Algoritmai ir duomenų struktūros

9 paskaita

2025-04-08

9 paskaitos tikslas

Susipažinti su grafų paieškos algoritmais:

- Paieška į plotį,
- Paieška į gylį,
- Kruskalio algoritmu,
- Primo algoritmu.

Paieška į plotį (angl. Breadth-first search)

- Tikslas – pradedant grafo (digrafo) viršūne *s* apeiti grafą keliaujant jo briaunomis t . y. atrasti visas pasiekiamas grafo (digrafo) viršūnes.
- Viršūnės grafe yra tarpusavyje pasiekiamos, jei jas jungia takas.
- Tako uv ilgiu vadinamas grafe pereinamų briaunų skaičius ir žymimas $\delta(u, v)$.
- Jei $\delta(u, v) = \infty$, tai viršūnė u nepasiekiama iš viršūnės v.
- Jei u, x, v – take esančios trys gretimos viršūnės, tai viršūnė u vadinama viršūnės x tėvu ir viršūnė v vadinama viršūnės x vaiku.

Paieškos į plotį algoritmas

P-Plot(G, s)**:**

1 for each $u \in V - \{s\}$
2 **do** $c[u] \leftarrow balta$
3 $d[u] \leftarrow \infty$
4 $\pi[u] \leftarrow NIL$
5 $c[s] \leftarrow pilka$
6 $d[s] \leftarrow 0$
7 $\pi[s] \leftarrow NIL$
8 $Q \leftarrow \{s\}$
9 while $Q \neq \emptyset$
10 **do** $u \leftarrow h(Q)$
11 **for each** $v \in Adj[u]$
12 **do if** $c[v] = balta$
13 **then** $c[v] \leftarrow pilka$
14 $d[v] \leftarrow d[u] + 1$
15 $\pi[v] \leftarrow u$
16 $ENQUEUE(Q, v)$
17 $DEQUEUE(Q)$
18 $c[u] \leftarrow juoda$
END

Sutartiniai žymėjimai:
h(Q) – sąrašo Q „galva“ (pirmasis elementas),
Adj[u] – viršūnei u gretimų viršūnių aibė,

Atributai:
c[u] – spalvos atributas,
d[u] – atstumo atributas,
π[u] – tėvystės atributas.

Operacijos sąraše Q:
ENQUEUE(Q,v) – naujos viršūnės įrašymas sąrašo Q gale,
DEQUEUE(Q) – galvos išėmimas iš sąrašo (pirmojo elemento pašalinimas).

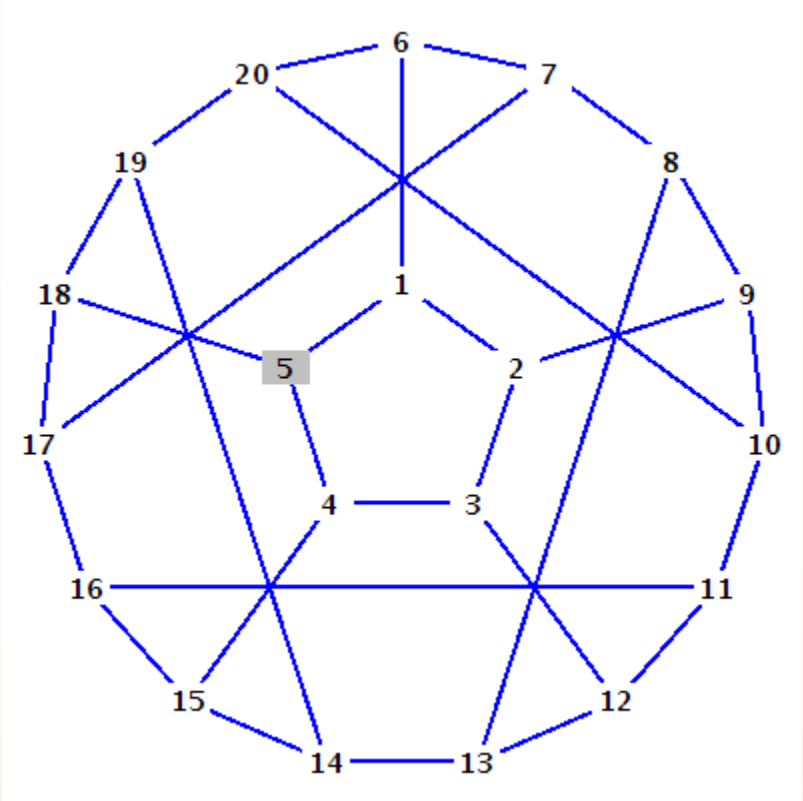
Paieškos į plotį algoritmo sudėtingumas

- ENQUEUE(Q, s) sudėtingumas O(1),
- DEQUEUE(Q) sudėtingumas O(1).

- Į eilę Q viršūnės patenka ir Q išimamos ne daugiau kaip 1 kartą, todėl šios procedūros užtrunka a O(|V|) = O(n) laiko vienetų, (n – grafo G viršūnių skaičius).
- |E| dydžio gretimumo sąrašas nuskaitomas tik vieną kartą, todėl šio sąrašo apdorojimo algoritmo sudėtingumas O(|E|) = O(m) , (m – grafo G briaunų skaičius).
- Vadinasi, paieškos į plotį algoritmo sudėtingumas

O(n + m).

Paieška į plotį grafe

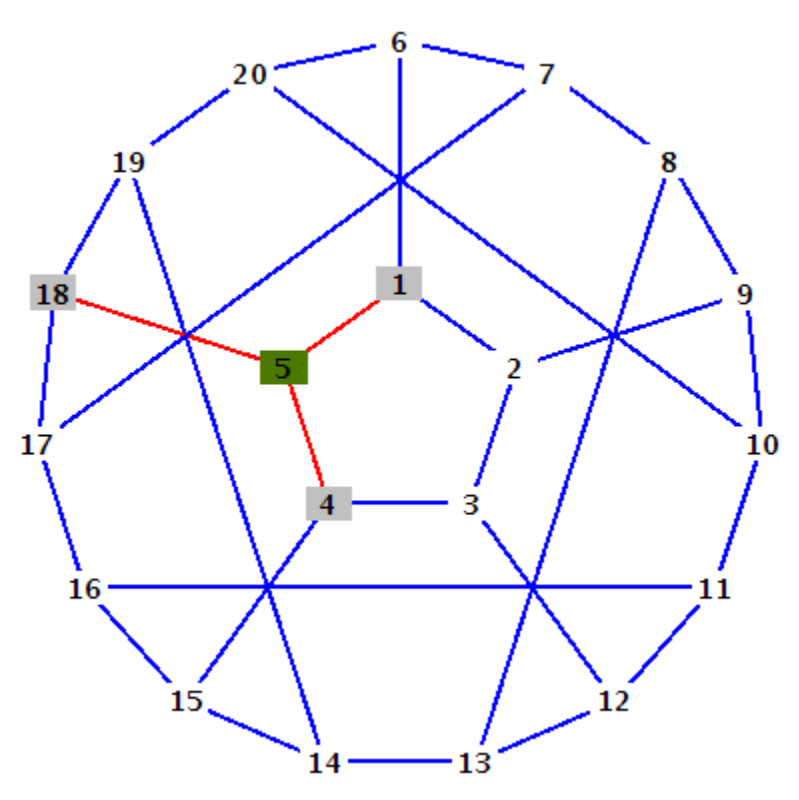


Gretimumo matrica:

0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	1
0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0

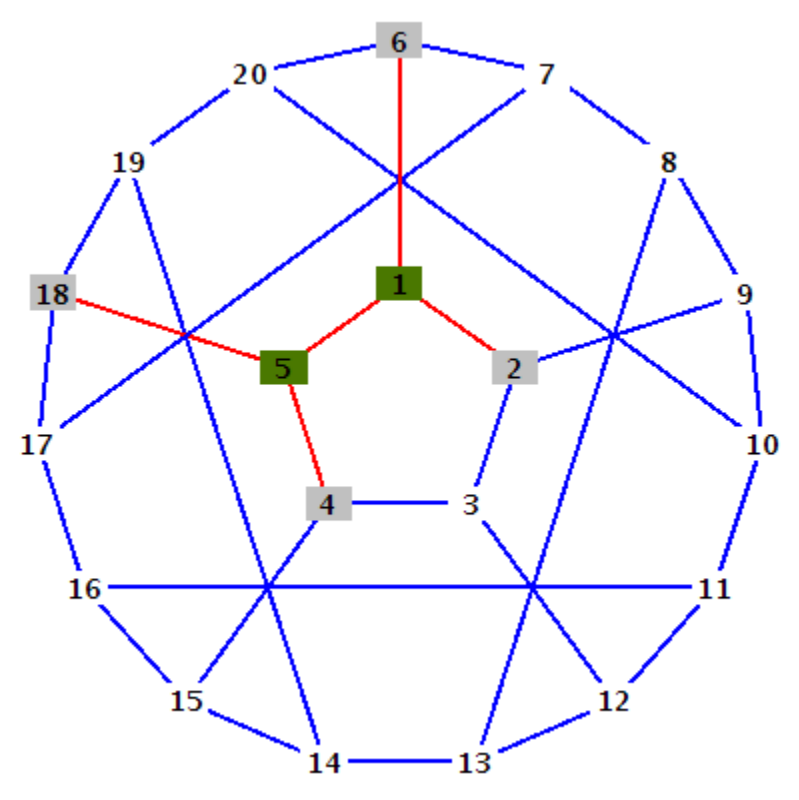
d[1]=∞, ... , d[20]=∞, π[5]=NIL

Paieška į plotį grafe



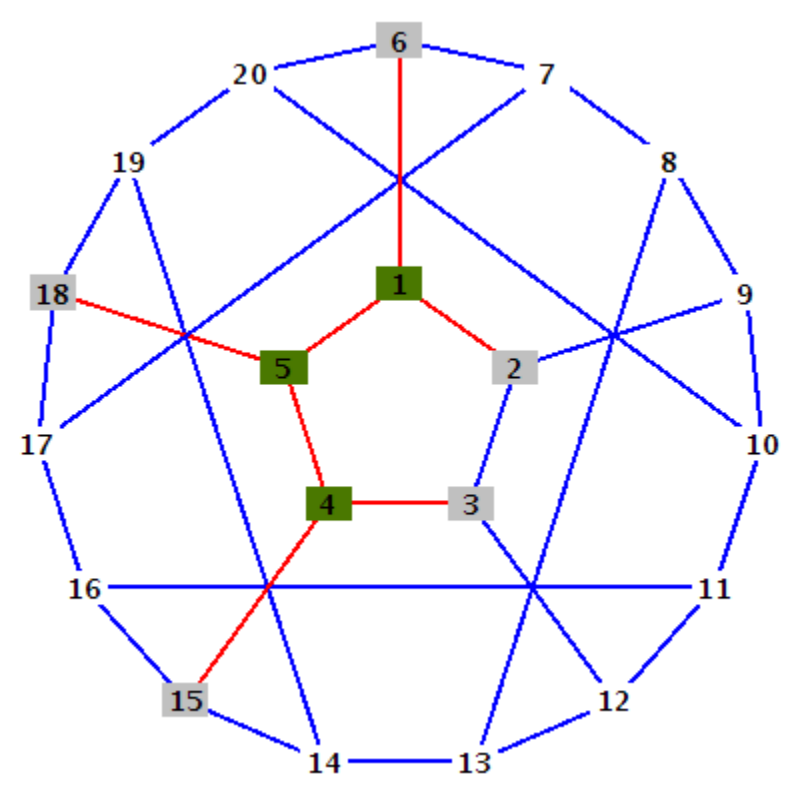
d[1]=1, π[1]=5, d[4]=1, π[4]=5, d[18]=1, π[18]=5

