Algoritmai

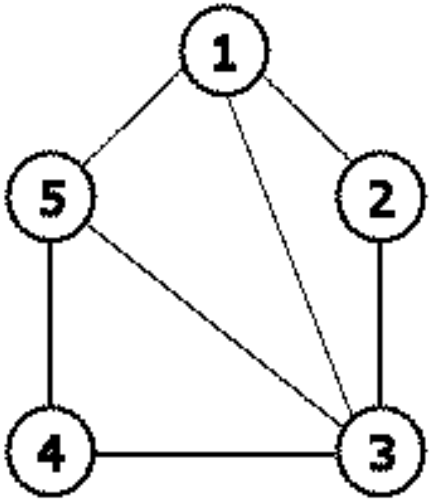
Trumpiausio kelio algoritmai

Trumpiausio kelio paieška

- Tai algoritmas, skirtas rasti trumpiausią kelią tarp dviejų taškų grafe.
- Pritaikymas:
  - Navigacijos sistemos (Google Maps, GPS).
  - Tinklo maršrutizavimas (interneto duomenų perdavimas).
  - Žaidimų kūrimas (personažų judėjimo trajektorijos).
- Tikslas – surasti kelią, kuris turi mažiausią "kainą" (svorį) tarp pradžios ir tikslo taškų

Kas yra grafas

Grafas yra duomenų struktūra, sudaryta iš **viršūnių** ir **briaunų**, kurios jungia šias viršūnes.



Grafų elementai:

- **Viršūnė (Vertex)**: atskiras taškas grafe, pvz., vieta žemėlapyje.
- **Briauna (Edge)**: ryšys tarp dviejų viršūnių, pvz., kelias tarp dviejų miestų.
- **Svoris (Weight)**: kiekviena briauna gali turėti skaičių, kuris rodo atstumą, laiką ar kainą

GRAFŲ TEORIJA: https://www.youtube.com/watch?v=1yXC3ersn3s

Grafo svoris

Svoris – tai skaičius, priskiriamas grafo briaunoms (ryšiams tarp viršūnių).

- Ką rodo svoris?
  - Atstumą tarp dviejų taškų.
  - Laiką, per kurį įveikiamas kelias.
  - Kaštus ar kainą tarp dviejų taškų.

Kaip apskaičiuojamas svoris?

- **Rankiniu būdu:** svoriai priskiriami pagal realius duomenis (pvz., atstumai tarp miestų, kelio kainos).
- **Automatiniai modeliai:** tam tikrose situacijose svoris apskaičiuojamas dinamiškai, naudojant tam tikrą funkciją (pvz., kelio ilgio, kainos ar kito parametro vertę).
- **Bendras svoris:** Kelių briaunų (kelio) svoris yra visų briaunų svorių suma.
  - Pvz., jei kelias iš A į B per C turi briaunų svorius 5 ir 10, bendras svoris = 5 + 10 = 15.

Neigiamas svoris

- Neigiamas svoris naudojamas situacijose, kai perėjimas tarp dviejų taškų yra „naudingas“ (pvz., gaunamos nuolaidos ar papildomi resursai).

  Pvz., jei neigiamas svoris yra -3, bendras kelio svoris gali sumažėti:
  - Kelias: A → B → C
  - Svoris: 2 + (-3) = -1.

- Problemos su neigiamais svoriais:
  - Neigiamų svorių naudojimas gali sukelti neigiamus ciklus, kai kelio svoris mažėja iki begalybės, pvz., važiuojant ratu galima nuolat „gauti naudą“.

Dijkstros algoritmas

Naudojamas trumpiausiam keliui rasti grafe su teigiamais svoriais.

- **Veikimo principas**:
  - Inicializuojamas pradinio taško atstumas kaip 0, o kitų – kaip begalybė.
  - Algoritmas pasirenka viršūnę su mažiausiu atstumu ir atnaujina jos kaimynų atstumus.
  - Procesas kartojamas, kol visi taškai aplankyti.

- **Trūkumas**: neveikia, jei grafas turi neigiamų svorių.

Bellman-Ford algoritmas

Naudojamas grafuose su neigiamais svoriais.

- **Veikimo principas**:
  - Kiekviena briauna peržiūrima maksimaliai (n-1) kartų, kur n yra viršūnių skaičius.
  - Jei randa trumpesnį kelią, atnaujina atstumą.
  - Gali aptikti neigiamus ciklus (kai kelio ilgis gali mažėti iki begalybės).
- **Privalumas**: gali apdoroti grafus su neigiamais svoriais.

