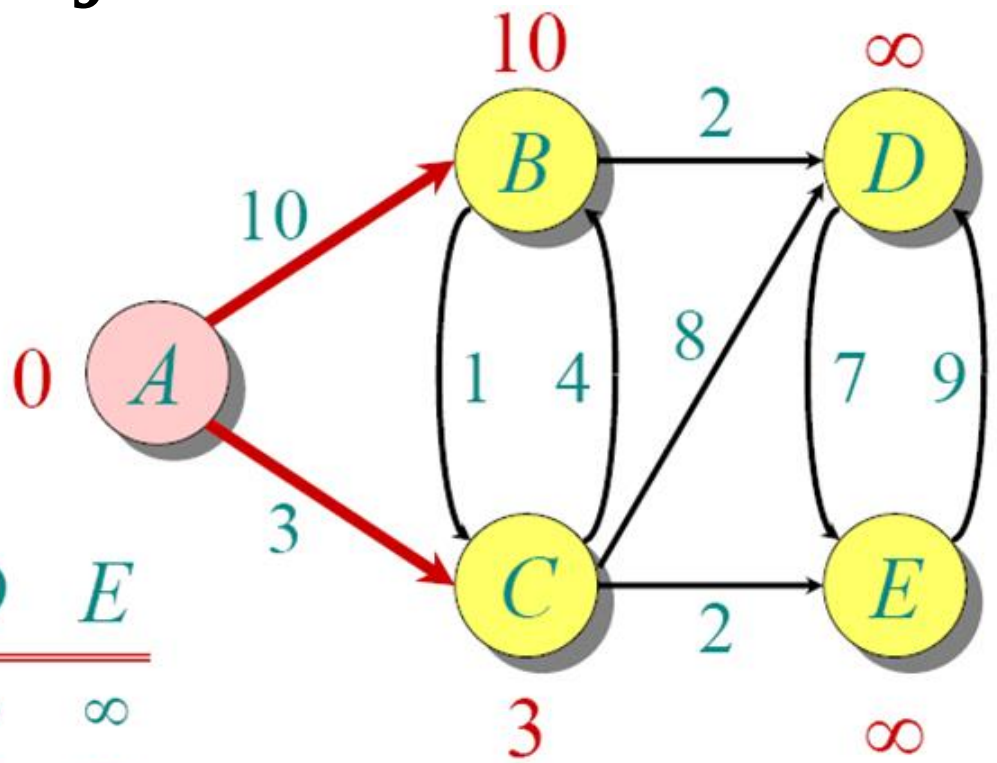
S: {}

Dijkstra Animacija



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

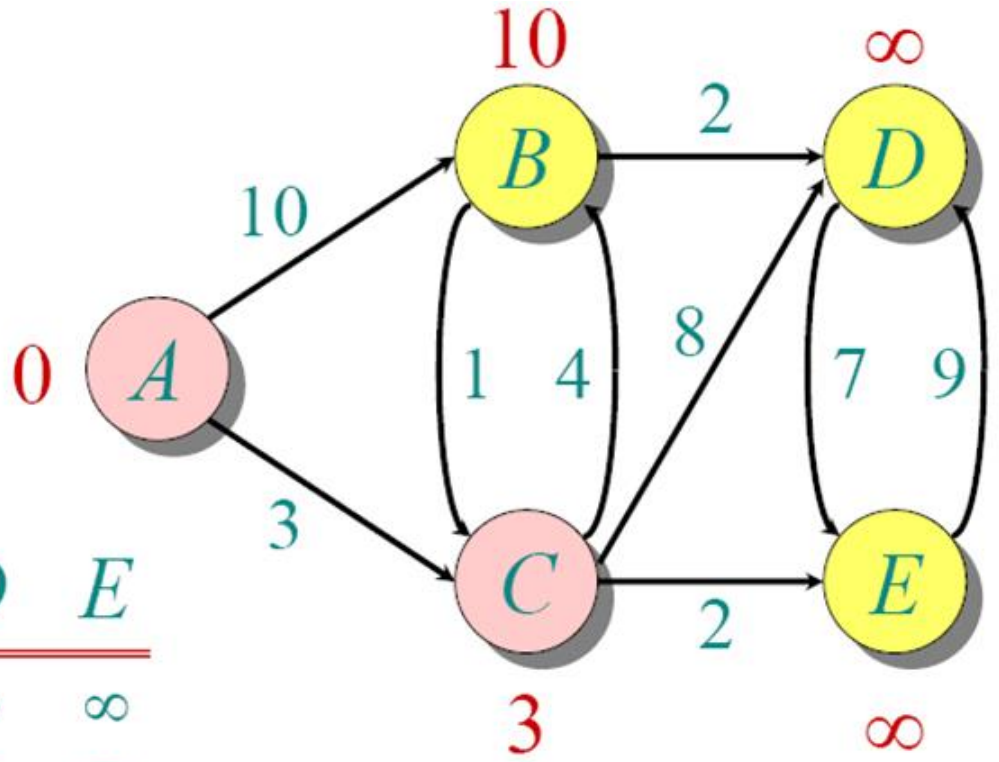
Dijkstra Animacija



