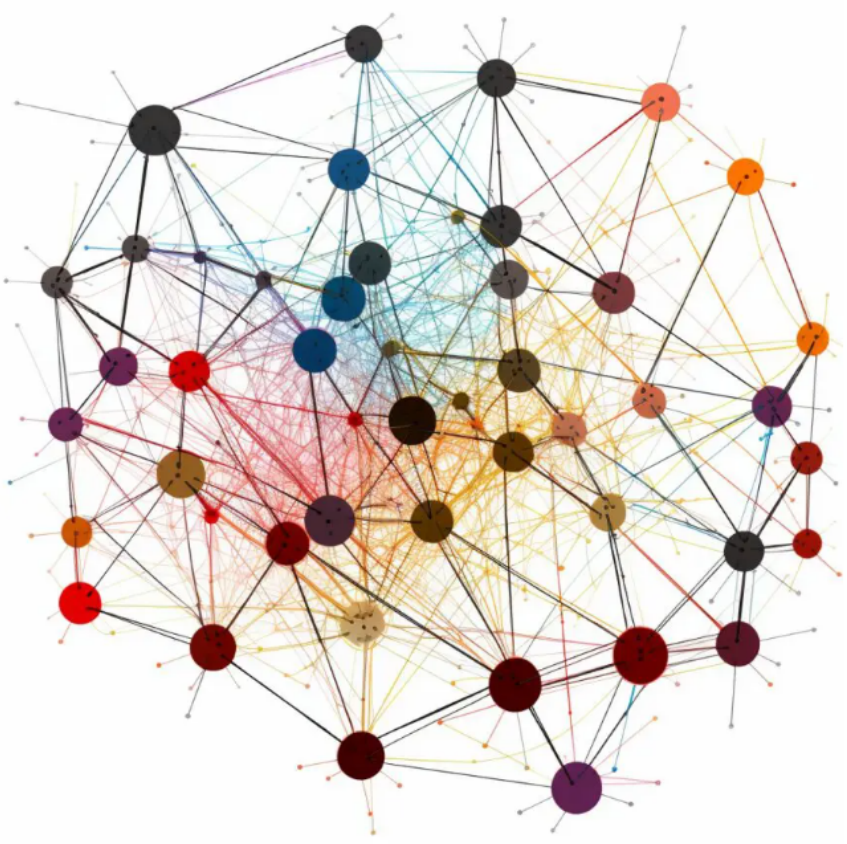
Tranzityvaus grafo uždarinys

Parengė Akvilė Ropytė ir Vasarė Pratuzaitė

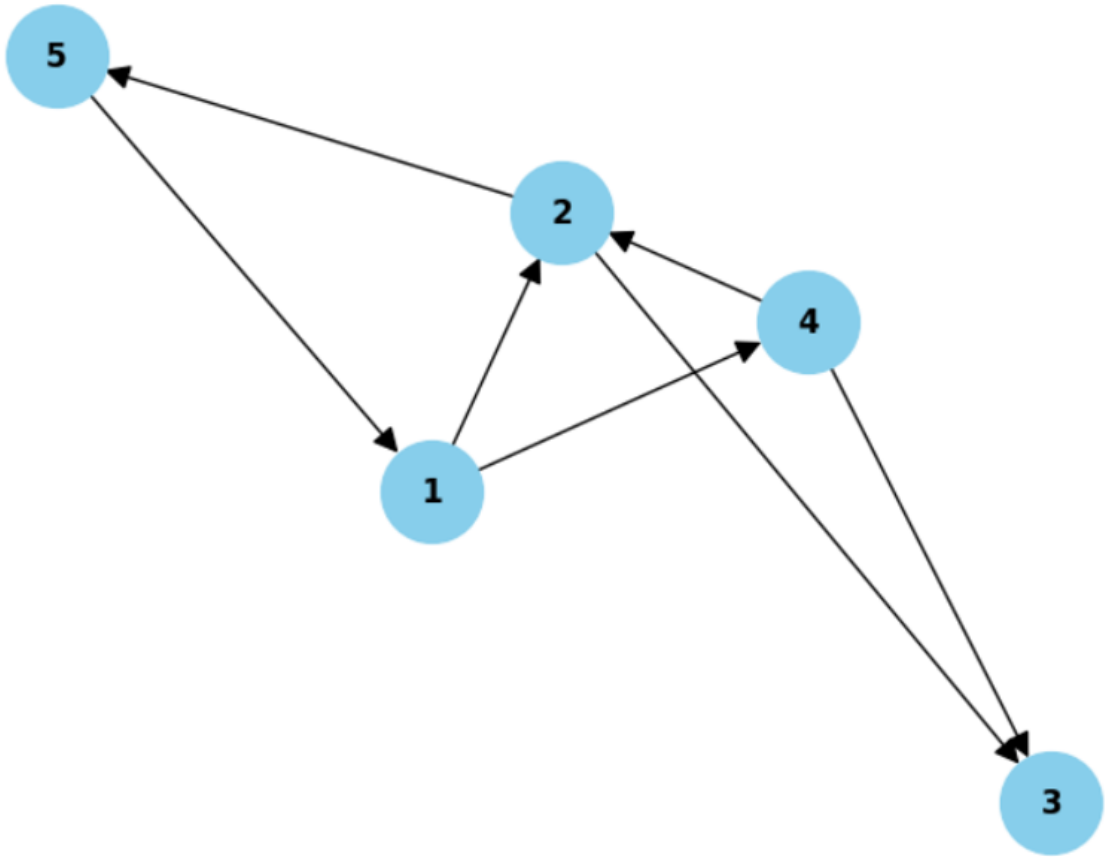


Turinys

Sąvokos

Orientuoto grafo apibrėžimas

Orientuotas grafas yra viršūnių V ir briaunų E aibių pora: G = (V,E), kur kiekviena briauna turi kryptį.