BFS algoritmas (Plotis Pirmas Paieška)

- Naudojamas, kai **visų briaunų svoriai vienodi** (pvz., nuliniai arba vienetiniai svoriai).

- **Veikimo principas**:
  - Pradinis taškas įtraukiamas į eilę.
  - Kiekvienas taškas aplankomas pagal atstumą nuo pradinio taško, einant „bangomis“ per grafą.
  - Idealiai tinka nedidelių vienodo svorio grafų paieškai.

- **Privalumas**: paprastas ir efektyvus, kai nėra skirtingų svorių.

Parametrų vaidmuo algoritmuose

- **Parametrai** – tai reikšmės, kurios įvedamos į algoritmą, kad jis galėtų atlikti užduotį.
- Jie lemia, kaip ir ką algoritmas skaičiuos.

Parametrai nulemia:

- Nuo kurio taško pradėti skaičiavimus.
- Kokį kelią algoritmas pasirinks.
- Kiek truks skaičiavimai ir kokie bus galutiniai rezultatai.

Algoritmo įėjimo parametrai

- Pradiniai taškai (Start Points):
  - Tai taškai, nuo kurių algoritmas pradeda savo paiešką.
  - Pvz., rasti trumpiausią kelią iš vieno miesto į kitą: pradinis miestas yra įėjimo parametras.

- Galutiniai taškai (End Points):
  - Taškai, į kuriuos algoritmas turi nukeliauti.
  - Pvz., kelionės tikslas tarp miestų – galutinis taškas algoritme.

Svoriai kaip parametrai

- Svorio reikšmės:
  - Kiekvienas grafas gali turėti skirtingas svorių reikšmes tarp taškų (briaunų).
  - Svoriai gali būti:
    - Teigiami (nustatantys atstumą ar kaštus).
    - Neigiami (parodantys naudą ar pelną).
- Kaip svoriai keičia rezultatą?
  - Skirtingos svorio reikšmės gali visiškai pakeisti algoritmo elgesį:
    - Mažesni svoriai reiškia trumpesnį kelią.
    - Neigiami svoriai leidžia algoritmui optimizuoti net „taupant“ resursus.

Kaip parametrai lemia algoritmo eigą

Algoritmo eiga priklauso nuo parametrų:

- Pradiniai taškai nustato, kur algoritmas pradeda paiešką:
  - Pavyzdžiui, rasti trumpiausią kelią nuo tam tikro miesto priklauso nuo to, kuris miestas pasirinktas kaip pradinis taškas.
- Galutiniai taškai nustato, kur algoritmas turi pasiekti tikslą:
  - Algoritmas keičia kelią, atsižvelgdamas į galutinį tašką, kurį siekia pasiekti.

Algoritmo logika keičiasi su parametrais:

- Priklausomai nuo **pradinių sąlygų**, skaičiavimai prasidės nuo skirtingų vietų grafe.
- Parametrai valdo **paieškos kryptį** ir taškus, kuriuos reikia apskaičiuoti.

Parametrai ir rezultatas

Svorio įtaka rezultatui:

- Skirtingi svoriai tarp briaunų gali lemti skirtingus rezultatus:
  - Mažesni svoriai lemia trumpesnį kelią, o didesni – ilgesnį.
  - Neigiami svoriai gali net sukelti ciklus arba „pelningus“ kelius (kai kelionė atneša naudą).

Parametrų pakeitimas keičia galutinį rezultatą:

- Keičiant pradinius taškus ar svorius, gauname skirtingus trumpiausius kelius:
  - Pavyzdžiui, pradėjus nuo kito taško arba pakeitus briaunos svorį, gali keistis optimalus kelias.

Pavyzdys su parametrais

Praktinis pavyzdys:

- Grafas su taškais A, B, C, D:
- Pradiniai taškai: A.
- Galutiniai taškai: D.
- Skirtingi svoriai tarp A-B (3), B-C (2), A-C (5), C-D (1).

- Kaip keičiasi rezultatas keičiant svorius?
  - Jei svoris tarp A ir B sumažinamas iki 1, algoritmas pasirenka trumpesnį kelią per B.