Paieška į plotį grafe



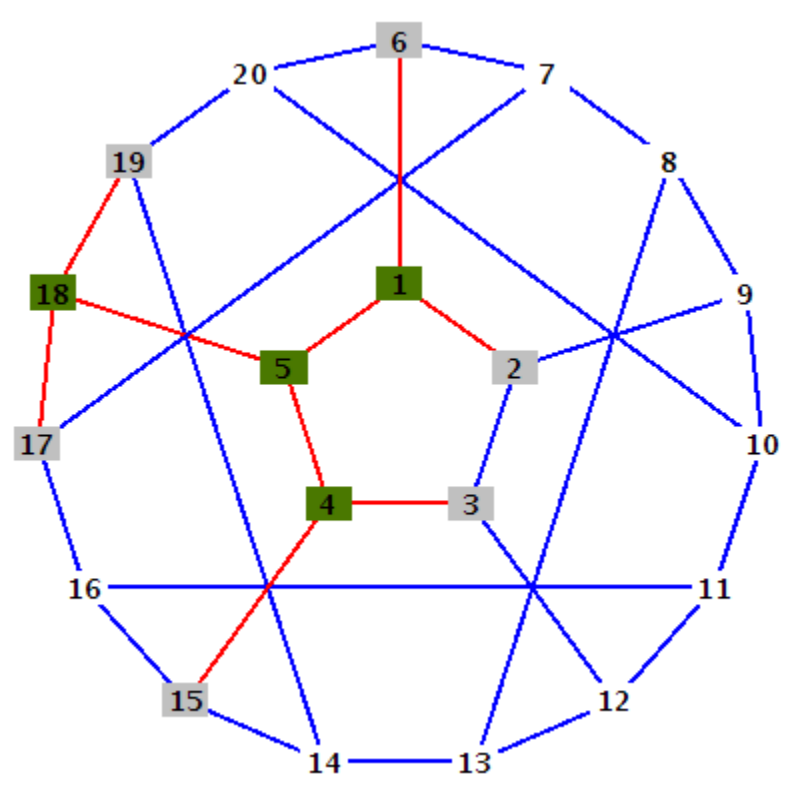
d[2]=2, π[2]=1, d[6]=2, π[6]=1

Paieška į plotį grafe



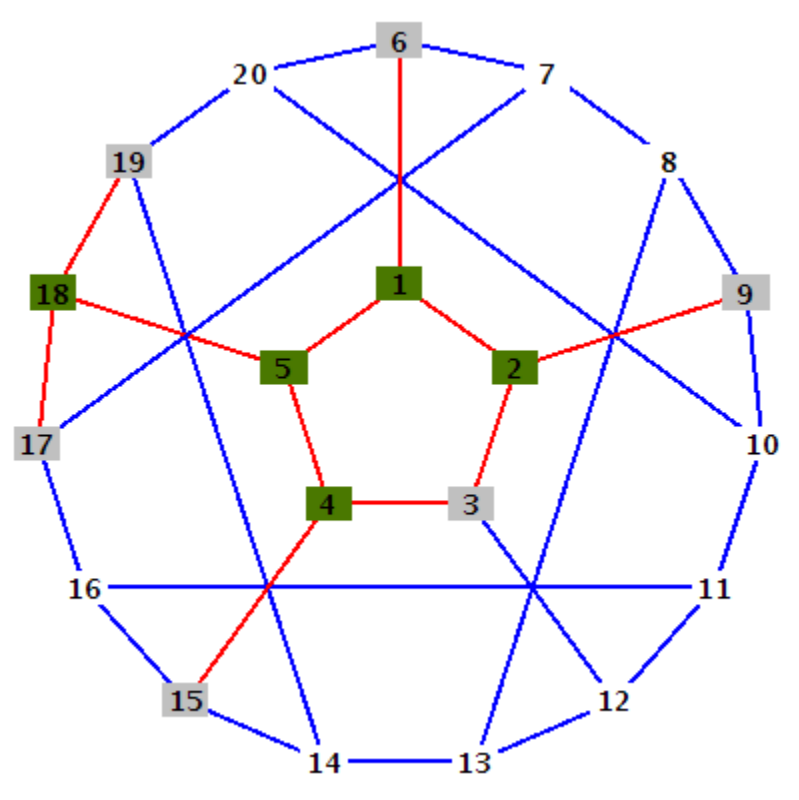
d[3]=2, π[3]=4, d[15]=2, π[15]=4

Paieška į plotį grafe



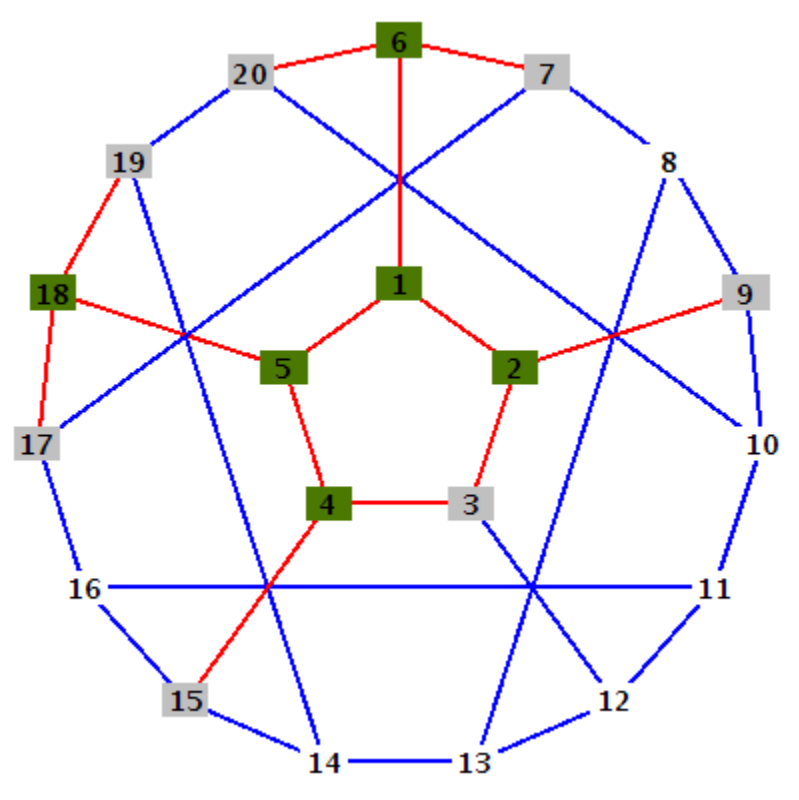
d[17]=2, π[17]=18, d[19]=2, π[19]=18

Paieška į plotį grafe



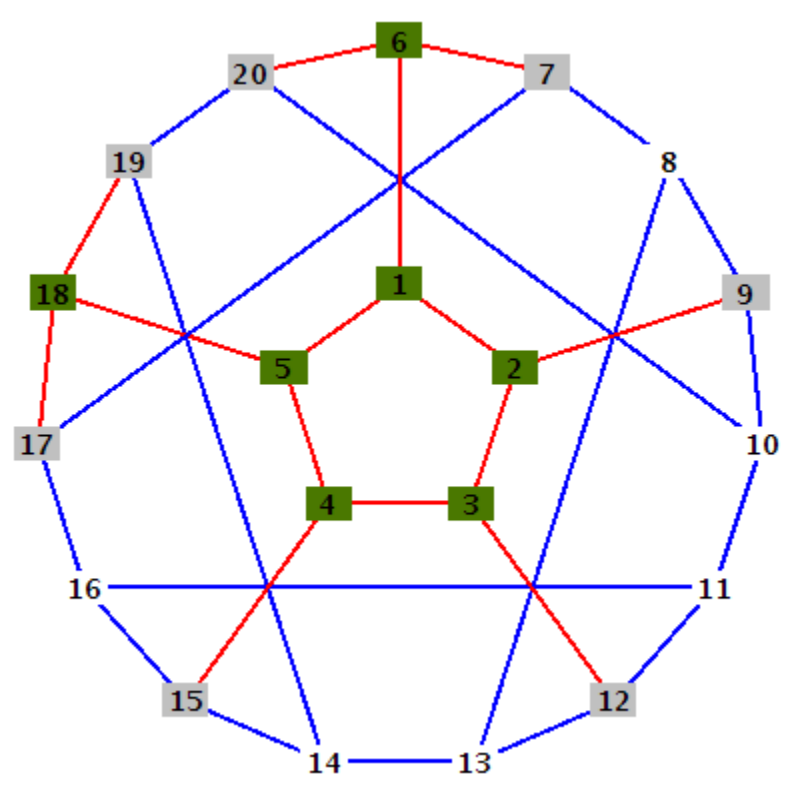
d[9]=3, π[9]=2

Paieška į plotį grafe



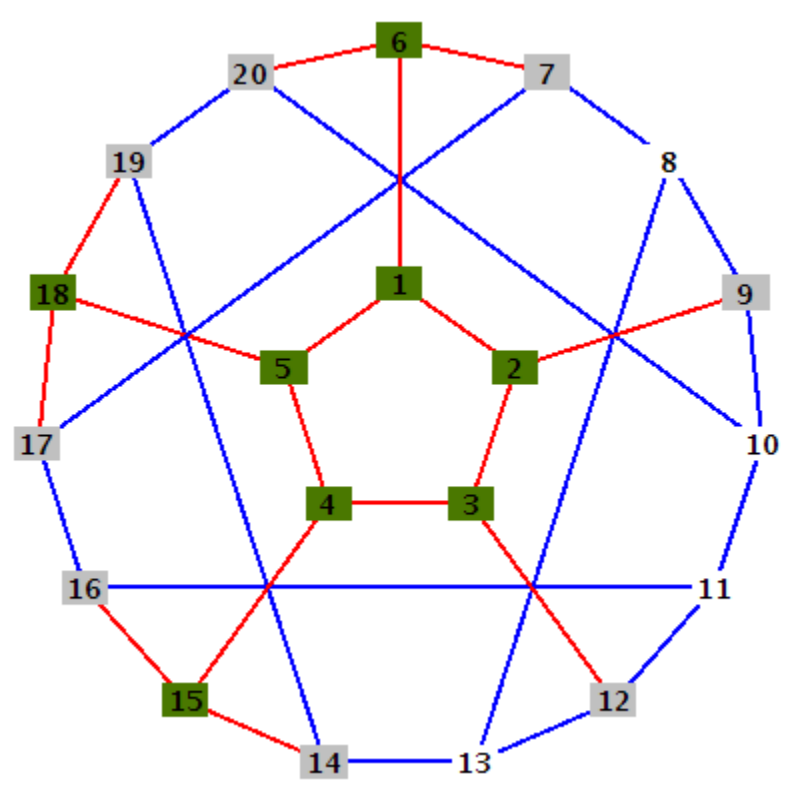
d[7]=3, π[7]=6, d[20]=3, π[20]=6

Paieška į plotį grafe



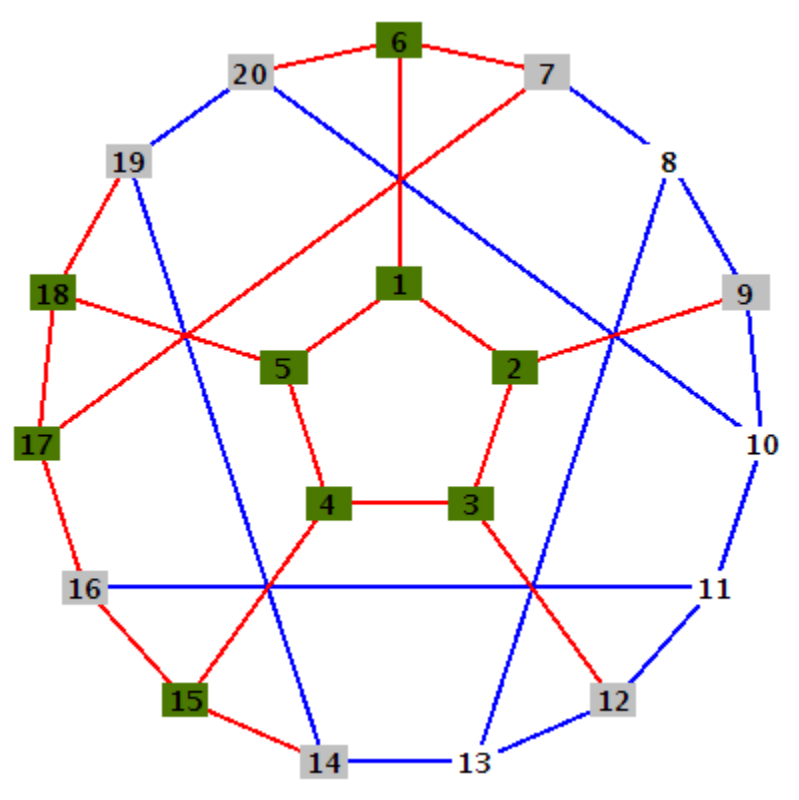
d[12]=3, π[12]=3

Paieška į plotį grafe

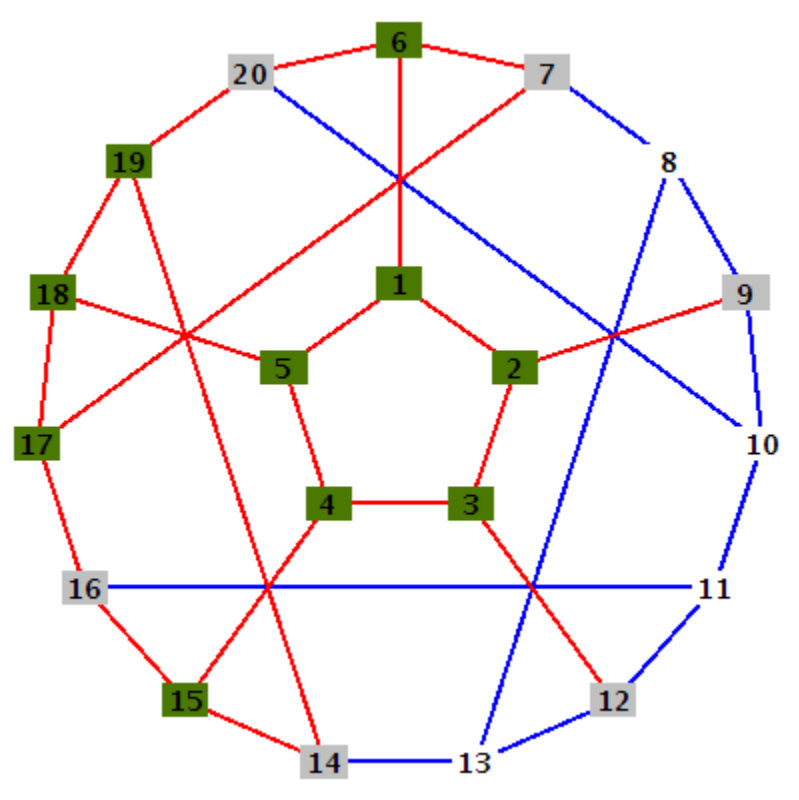


d[14]=3, π[14]=15, d[16]=3, π[16]=15

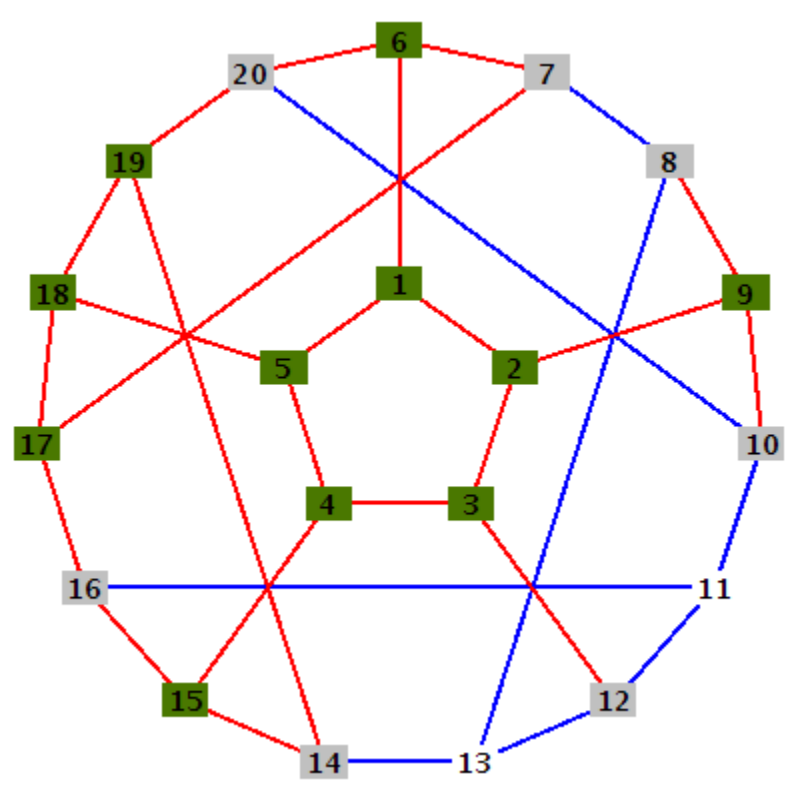
Paieška į plotį grafe



Paieška į plotį grafe

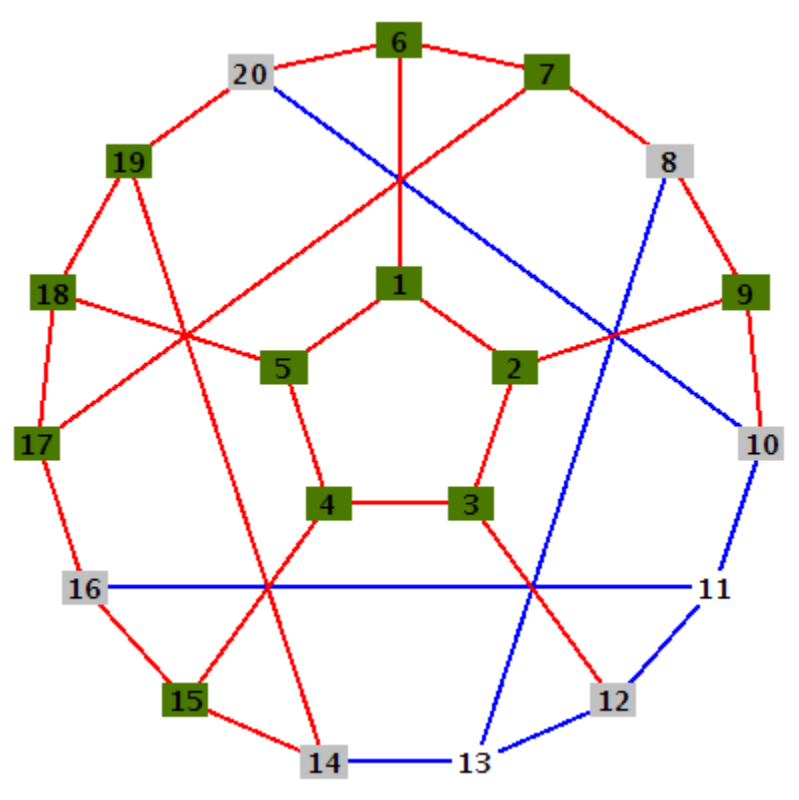


Paieška į plotį grafe

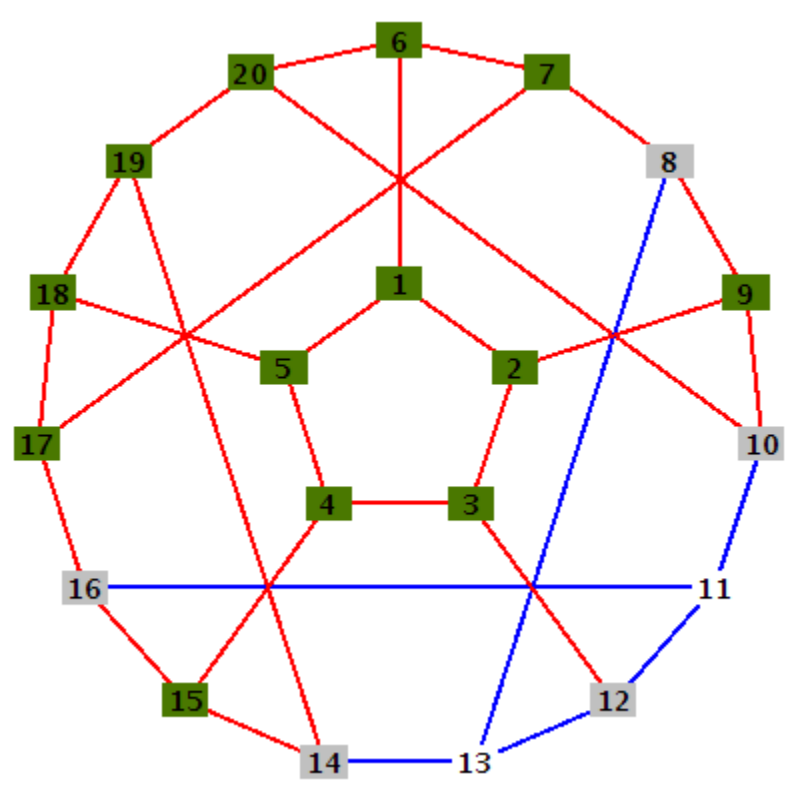


d[8]=4, π[8]=9, d[10]=4, π[10]=9

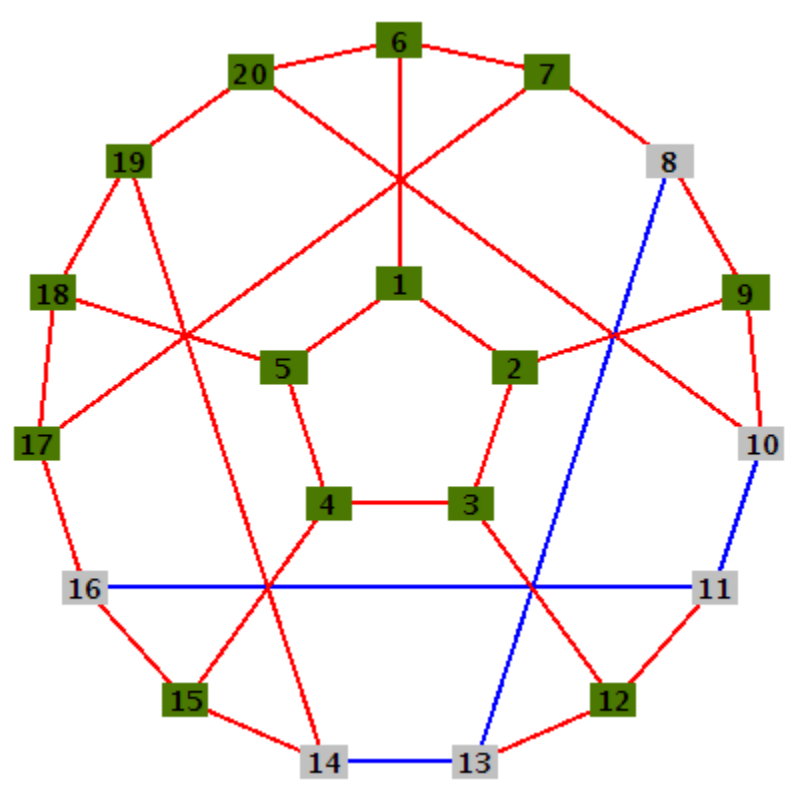
Paieška į plotį grafe



Paieška į plotį grafe

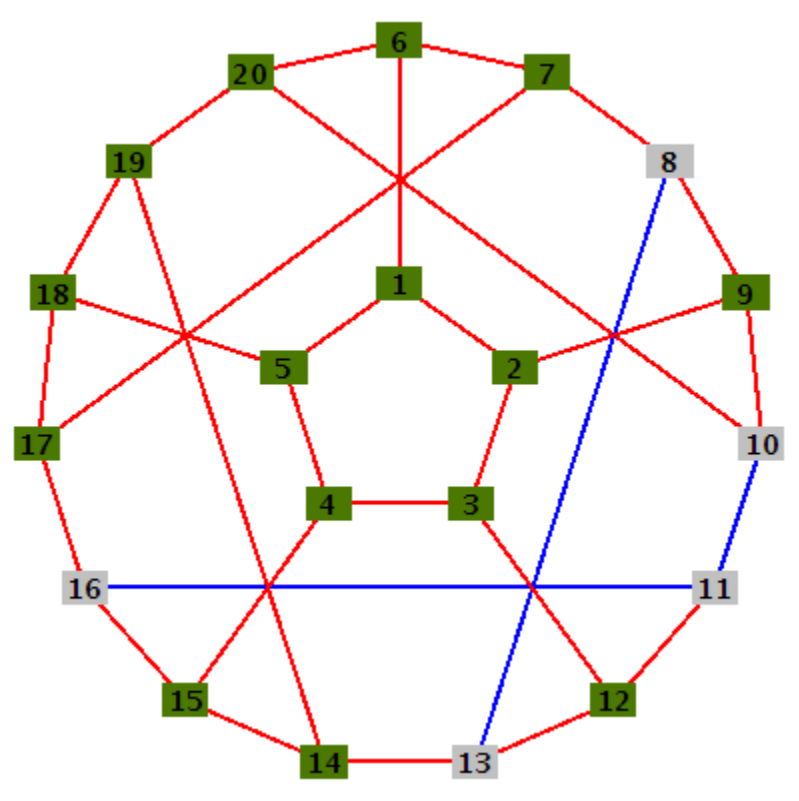


Paieška į plotį grafe

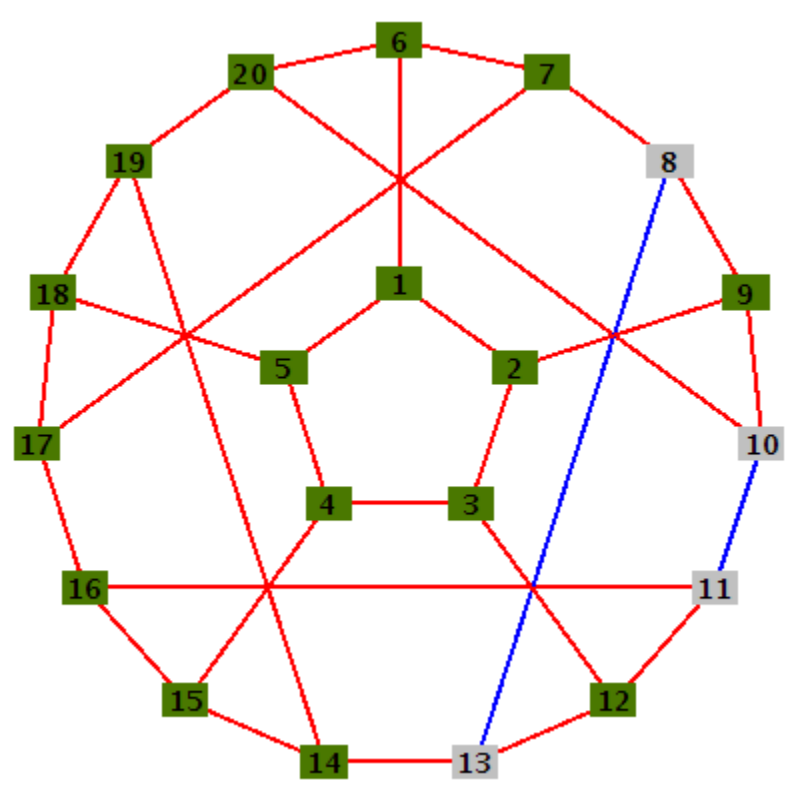


d[11]=4, π[11]=12, d[13]=4, π[13]=12

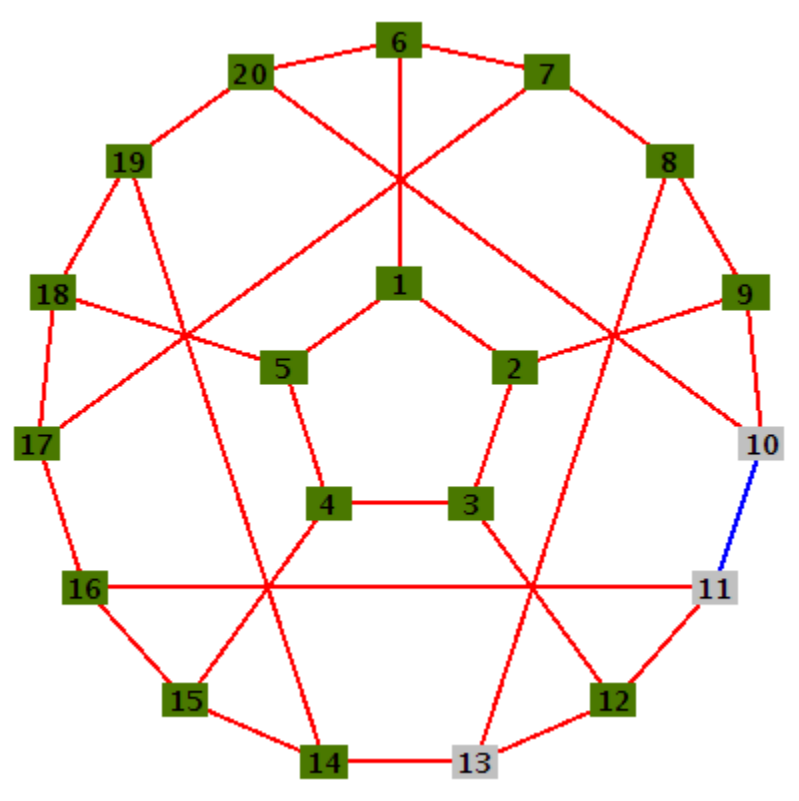
Paieška į plotį grafe



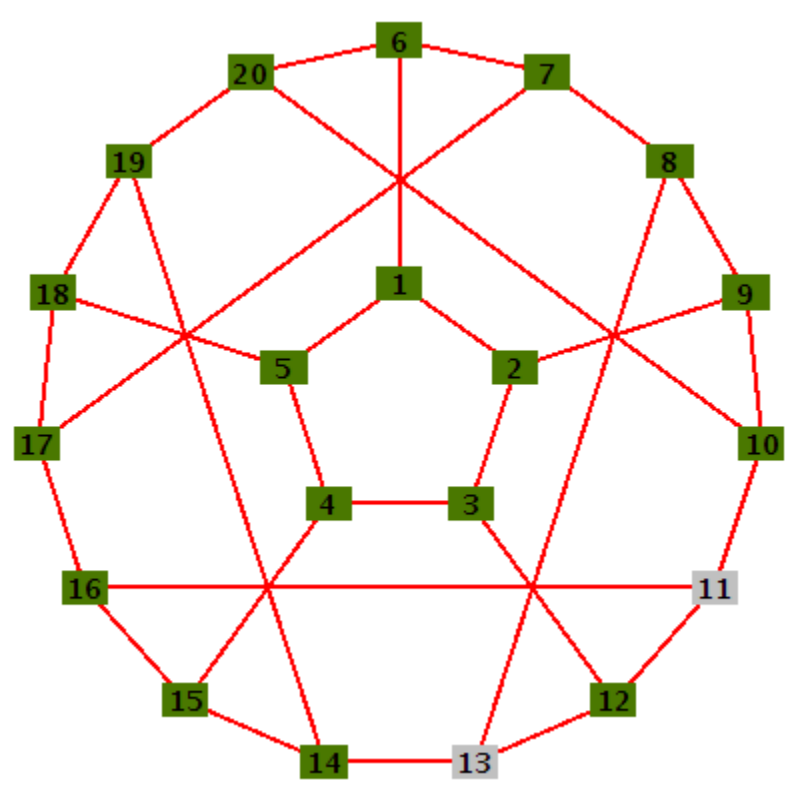
Paieška į plotį grafe



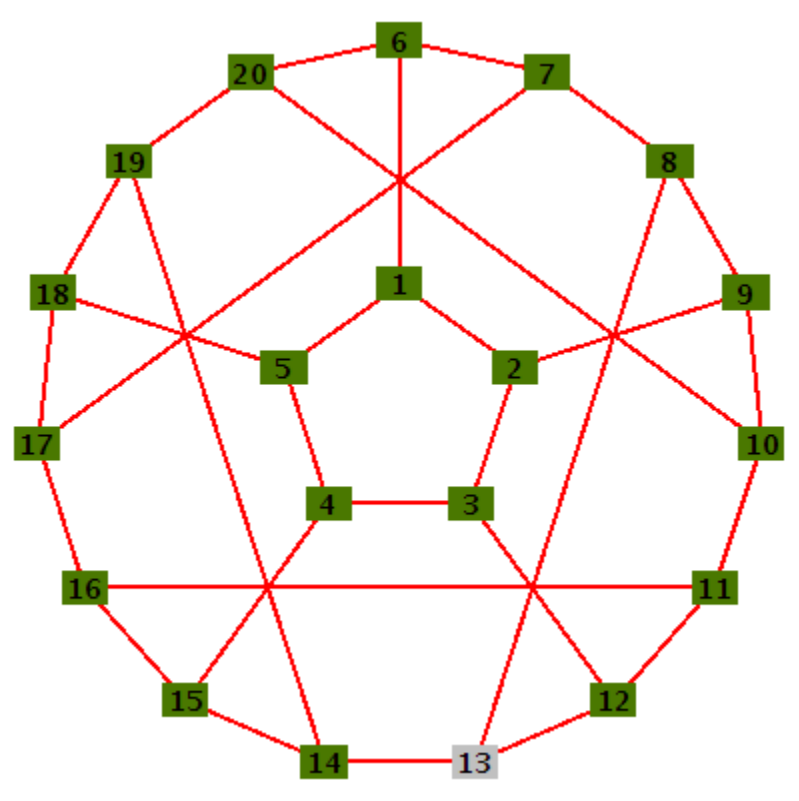
Paieška į plotį grafe



Paieška į plotį grafe

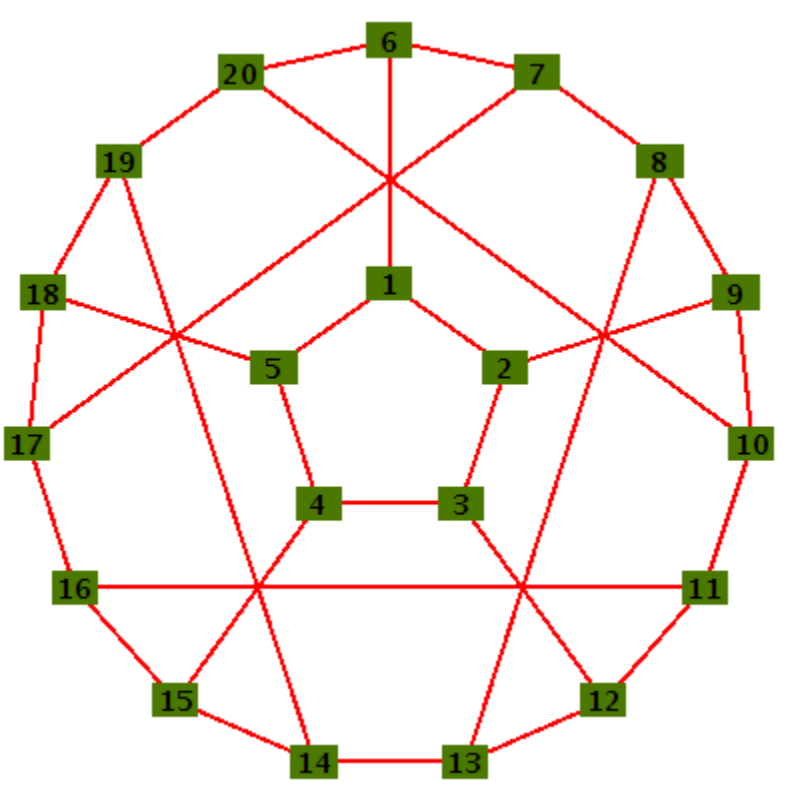


Paieška į plotį grafe



Paieška į plotį grafe

Atstumų atributai:
d[1]=1, d[2]=2,
d[3]=2, d[4]=1,
d[5]=0, d[6]=2,
d[7]=3, d[8]=4,
d[9]=3, d[10]=4,
d[11]=4, d[12]=3,
d[13]=4, d[14]=3,
d[15]=2, d[16]=3,
d[17]=2, d[18]=1,
d[19]=2, d[20]=3.



Tėvystės atributai:
π[1]=5, π[2]=1,
π[3]=4, π[4]=5,
π[5]=NIL, π[6]=1,
π[7]=6, π[8]=9,
π[9]=2, π[10]=9,
π[11]=12, π[12]=3,
π[13]=12, π[14]=15,
π[15]=4, π[16]=15,
π[17]=18, π[18]=5,
π[19]=18, π[20]=6.

Paieška į gylį (angl. Depth-first search)

- Tikslas – pradedant grafo (digrafo) viršūne *s,* apeiti grafą keliaujant jo briaunomis t . y. atrasti visas pasiekiamas grafo (digrafo) viršūnes.

- Viršūnių paieška vyksta pagal tokią strategiją:
  - Kita viršūnė atrandama iš ką tik atrastos viršūnės einant briauna.
  - Jei tokių briaunų nėra, grįžtama į prieš tai nagrinėtą viršūnę, ir kurios vykdoma paieška.

Paieškos į gylį algoritmas

rekursija

```
P-Gyl(G):
1 for each u ∈ V
2       do c[u] ← balta
3             π[u] ← NIL
4 laikas ← 0
5 for each u ∈ V
6       do if c[u]=balta
7             then VISIT(u)
```

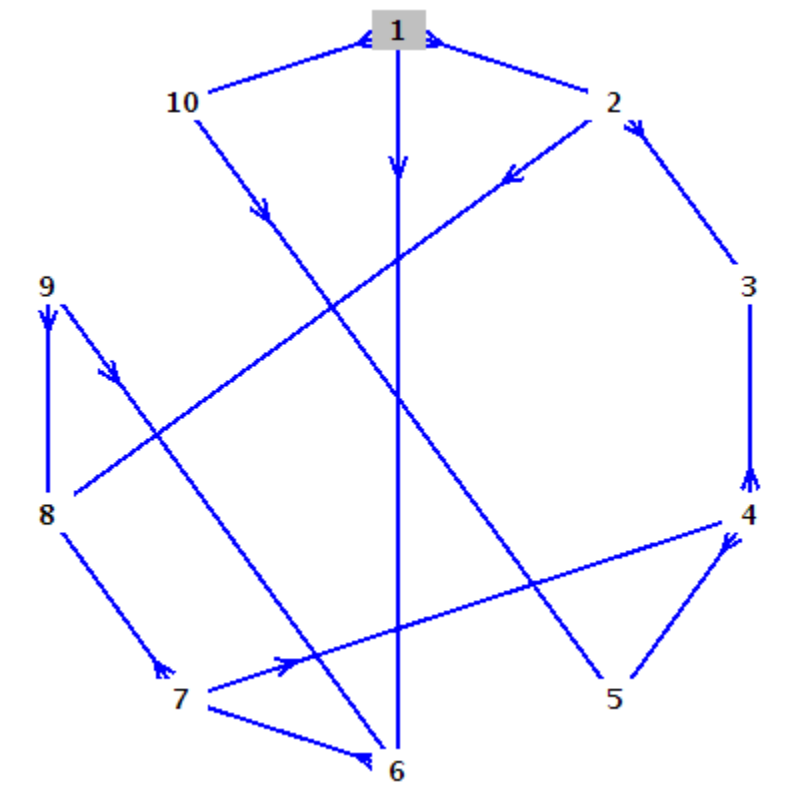
```
VISIT(u):
1 c[u] ← pilka
2 d[u] ← laikas ← laikas+1
3 for each v ∈ Adj[u]
4       do if c[v]=balta
5             then π[v] ← u
6                  VISIT(v)
7 c[u] ← juoda
8 f[u] ← laikas ← laikas+1
END
```

Atributai:
c[u] – spalvos atributas,
d[u] – laiko atributas (viršūnės u apdorojimo pradžia),
f[u] – laiko atributas (viršūnės u apdorojimo pabaiga),
π[u] – tėvystės atributas.