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

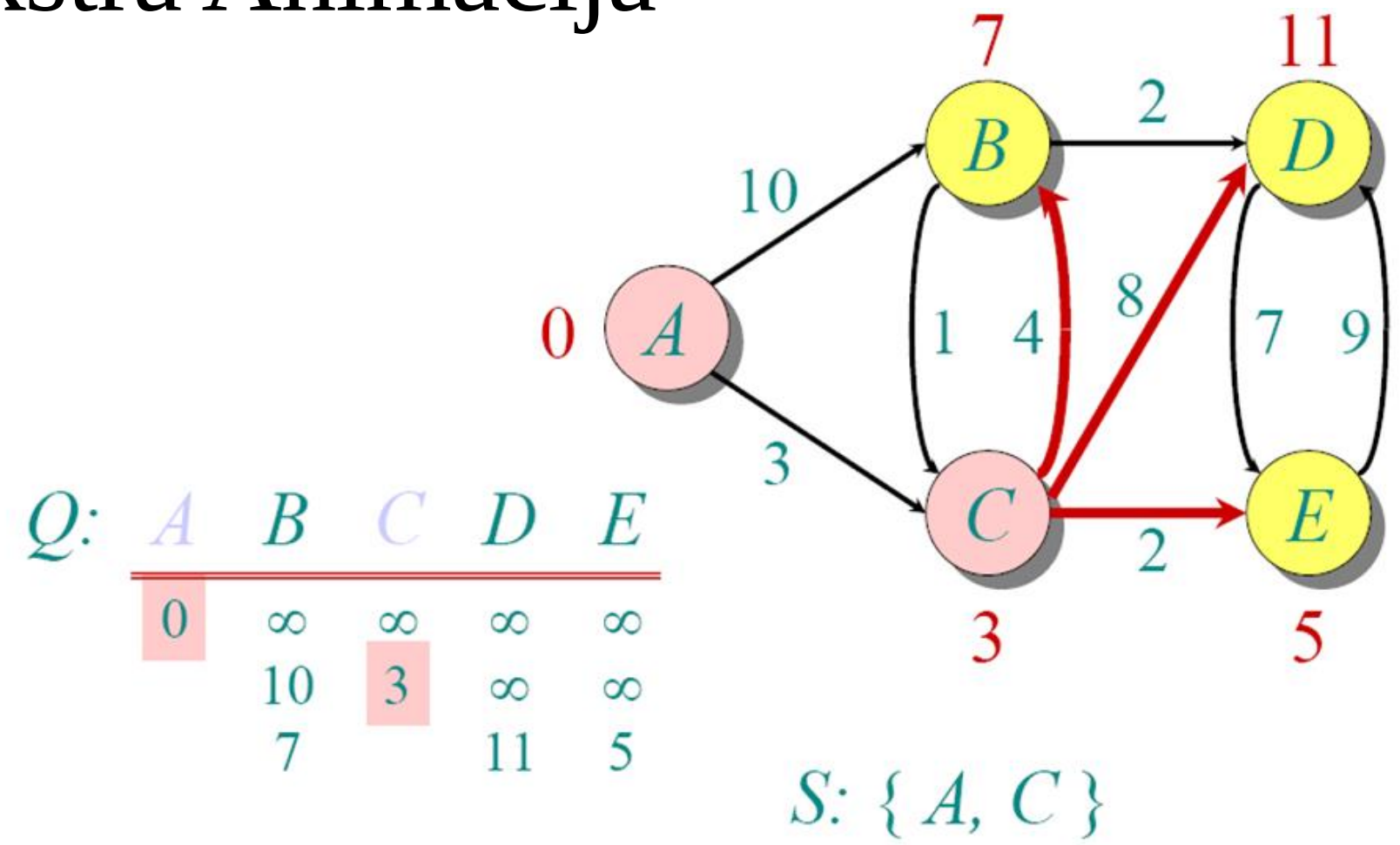
Dijkstra Animacija



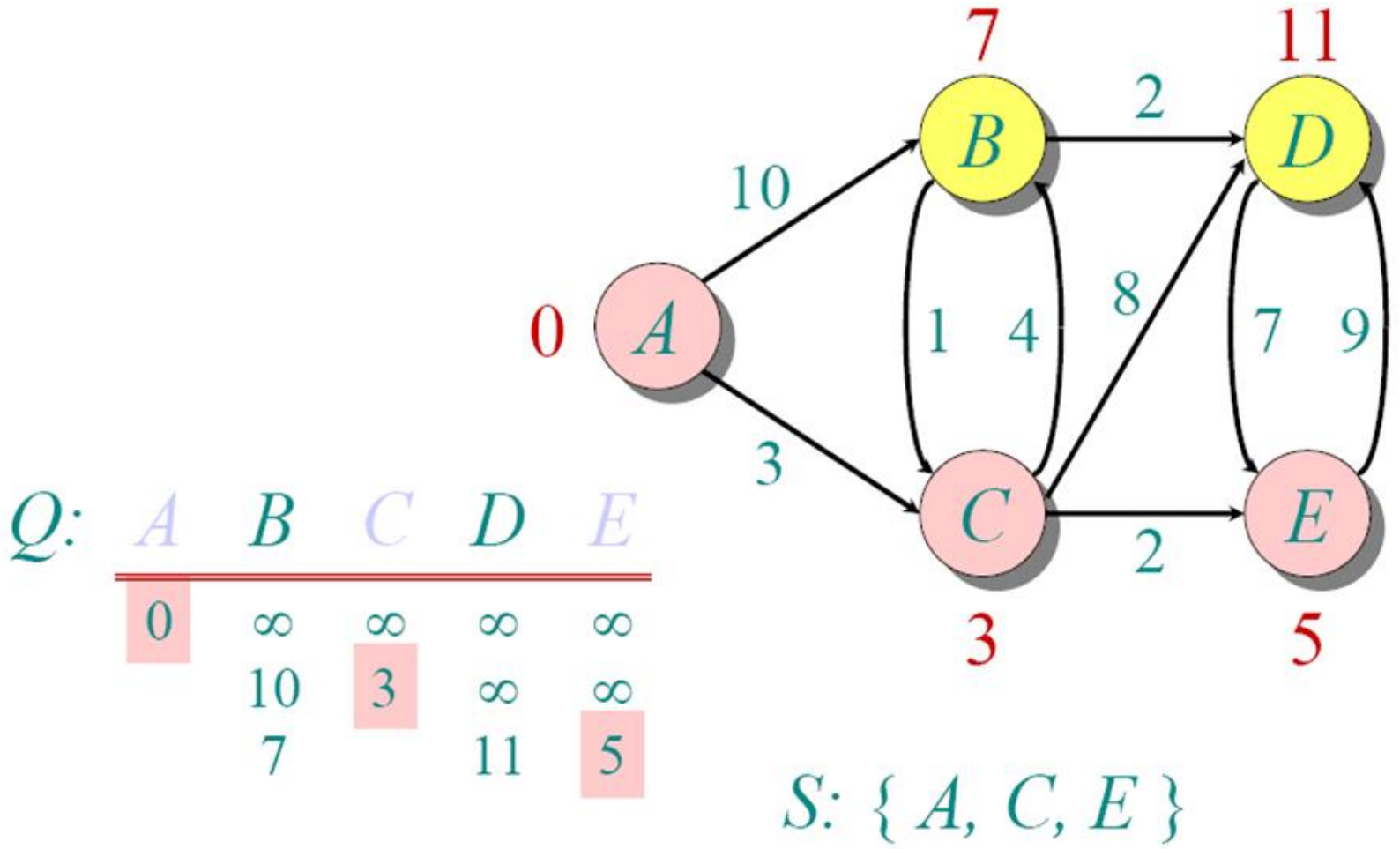
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }

