Dijkstran algoritmi

Henrik Aalto

Algokerho

24.11.2022

Kolme tasoa algoritmin ymmärtämiseen

Algoritmin ymmärrys voidaan jakaa karkeasti kolmelle tasolle:

1. Tietää, mitä algoritmi tekee, ja osaa käyttää sitä
2. Ymmärtää algoritmin toiminnan ja osaa soveltaa sitä
3. Osaa kehittää omia *algoritmeja*, jotka hyödyntävät Dijkstran ideoita

Taso 1. Perusteet

Dijkstran algoritmi:

- on yksi tärkeimmistä ja tunnetuimmista verkkoalgoritmeista.
- löytää painotetussa suunnatussa verkossa valitusta lähtösolmusta lyhimmät polut kaikkiin muihin solmuihin.
- on tehokas ja täten skaalautuu hyvin suuriin verkkoihin.

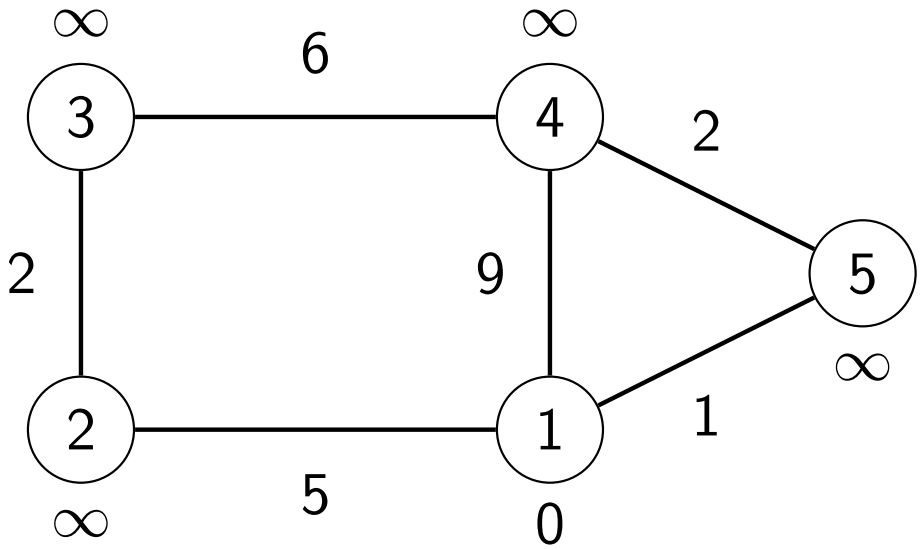
Taso 1. Toiminta

Toimintaperiaate

Dijkstran algoritmi käsittelee jokaisen solmun kerran. Lähtösolmu käsitellään ensin, minkä jälkeen valitaan käsittelemättömistä solmuista aina se, johon löydetty etäisyys on pienin.

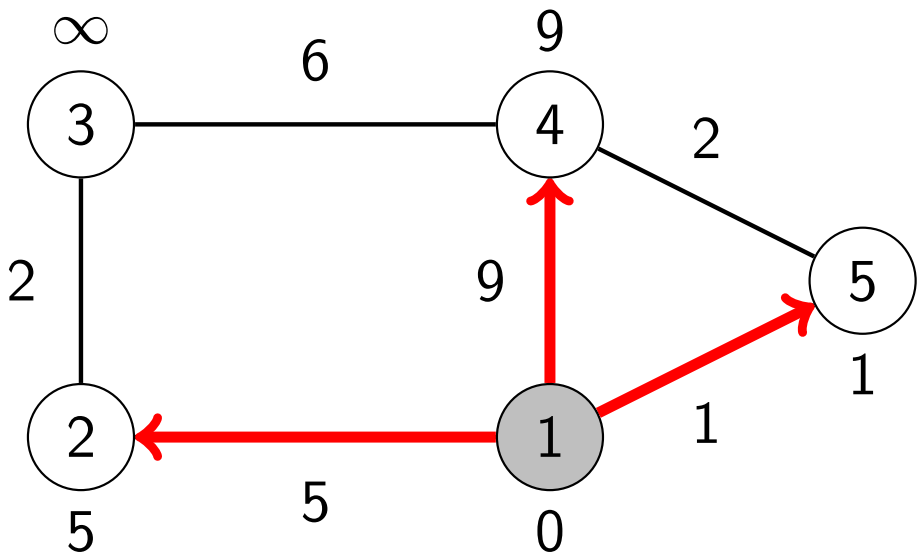
Taso 1. Toiminta

Seuraavaksi esimerkki algoritmin toiminnasta.[1] Aloitetaan haku solmusta 1, ja täytetään etäisyystaulukko.