Paieškos į gylį algoritmo sudėtingumas

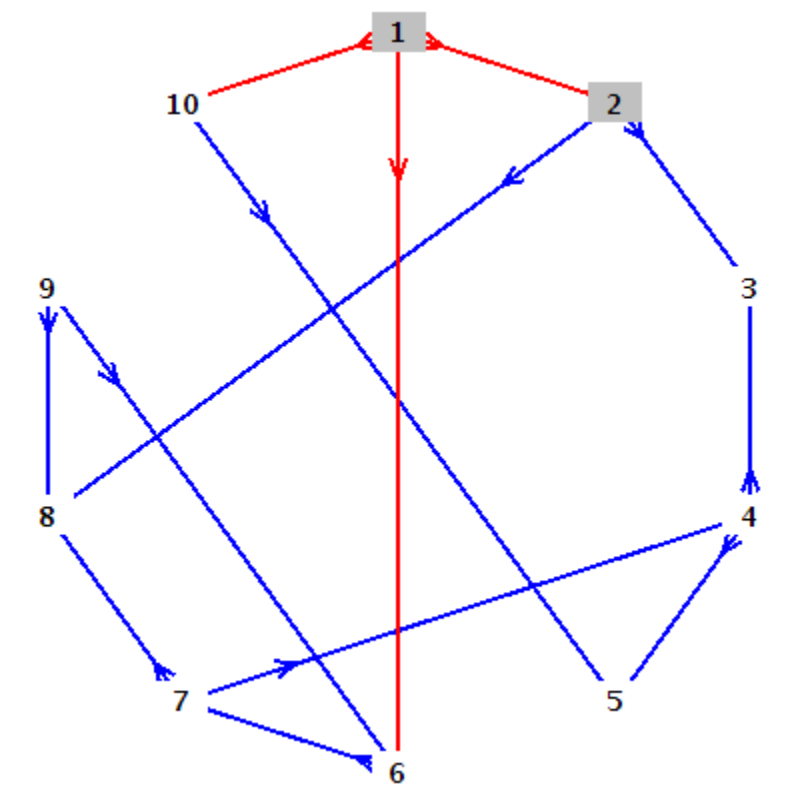
- 1–3 eilutėse komandos vykdomos O(|V|)=O(n) laiko.
- 5–6 eilutėse rekursija VISIT(u) iškviečiama ne daugiau |V| kartų.
- VISIT(u) vykdoma kiekvienai sąrašo Adj[u] viršūnei, todėl iš viso ne daugiau kaip O(|E|)=O(m) laiko vienetų.
- Vadinasi, paieškos į gylį sudėtingumas O(n+m).