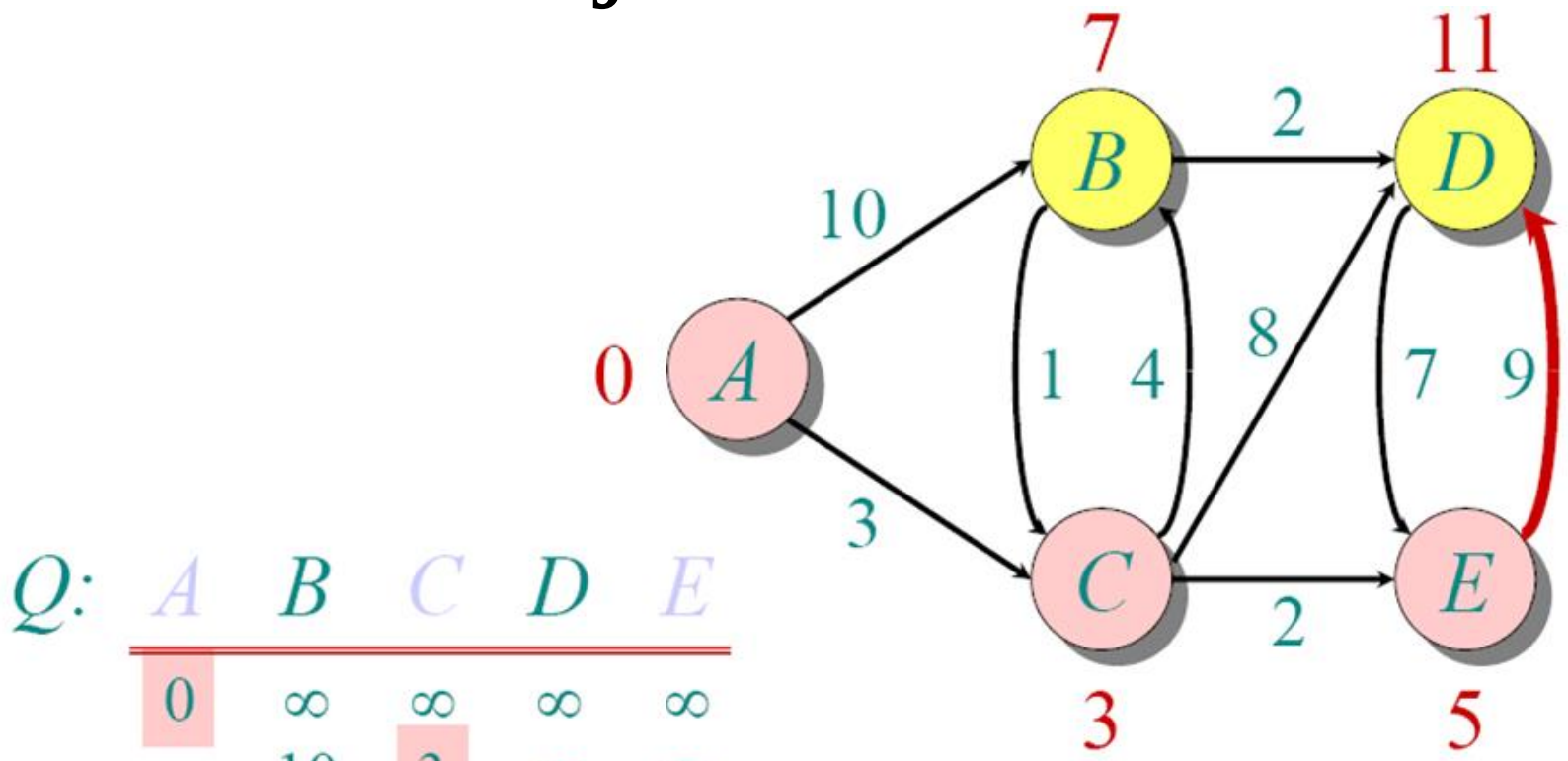
Dijkstra Animacija



Dijkstra Animacija



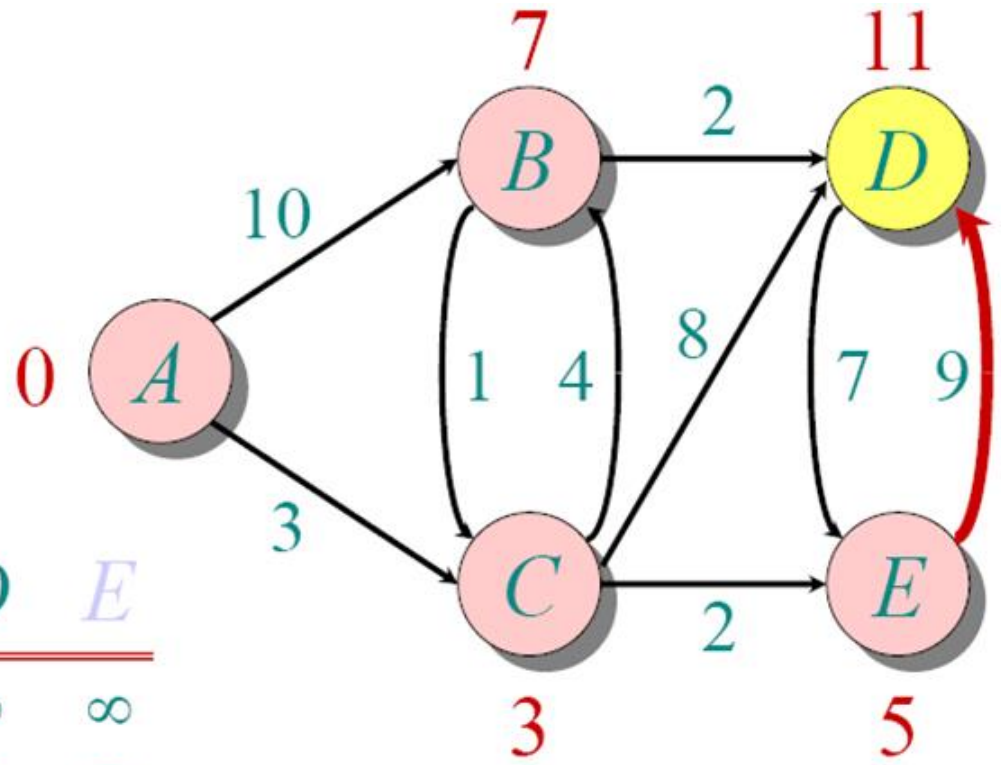