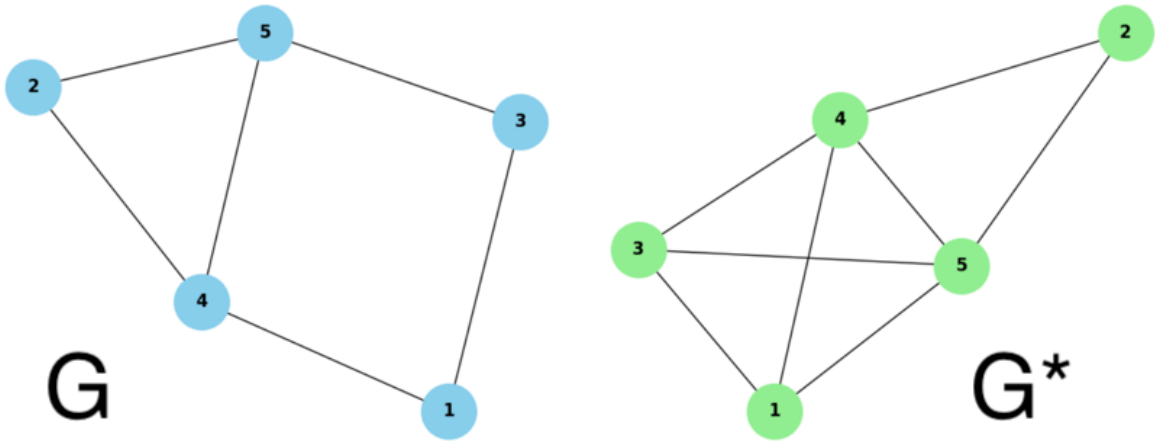


Grafo uždarinio apibrėžimas

Grafo G uždarinys yra grafas G*, gautas iš G, vis pridedant kraštines tarp neincidenčių viršūnių, kurių laipsnių suma yra nemažesnė nei grafo G viršūnių skaičius.

