

LYHIMPIEN POLKUJEN HAKUALGORITMIT JA -JÄRJESTELMÄT

Rodion Efremov

Tietojenkäsittelytieteen laitos, Helsingin yliopisto

- Suunnattu verkko on $G = (V, A)$, missä V on solmujen joukko ja $A \subset V \times V$ on suunnattujen kaarien joukko.
- Suuntaamaton verkko $G = (V, E)$ voidaan aina simuloida suunnatulla verkolla (V, A) laittamalla A :han kaaret (u, v) ja (v, u) jokaisella suuntaamattomalla kaarella $\{u, v\} \in E$.
- Jatkossa merkitsemme $n = |V|$ ja $m = |E|$.

- k :n kaaren polku on $\gamma = \langle u_0, u_1, \dots, u_k \rangle$, missä kukin solmu esiintyy vain kerran ja jokaisella $i \in \{0, 1, \dots, k-1\}$ $(u_i, u_{i+1}) \in A$.
- Painotettujen verkkojen kohdalla otaksumme kunkin kaaren (u, v) painon $w(u, v)$ olevan **ei-negatiivinen**.
- Suuntamattomassa verkossa solmun u "aste" on $d(u) = |\{\{u, v\} : \{u, v\} \in E\}|$.
- Suunnatussa verkossa solmun u "sisäänaste" (engl. *indegree*) on $\deg^-(u) = |\{(v, u) : (v, u) \in A\}|$ ja "ulosaste" (engl. *outdegree*) on $\deg^+(u) = |\{(u, v) : (u, v) \in A\}|$.

Leveyssuuntainen haku

- Löytää lyhimmän polun (yhden monesta mahdollisesta) painottamattomassa verkossa.
- Toteutus vaatii vain jonon ja hajautustaulun.
- Toimii ajassa $\mathcal{O}(n + m) \approx \sum_{i=0}^N d_i$, missä N on lyhimmän polun solmujen määrä ja d keskiarvoinen solmun aste tai solmun ulosaste verkon tyypistä riippuen.

Algoritmi 2: BREADTH-FIRST-SEARCH(G, s, t)

```
1  $Q = \langle s \rangle$ 
2  $\pi(s) = \text{nil}$ 
3 while  $|Q| > 0$  do
4      $u = \text{DEQUEUE}(Q)$ 
5     if  $u$  is  $t$  then
6         return  $\text{TRACEBACK-PATH}(u, \pi, \text{nil})$ 
7     for  $(u, v) \in G.A$  do
8         if  $v$  is not yet mapped in  $\pi$  then
9              $\pi(v) = u$ 
10            ENQUEUE( $Q, v$ )
11 return  $\langle \rangle$ 
```

Voiko leveyssuuntaisen haun nopeuttaa?

Voi!

Kaksisuuntainen leveyssuuntainen haku

- Ajaa kaksi hakuavaruutta: yksi normaalin tapaan lähtösolmusta, ja toinen "takaperin" maalisolmusta.
- Kun kaksi yllä mainittua hakuavaruutta kohtavat jossain "keskellä", rakennetaan lyhin polku.
- Aikavaativuus on $2 \sum_{i=0}^{\lceil N/2 \rceil} d^i$.
- Verkosta riippuen voi olla jopa ~ 1000 kertaa nopeampi kuin yksisuuntainen BFS.

Algoritmi 3: BIDIRECTIONAL-BREADTH-FIRST-SEARCH(G, s, t)

```
1  $Q, \pi, d = (\langle s \rangle, (s, \text{nil}), (s, 0))$ 
2  $Q_{REV}, \pi_{REV}, d_{REV} = (\langle t \rangle, (t, \text{nil}), (t, 0))$ 
3  $\tau, \mu = (\text{nil}, \infty)$ 
4 while  $|Q||Q_{REV}| > 0$  do
5   if  $\tau$  is not nil and  $d(\text{HEAD}(Q)) + d_{REV}(\text{HEAD}(Q_{REV})) \geq \mu$  then
6     return  $\text{TRACEBACK-PATH}(\tau, \pi, \pi_{REV})$ 
7    $u = \text{DEQUEUE}(Q)$ 
8   if  $u$  is mapped in  $\pi_{REV}$  and  $\mu > d(u) + d_{REV}(u)$  then
9      $\mu = d(u) + d_{REV}(u)$ 
10     $\tau = u$ 
11   for  $(u, v) \in G.A$  do
12     if  $v$  is not yet mapped in  $\pi$  then
13        $\pi(v) = u$ 
14        $\text{ENQUEUE}(Q, v)$ 
15        $d(v) = d(u) + 1$ 
16    $u = \text{DEQUEUE}(Q_{REV})$ 
17   if  $u$  is mapped in  $\pi$  and  $\mu > d(u) + d_{REV}(u)$  then
18      $\mu = d(u) + d_{REV}(u)$ 
19      $\tau = u$ 
20   for  $(v, u) \in G.A$  do
21     if  $v$  is not yet mapped in  $\pi_{REV}$  then
22        $\pi_{REV}(v) = u$ 
23        $\text{ENQUEUE}(Q_{REV}, v)$ 
24        $d_{REV}(v) = d_{REV}(u) + 1$ 
25 return  $\langle \rangle$ 
```