Dijkstra Animacija



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

$S: \{ A, C, E \}$

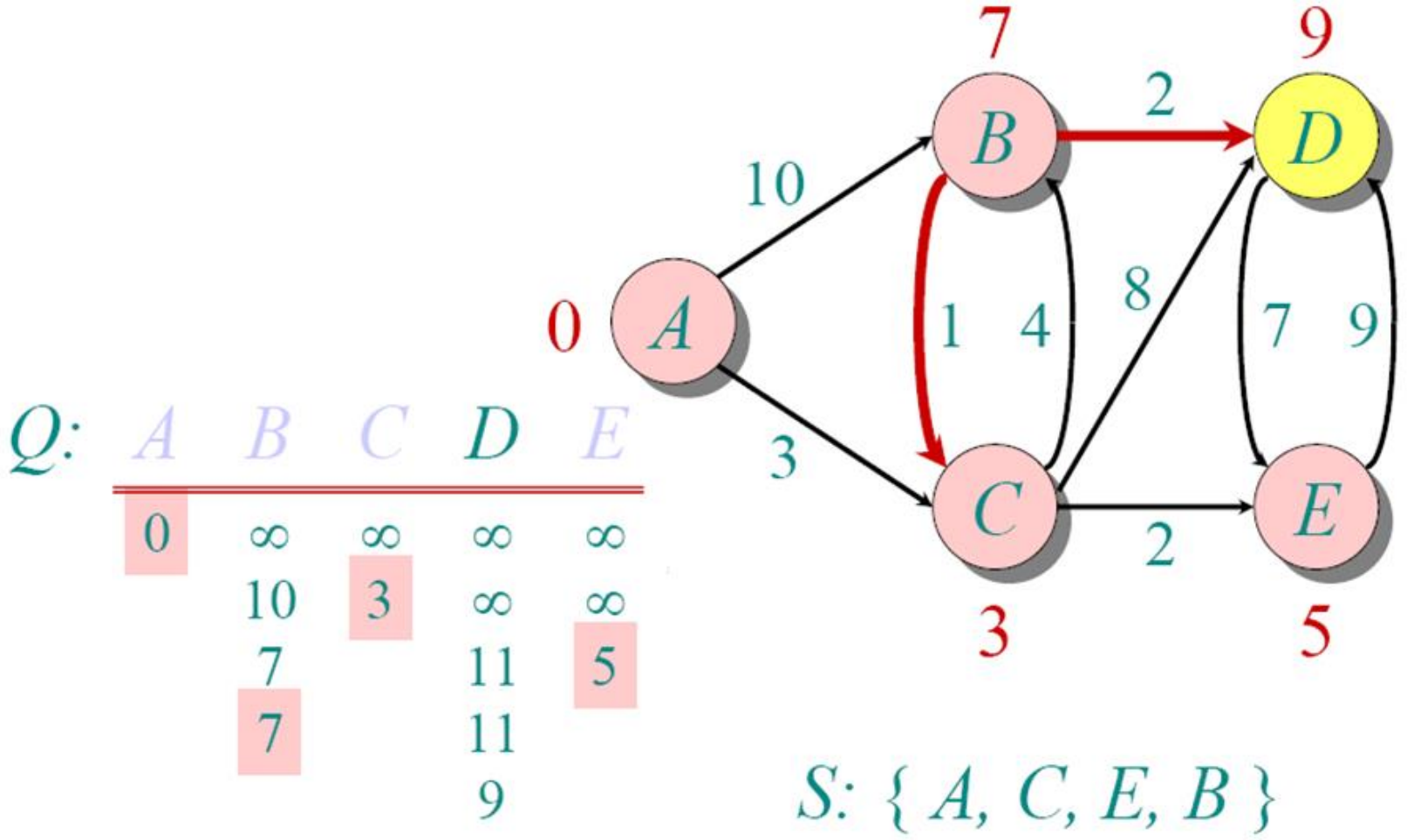