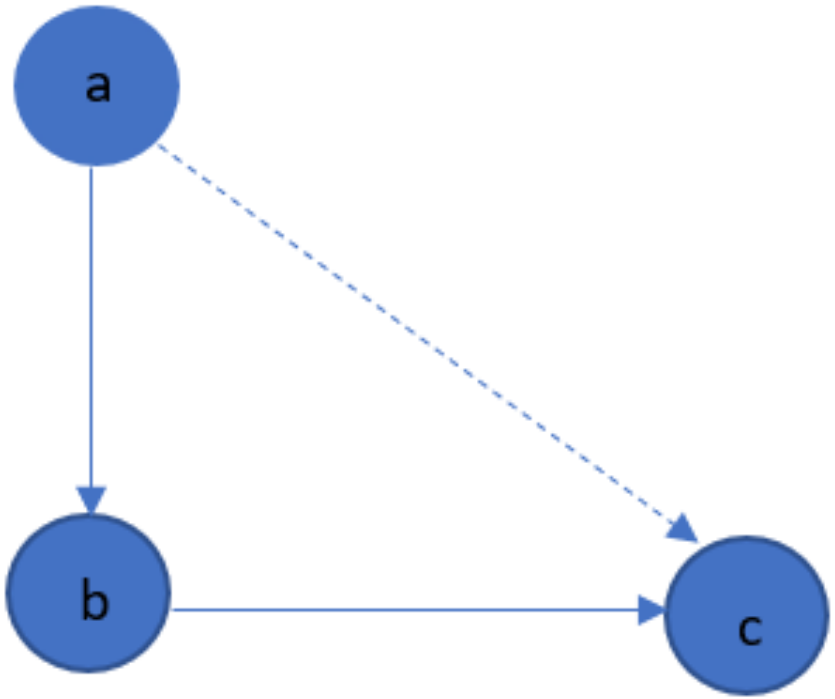
Pavyzdys

d(1) + d(5) = 5
d(3) + d(4) = 5



Tranzityvaus grafo apibrėžimas

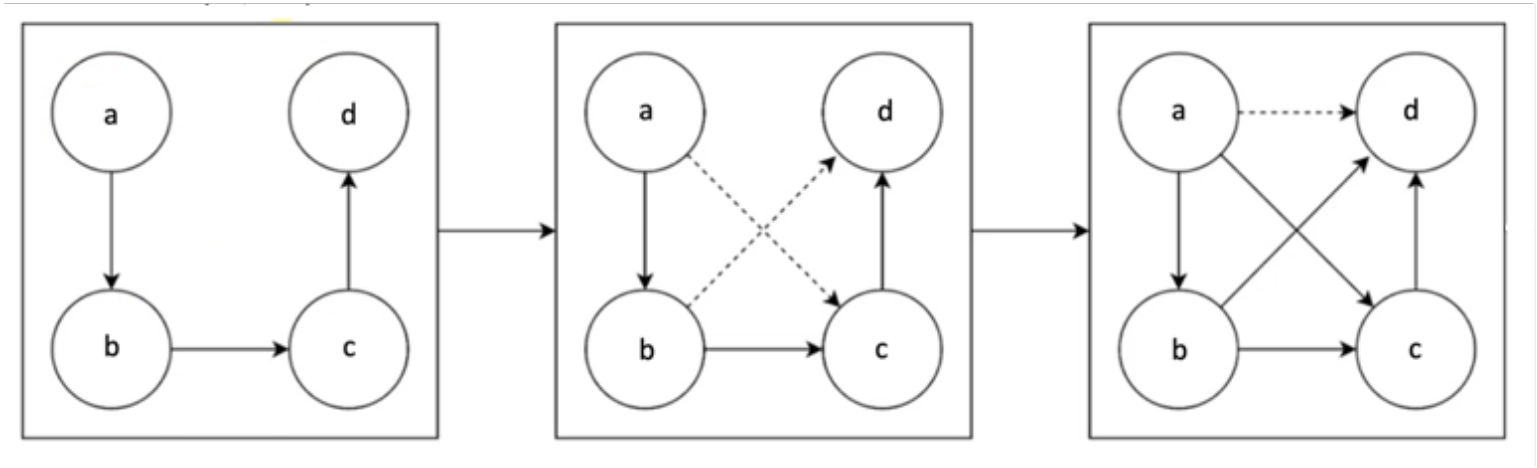
Tai toks grafas G(V,E), kurio viršūnių poroms (a,b) ir (b,c) egzistuoja viršūnių pora (a,c). Paprasčiau: jeigu iš a galime nueiti į b ir iš b galime nueiti į c, tai iš a galime nueiti į c.