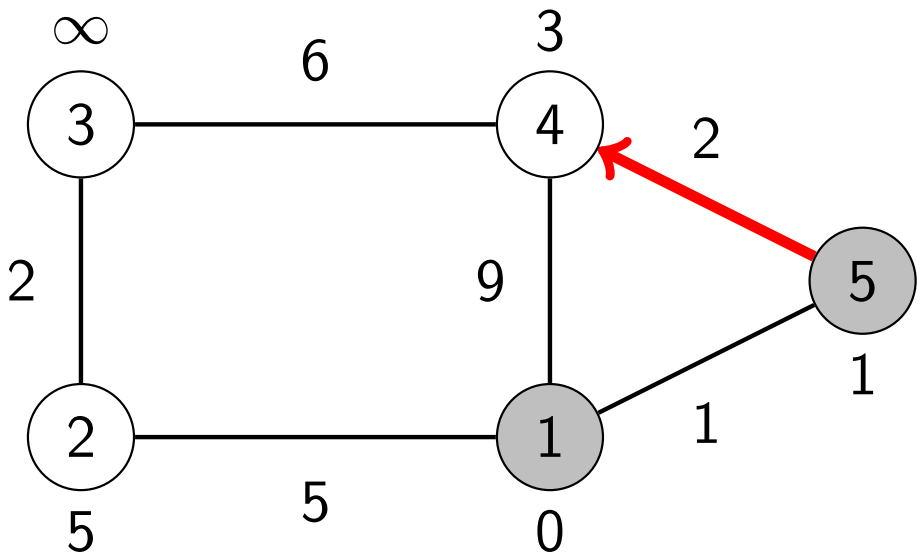


[1] Kuvat Kisakoodarin käsikirjasta

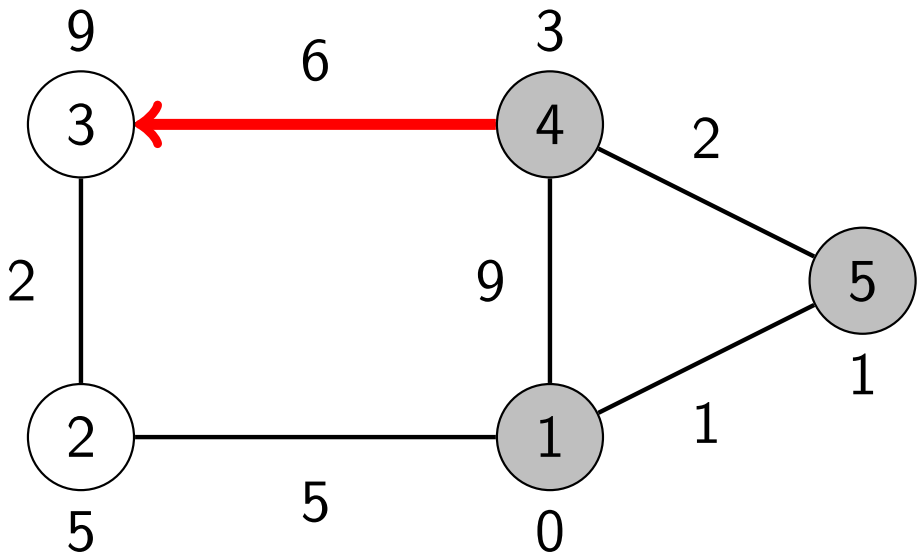
Taso 1. Toiminta



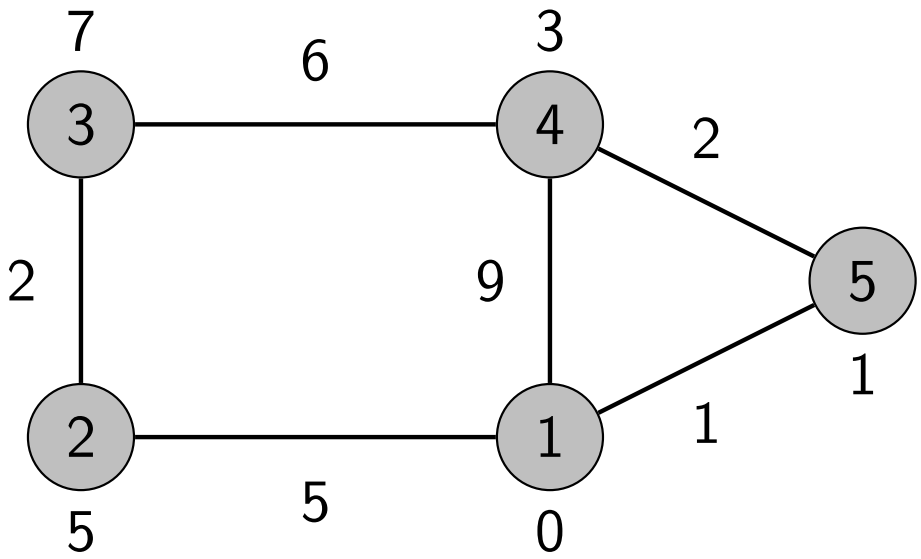
Taso 1. Toiminta



Taso 1. Toiminta



Taso 1. Toiminta



Taso 1. Implementaatio

```
distance = {node: inf if node != start else 0 for node in nodes}
visited = set()
while len(visited) < n:
  node = ???
  if node in visited:
    continue
  visited.add(node)
  for other, edge_weight in neighbours(node):
    if other not in visited:
      ???
```

Nyt `distance`-taulu sisältää lyhimmän etäisyyden jokaiseen solmuun.

