Määrittelydokumentti

Johdatus

Tarkoituksena on tehdä kattava ja helppokäyttöinen verkkokirjasto, joka tarjoaa mahdollisuuden verkon luomiseen, muokkaamiseen, ja tarkasteluun sekä yleisimmät algoritmit ongelmien ratkaisuun.

Polunetsintä solmujen välillä on pääosassa, joten verkon täytyy itse tietää, millainen se on (painoton verkko käyttää BFS-hakua, pelkästään positiivisia painoja sisältävä Dijkstraa, negatiivisia sisältävä Bellman-Fordia ja heuristiikan omaava A staria). Näin ohjelmoijan ei tarvitse itse valita algoritmia polunetsinnälle. Verkon tulee tarjota myös muita yleisiä verkkoalgoritmeja, kuten Flood fill, syklin tunnistus, jne.

Tietorakenteet ja algoritmit

Lista tarvittavista algoritmeista ja niiden käyttämistä tietorakenteista:

- BFS
  - Aikavaativuudeksi tulisi tulla O(|V|+|E|) ja tilavaativuuden O(|V|). Aputietorakentaita: Hahstable, LinkedList, Stack, Queue
- Dijkstra
  - Minikeko, jossa on decrease-key ominaisuus, jonka aikavaativuus on O(log(n)). Näin Dijkstra saadaan toimimaan ajassa O((|V|+|E|)log(|V|)) ja tilavaativuudeksi tulisi O(|V|). Aputietorakentaita: Hahstable, LinkedList, Stack
- Bellman-Ford
  - Aikavaativuudeksi tulisi tulla O(|V||E|) ja tilavaativuudeksi O(|V|). Aputietorakentaita: Hahstable, LinkedList, Stack
- A*
  - Aika- ja tilavaativuudeksi tulisi tulla sama kuin Dijkstran, mutta käytännössä algoritmi toimisi hieman nopeammin heuristiikan takia. Aputietorakentaita: MinHeap, Hahstable, LinkedList, Stack
- Flood fill
  - Aikavaativuudeksi tulisi tulla O(|V|+|E|) ja tilavaativuuden O(|V|). Aputietorakentaita: Hahstable, LinkedList, Stack, Queue
- DFS (syklien tunnistus)
  - Aikavaativuudeksi tulisi tulla O(|V|+|E|) ja tilavaativuuden O(|V|). Aputietorakentaita: Hahstable, LinkedList

Itse verkko on tärkein tietorakenne, joka käyttää aputietorakenteina Hashtablea ja LinkedListia.

Aika- ja tilavaativuuksien lähteet: en.wikipedia.org

Syötteet

Verkkoluokkaa (Graph) käytetään yhdistämään solmuja (Vertex) verkkoon luomalla kaaria (Edge) niiden välille. Koodina tämä toimisi seuraavasti:

```
Graph graph = new Graph();
Vertex a = new Vertex(”A”); // Solmu, jonka nimi on ”A” (vapaaehtoinen)
Vertex b = new Vertex(”B”);
Vertex c = new Vertex(”C”);
graph.connect(a,b,3);      // Alku- ja loppusolmu, sekä kaaren paino
graph.connect(b,c);        // ”Painoton kaari” (paino on 1)
```

Lyhyin polku solmusta a, solmuun c löytyisi seuraavasti:

```
graph.shortestPath(a,c);   // Palauttaa polun linkitettynä listana
```

Polunetsintäluokat ottavat konstruktorissa Graph-olion. Jokainen näistä algoritmeista toteuttaa rajapinnan PathFinder, johon liittyy metodi shortestPath(Vertex a, Vertex b).

Lopuista verkon toiminnallisuuksista ja niiden käytöstä javadocissa!