Pavyzdžiai su Python, naudojant grafus

```python
# Viršūnių sąrašas (vertices)
vertices = ['A', 'B', 'C', 'D', 'E']

# Briaunų sąrašas su svoriais (edges)
edges = [
    ('A', 'B', 4),
    ('A', 'C', 2),
    ('B', 'C', 5),
    ('B', 'D', 10),
    ('C', 'E', 3),
    ('D', 'E', 1)
]

# Grafo atvaizdavimas: viršūnių ir briaunų su svoriais sąrašai
print("Viršūnių sąrašas:", vertices)
print("Briaunų sąrašas (pradinis taškas, galutinis taškas, svoris):")
for edge in edges:
    print(f"{edge[0]} -> {edge[1]} (svoris: {edge[2]})")
```

```
Viršūnių sąrašas: ['A', 'B', 'C', 'D', 'E']
Briaunų sąrašas (pradinis taškas, galutinis taškas, svoris):
A -> B (svoris: 4)
A -> C (svoris: 2)
B -> C (svoris: 5)
B -> D (svoris: 10)
C -> E (svoris: 3)
D -> E (svoris: 1)
```

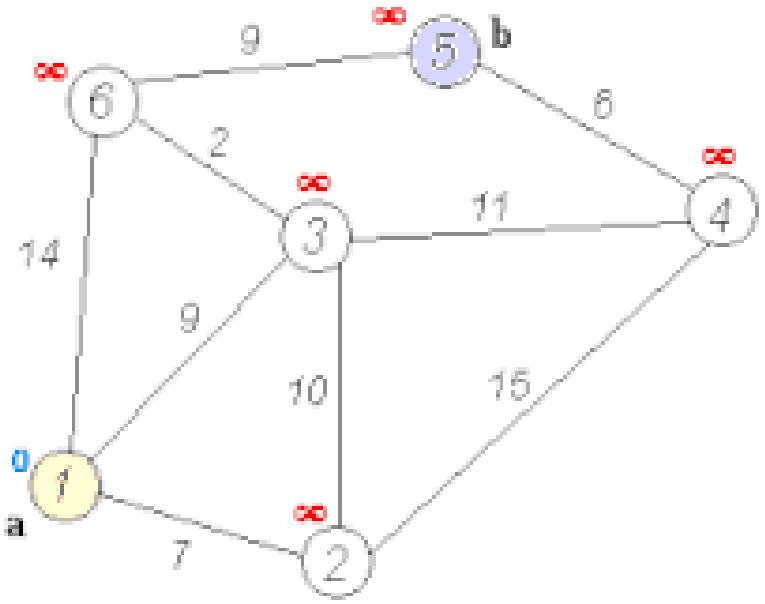
Pavyzdžiai su C++, naudojant grafus

```cpp
// Viršūnių sąrašas (vertices)
std::vector<char> vertices = {'A', 'B', 'C', 'D', 'E'};

// Briaunų sąrašas su svoriais (edges), naudojant std::tuple
std::vector<std::tuple<char, char, int>> edges = {
    {'A', 'B', 4},
    {'A', 'C', 2},
    {'B', 'C', 5},
    {'B', 'D', 10},
    {'C', 'E', 3},
    {'D', 'E', 1}
};

// Spausdiname viršūnių sąrašą
std::cout << "Viršūnių sąrašas: ";
for (char v : vertices) {
    std::cout << v << " ";
}
std::cout << std::endl;

// Spausdiname briaunų sąrašą su svoriais
std::cout << "Briaunų sąrašas (pradinis taškas, galutinis taškas, svoris):"
          << std::endl;
for (auto edge : edges) {
    std::cout << std::get<0>(edge) << " -> " << std::get<1>(edge)
              << " (svoris: " << std::get<2>(edge) << ")" << std::endl;
}
```

```
Viršūnių sąrašas: A B C D E
Briaunų sąrašas (pradinis taškas, galutinis taškas, svoris):
A -> B (svoris: 4)
A -> C (svoris: 2)
B -> C (svoris: 5)
B -> D (svoris: 10)
C -> E (svoris: 3)
D -> E (svoris: 1)
```

Dijkstros algoritmas

- Dijkstros algoritmas yra skaičiavimo metodas, naudojamas rasti trumpiausią kelią grafuose su teigiamais svoriais.
- Algoritmas nustato trumpiausią atstumą nuo pradinio taško iki visų kitų taškų grafuose.



