Kauppamatkustajan ongelma

Sisällysluettelo

[Johdanto 1](#_Toc337885816)

[Tarkka ratkaisu 1](#_Toc337885817)

[Approksimointi 1](#_Toc337885818)

[Ratkaisun tietorakenteet ja algoritmit 2](#_Toc337885819)

[Algoritmit 2](#_Toc337885820)

[Muut menetelmät 2](#_Toc337885821)

[Tietorakenteet (suluissa menetelmä/algoritmi, missä tietorakennetta sovelletaan) 2](#_Toc337885822)

[Ohjelman toiminta 2](#_Toc337885823)

[Aika- ja tilavaativuudet 2](#_Toc337885824)

[Lähteet 3](#_Toc337885825)

# Johdanto

Tehtävässä ratkaistaan klassista verkkoihin liittyvää kauppamatkustajan ongelmaa. Kauppamatkustaja lähtee paikasta A tarkoituksenaan käydä *n*-määrässä kohteita palaten takaisin lähtöpisteeseen. Ongelma on, missä järjestyksessä kohteet tulee käydä, jotta kokonaisreitin pituus on lyhin.

# Tarkka ratkaisu

Tehtävän tarkoituksena on ratkaista ongelma ensin tarkasti käyttämällä ns. *branch and bound* –menetelmää. Toinen vaihtoehto tehtävän ratkaisemiseksi olisi käyttää *brute force* –menetelmää, jossa jokainen mahdollinen reittivaihtoehto lasketaan ja lopuksi valitaan lyhyin lasketuista reiteistä. *Branch and bound* –menetelmän etu tähän verrattuna on se, että kyseisessä menetelmässä reitin laskeminen lopetetaan mikäli laskennan edetessä ylitetään jo aiemmin saavutettu minimireitin pituus. Tällöin kaikkia reittejä ei tulla laskemaan, koska tiedetään, että ne eivät johda parempaan lopputulokseen kuin mikä aiemmin on saavutettu.

# Approksimointi

Tarkan laskennan lisäksi ratkaisu approksimoidaan kahdella valitulla algoritmilla (Prim ja Greedy). Approksimoinnilla pyritään selvittämään saadaanko algoritmeilla paremmalla vasteajalla samoja tai ainakin hyvin lähellä olevia ratkaisuja. Lisäksi selvitetään mikä algoritmeista antaa parhaita tuloksia kun solmujen lukumäärä kasvaa hyvin suureksi.

**Primin** algoritmi selvittää verkon lyhyimmän viritetyn puun. Approksimaatio saadaan luomalla algoritmilla lähtöpisteestä alkaen lyhyin virittävä puu, suoritetaan syvyyssuuntainen läpikäynti ja lopuksi yhdistetään viimeinen piste alkupisteeseen.

**Greedy** algoritmi toimii ahneesti. Algoritmi etenee verkossa aina lähimpään solmuun, jota ei ole vielä tutkittu. Näin käydään läpi kaikki solmut kunnes viimeisestä palataan alkupisteeseen.

# Ratkaisun tietorakenteet ja algoritmit

Tässä kappaleessa kuvataan algoritmit ja menetelmät, joita ohjelmassa tullaan käyttämään. Lisäksi kuvataan algoritmeissa käytettävät tietorakenteet, jotka tullaan toteuttamaan itse.

### Algoritmit

* Prim, DFS, Greedy

### Muut menetelmät

* Branch-and-bound

### Tietorakenteet (suluissa menetelmä/algoritmi, missä tietorakennetta sovelletaan)

* Puu (Branch and Bound)
* Minimikeko (Prim, DFS)
* Pino (Brute Force, Branch and Bound, Prim, DFS, Greedy)

# Ohjelman toiminta

Ohjelma saa syötteenään etäisyysverkon vierusmatriisimuodossa. Vierusmatriisi muodostetaan staattisesti. Minimisyöte, millä ratkaisua ja approksimaatioita testataan on 4 solmua (mukaan lukien lähtösolmu). Tätä suurempia verkkoja testataan aina n.16 solmun verkkoihin asti, joka alkaa olla tarkan ratkaisun maksimiraja. Tämän lisäksi tutkitaan suurempia verkkoja approksimaatioilla.

Ohjelma tulostaa seuraavat asiat:

1. Lyhyimmän reitin pituus
2. Lyhyin reitti (tulostetaan solmut järjestyksessä)
3. Ratkaisuun kulutettu aika sekunteina

# Aika- ja tilavaativuudet

* Branch and Bound (n = solmujen lukumäärä)
  + aikavaativuuden worst case 2^n, käytännössä tulisi olla nopeampi
  + tilavaativuus: n^2
* Prim (|E| = solmujen lukumäärä, |V| = kaarien lukumäärä)
  + Aikavaativuuden worst-case: (|E| log|V|)
  + Tilavaativuus: |V|

# Lähteet

Primin ja DFS algoritmit ja muut tietorakenteet kuvattuna Tietorakenteet kevät 2012 [kurssimateriaalissa](http://www.cs.helsinki.fi/u/floreen/tira2012/tira.pdf).