Miten rakentaa polut lyhimpien polkujen puusta?

Tarvitaan vain kuvaus π (ja myös π_{REV} mikäli polku oli haettu kaksisuuntaisella haulla).

Algoritmi 1: TRACEBACK-PATH(x, π, π_{REV})

```
1  $u = x$ 
2  $p = \langle \rangle$ 
3 while  $u$  is not nil do
4    $p = \langle u \rangle \circ p$ 
5    $u = \pi(u)$ 
   Kaksisuuntainen haku?
6 if  $\pi_{REV}$  is not nil then
7    $u = \pi_{REV}(x)$ 
8   while  $u$  is not nil do
9      $p = p \circ \langle u \rangle$ 
10     $u = \pi_{REV}(u)$ 
11 return  $p$ 
```

Dijkstran algoritmi

- Vuonna 1959 Edsger W. Dijkstra esitti kuuluisan polunhakualgoritminsa, joka toimii polynomisessa ajassa.
- FIFO jonon sijasta prioriteettijono; kutsutaan usein "avoimeksi" listaksi (engl. *open set*).
- Hajautustauluun perustuva joukkorakenne; kutsutaan usein "suljetuksi" listaksi (engl. *closed set*).
- g -kuvaus, joka kuvaa kunkin saavutetun solmun toistaiseksi pienempään etäisyyteen lähtösolmusta laskettuna.
- π -kuvaus, aivan kuten BFS:ssä (kuvaa solmun edeltäjänsä lyhimpien polkujen puussa).
- Kun solmu poistetaan avoimesta listasta, sen g -arvo on optimaali.

Algoritmi 4: DIJKSTRA-SHORTEST-PATH(G, s, t, w)

```
1 OPEN, CLOSED,  $g, \pi = (\{s\}, \emptyset, \{(s, 0)\}, \{(s, \text{nil})\})$ 
2 while  $|OPEN| > 0$  do
3    $u = \arg \min_{x \in OPEN} g(x)$ 
4   if  $u$  is  $t$  then
5     return TRACEBACK-PATH( $u, \pi, \text{nil}$ )
6   OPEN = OPEN -  $\{u\}$ 
7   CLOSED = CLOSED  $\cup \{u\}$ 
8   for  $(u, x) \in G.A$  do
9     if  $x \in CLOSED$  then
10      continue
11      $g' = g(u) + w(u, x)$ 
12     if  $x \notin OPEN$  then
13       OPEN = OPEN  $\cup \{x\}$ 
14        $g(x) = g'$ 
15        $\pi(x) = u$ 
16     else if  $g(x) > g'$  then
17        $g(x) = g'$ 
18        $\pi(x) = u$ 
19 return  $\langle \rangle$ 
```

Dijkstran algoritmi

- Dijkstran algoritmi voidaan mieltää BFS:n yleistykseksi painotetuissa verkoissa: samoin kuten BFS, Dijkstran algoritmi kasvattaa hakuavaruutensa "kaikkiin suuntiin" laajenevan pallon tavoin.

A* - haku

- Pseudokoodi tasan sama kuten Dijkstran algoritmilla, paitsi että rivillä 3 $g(x)$ on korvattava $f(x)$:llä, jolle siis $f(x) = g(x) + h(x)$, missä $h(x)$ on optimistinen (eli aliarvioitu) kustannus solmusta x maalisolmuun.
- Intuitio järjestelyn takana on se, että A* "tietää" mihin suuntaan kannattaa lähteä kasvattamaan hakuavaruuden päästääkseen maalisolmuun nopeammin.
- Määrittämällä $h(u) = 0$ kaikilla $u \in V$, A* palautuu Dijkstran algoritmiksi.