Šaltinis: https://lt.wikipedia.org/wiki/Dijkstros_algoritmas

Algoritmo sudėtingumas

- Paprasčiausias Djikstros algoritmo įvykdymas laiko rinkinio Q viršūnes paprastame tiesiniame sąraše ar masyve, ir operacija Išrinkti mažiausią yra paprasta tiesinė paieška per visas viršūnes Q. Šiuo atveju algoritmo sudėtingumas yra

$$O(V^2)$$

Algoritmo žingsniai

1. Inicializavimas

- Nustatome pradinių taškų atstumus:
  - Atstumas nuo A iki A: 0
  - Atstumas nuo A iki B: ∞
  - Atstumas nuo A iki C: ∞
  - Atstumas nuo A iki D: ∞
  - Atstumas nuo A iki E: ∞
- Kuriame **neaplankytų viršūnių** sąrašą: {A, B, C, D, E}

- Grafas su viršūnėmis A, B, C, D, E ir atstumais:
  - A -> B (4)
  - A -> C (2)
  - B -> C (5)
  - B -> D (10)
  - C -> E (3)
  - D -> E (1)

Algoritmo žingsniai (tęs)

2. Keliaujame nuo pradinio taško (A):

- Patikriname kaimynus:
  - Atstumas į B: 0 + 4 = 4 (naujas atstumas)
  - Atstumas į C: 0 + 2 = 2 (naujas atstumas)
- Atstumas po pirmo žingsnio:
  - A: 0
  - B: 4
  - C: 2
  - D: ∞
  - E: ∞
- Pažymime A kaip aplankytą: {B, C, D, E}

- Grafas su viršūnėmis A, B, C, D, E ir atstumais:
  - A -> B (4)
  - A -> C (2)
  - B -> C (5)
  - B -> D (10)
  - C -> E (3)
  - D -> E (1)

Algoritmo žingsniai (tęs)

2. Pasikartojame per C (mažiausias atstumas):
   - Atstumas į E: 2 + 3 = 5 (naujas atstumas)
- Atstumas po šio žingsnio:
  - A: 0
  - B: 4
  - C: 2
  - D: ∞
  - E: 5

3. Pasikartojame per B:
   - Atstumas į D: 4 + 10 = 14 (naujas atstumas)

- Grafas su viršūnėmis A, B, C, D, E ir atstumais:
  - A -> B (4)
  - A -> C (2)
  - B -> C (5)
  - B -> D (10)
  - C -> E (3)
  - D -> E (1)

Galutinis rezultatas

- Atstumas po visų perėjimų:
  - A: 0
  - B: 4
  - C: 2
  - D: 14
  - E: 5
- Trumpiausi atstumai nuo A:
  - A -> A: 0
  - A -> B: 4
  - A -> C: 2
  - A -> D: 14
  - A -> E: 5

- Grafas su viršūnėmis A, B, C, D, E ir atstumais:
  - A -> B (4)
  - A -> C (2)
  - B -> C (5)
  - B -> D (10)
  - C -> E (3)
  - D -> E (1)

Pseudokodas

```
FOR i=0 to |V|-1
        dist(i)=INFINITY
        prev(i)=NULL
END FOR

WHILE F nepilnas
        imame viršūnę v iš U su artimiausiu keliu iki S
        pridedame V į F
        FOR V briaunai(v1,v2)
                IF (dist(v1)+ilgis(v1,v2) < dist(v2)
                        dist(v2)=dist(v1)+ilgis(v1,v2)
                        prev(v2)=v1
                        [galbūt reikia atnaujinti U]
                END IF
        END FOR
END WHILE
```

Kintamieji

S – pradžios viršūnė

F – aplankytų viršūnių sąrašas

G – grafas

E – briauna su viršūnėmis (v1,v2)

V – viršūnė

dist(i) – atstumų nuo S iki kiekvienos V masyvas

prev(i) – rodyklės į ankstesnes V

i – indeksas

U – neaplankytų viršūnių sąrašas

Apibendrinimas

- Trumpiausio kelio paieška: aptarėme, kaip trumpiausio kelio algoritmai taikomi realiame gyvenime, pvz., navigacijos sistemose ir tinklo maršrutizavime.
- Grafų sąvokos: kas yra grafas, viršūnės, briaunos ir svoriai.
- Dijkstra's algoritmas: veikia tik su teigiamiems svoriams ir kaip jis nustato trumpiausius kelius.
- Bellman-Ford algoritmas: gali dirbti su neigiamais svoriais ir kaip jis apdoroja grafus su neigiamais ciklais.
- BFS algoritmas: naudojamas, kai visos briaunos turi vienodą svorį.
- Grafo svoris ir neigiamas svoris.
- Kaip neigiami svoriai, kaip jie gali paveikti kelių paiešką.
- Dijkstros algoritmo įgyvendinimas