Käyttöohje - Kauppamatkustajan ongelma

Sisällysluettelo

[Johdanto 1](#_Toc337885397)

[Ohjelmointirajapinta 1](#_Toc337885398)

[Alustaminen 1](#_Toc337885399)

[Tarkat laskennat 2](#_Toc337885400)

[Approksimoinnit 2](#_Toc337885401)

[Tulokset 2](#_Toc337885402)

# Johdanto

Tässä dokumentissa kuvataan Kauppamatkustajan ongelman toteuttavan moduulin TSP (Traveling Salesman Problem) käyttöohje ja sen ohjelmointirajapinta.

# Ohjelmointirajapinta

Traveling Salesman moduuli on toteutettu abstraktina ohjelmointirajapintana. Sen tarkoitus on ainoastaan tuottaa ratkaisu kauppamatkustajan ongelmaan. Lähtödatan muodostaminen ja sen oikeellisuuden varmistaminen on täysin rajapintaa hyödyntävän järjestelmän vastuulla.

## Alustaminen

Alustusoperaatiolle annetaan etäisyysmatriisi, joka on *n X n* –matriisi, jossa on esitetty kohteiden väliset etäisyydet. Tämän matriisin luominen ja oikeellisuus on siis kokonaan moduulia hyödyntävän järjestelmän vastuulla. Alla esimerkki oikeanlaisesta etäisyysmatriisista, jossa on 5 kohdetta. **Huomioitavaa on, että lähtöpiste tulee olla aina indeksissä 0!**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** |
| **0** | 0 | 34 | 66 | 21 | 96 |
| **1** | 34 | 0 | 57 | 43 | 9 |
| **2** | 66 | 57 | 0 | 88 | 13 |
| **3** | 21 | 43 | 88 | 0 | 25 |
| **4** | 96 | 9 | 13 | 25 | 0 |

Alustuksessa luodaan TravelingSalesman olio etäisyysmatriisilla:

***TravelingSalesman travelingSalesman = new TravelingSalesman(int[][] graph);***

## Tarkat laskennat

Nämä metodit laskevat tarkan vastauksen. Johtuen suuresta laskentamäärästä, nämä ovat tehokkaita vain tiettyyn rajaan asti.

Tämä metodi laskee tarkan vastauksen käyttämällä branch-and-bound –metodia (tehokas vielä n.14-18 kohteen kanssa riippuen palvelimen tehokkuudesta):  
***travelingSalesman.calculateWithBranchAndBound()***

Tämä metodi laskee tarkan vastauksen käyttämällä brute forcea (tehokas vielä n.10-12 kohteen kanssa riippuen palvelimen tehokkuudesta):  
***travelingSalesman.calculateWithBruteForce()***

## Approksimoinnit

Nämä metodit antavat arvion parhaasta mahdollisesta reitistä eri laskenta-algoritmeilla, jotka eivät ole niin raskaita kuin tarkan vastauksen antavat algoritmit. Tämän johdosta näiden antamat tulokset eivät voivat poiketa merkittävästikin optimaalisesta, mutta ovat hyödyllisiä kun pyritään laskemaan optimoitua reittiä erittäin suurilla kohdemäärillä.

Approksimaatio käyttäen Primin algoritmia pienimmän virittäjäpuun löytämiseksi ja DFS-algoritmia puun läpikäymiseksi (tehokas vielä n.40 kohteen kanssa):  
**travelingSalesman.approximateWithPrim()**

Approksimaatio käyttäen Primin algoritmia pienimmän virittäjäpuun löytämiseksi ja yksinkertaista pinorakennetta, jonka avulla käydään löydetyt solmut läpi järjestyksessä (tehokas vielä n.40 kohteen kanssa):  
***travelingSalesman.approximateWithAlternativePrim()***

Approksimaatio yksinkertaisella ahneella algoritmilla, joka etsii aina lähimmän uuden solmun, jota ei ole vielä löydetty:   
***travelingSalesman.approximateWithGreedyAgorithm()***

## Tulokset

Moduuli palauttaa aina olion TSPResultHandler, joka sisältää laskennan tulokset.

Tämä metodi palauttaa int[] –taulukossa parhaan reitin. Taulukossa on järjestyksessä ne indeksit, minkä kautta kyseisen laskennan mielestä paras reitti kulkee. Huomioitavaa on, että ensimmäinen ja viimeinen indeksi on aina 0! ***TSPResultHandler.getBestRoute()***

Tämä metodi palauttaa lyhyimmän reitin pituuden.  
***TSPResultHandler.*getMinimumRouteLength()**