Dijkstra Animacija



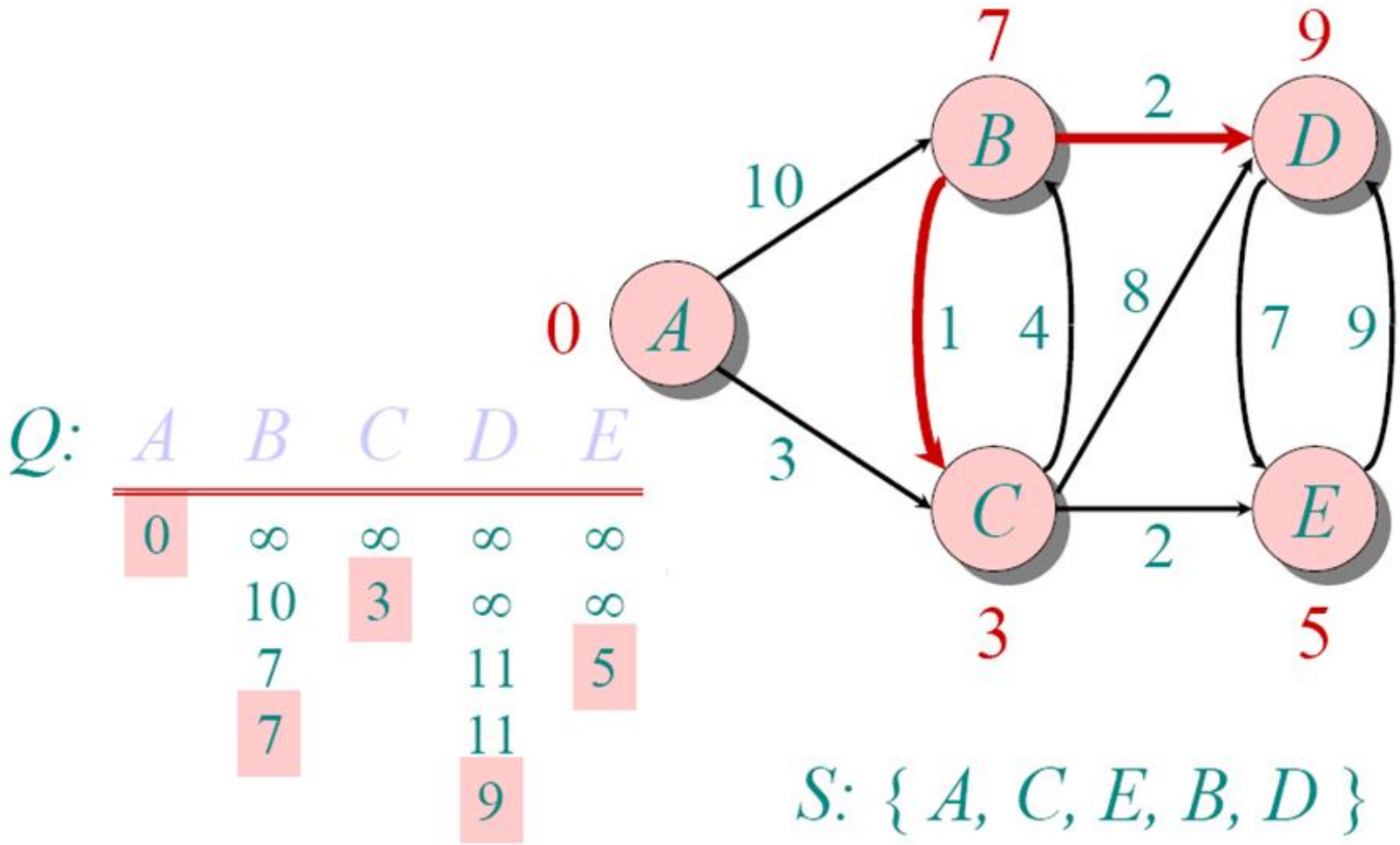
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