Paieška į gylį digrafe



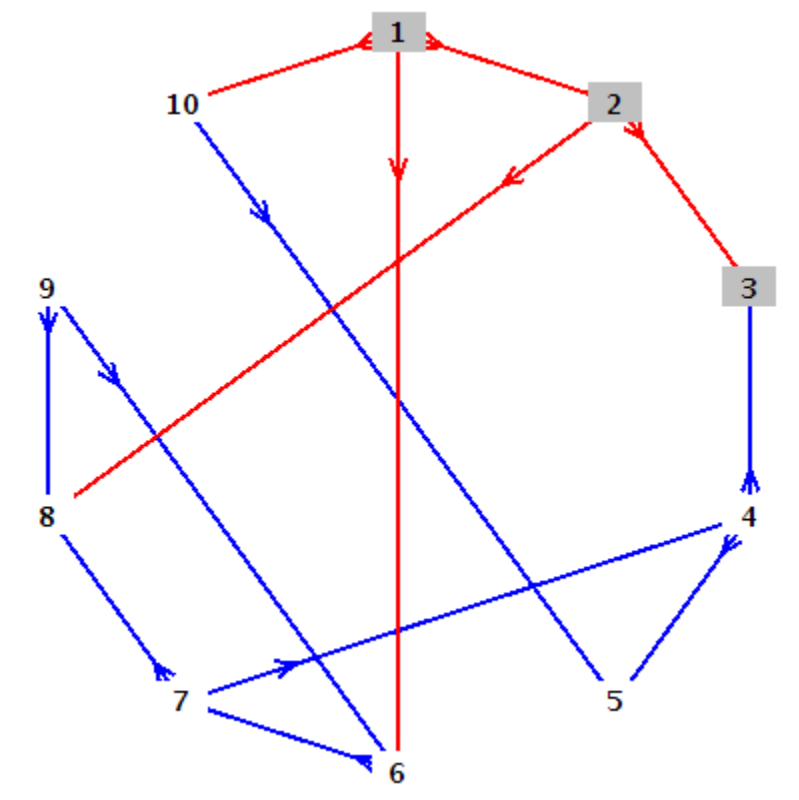
π[1]=NIL, d[1]=1

Paieška į gylį digrafe



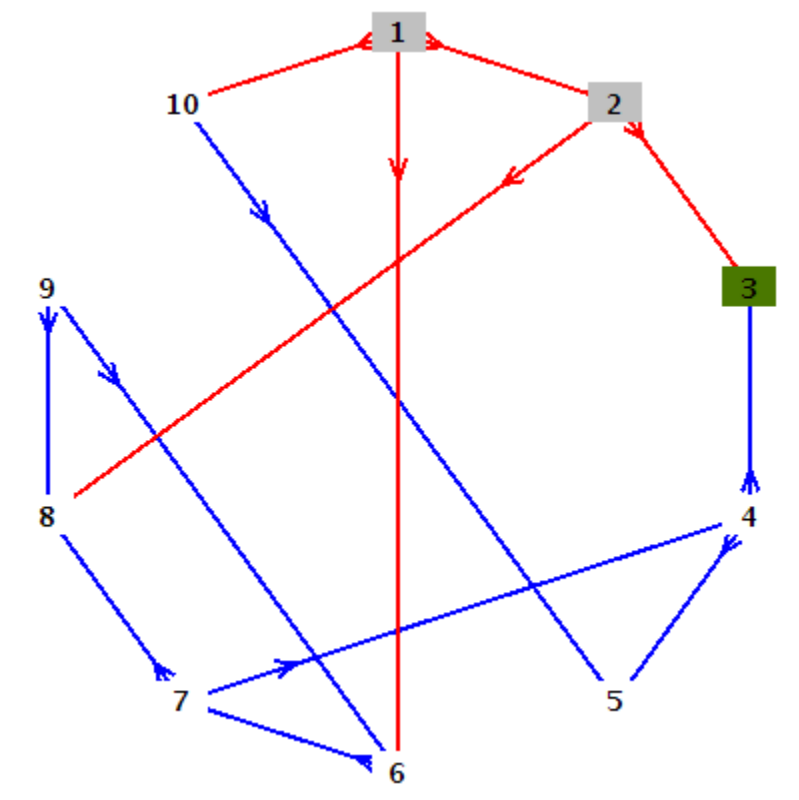
π[2]=1, d[2]=2

Paieška į gylį digrafe



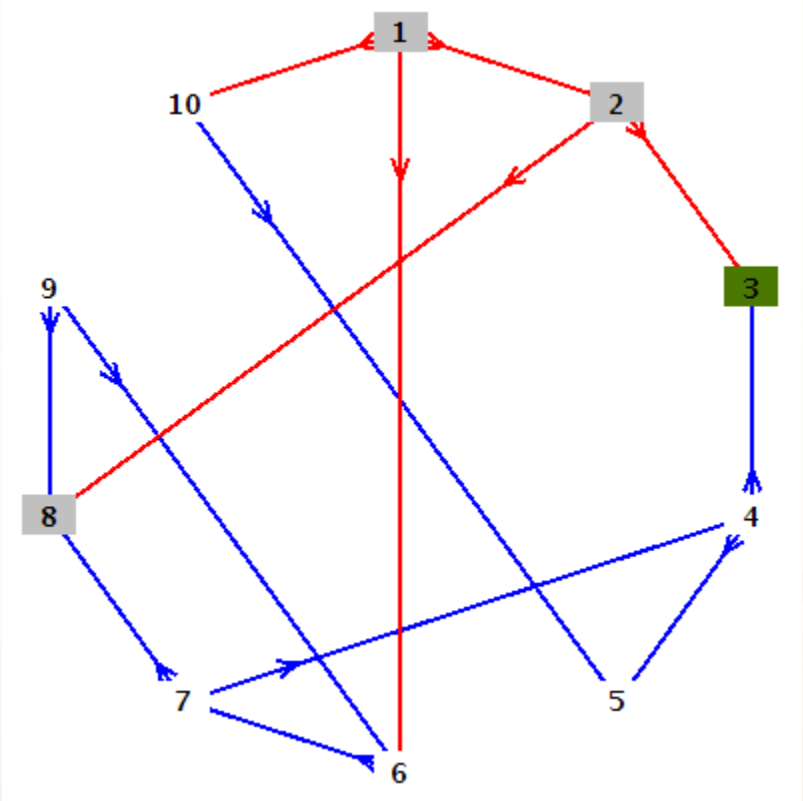
π[3]=2, d[3]=3

Paieška į gylį digrafe



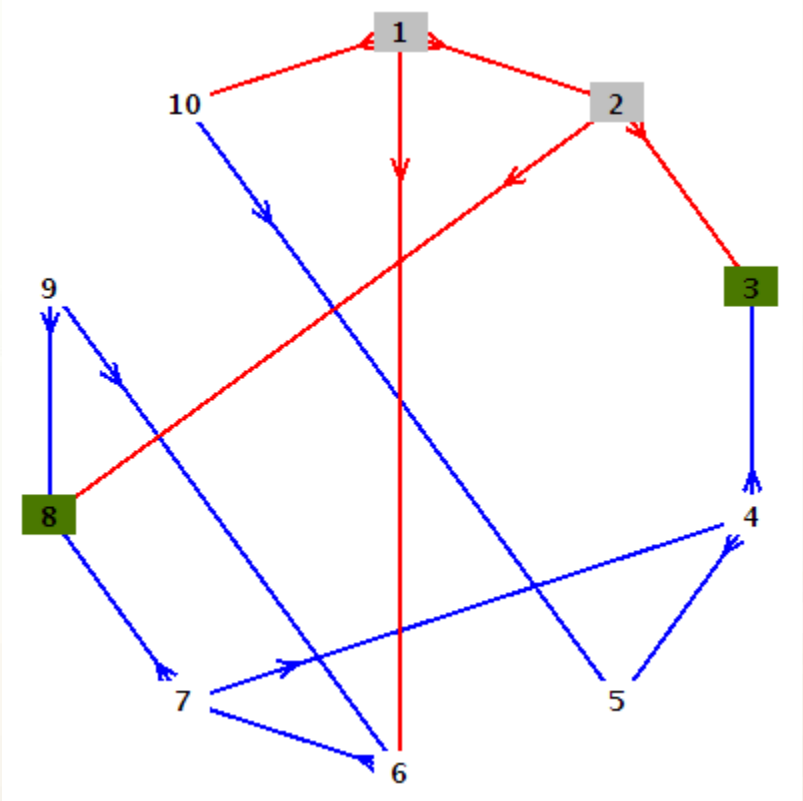

```
f[3]=4
```

Paieška į gylį digrafe



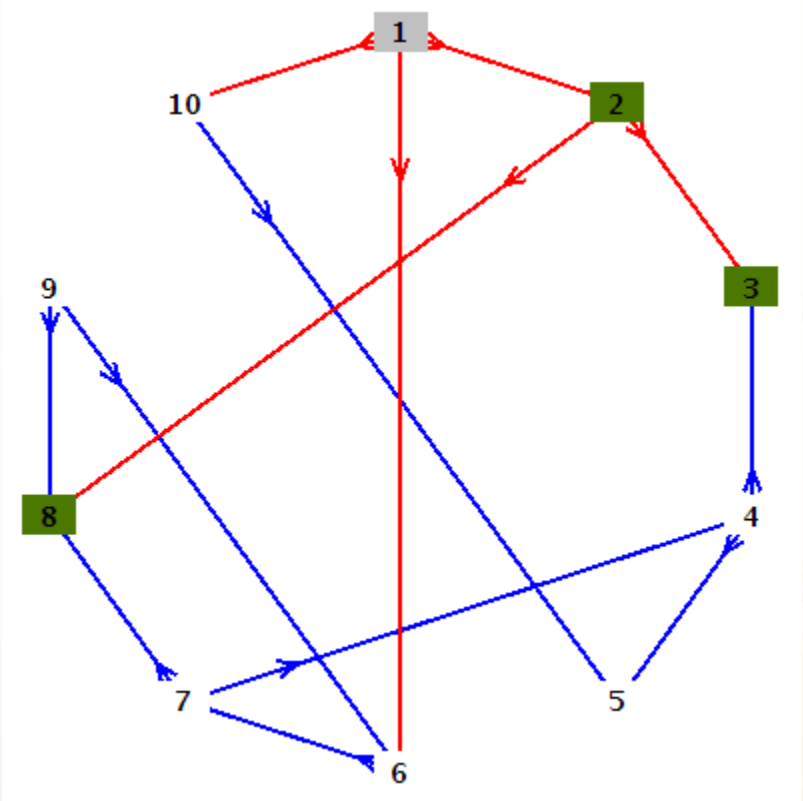
π[8]=2, d[8]=5

Paieška į gylį digrafe



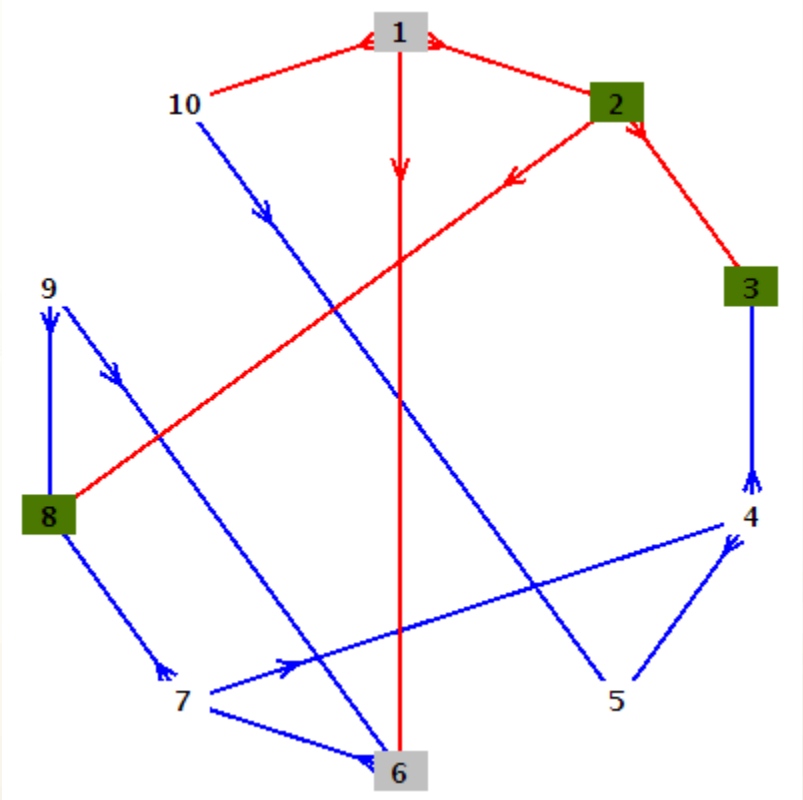

```
f[8]=6
```

Paieška į gylį digrafe



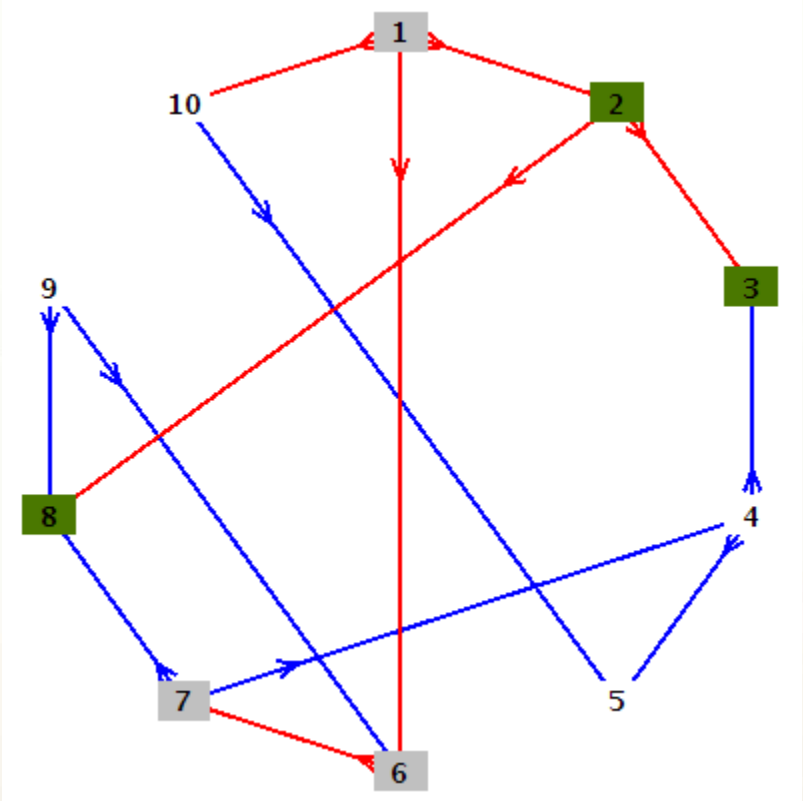

```
f[2]=7
```

Paieška į gylį digrafe



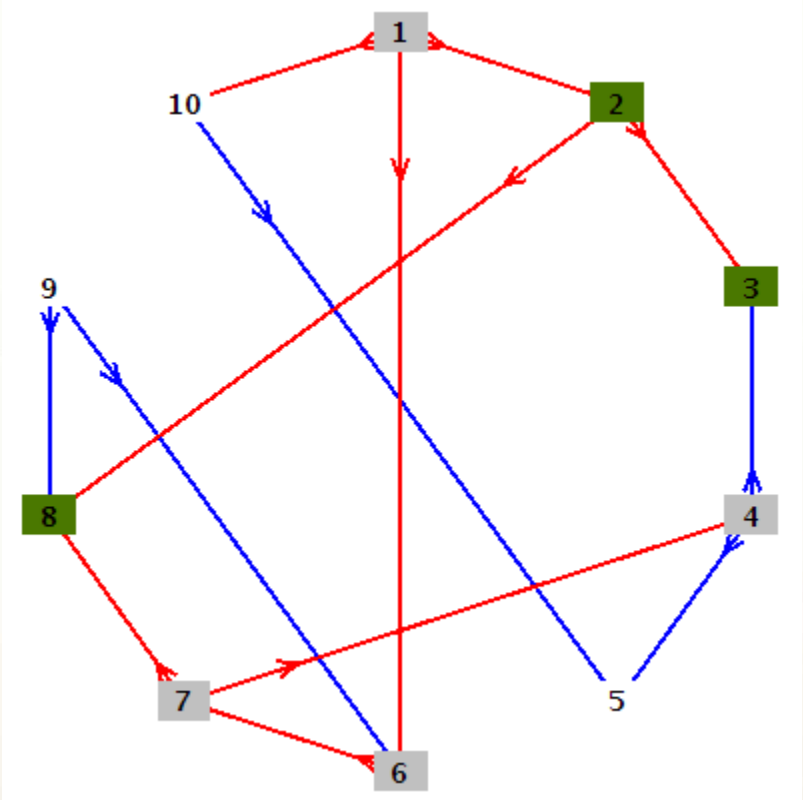
π[6]=1, d[6]=8

Paieška į gylį digrafe



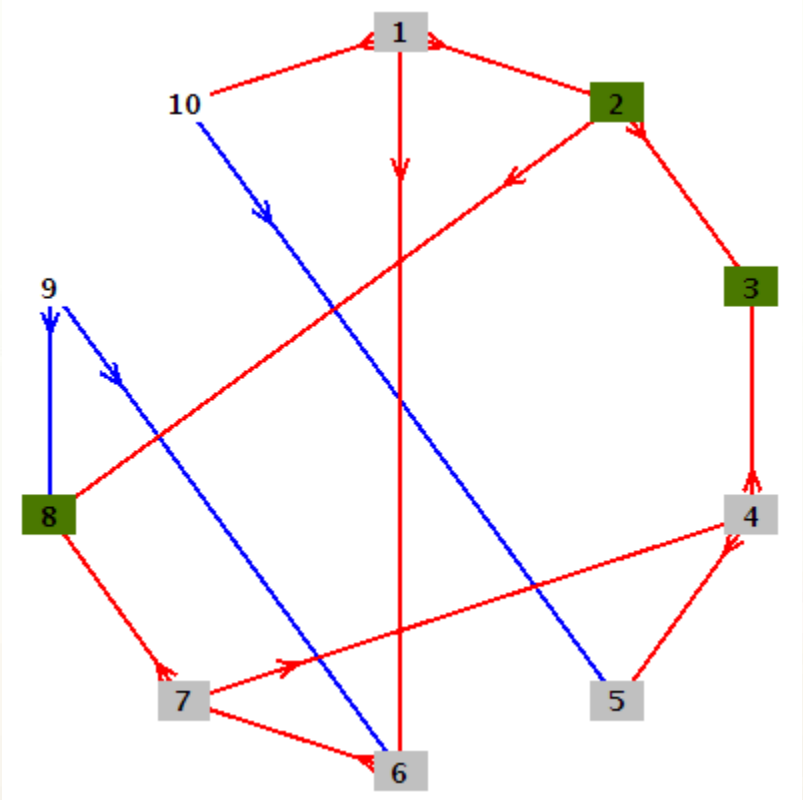
π[7]=6, d[7]=9

Paieška į gylį digrafe



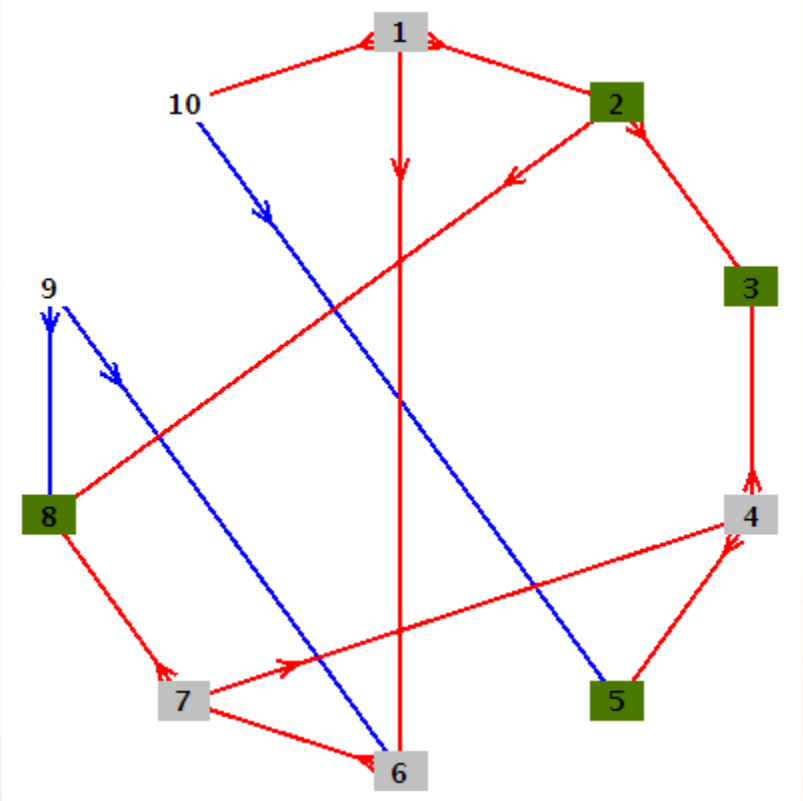
π[4]=7, d[4]=10

Paieška į gylį digrafe



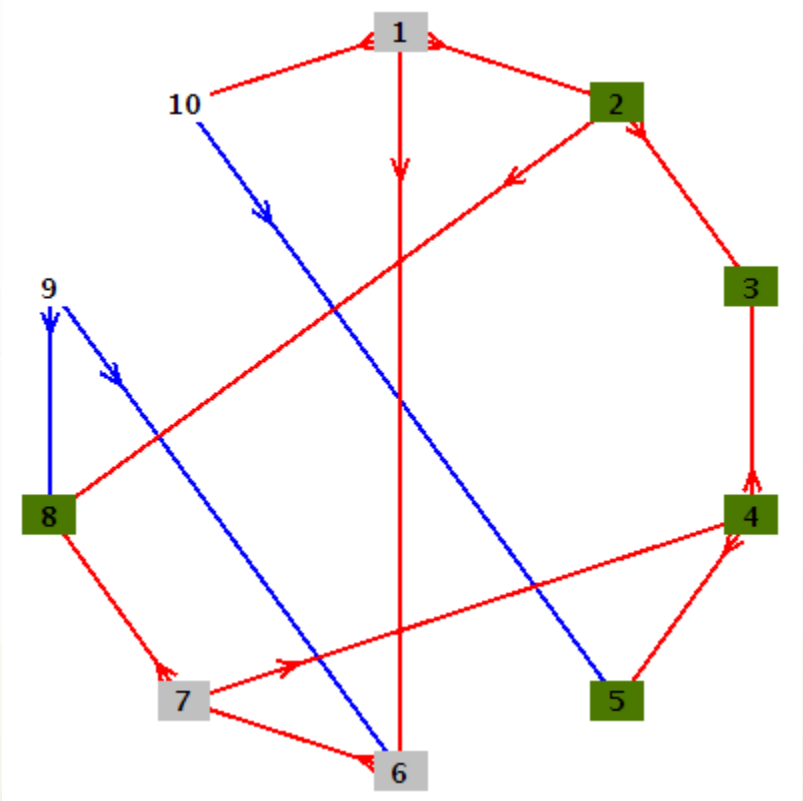
π[5]=4, d[5]=11

Paieška į gylį digrafe



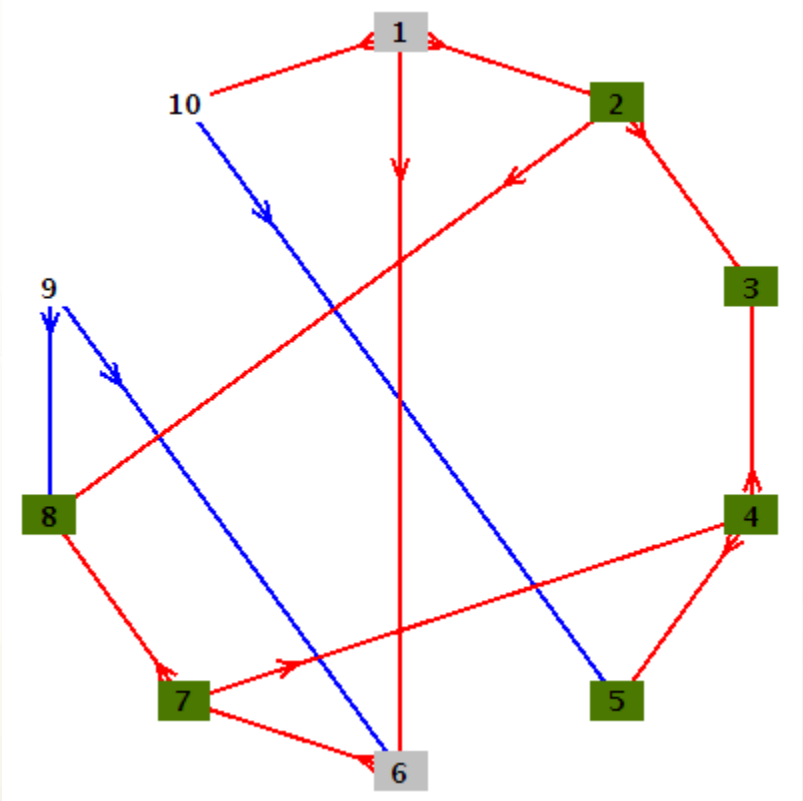

```
f[5]=12
```

Paieška į gylį digrafe



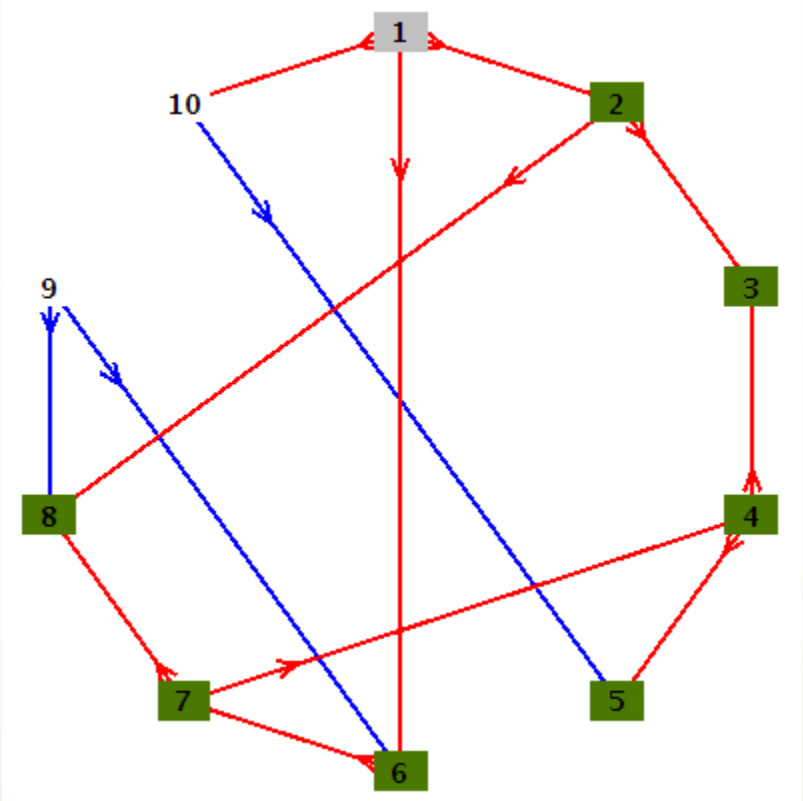

```
f[4]=13
```

Paieška į gylį digrafe



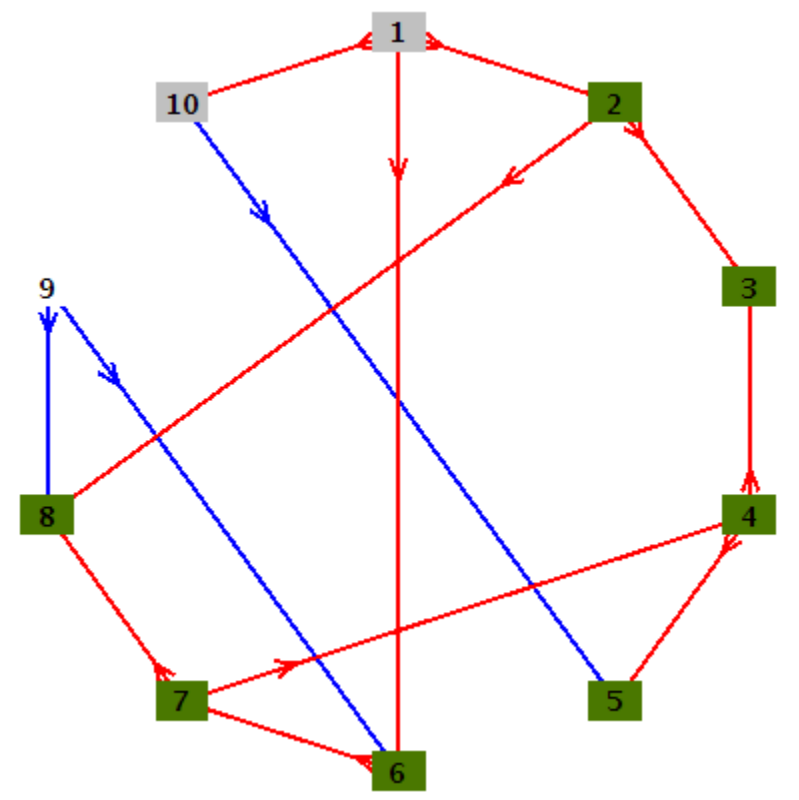

```
f[7]=14
```

Paieška į gylį digrafe



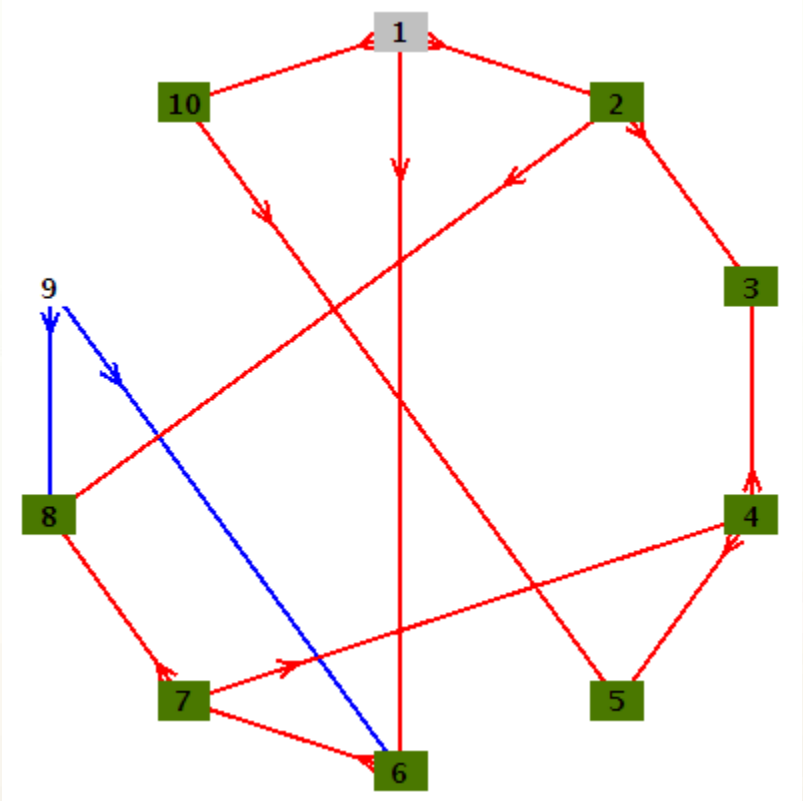
f[6]=15

Paieška į gylį digrafe



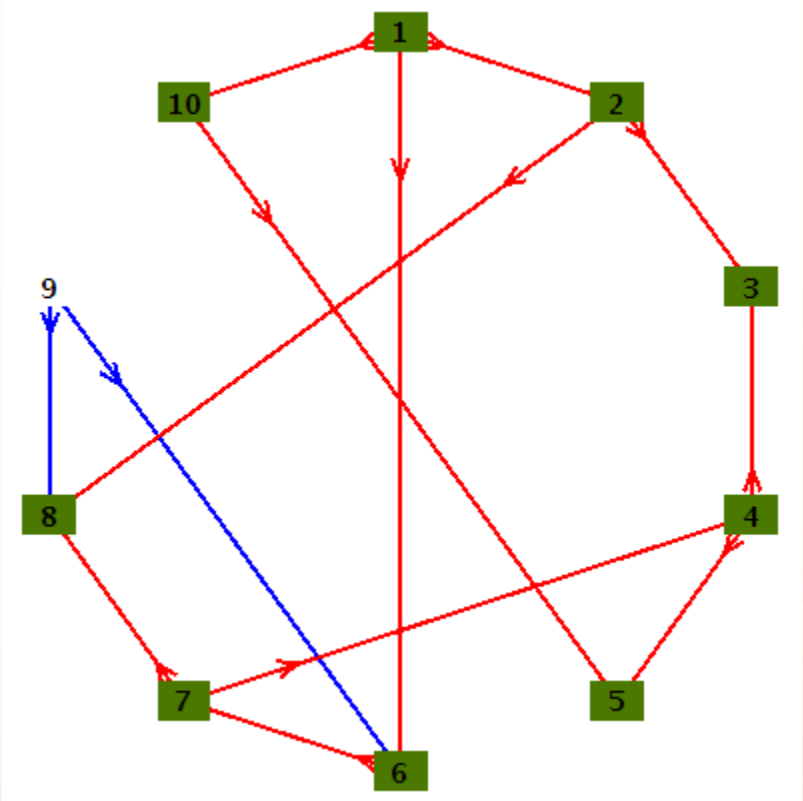
π[10]=1, d[10]=16

Paieška į gylį digrafe



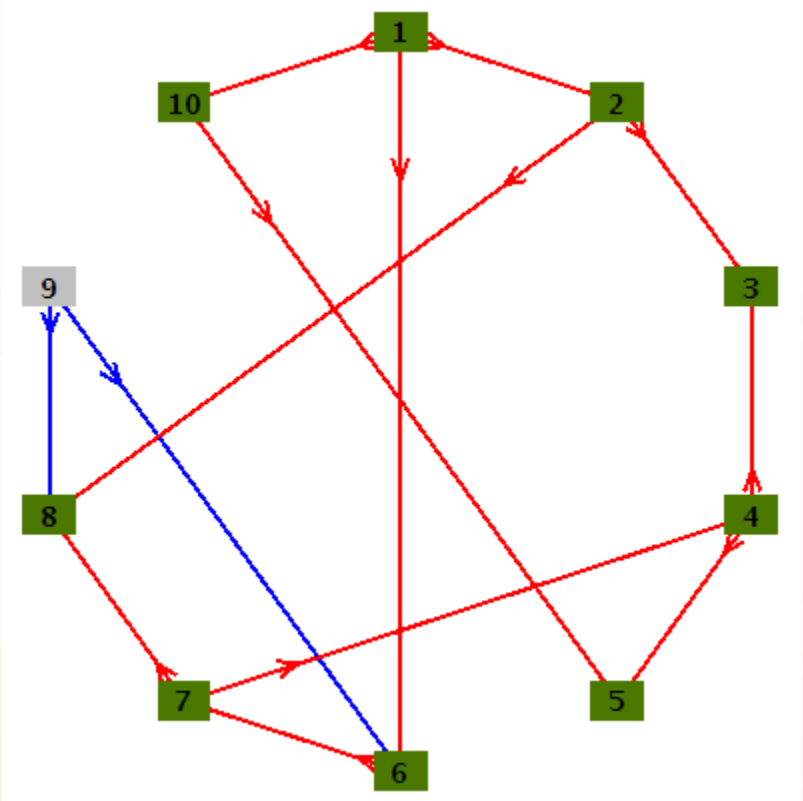

```
f[10]=17
```

Paieška į gylį digrafe



f[1]=18

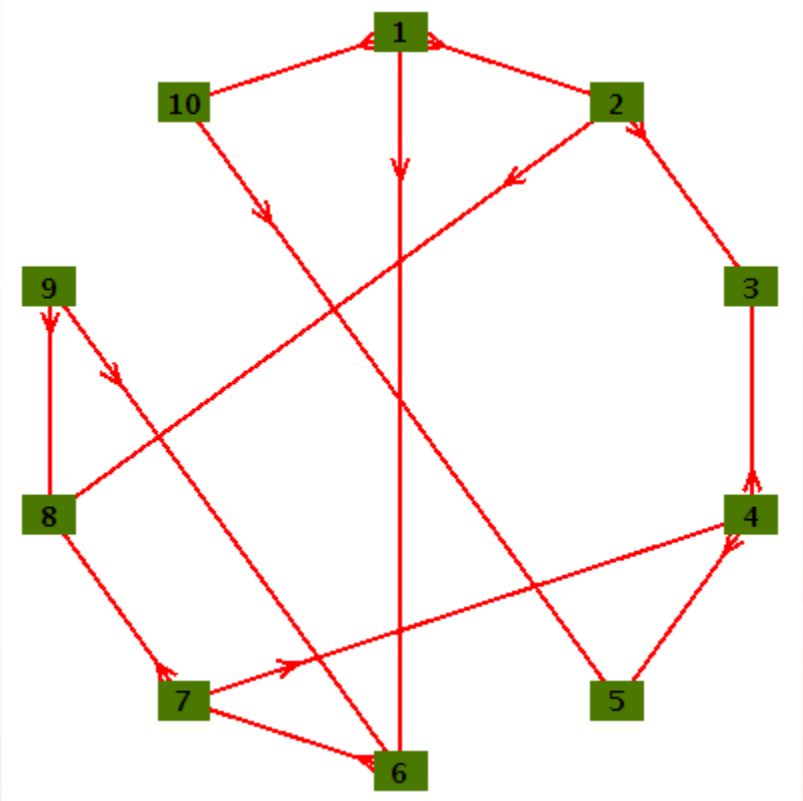
Paieška į gylį digrafe



d[9]=19

Paieška į gylį digrafe

Laiko atributai:
d[1]=1, d[2]=2,
d[3]=3, f[3]=4,
d[8]=5, f[8]=6,
f[2]=7, d[6]=8,
d[7]=9, d[4]=10,
d[5]=11, f[5]=12,
f[4]=13, f[7]=14,
f[6]=15, d[10]=16,
f[10]=17, f[1]=18,
d[9]=19, f[9]=20.

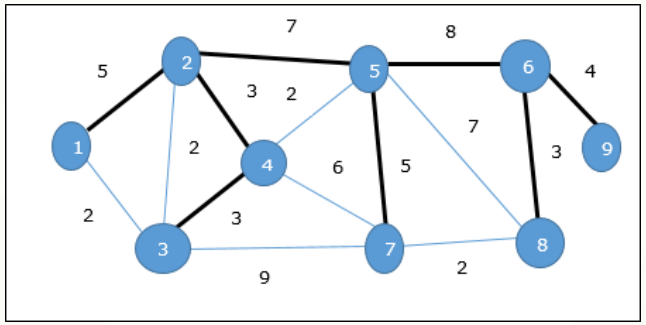


Tėvystės atributai:
π[1]=NIL, π[2]=1,
π[3]=2, π[8]=2,
π[6]=1, π[7]=6,
π[4]=7, π[5]=4,
π[10]=1.

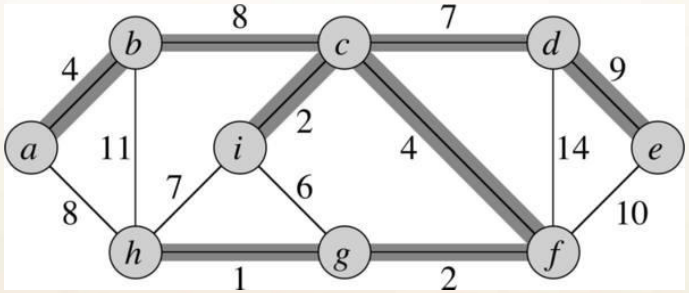
f[9]=20

Minimalūs jungiantieji medžiai

- Jungiančiuoju medžiu vadinamas jungaus svorinio grafo G pografis (medis), kuriam priklauso visos grafo G viršūnės:



- Minimaliu jungiančiuoju medžiu vadinamas jungaus svorinio grafo G jungiantysis medis, kurio briaunų svorių suma yra minimali:



Minimalaus jungiančiojo medžio paieška

- Kruskalio algoritmas (1956, Joseph Kruskal)
- Primo algoritmas (1957, Robert C. Prim)
- Solino algoritmas (1965, Sollin):

It was first published in **1926 by Otakar Borůvka** as a method of constructing an efficient electricity network for Moravia. The algorithm was **rediscovered by Choquet in 1938**; **again by Florek, Łukasiewicz, Perkal, Steinhaus, and Zubrzycki in 1951**; and **again by Sollin in 1965**. Because Sollin was the only computer scientist in this list living in an English speaking country, this algorithm is frequently called Sollin's algorithm, especially in the parallel computing literature.

Kruskalio algoritmas

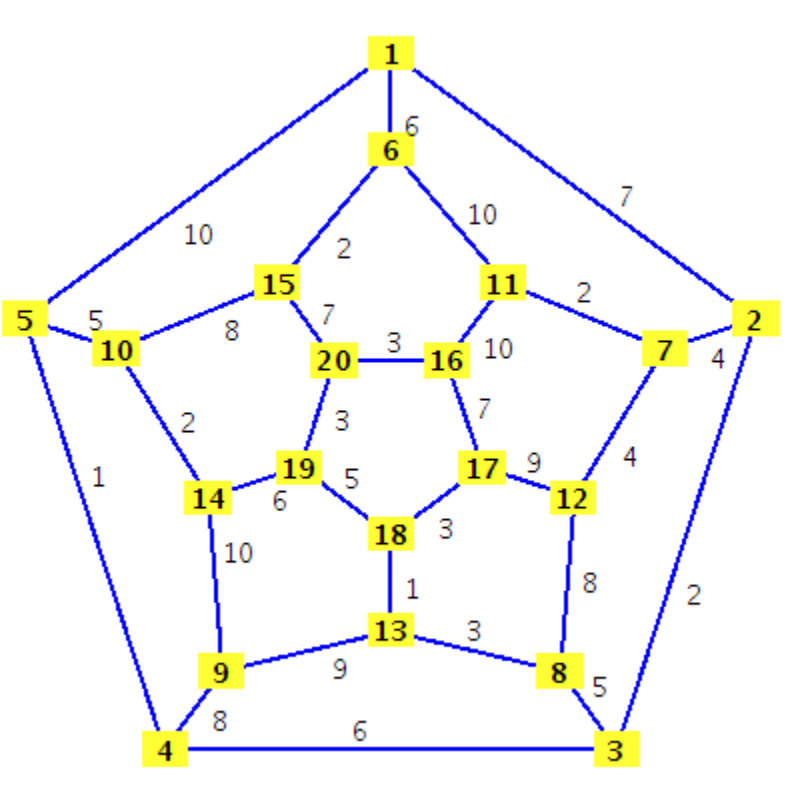
MJM-Kruskal (G, w):

1. $A \leftarrow \emptyset$
2. **for each** $v \in V$
3. **do** sukurk $S(v)$
4. surūšiuok E briaunas pagal $w(uv)$ didėjimą
5. **for each** $uv \in E$
6. **do if** $S(u) \neq S(v)$
7. **then** $A \leftarrow A \cup \{uv\}$
8. sukurk $S(u) \cup S(v)$
9. RETURN A

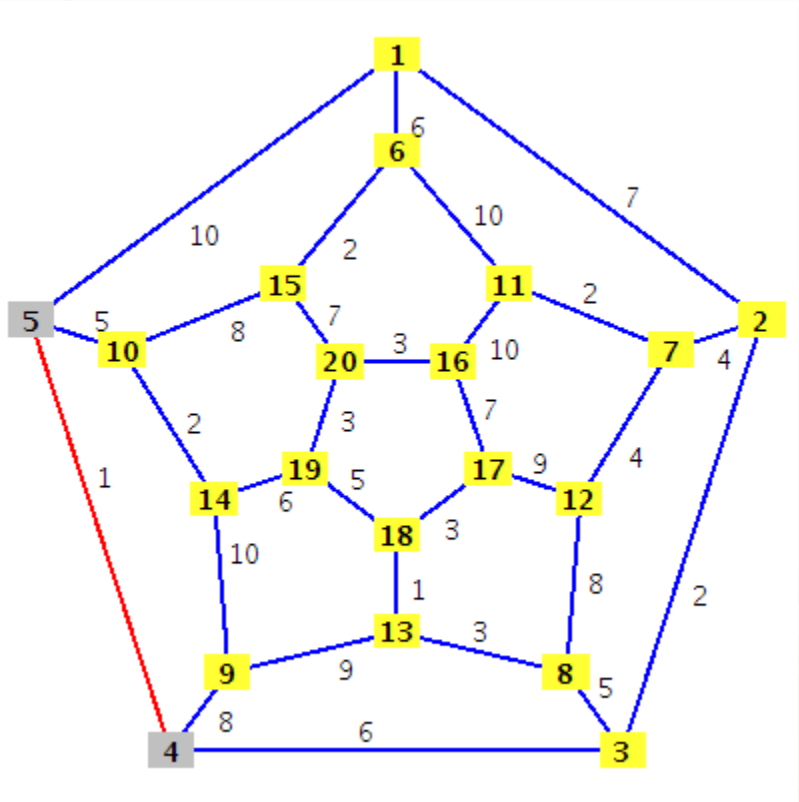
Kruskalio algoritmo vykdymo metu generuojamos laikinos grafo viršūnių aibės S(v), kurios lyginamos ir apjungiamos (formuojamas jungiantysis miškas).

Ilgiausiai vykdoma procedūra 4 eilutėje (galima pritaikyti, pavyzdžiui, greito rūšiavimo algoritmą *QuickSort*), todėl Kruskalio algoritmo sudėtingumas O(m log m).