Taso 2. Ymmärrys

- Jos haluaa laajentaa algoritmia tai muokata sen toimintaa, on tärkeää ymmärtää, miten ja miksi se toimii.
- Onkin hyvä harjoitustehtävä todistaa, miksi Dijkstran algoritmi antaa lyhimmät etäisyydet.
- Algoritmi lukitsee solmun etäisyysarvon, kun solmu käsitellään. Miksi ei ole mahdollista, että jokin solmu käsiteltäisiin ”liian aikaisin”, eli ennen kuin lyhin polku siihen on löytynyt?

Taso 2. Muunnelmat

Käyttämällä kekoa tai binäärihakupuuta saa solmun, johon etäisyys on pienin, löydettyä tehokkaasti.
Pythonissa:

```
import heapq
queue = []
heapq.heappush(queue, (0, start))
shortest_distance, node = heapq.heappop(queue)
```

C++:ssa:

```
priority_queue<pair<int, int>> pq;
int distance = ???, start = ???;
pq.push({-distance, start); // Huom. käänteinen etäisyys
auto [shortest_distace, node] = pq.top();
pq.pop();
shortest_distance = -shortest_distance;
```

Taso 2. Sovellukset

- Dijkstran algoritmi on itsessään yksinkertainen, ja harjoittelun jälkeen myös sen implementointi on helppoa.
- Ongelmien haastavuus onkin usein keksiä, miten Dijkstraa voi hyödyntää.
- Välillä annettuun verkkoon voi joutua lisäämään solmuja/tiloja (harjoitustehtävä Alennus)
- Toisinaan verkkorakenne ei itsestäänselvä, vaan haasteena on rakentaa verkko, jossa etäisyydet etsiä (Jakarta Skyscrapers)

Taso 3. *Fracturing Search*

Yksi esimerkki Dijkstran algoritmin ideoiden soveltamisesta muihin algoritmeihin on tekniikka nimeltä *Fracturing Search*[2], jolla voi ratkaista hakuongelmia suurissa hakuavaruuksissa.

Perusongelma

Annettuna on juurrettu puu, jossa jokaiseen solmuun on liitetty lukuarvo. Vanhemman arvo on aina lapsen arvoa pienempi. Jokaisella solmulla on enintään D lasta. Etsi puun K pienintä solmuarvoa.

Ratkaisu: Dijkstra puussa. Solmuja käsitellään enintään $\mathcal{O}(KD)$ kappaletta. Aikavaativuus $\mathcal{O}(KD \log(KD))$.

[2] `https://usaco.guide/adv/fracturing-search`

Taso 3. *Fracturing Search*

Ratkaisu perusongelmaan yleistyy. Moni hakuongelma voidaan redusoida samantyyppiseen tilanteeseen.

Vaatimukset:

- Etsitään K:nneksi pienintä arvoa (suuresta) hakuavaruudesta
- Hakuavaruus tulee muokata samanlaiseksi puurakenteeksi:
  - Juurella on pienin arvo
  - Lapsella on vanhempaa suurempi arvo
  - Kaikki objektit löytyvät juuren alipuusta
  - Lapsien alipuut jakavat hakualueen toisistaan erillisiin osiin

Taso 3. *Fracturing Search*

Seuraava tehtävä on Usaco Training Campilta[3]

Esimerkkitehtävä

Annettuna on painotettu, suuntaamaton verkko, jossa on $N \leq 50$ solmua ja $M \leq \binom{N}{2}$ kaarta. Etsi K:nneksi ($K \leq 10^4$) pienin virittävä puu.

[3] Lähde. `https://usaco.guide/adv/fracturing-search`

Taso 3. *Fracturing Search*

Ratkaisu

- Luodaan virittävien puiden joukosta puu, jossa jokaisella virittävällä puulla on enintään $\mathcal{O}(N)$ lasta.
- Tämä on hankalaa tehdä niin, että lapsien alipuut ovat täysin erillisiä, mutta kuitenkin sisältävät kaikki mahdolliset virittävät puut
- Juuri on tietysti verkon pienin virittävä puu.

Taso 3. *Fracturing Search*

Ratkaisu

- Luodaan virittävien puiden joukosta puu, jossa jokaisella virittävällä puulla on enintään $\mathcal{O}(N)$ lasta.
- Tämä on hankalaa tehdä niin, että lapsien alipuut ovat täysin erillisiä, mutta kuitenkin sisältävät kaikki mahdolliset virittävät puut
- Juuri on tietysti verkon pienin virittävä puu.
- Olkoon minkä tahansa virittävän puun $N-1$ kaarta numeroitu numeroin $1 \ldots N-1$. Tällöin määritellään sille $N-1$ lasta seuraavasti: lapsen i $(1 \leq i \leq N-1)$ alipuu sisältää vain ratkaisuja (virittäviä puita), jotka sisältää kaaret $[1, i-1]$, mutta ei kaarta i.

Tehtäviä!

Harjoitustehtäviä löytyy osoitteesta `cses.fi/algo/list`.