Grafo tranzityvus uždarinys

Tranzityvaus grafo uždarinio apibrėžimas

Tai duomenų struktūra, kurioje pavaizduoti visi galimi trumpiausi keliai tarp kiekvienos viršūnės. Kitaip, iš grafo G = (V, E) gauname tranzityvų grafo uždarinį G*(V, E*), kur yra briauna (u, v), priklausanti aibei E* jei grafe G yra kelias iš u į v.



Floyd-Warshall algoritmas

Algoritmo istorija

Dabar žinomas algoritmas buvo paskelbtas Roberto Floido (angl. Robert Floyd) 1962 m. Tačiau šis algoritmas buvo labai panašus į anksčiau išleistą Stefano Varšalo (angl. Stephen Warshall) algoritmą tranzityvaus grafo uždariniui rasti.

Robert W. Floyd

Stephen Warshall

Algoritmo panaudojimas

Floyd-Warshall algoritmas skirtas rasti trumpiausius kelius tarp kiekvienos viršūnės orientuotame svoriniame grafe.

Algoritmo esmė

Algoritmas sukonstruoja naują svorinį grafą iš duoto svorinio grafo, kuriame kiekvienai briaunai pateikiamas svoris, atitinkantis minimaliam svoriui kažkokio kelio tarp tos briaunos viršūnių.

Svarbu!

Algoritmas tinka ir grafui, kurio briaunų svoriai yra neigiami, tačiau, jei grafe bus neigiamų ciklų, tai algoritmas neveiks korektiškai.

Algoritmo žingsniai

1-as žingsnis

Į pradinę matricą surašome grafo kraštinių svorius, įstrižainėje paliekant 0. Jei viršūnių nejungia briauna, rašome ∞

$$D_0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{array}$$

2-as žingsnis

Iš pradinės matricos, kuriame sekančią:

1. Perrašome elementus iš pirmo stulpelio ir pirmos eilutės, įstižainėje paliekant nulius.
2. Briaunos (i,j) svorį keičiame, jei briaunų (i, iteracijos skaičius) ir (iteracijos skaičius, j) svorių suma yra mažesnė už pradinėje matricoje buvusį svorį. Jei ne, perrašome svorį iš pradinės matricos.

2-as žingsnis

$$D_0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{array}$$

$$D_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & & \\ 4 & & 0 & \\ \infty & & & 0 \end{bmatrix} \end{array} \Rightarrow D_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{array}$$

3-ias žingsnis

Toliau kuriame sekančią matricą tuo pačiu principu, tik svorius lyginime su prieš tai buvusios matricos svoriais (o ne su pirmutinės). Atitinkamai kartojame procesą tiek kartų, kiek grafe yra viršūnių.

3-ias žingsnis

$$D_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{array}$$

$$D_2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & & \\ \infty & 0 & 1 & \infty \\ & 12 & 0 & \\ & 2 & & 0 \end{bmatrix} \end{array} \Rightarrow D_2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & 9 & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 3 & 0 \end{bmatrix} \end{array}$$

$$D_2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & 9 & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 3 & 0 \end{bmatrix} \end{array}$$

$$D_3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & & 9 & \\ & 0 & 1 & \\ 4 & 12 & 0 & 5 \\ & & 3 & 0 \end{bmatrix} \end{array} \quad \Rightarrow \quad D_3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \end{array}$$

$$D_3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \end{array}$$

$$D_4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & & & 1 \\ & 0 & & 6 \\ & & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \end{array} \Rightarrow D_4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \end{array}$$

Floyd-Warshall algoritmo pseudokodas

```
let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
for each edge (u, v) do
    dist[u][v] ← w(u, v)  // The weight of the edge (u, v)
for each vertex v do
    dist[v][v] ← 0
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j]
                dist[i][j] ← dist[i][k] + dist[k][j]
            end if
```

Šaltiniai

- Wikipedia
  Floyd-Warshall algorithm
  https://en.wikipedia.org/wiki/Floyd-Warshall_algorithm
- Programiz.com
  Floyd-Warshall algorithm
  https://www.programiz.com/dsa/floyd-warshall-algorithm