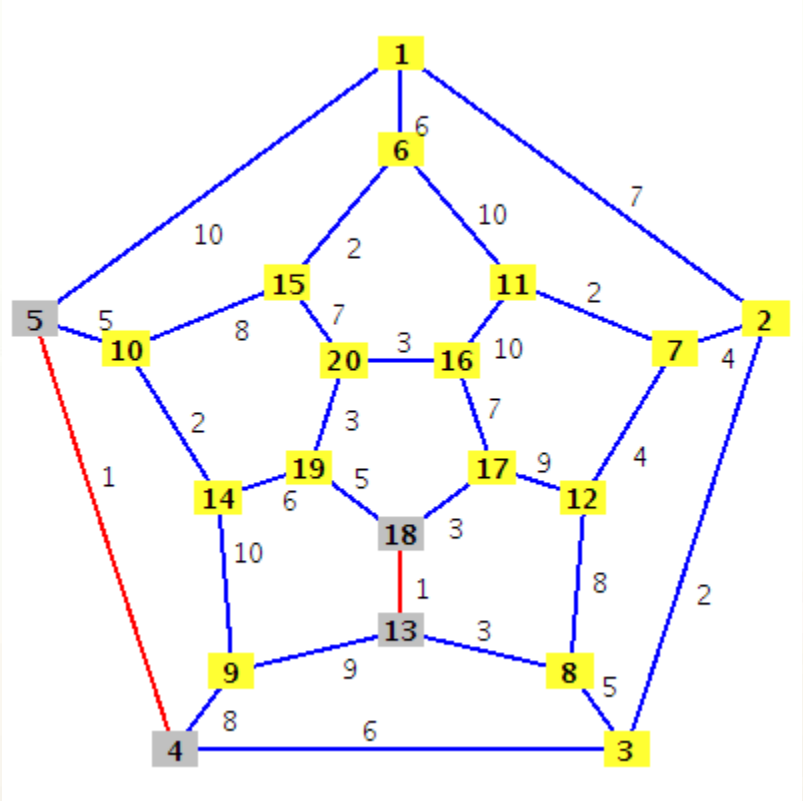
Kruskalio algoritmo vizualizacija



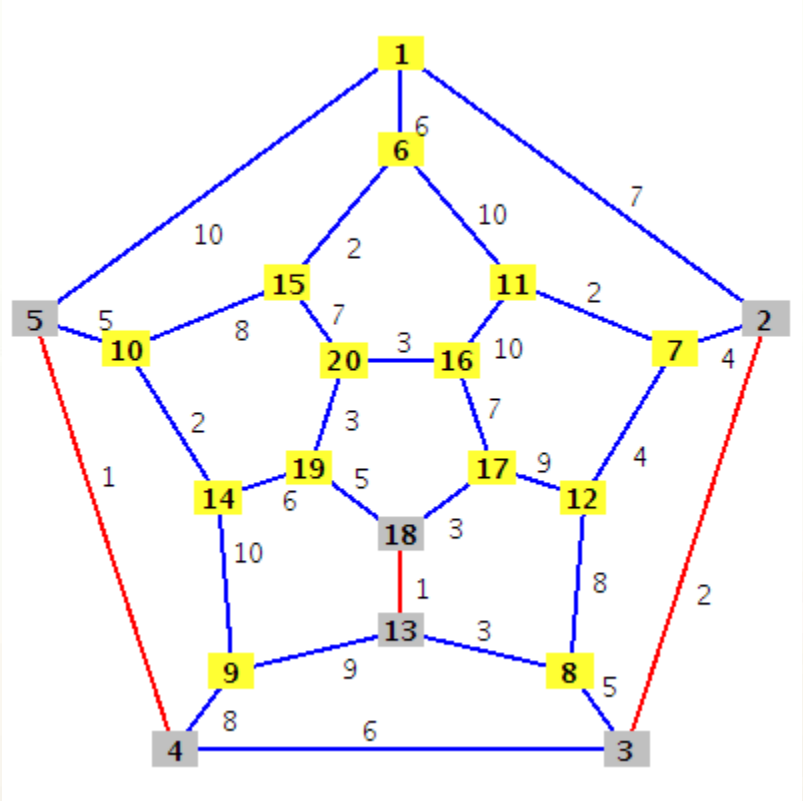
Kruskalio algoritmo vizualizacija



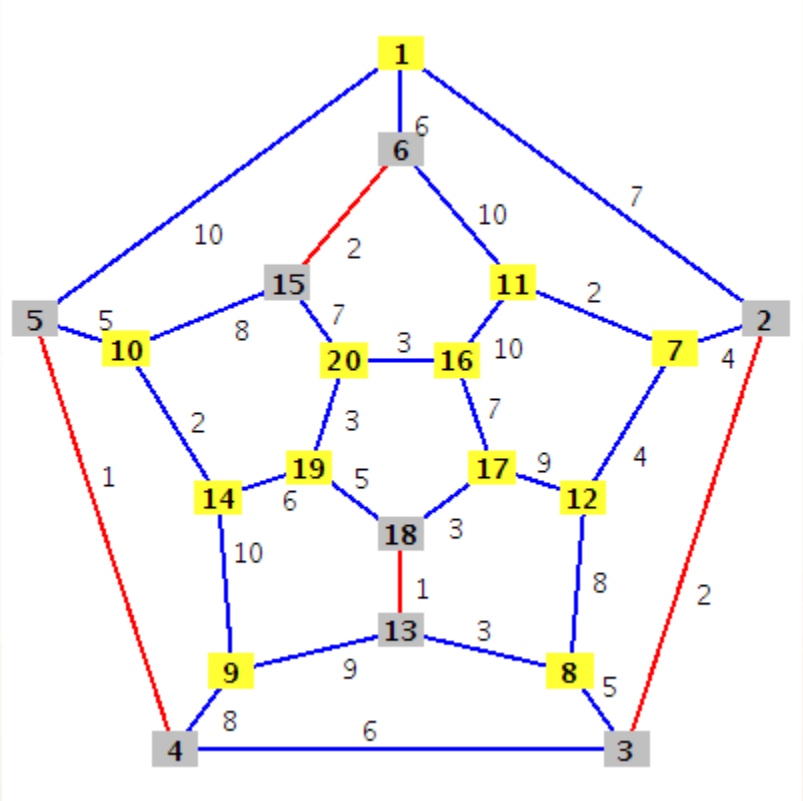
Kruskalio algoritmo vizualizacija



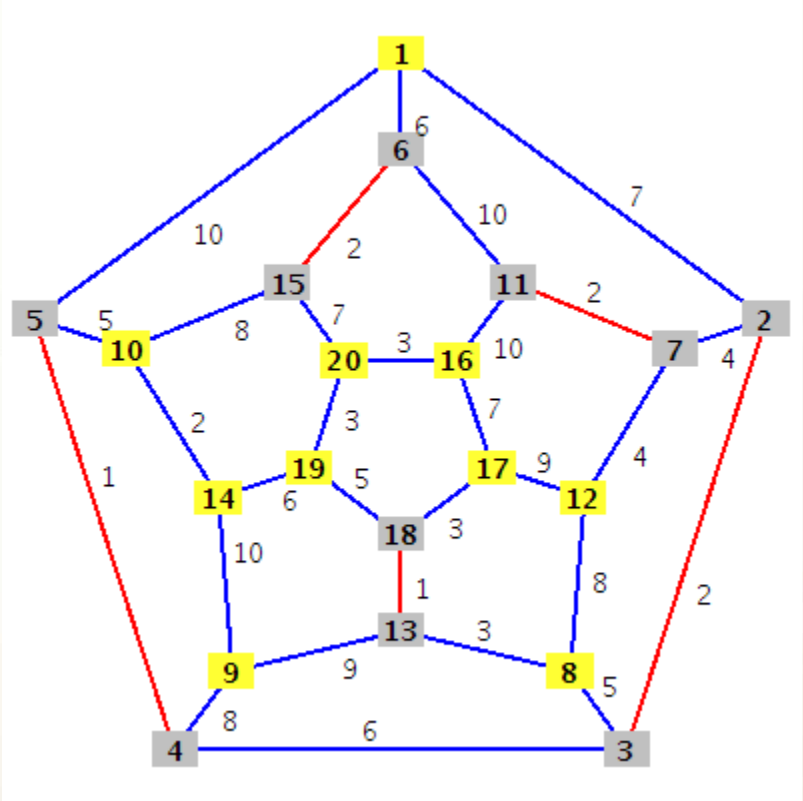
Kruskalio algoritmo vizualizacija



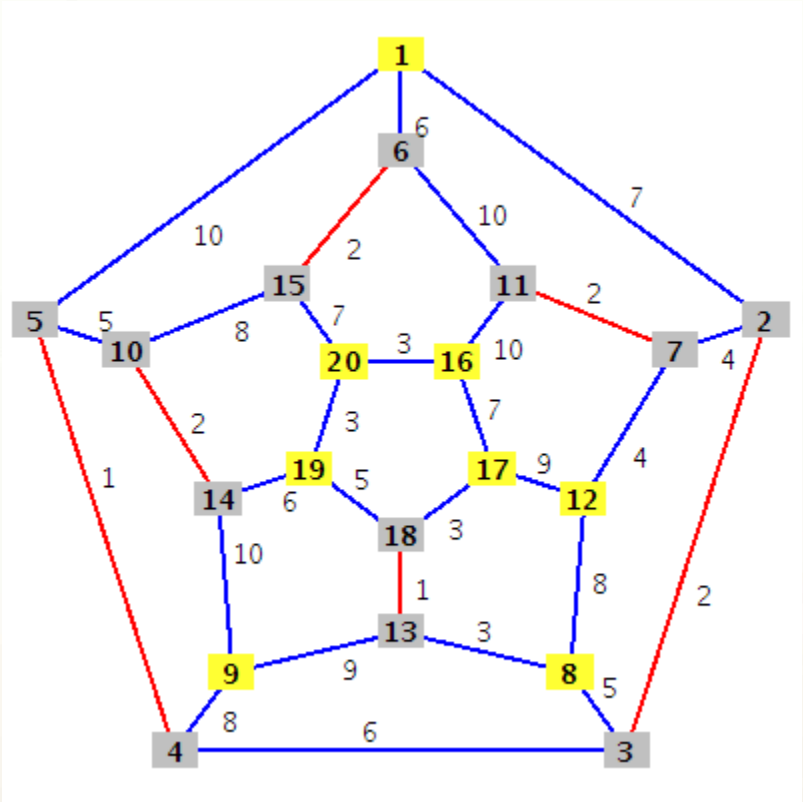
Kruskalio algoritmo vizualizacija



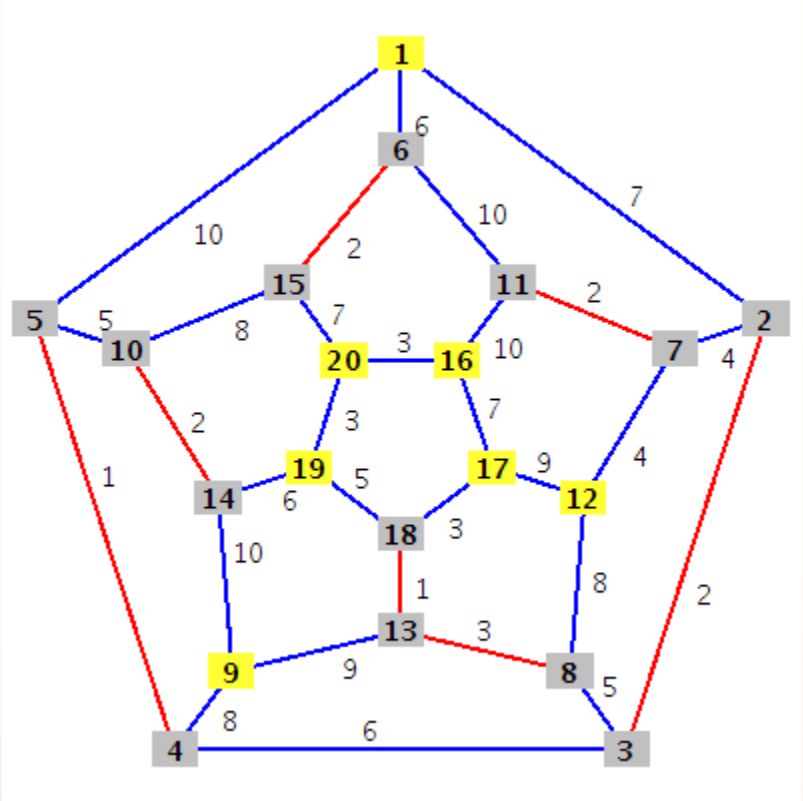
Kruskalio algoritmo vizualizacija



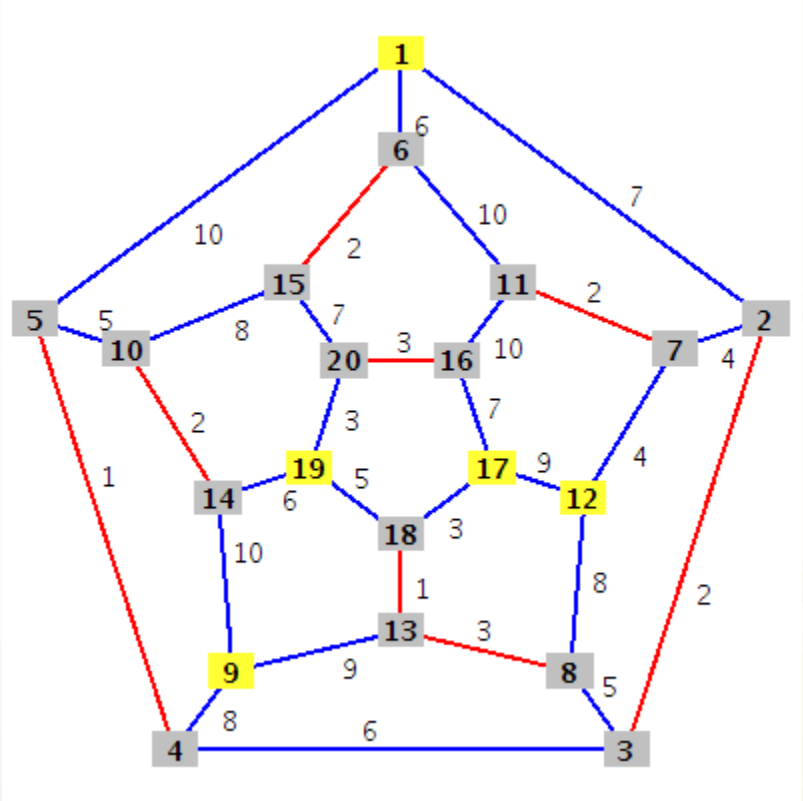
Kruskalio algoritmo vizualizacija



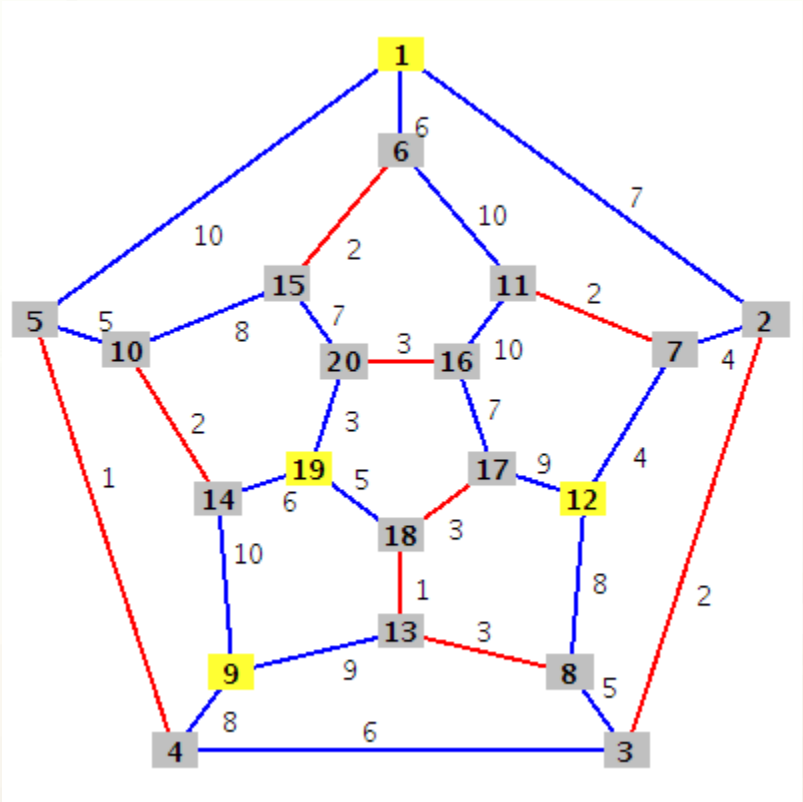
Kruskalio algoritmo vizualizacija



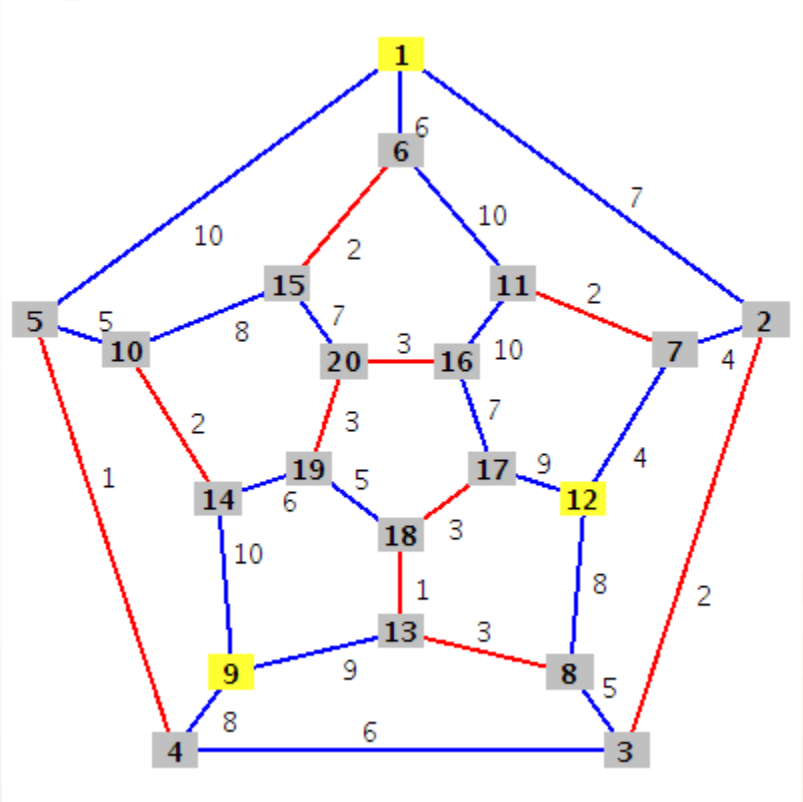
Kruskalio algoritmo vizualizacija



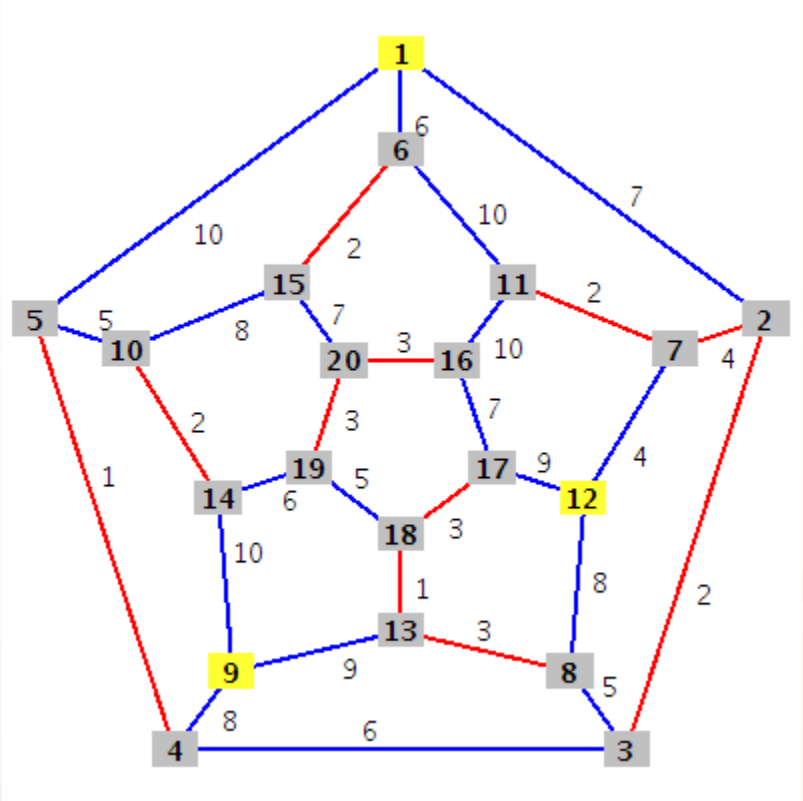
Kruskalio algoritmo vizualizacija



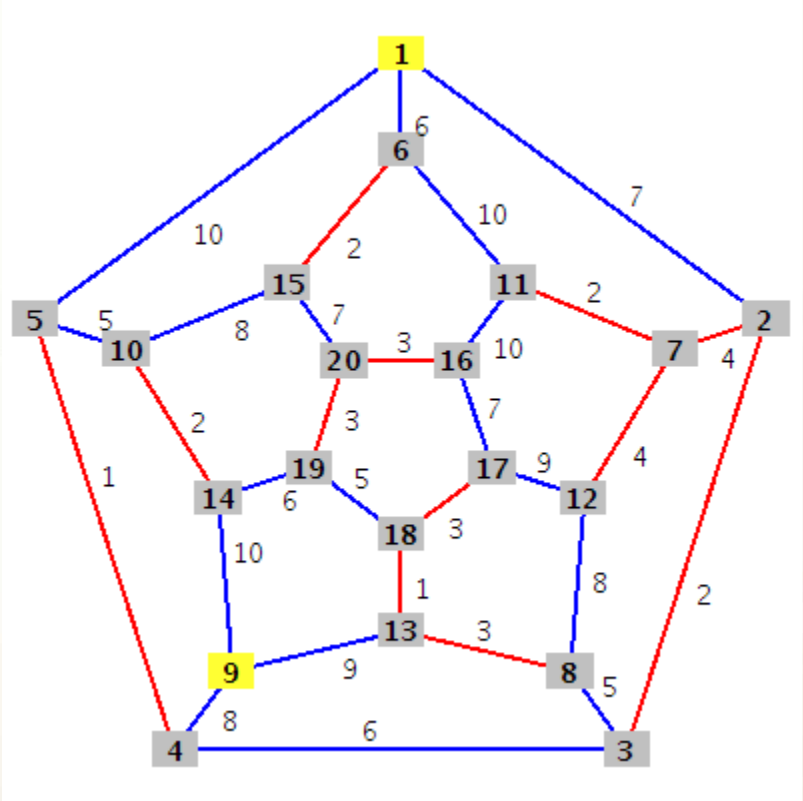
Kruskalio algoritmo vizualizacija



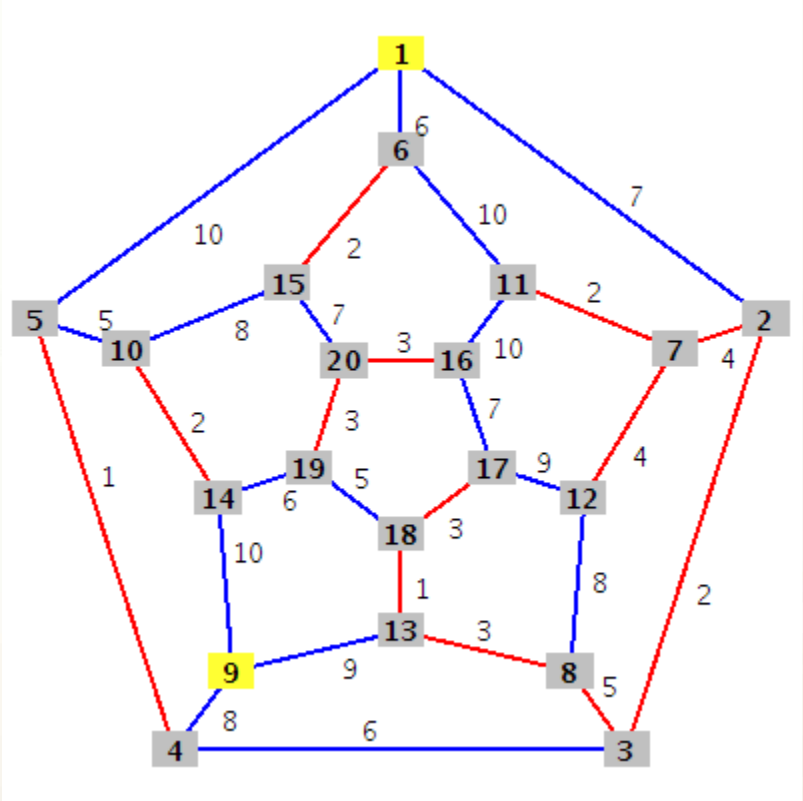
Kruskalio algoritmo vizualizacija



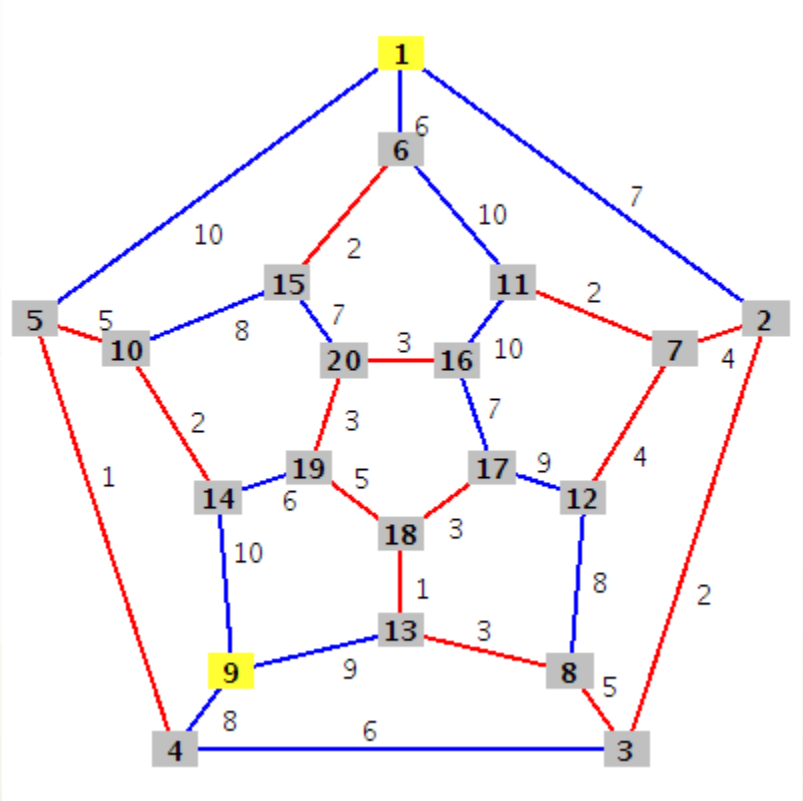
Kruskalio algoritmo vizualizacija



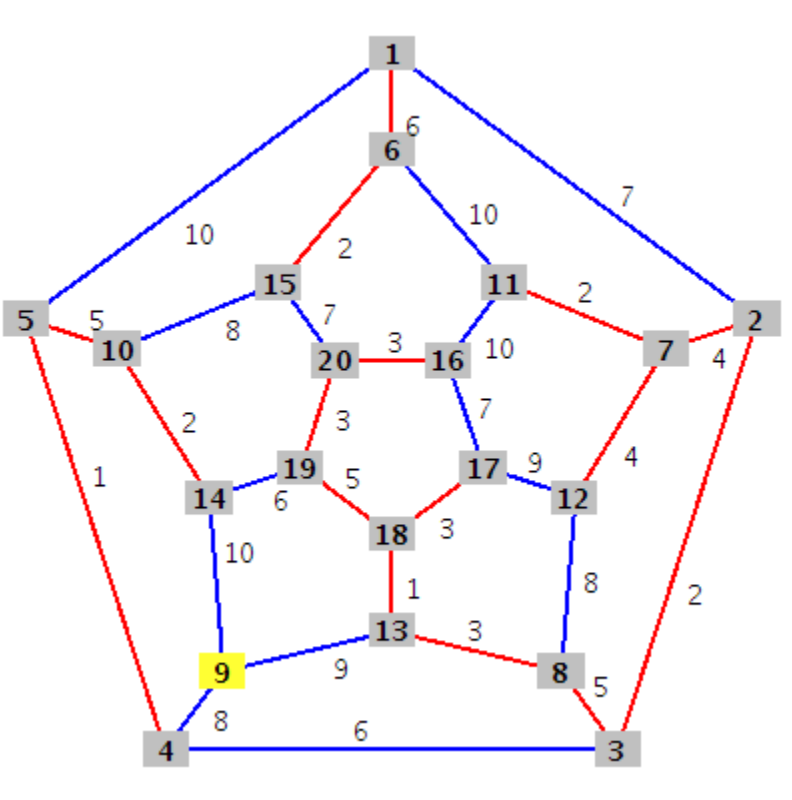
Kruskalio algoritmo vizualizacija



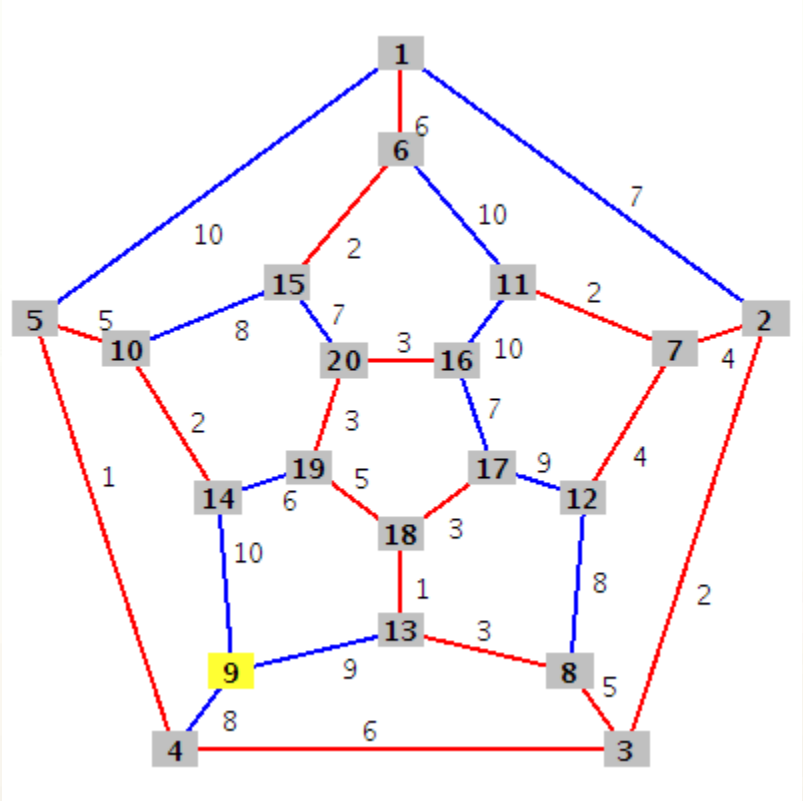
Kruskalio algoritmo vizualizacija



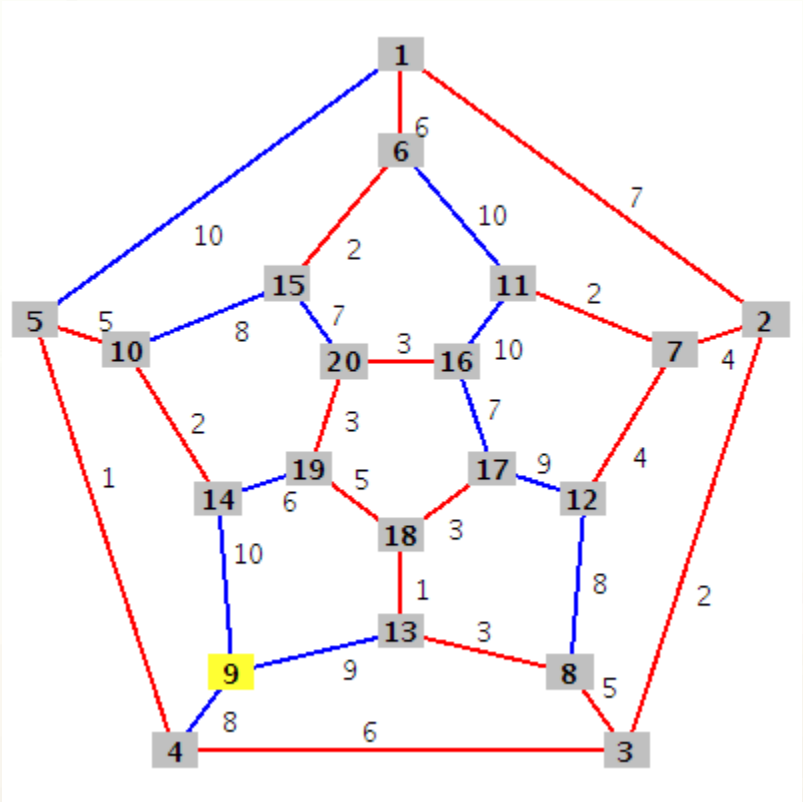
Kruskalio algoritmo vizualizacija



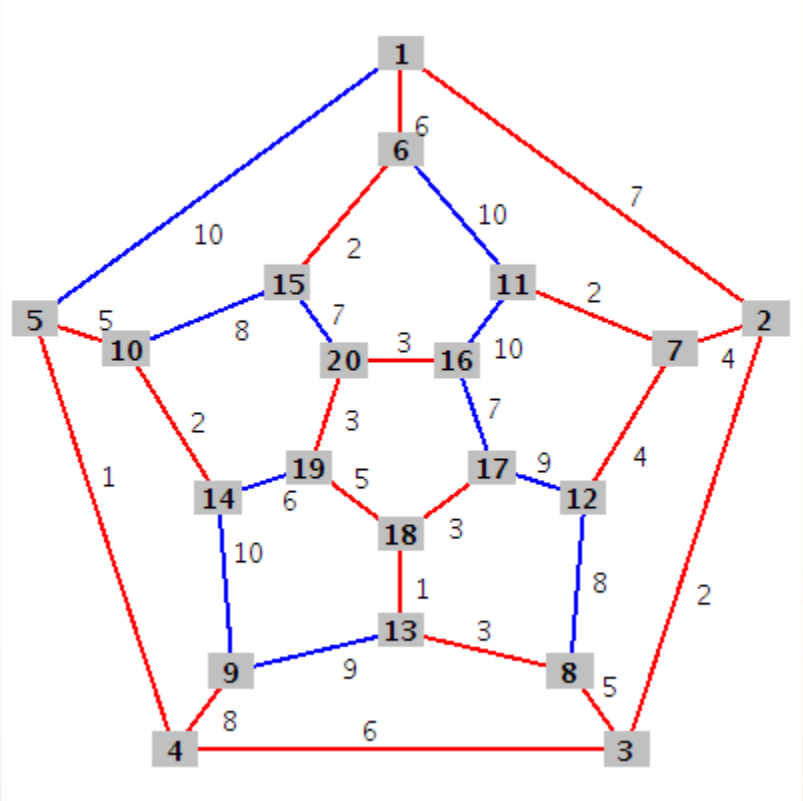
Kruskalio algoritmo vizualizacija



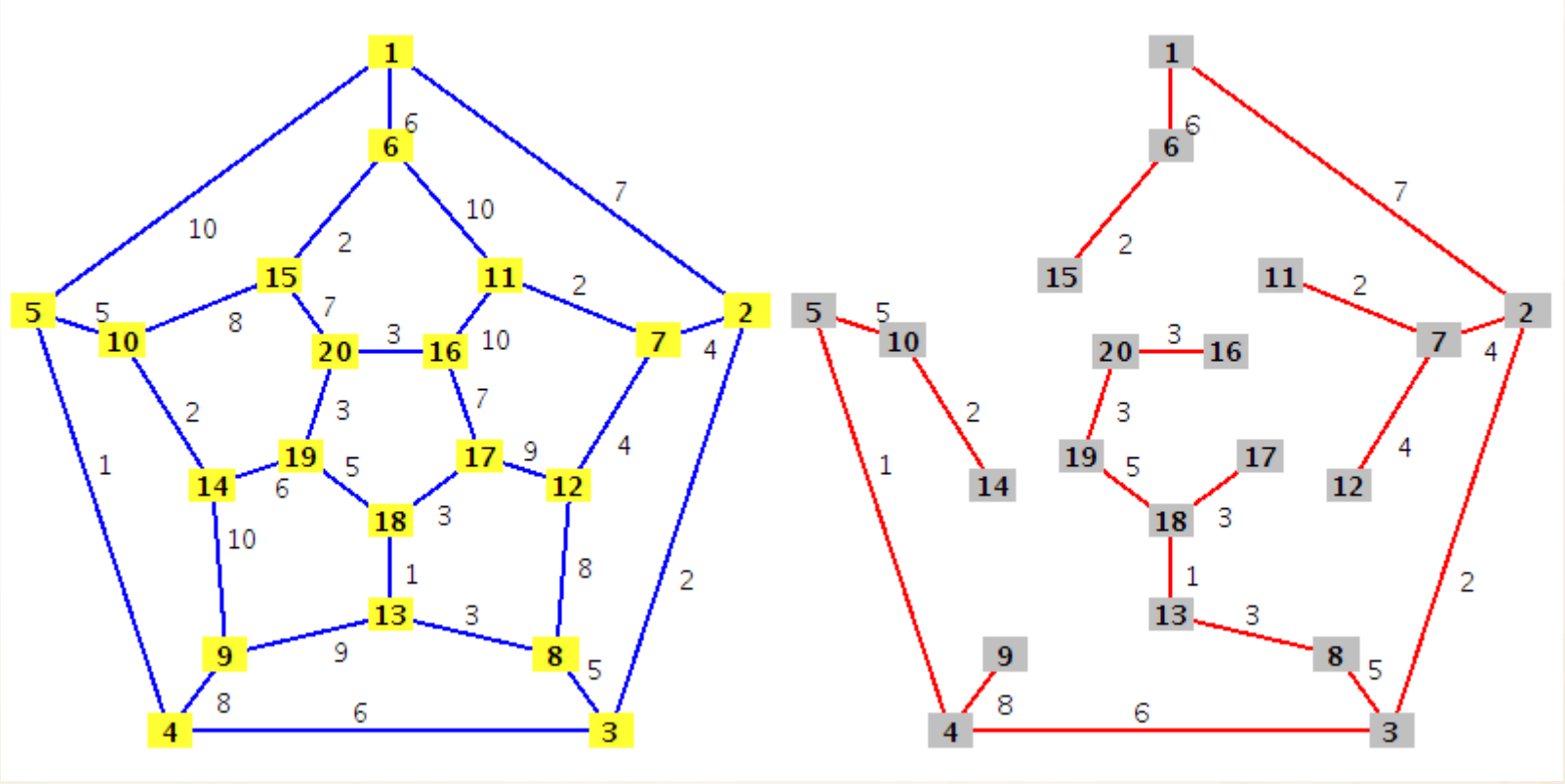
Kruskalio algoritmo vizualizacija



Kruskalio algoritmo vizualizacija



Kruskalio algoritmo vizualizacija