$S: \{ A, C, E, B \}$

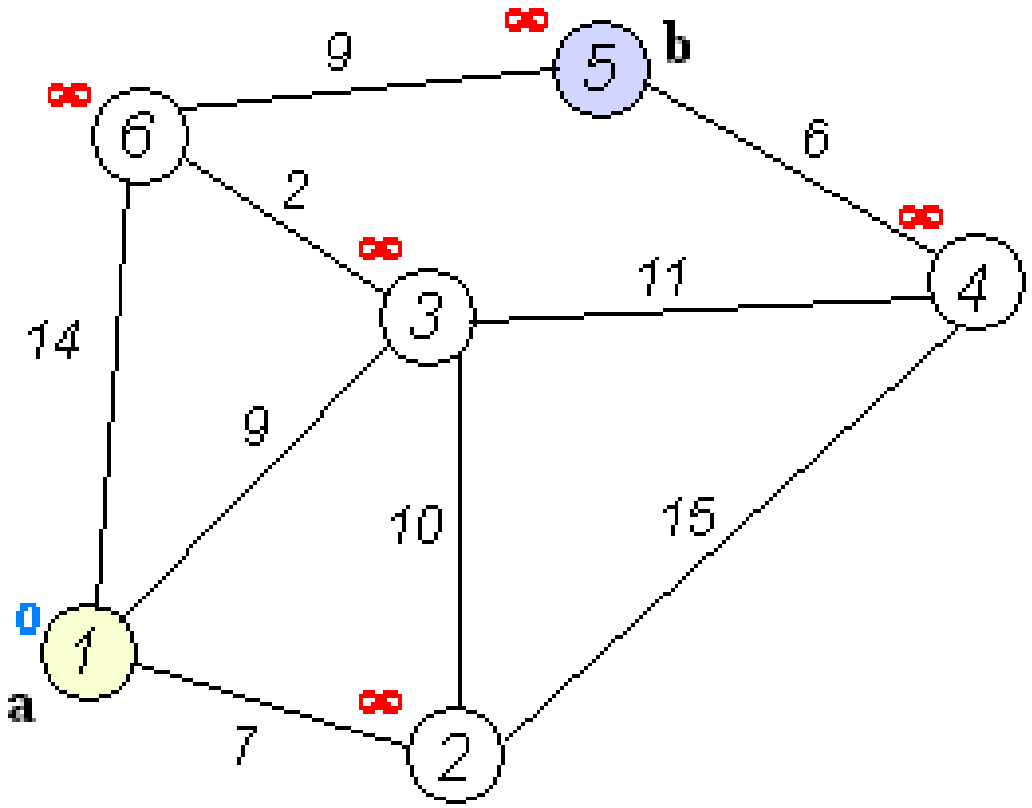
Dijkstra Animacija



Dijkstra Animacija



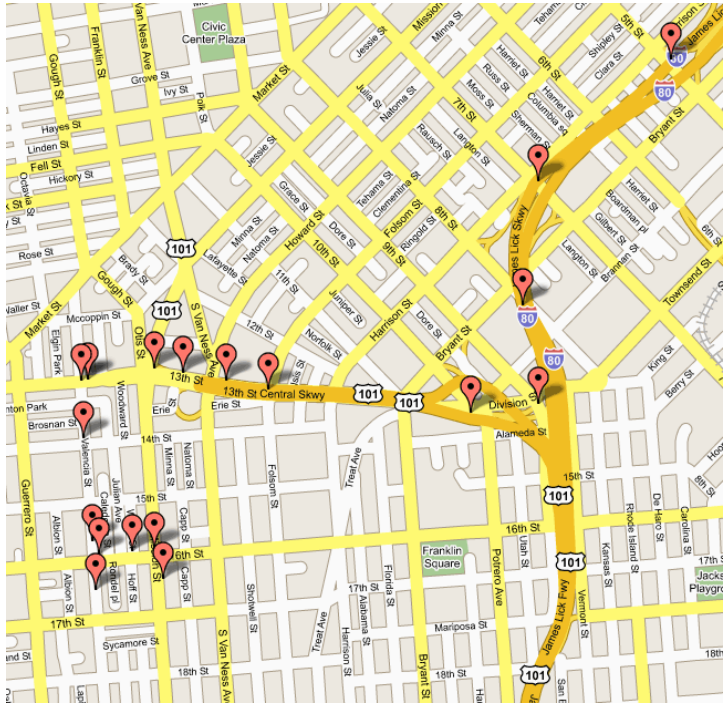
Dijkstros algoritmo animacija

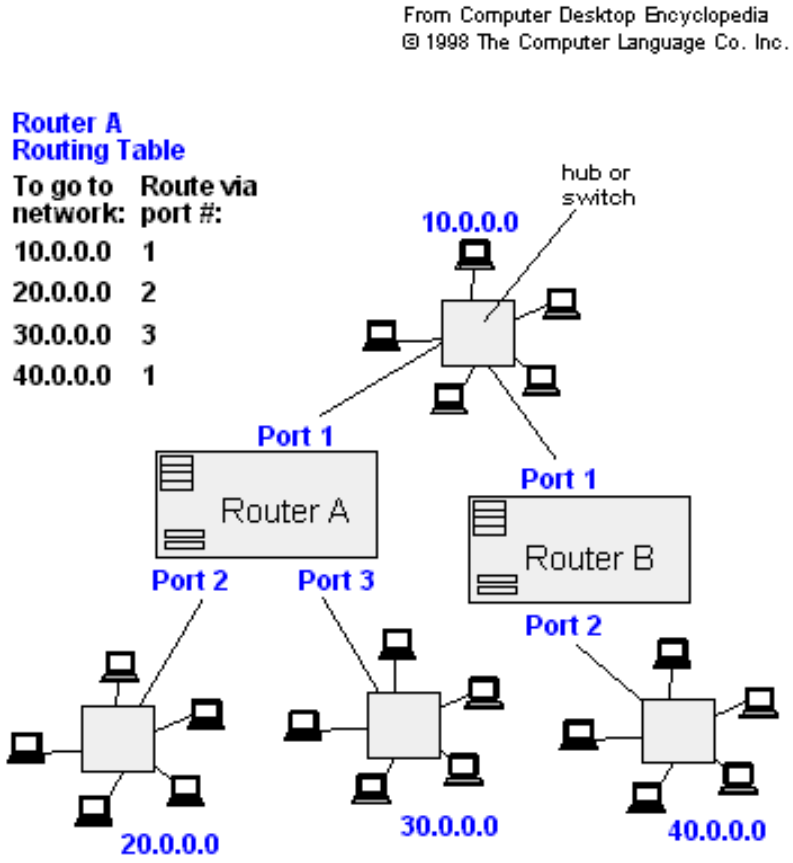


http://optlab-server.sce.carleton.ca/POAnimations2007/DijkstrasAlgo.html

Dijkstra's Algoritmo taikymo sritys

- Traffic Information Systems are most prominent use
- Mapping (Map Quest, Google Maps)
- Routing Systems





KELIOS SUDĖTINGOS PROBLEMOS

1. Prekybos agento uždavinys.
2. Trijų komunalinių paslaugų uždavinys.
3. Keturių spalvų uždavinys.

Prekybos agento uždavinys.

Grandinė (*circuit*) yra kelias, kuris prasideda viršūnėje v, apeina visas grafo viršūnes ir baigiasi viršūnėje v. Akivaizdu, kad nejungiame grafe negali būti grandinės. Tačiau nustatyti ar duotame grafe egzistuoja grandinė gali būti sudėtinga.

Prekybos agento uždavinys.

Gerai žinomas šio uždavinio variantas – prekybos agento uždavinys: grafas su svoriais vaizduoja kelių žemėlapį, briaunų svoriai išreiškia atstumus tarp miestų (arba laiką);