Rezultatas: A = {{1, 2}, {1, 6}, {2, 3}, {2, 7}, {3, 4}, {3, 8}, {4, 5}, {4, 9}, {5, 10}, {6, 15}, {7, 11}, {7, 12}, {8, 13}, {10, 14}, {13, 18}, {16, 20}, {17, 18}, {18, 19}, {19, 20}}

Primo algoritmas

MJM-Prim (G, w, r):

1. $Q \leftarrow V$
2. **for each** $u \in Q$
3. **do** $key[u] \leftarrow \infty$
4. $key[r] \leftarrow 0$
5. $\pi[r] \leftarrow NIL$
6. **while** $Q \neq \emptyset$
7. **do** $u \leftarrow \min(Q)$, (u viršūnė iš Q pašalinama)
8. **for each** $v \in Adj[u]$
9. **do if** $v \in Q$ ir $w(uv) < key[v]$
10. **then** $\pi[v] \leftarrow u$
11. $key[v] \leftarrow w(uv)$
12. RETURN MJM= $\{v\pi[v] : v \in V - r\}$

Skirtingai nei Kruskalio algoritmas, Primo algoritmas augina vieną medį A:

A = {vπ[v] : v ∈ V − r − Q}, čia

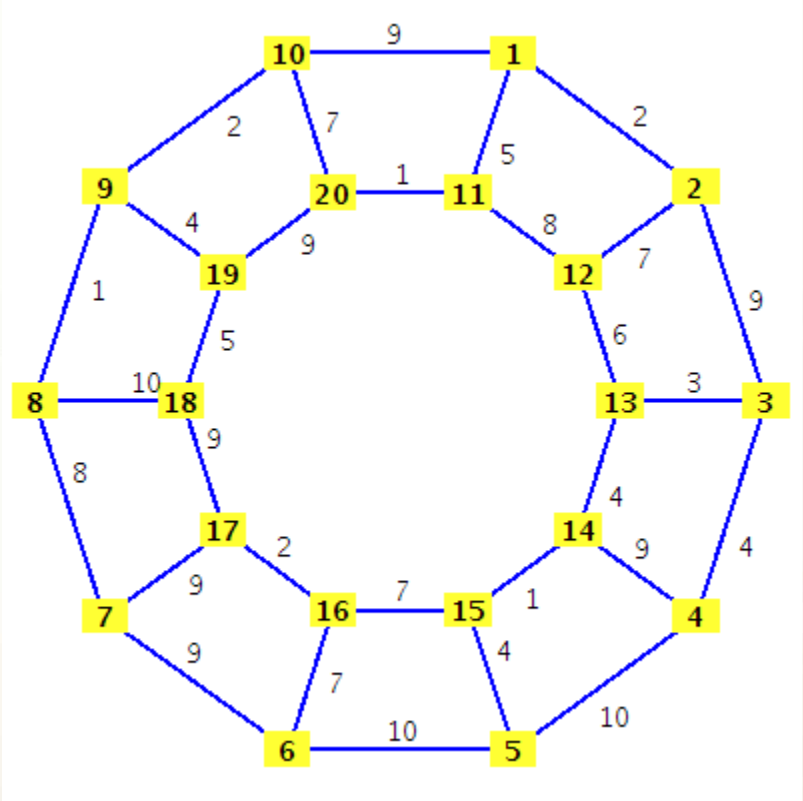
r – pradinė viršūnė (minimalaus jungiančio medžio sėkla),

π[u] – tėvystės atributas,

key[u] – atstumo atributas.

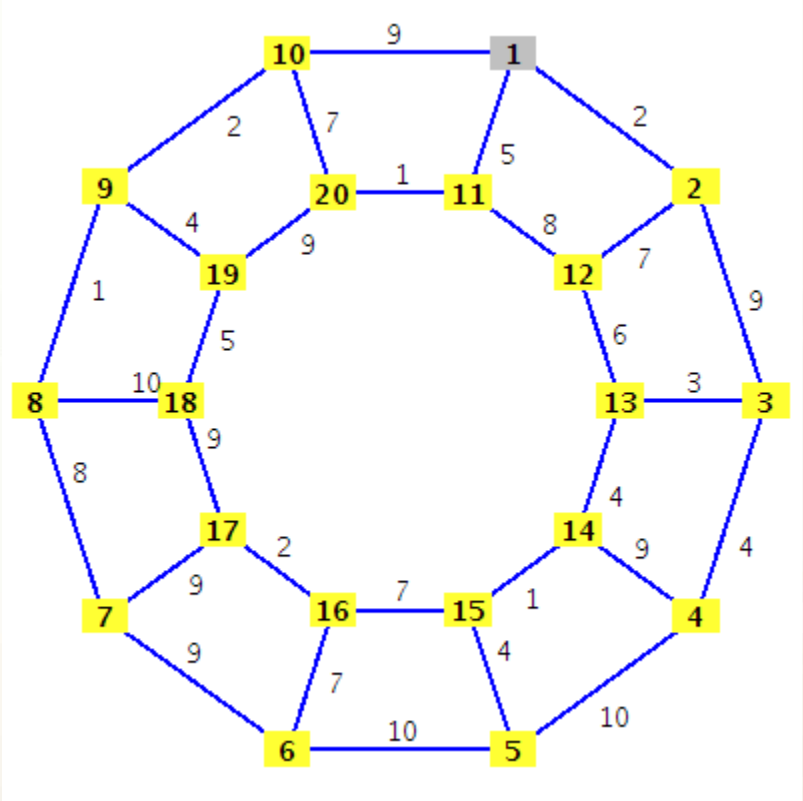
Primo algoritmo sudėtingumas ~O(m log n).

Primo algoritmo vizualizacija



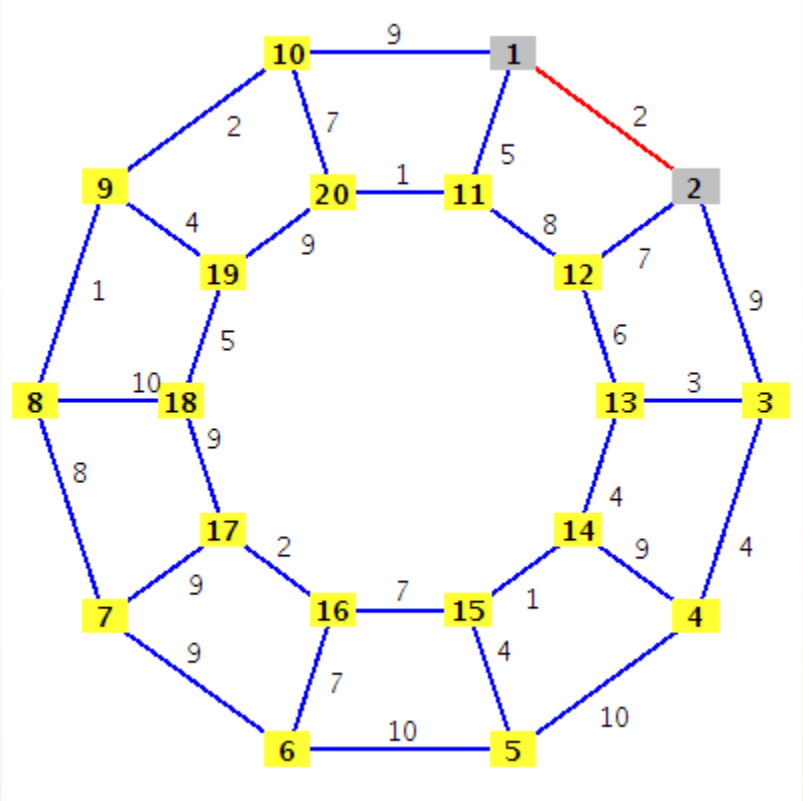
key[1]=∞, key[2]=∞,…,key[20]=∞, key[1]=0, π[1]=NIL

Primo algoritmo vizualizacija



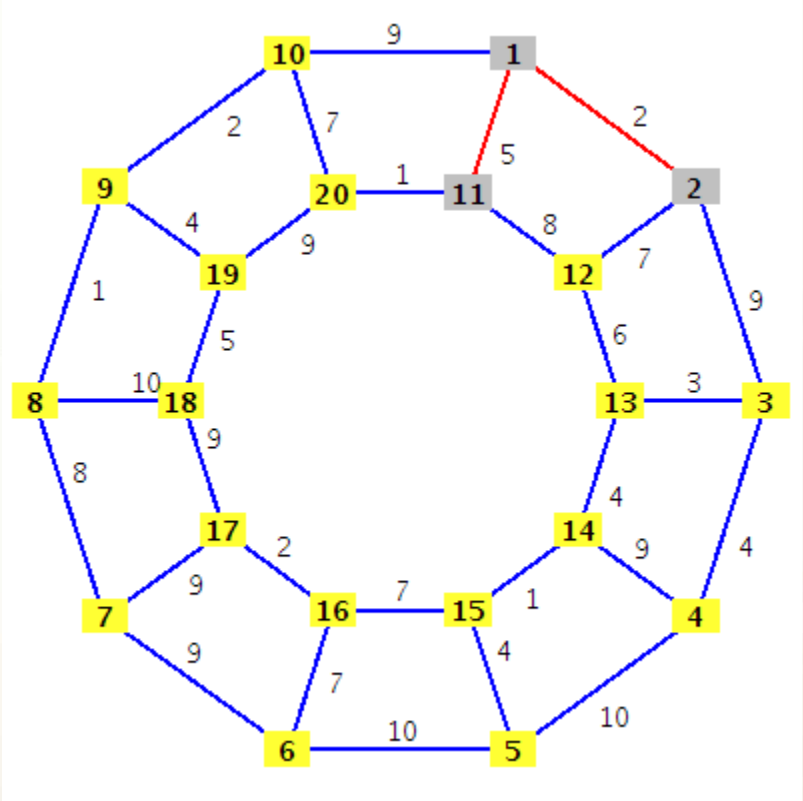
π[2]=1, key[2]=2, π[10]=1, key[10]=9, π[11]=1, key[11]=5,
Q=[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]

Primo algoritmo vizualizacija



π[3]=2, key[3]=9, π[12]=2, key[12]=7,
Q=[3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]

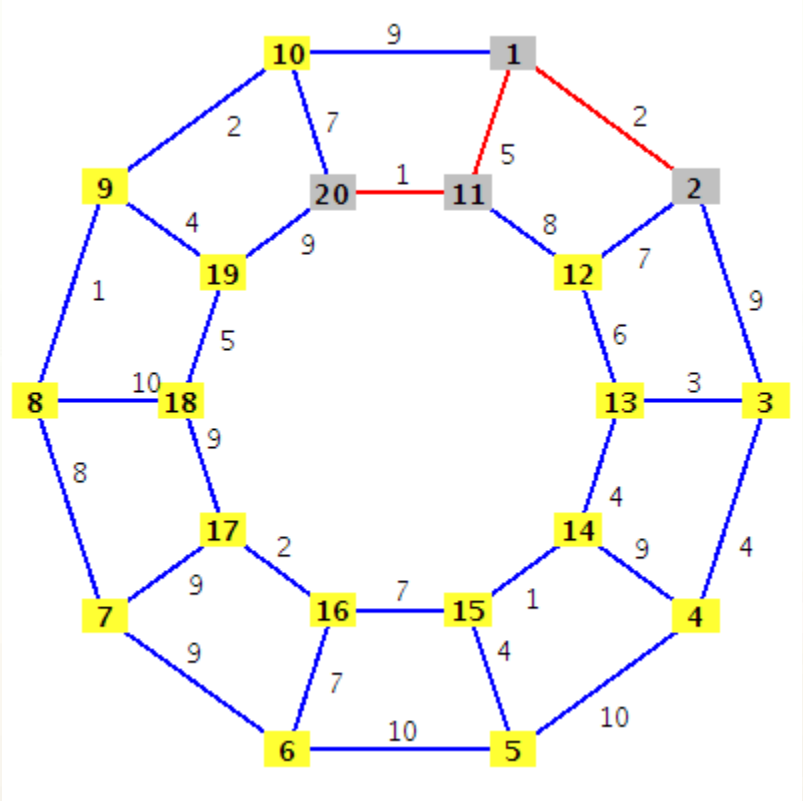
Primo algoritmo vizualizacija



π[20]=11, key[20]=1,

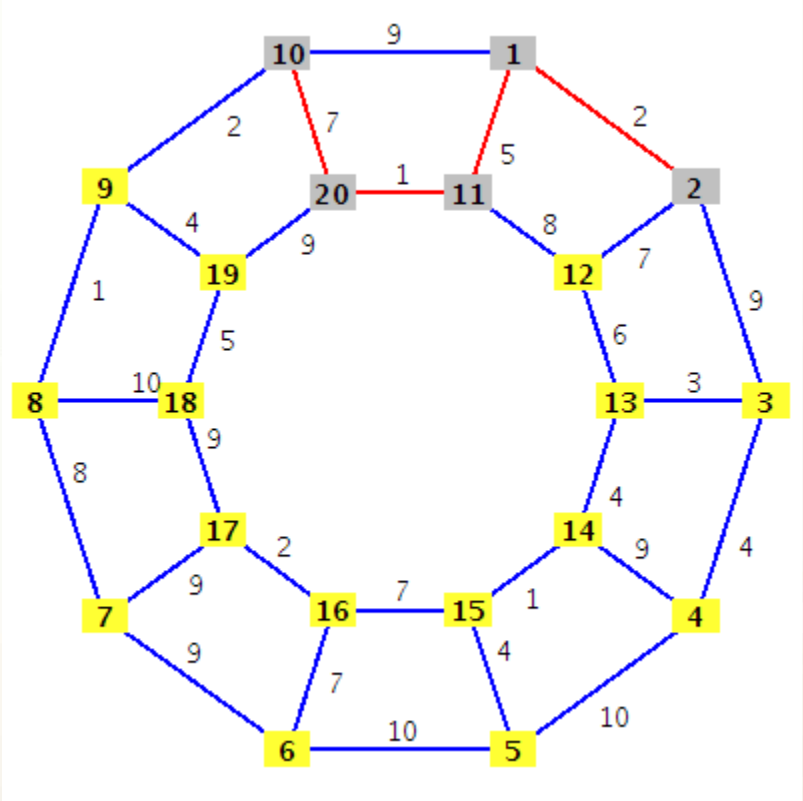
Q=[3,4,5,6,7,8,9,10,12,13,14,15,16,17,18,19,20]

Primo algoritmo vizualizacija



π[10]=20, key[10]=7, π[19]=20, key[19]=9,
Q=[3,4,5,6,7,8,9,10,12,13,14,15,16,17,18,19]

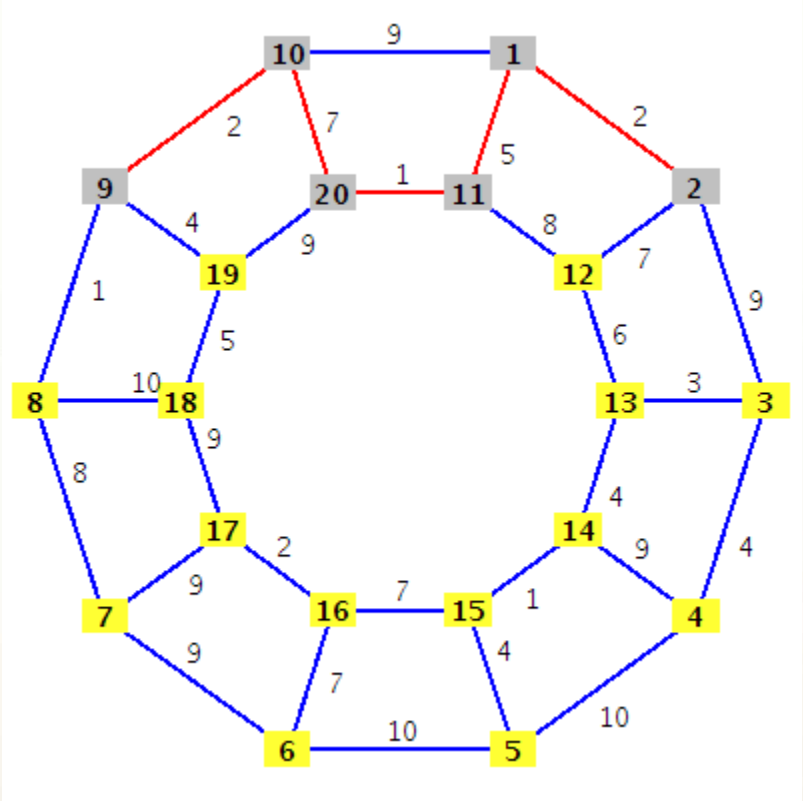
Primo algoritmo vizualizacija



π[9]=10, key[9]=2,

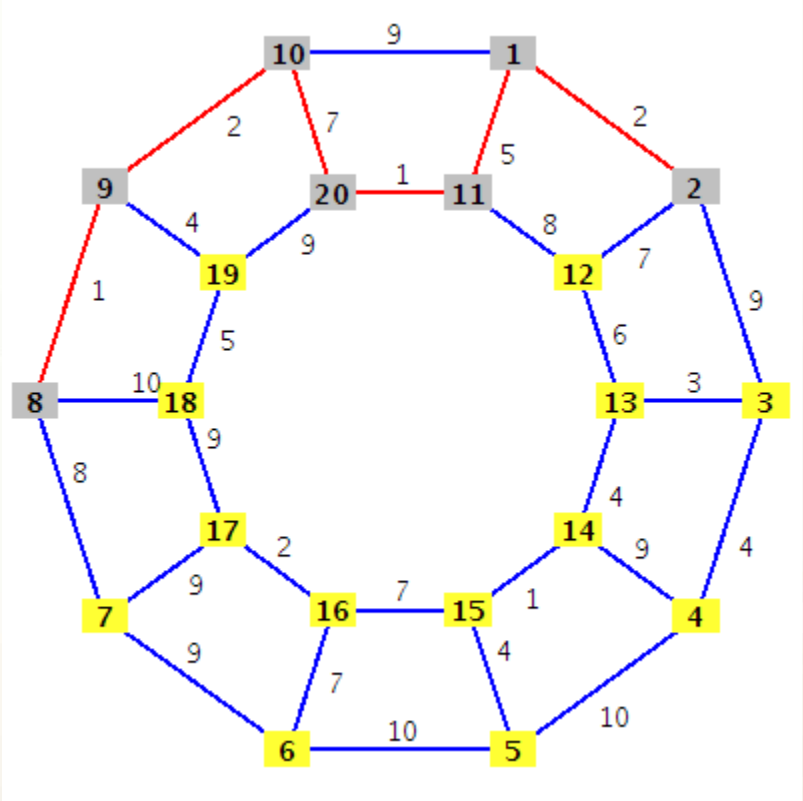
Q=[3,4,5,6,7,8,9,12,13,14,15,16,17,18,19]

Primo algoritmo vizualizacija



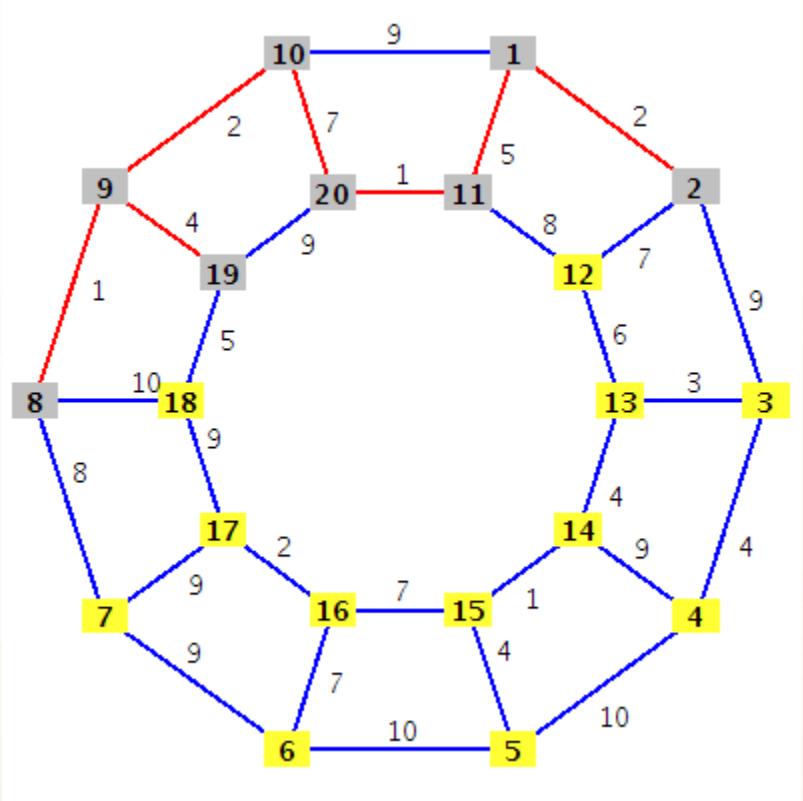
π[8]=9, key[8]=1, π[19]=9, key[19]=4,
Q=[3,4,5,6,7,8,12,13,14,15,16,17,18,19]

Primo algoritmo vizualizacija



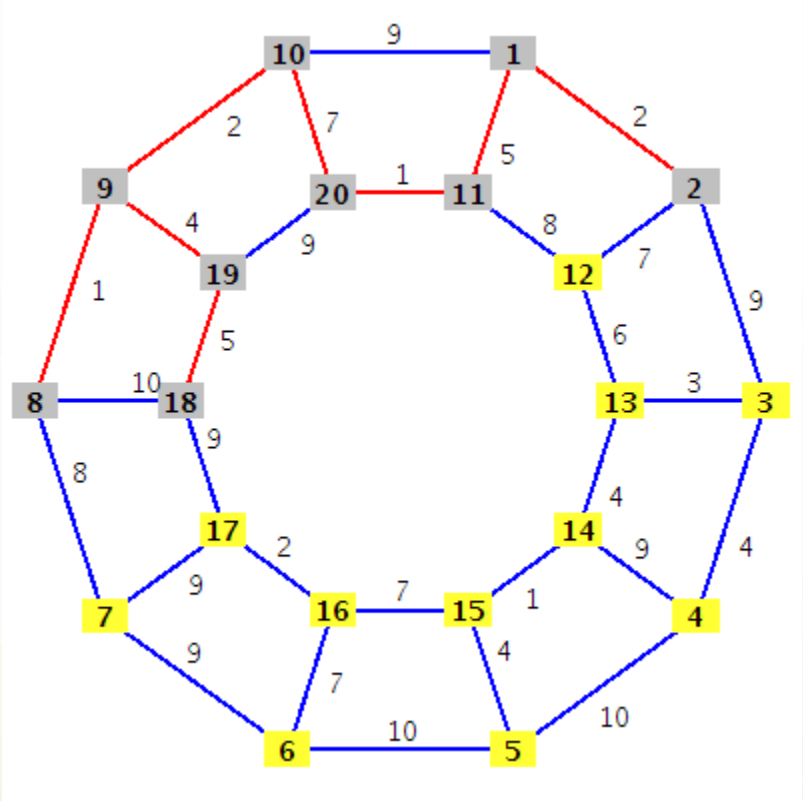
π[7]=8, key[7]=8, π[18]=8, key[18]=10,
Q=[3,4,5,6,7,12,13,14,15,16,17,18,19]

Primo algoritmo vizualizacija



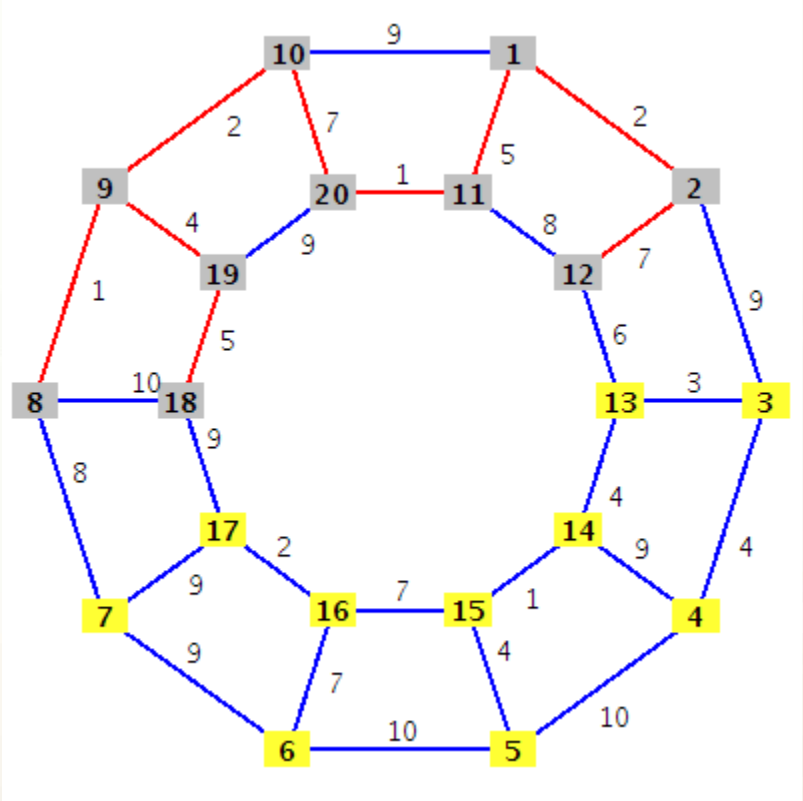
π[18]=19, key[18]=5,
Q=[3,4,5,6,7,12,13,14,15,16,17,18]

Primo algoritmo vizualizacija



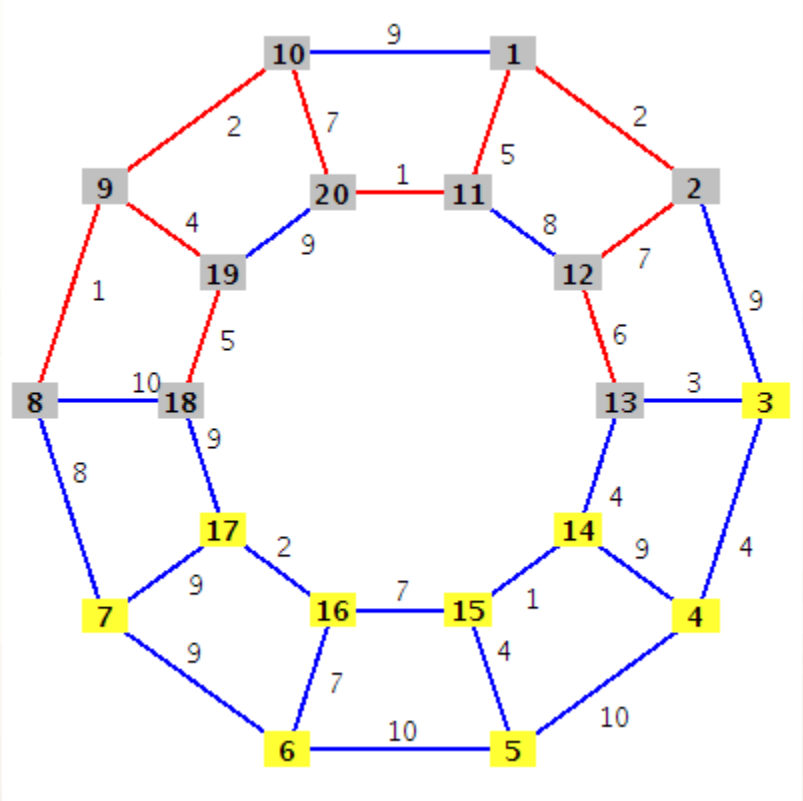
π[17]=18, key[17]=9,
Q=[3,4,5,6,7,12,13,14,15,16,17]

Primo algoritmo vizualizacija



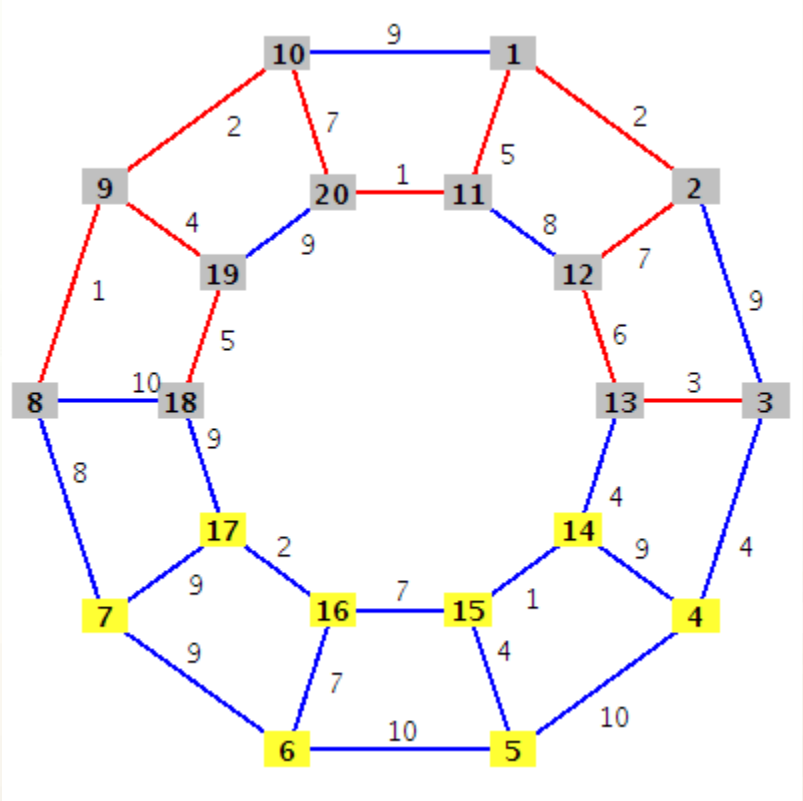
π[13]=12, key[13]=6,
Q=[3,4,5,6,7,13,14,15,16,17]

Primo algoritmo vizualizacija



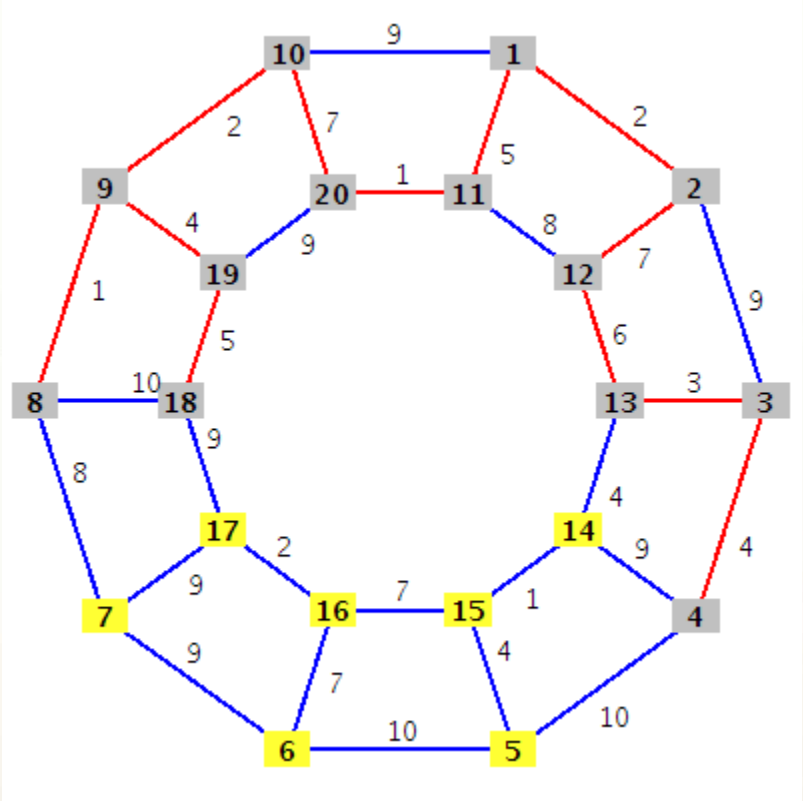
π[3]=13, key[3]=3, π[14]=13, key[14]=4,
Q=[3,4,5,6,7,14,15,16,17]

Primo algoritmo vizualizacija



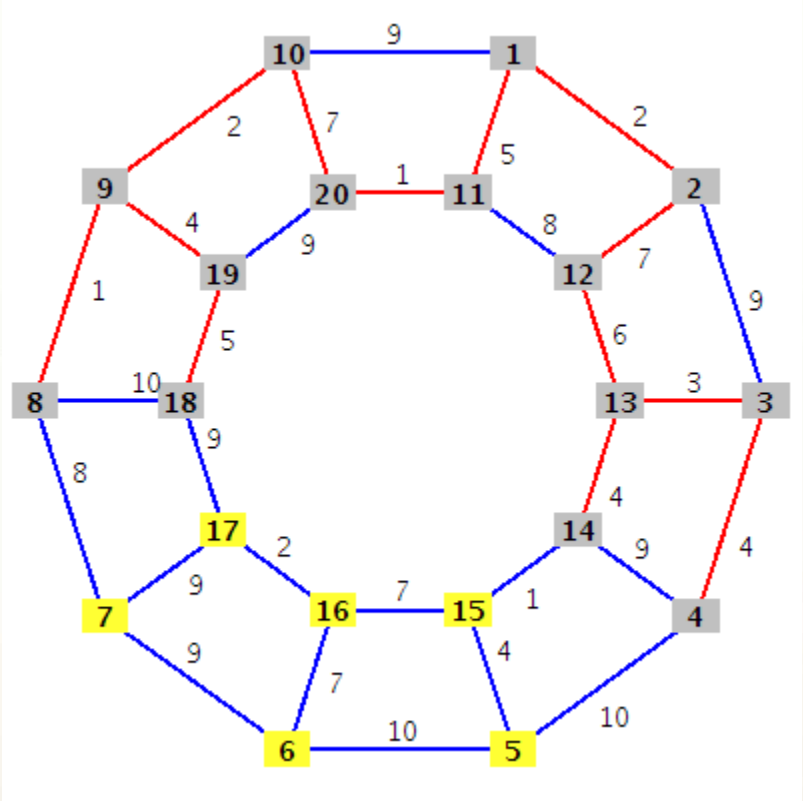
π[4]=3, key[4]=4,
Q=[4,5,6,7,14,15,16,17]

Primo algoritmo vizualizacija



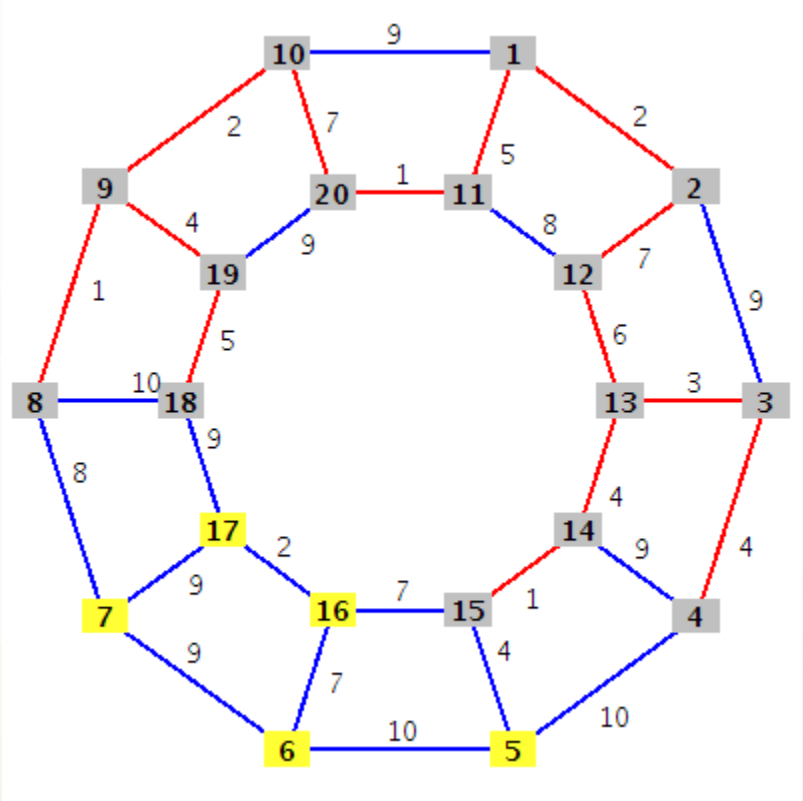
π[5]=4, key[5]=10,
Q=[5,6,7,14,15,16,17]

Primo algoritmo vizualizacija



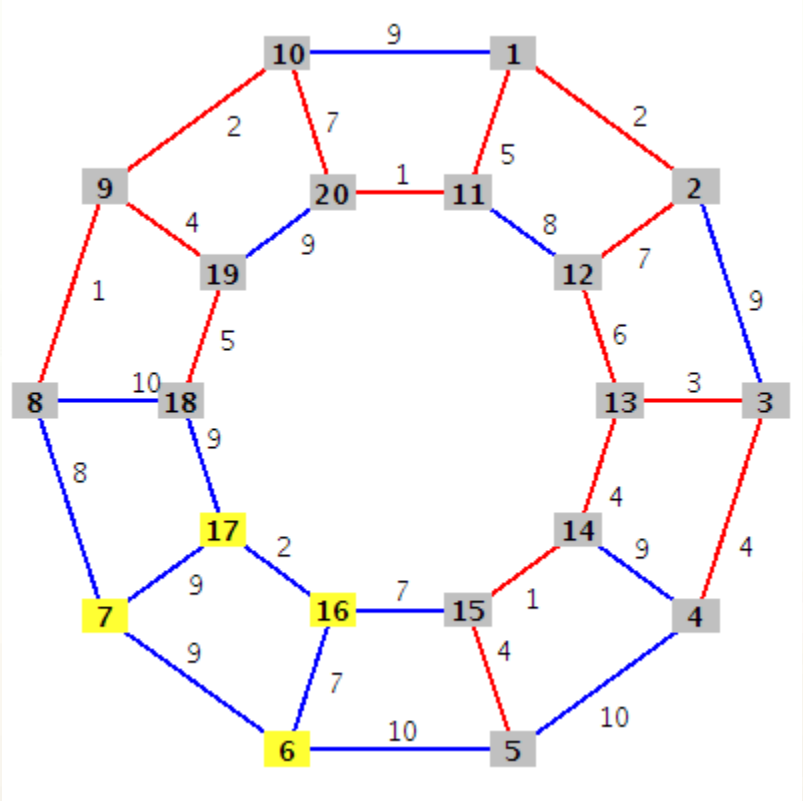
π[15]=14, key[15]=1,
Q=[5,6,7,15,16,17]