prekybos agentas turi pradėti kelionę duotame mieste, aplankyti kiekvieną miestą vienintelį kartą ir grįžti į pradinį miestą, tačiau grandinė, kuria keliauja prekybos agentas, turi būti pigiausia.

Žinoma, šio prekybos agento uždavinio sprendimas yra ne paprastesnis nei nustatymas, ar grandinė egzistuoja.

Trijų komunalinių paslaugų uždavinys.

Tarkime, yra 3 namai A, B ir C bei 3 komunalinės paslaugos X, Y ir Z (pavyzdžiui, telefonas, vanduo ir elektra).

Jei namai ir komunalinės paslaugos yra grafo viršūnės, ar galima sujungti kiekvieną namą su kiekviena komunaline paslauga taip, kad briaunos nepersikirstų.

Atsakymas – ne, kadangi grafas yra izomorfinis $K_{3,3}$, kuris nėra plokščias.

Trijų komunalinių paslaugų uždavinys.

Trijų komunalinių paslaugų uždavinio apibendrinimas – nustatyti, ar duotas grafas yra planarinis. Ši grafo savybė yra svarbi daugelyje taikymų.

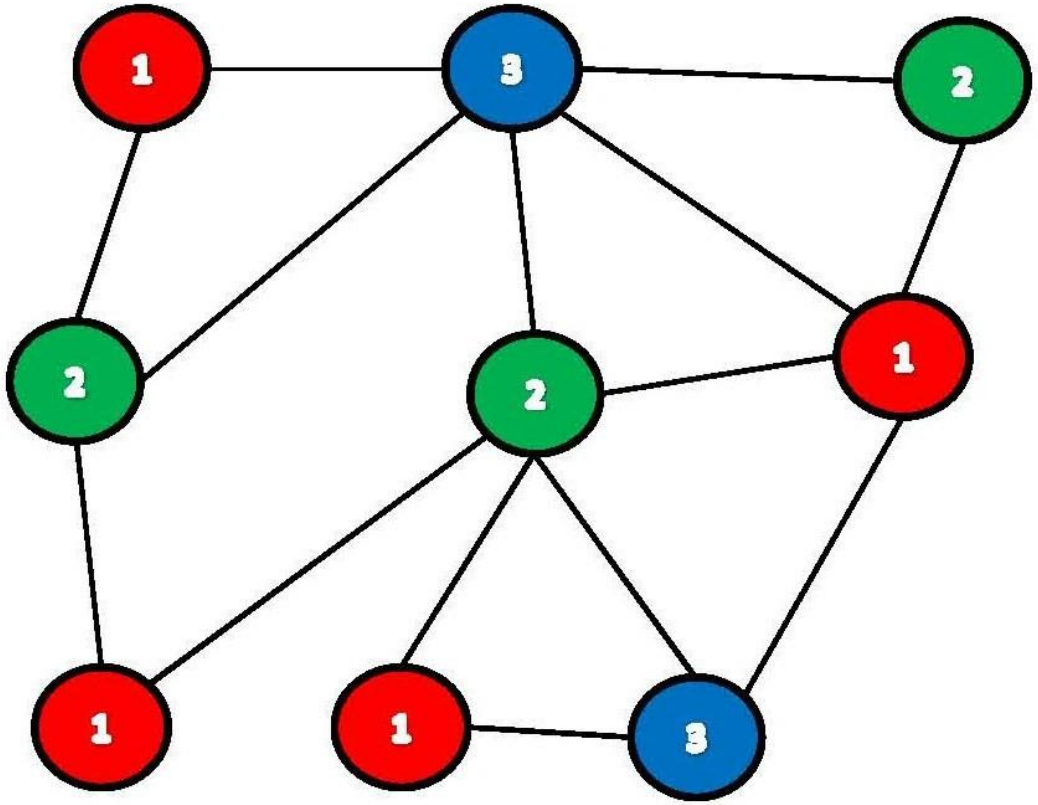
Pavyzdžiui, grafas gali vaizduoti elektrinę grandinę, kur viršūnės atitinka elementus, o briaunos ryšius tarp elementų. Ar galima suprojektuoti elektrinę grandinę taip, kad ryšiai nesikirstų?

Keturių spalvų uždavinys

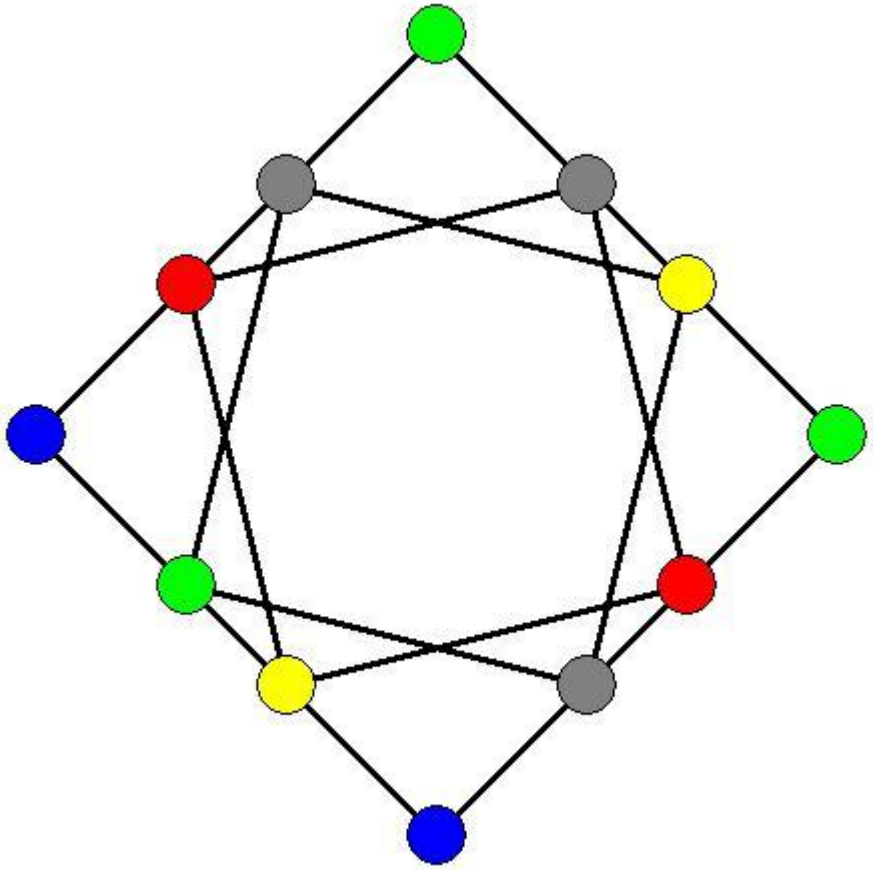
Ar duotą planarinį grafą galima nuspalvinti ne daugiau kaip 4 spalvomis taip, kad nebūtų vienos spalvos kaimyninių viršūnių? Atsakymas į šį klausimą – taip, bet įrodyti tai yra sudėtinga.

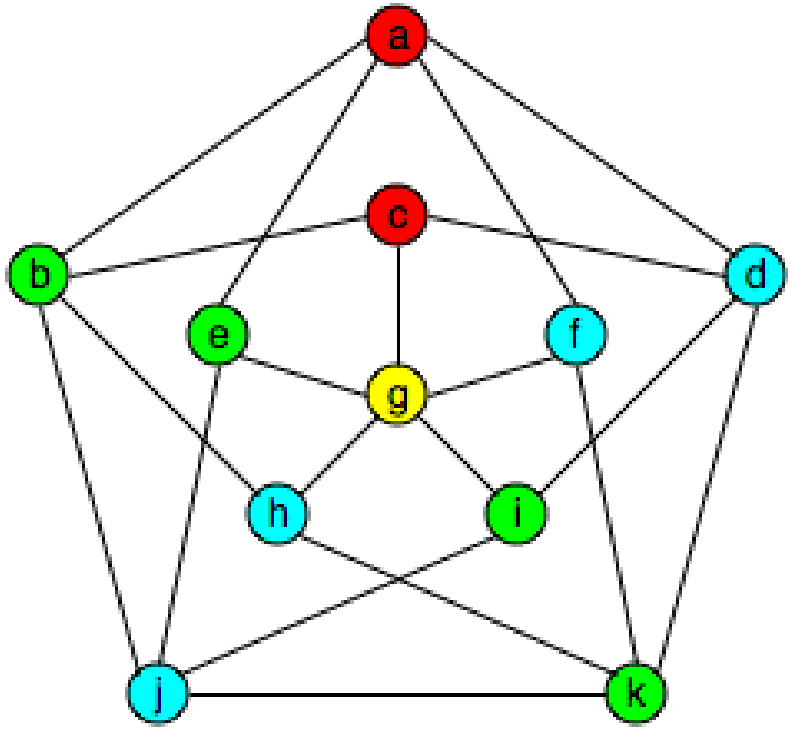
Iš tikrųjų šis uždavinys buvo suformuluotas daugiau kaip prieš 100 metų, tačiau įrodytas tik 1970-aisiais panaudojus kompiuterius.

Keturių spalvų uždavinys



Keturių spalvų uždavinys





Keturių spalvų uždavinys