Primo algoritmo vizualizacija



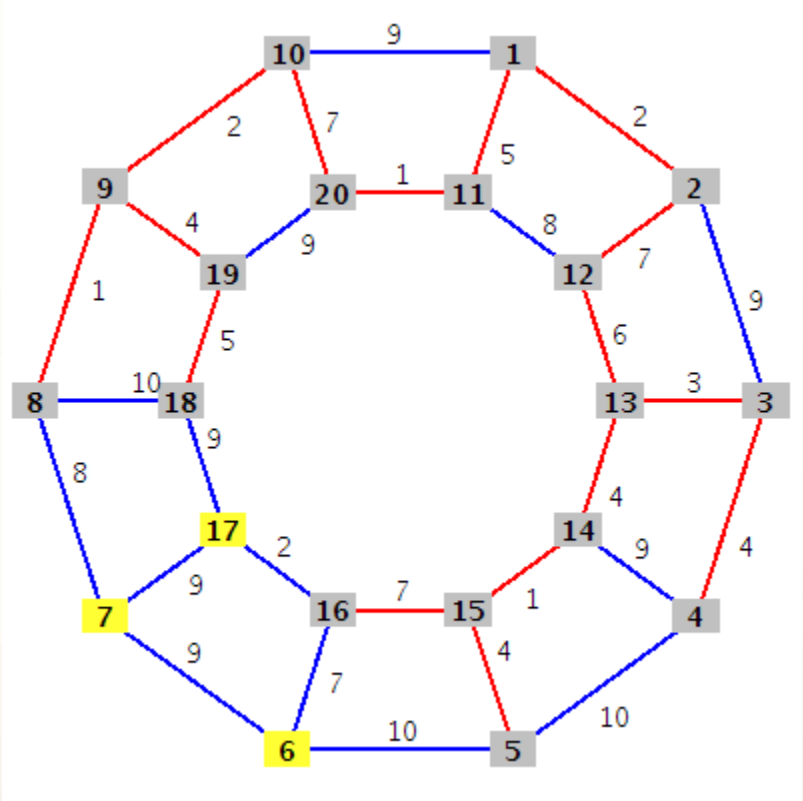
π[5]=15, key[5]=4, π[16]=15, key[16]=7,
Q=[5,6,7,16,17]

Primo algoritmo vizualizacija



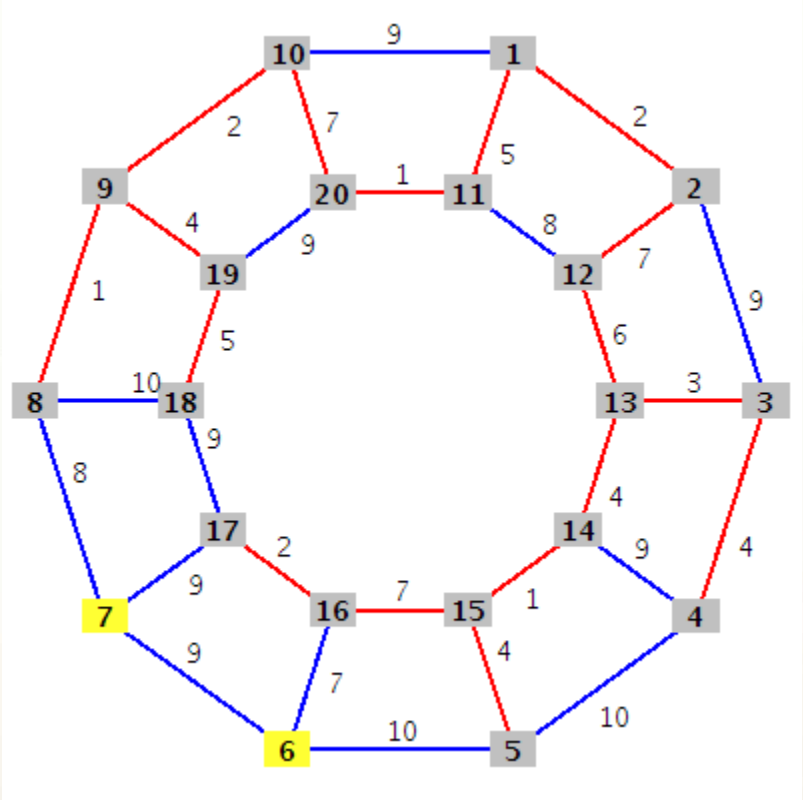
π[6]=5, key[6]=10,
Q=[6,7,16,17]

Primo algoritmo vizualizacija



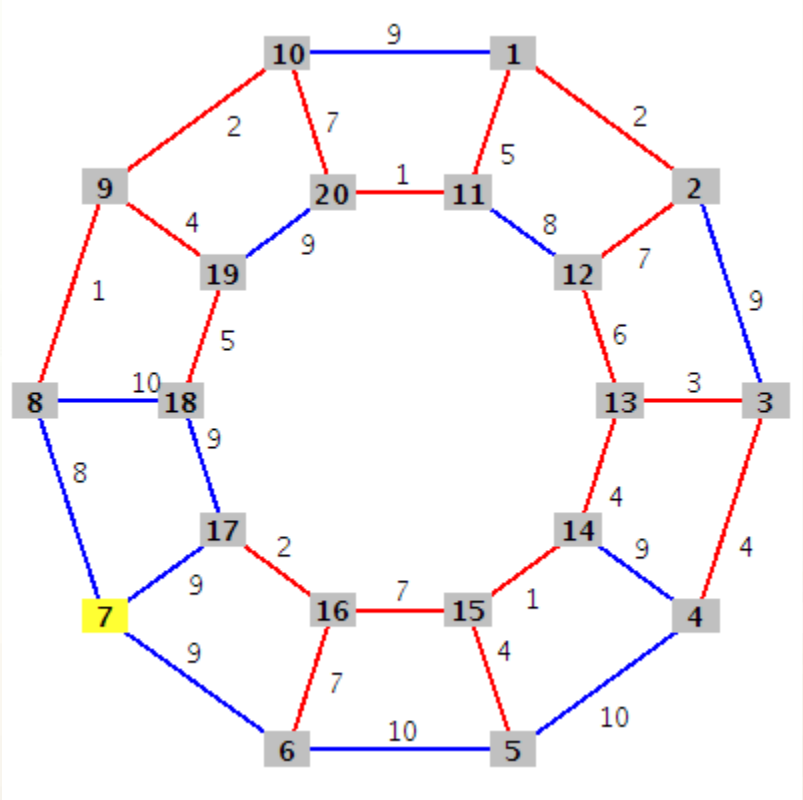
π[6]=16, key[6]=7, π[17]=16, key[17]=2,
Q=[6,7,17]

Primo algoritmo vizualizacija



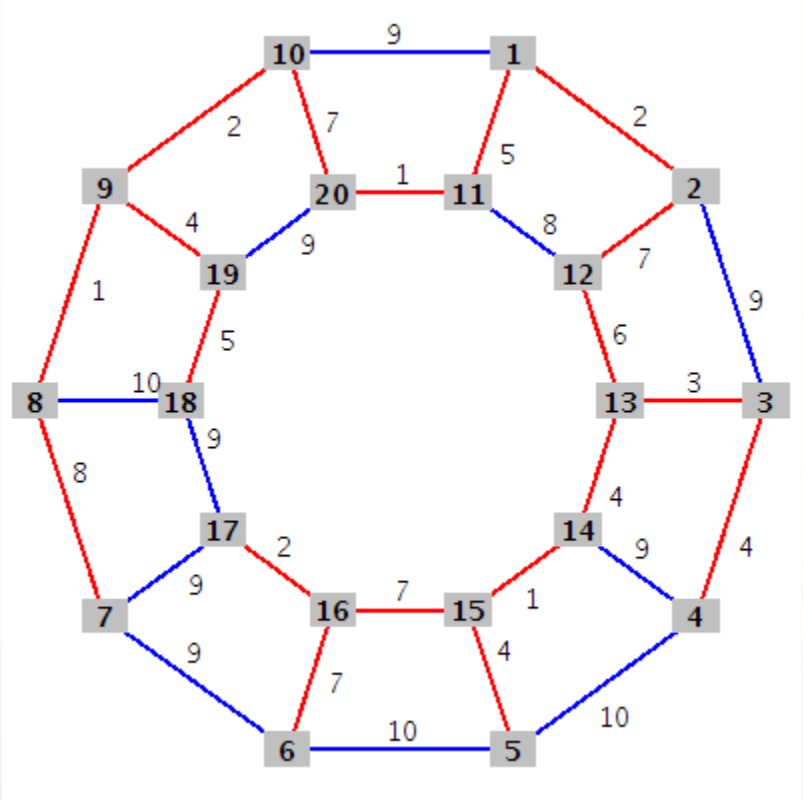
Q=[6,7]

Primo algoritmo vizualizacija



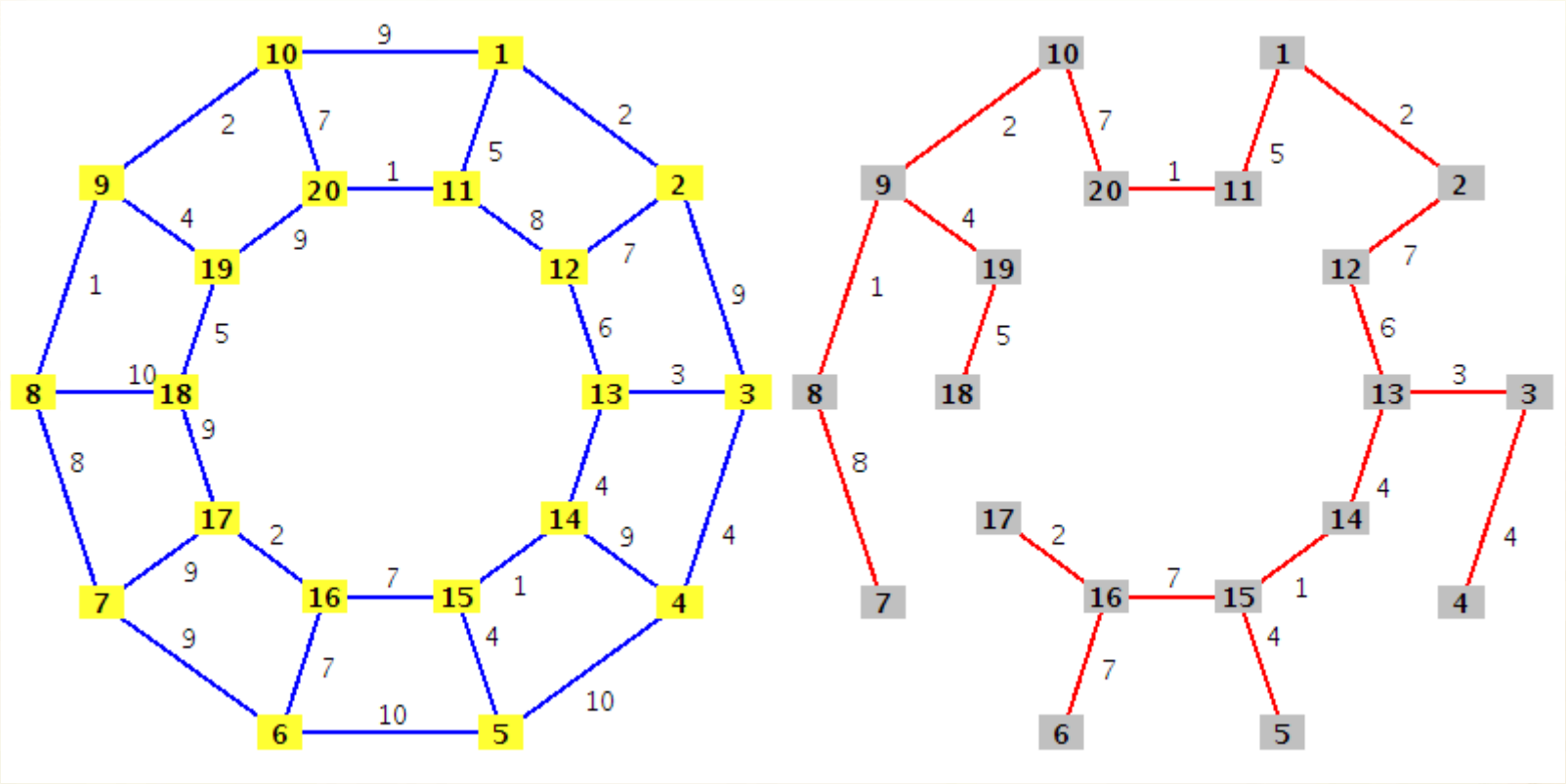
Q= [7]

Primo algoritmo vizualizacija



Q= [ ]

Primo algoritmo vizualizacija



Rezultatas: π[1]=NIL, π[2]=1, π[3]=13, π[4]=3, π[5]=15, π[6]=16, π[7]=8, π[8]=9, π[9]=10, π[10]=20, π[11]=1, π[12]=2, π[13]=12, π[14]=13, π[15]=14, π[16]=15, π[17]=16, π[18]=19, π[19]=9, π[20]=11.

Solino (Boruvkos) algoritmas

Skirtingai nei Primo ir Kruskalio algoritmai, Solino algoritmas išrenka keletą briaunų, formuojančių tarpusavyje atskirus jungiančiuosius medžius. Tęsiant algoritmą, pagal minimalius briaunų svorius jungiantieji medžiai jungiami tarpusavyje iki kol gaunamas minimalus jungiantysis medis.

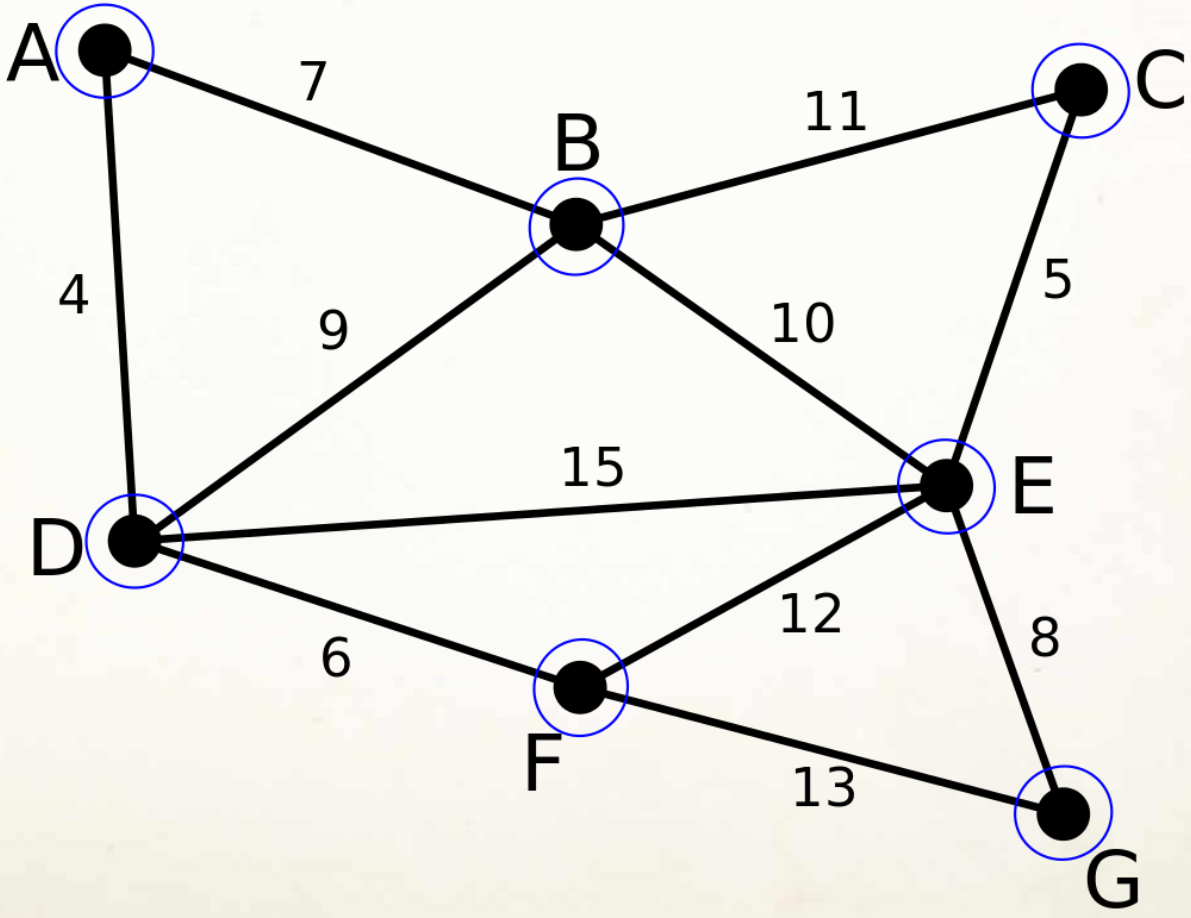
Pseudokodas:

Input: A graph G whose edges have distinct weights

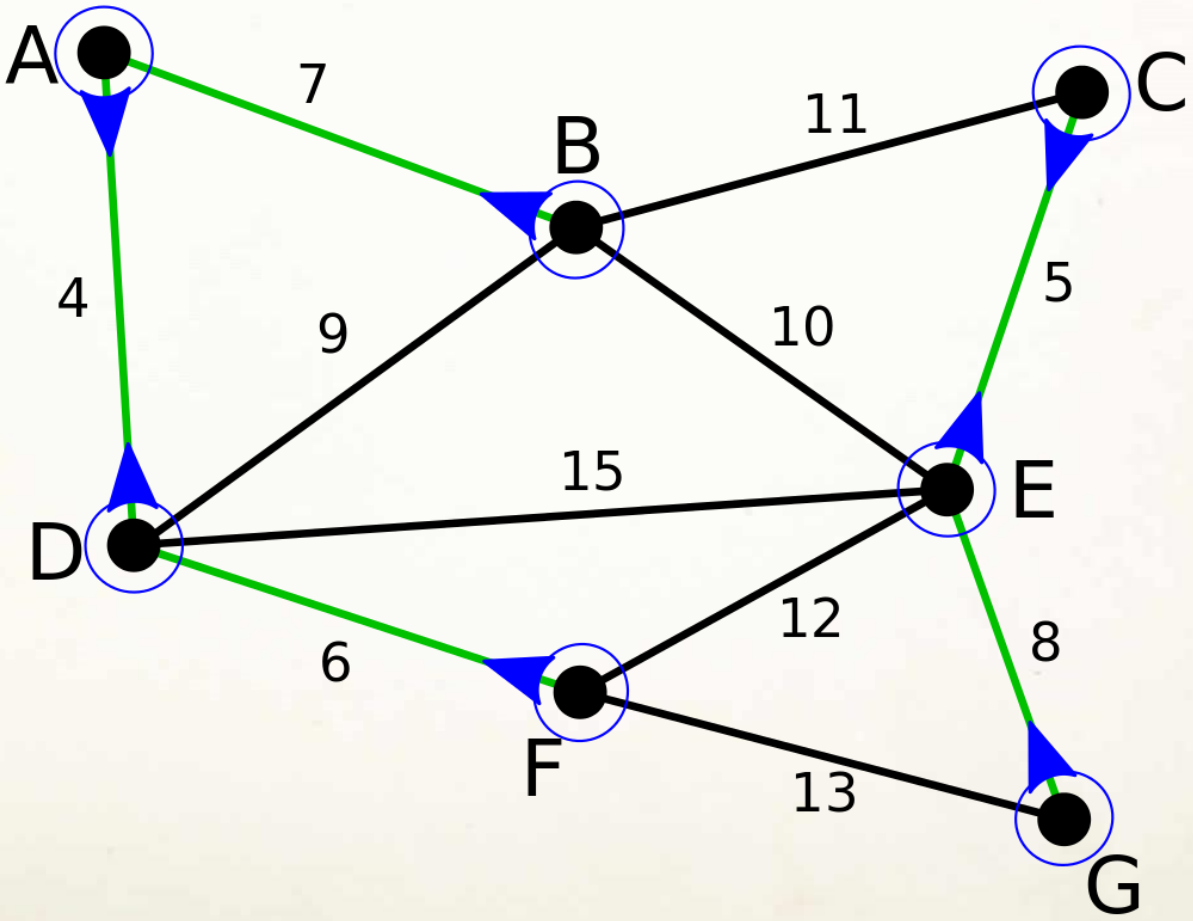
```
Initialize a forest F to be a set of one-vertex trees, one for each vertex of the graph.
While F has more than one component:
  Find the connected components of F and label each vertex of G by its component
  Initialize the cheapest edge for each component to "None"
  For each edge uv of G:
    If u and v have different component labels:
      If uv is cheaper than the cheapest edge for the component of u:
        Set uv as the cheapest edge for the component of u
      If uv is cheaper than the cheapest edge for the component of v:
        Set uv as the cheapest edge for the component of v
  For each component whose cheapest edge is not "None":
    Add its cheapest edge to F
```

Output: F is the minimum spanning forest of G.

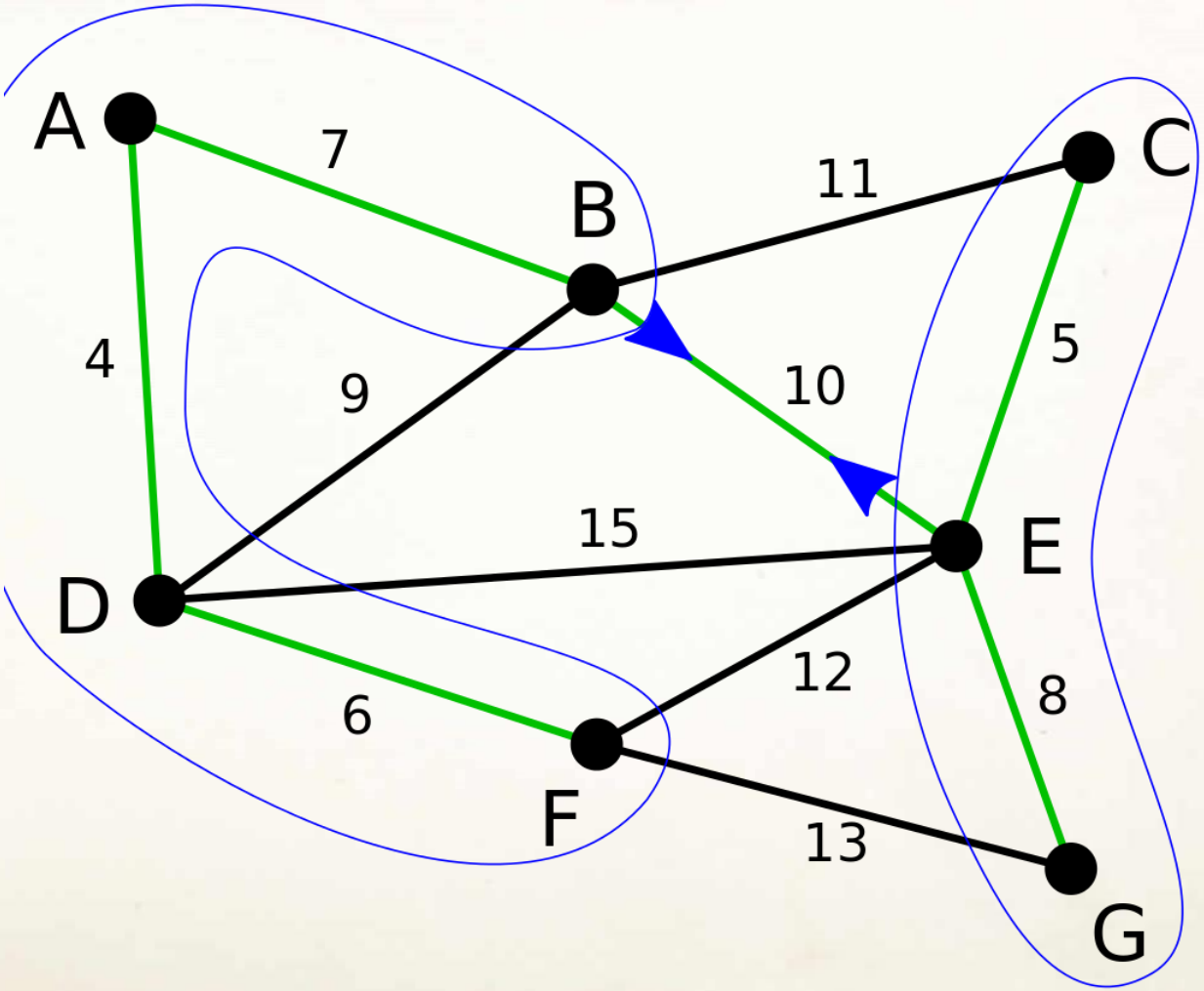
Solino algoritmo pavyzdys (1)



Solino algoritmo pavyzdys (2)



Solino algoritmo pavyzdys (3)



Ačiū už dėmesį.

Klausimai?