Praca domowa 10 – path

Michał Szczygieł

Wstęp

Zadanie składa się z 2 plików:

- Main.java
- Path.java

CEL:

Należy wyznaczyć ścieżkę o najmniejszym koszcie transportu łączącą wierzchołek o numerze 1 z wierzchołkiem k którego indeks określony jest parametrem linii komend <indeks> oraz wyznaczyć wartość tego kosztu. Koszt transportu definiowany dla każdego z węzłów ścieżki (dla węzła początkowego oraz końcowego z definicji przyjmujemy 0) jak bezwzględna wartość różnicy przepustowości krawędzi ścieżki o końcach w tym wierzchołku, czyli np. w przypadku węzła y przez który przechodzi ścieżka z wykorzystaniem krawędzi (x,y) oraz (y,z) o przepustowości odpowiednio p_{xy} i p_{yz} koszt transportu wyniesie | p_{xy} - p_{yz} |. Dla ścieżki koszt transportu definiowany jest jako suma kosztów transportu wszystkich wierzchołków wchodzących w skład ścieżki.

Klasa Main

Zadaniem tej klasy jest wczytanie danych z bazy danych oraz przekazanie otrzymanych wartości do klasy Path. Sposób tworzenia zapytania dla bazy danych w celu uzyskania danych odbywa się z użyciem:

```
/**
     * Query to be executed on database
     */
    private static final String QUERY = "SELECT x, y, p FROM Gtable";

…
 Connection conn = DriverManager.getConnection(connector);
            Statement statement = conn.createStatement();


            coordinates = retriveData(statement.executeQuery(QUERY));
```

Oprócz tego jak kolejnym parametrem jest indeks, wskazujący cel dla którego ma być wyliczona optymalna, co znaczy z namnijeszym kosztem przepływu.

Metody:

```
public static void main(final String... args)
```

- Metoda główna, przyjmuje dwa parametry connection string i indeks, przekazuje wczytane dane i rozmiar do klasy Path, wykonując cel, oraz wyświetlenie wynik znajdywania ścieżki, z dokładnością do 5 miejsc po przecinku.

```
public static Map<Point2D, Float> dbconnection(final String connector)
```

– Metoda ta ma na celu obsłużenie połączenia bazodanowego, oraz wykonanie zapytania SQL dla bazy danych. Oraz zwrócenie poprawnie sformatowanych danych poprzez funkcję umieszczoną poniżej.

```
private static Map<Point2D, Float> retriveData(final ResultSet result)
        throws SQLException
```

– Metoda ta odbiera dane po wykonaniu zapytania SQL do bazy danych, oraz upakowywuje dane do mapy gdzie klucz stawowi czas reprezentowany przez timestamp, a wartość jest wynikiem funkcji y=f(t) .

```
private static Timestamp getDate(Long milliPerDay, final String... date)
```

- Metoda ta zwraca czas w postaci Timestampa.

Klasa Path

Klasa path posiada wewnętrzną klasę:

```
public class Edge
```

Która, służy do przechowywania danych na temat kosztu przypływu jak i wierzchołków które są połączone tą krawędzią. Klasa ta również zawiera metody do budowania grafu, Depth-first search oraz Breadth-first search.

Realizacja algorytmu Path

Do rozwiązania problemu użyłem Algorytm Dynica. Algorytm ten w teorii optymalizacji, algorytm o złożoności czasowej $O(V^2E)$ rozwiązujący problem maksymalnego przepływu w sieci przepływowej umożliwiający odnajdywanie przepływu blokującego w sieci warstwowej. Algorytm skonstruowany został w 1970 roku przez izraelskiego profesora - Chaima Dynica. Strukturą zbliżony jest do alg. Edmondsa-Karpa.

Algorytm postępowania

1 - dziel graf na L warstw (przegląd wszerz)

2- utwórz ścieżki powiększające (przegląd w głąb), nie przemieszczając się względem tej samej warstwy

3- wyznacz maksymalny przepływ

Oczywiście w celach tego zadania algorytm został zmieniony w celu spełnienia wymagań projektu.

Złożoność obliczeniowa dla tego programu wynosi:

- Przeszukiwanie grafu tą metodą ma złożoność obliczeniową $O(V^2E)$

Dyskusja

Jest to jedna z lepszych metod co pokazuje poniższa tabela. Ciekawym rozwiązaniem musi się wydawać praca Jim Orlinsa, natomiast nie znalazłem wystarczających informacji aby móc zaimplementować to rozwiązanie używając właśnie tej metody.

Method	Complexity	Description
Linear programming		Constraints given by the definition of a legal flow. See the linear program here.
Ford–Fulkerson algorithm	$O(E \max\|f\|)$	As long as there is an open path through the residual graph, send the minimum of the residual capacities on the path. The algorithm works only if all weights are integers. Otherwise it is possible that the Ford–Fulkerson algorithm will not converge to the maximum value.
Edmonds–Karp algorithm	$O(VE^2)$	A specialization of Ford–Fulkerson, finding augmenting paths with breadth-first search.
Dinitz blocking flow algorithm	$O(V^2E)$	In each phase the algorithms builds a layered graph with breadth-first search on the residual graph. The maximum flow in a layered graph can be calculated in $O(VE)$ time, and the maximum number of the phases is n-1. In networks with unit capacities, Dinic's algorithm terminates in $O(E\sqrt{V})$ time.
General push-relabel maximum flow algorithm	$O(V^2E)$	The push relabel algorithm maintains a preflow, i.e. a flow function with the possibility of excess in the vertices. The algorithm runs while there is a vertex with positive excess, i.e. an active vertex in the graph. The push operation increases the flow on a residual edge, and a height function on the vertices controls which residual edges can be pushed. The height function is

		changed with a relabel operation. The proper definitions of these operations guarantee that the resulting flow function is a maximum flow.
Push-relabel algorithm with *FIFO* vertex selection rule	$O(V^3)$	Push-relabel algorithm variant which always selects the most recently active vertex, and performs push operations until the excess is positive or there are admissible residual edges from this vertex.
Dinitz blocking flow algorithm with dynamic trees	$O(VE \log(V))$	The dynamic trees data structure speeds up the maximum flow computation in the layered graph to $O(E\log(V))$.
Push-relabel algorithm with dynamic trees	$O(VE \log(V^2/E))$	The algorithm builds limited size trees on the residual graph regarding to height function. These trees provide multilevel push operations.
Binary blocking flow algorithm[8]	$O(E \min(V^{2/3}, \sqrt{E}) \log(V^2/E) \log U)$	The value U corresponds to the maximum capacity of the network.
MPM (Malhotra, Pramodh-Kumar and Maheshwari) algorithm	$O(V^3)$	Refer to the Original Paper.
Jim Orlin's + KRT (King, Rao, Tarjan)'s algorithm	$O(VE)$	Orlin's algorithm solves max-flow in $O(VE)$ time for $m \leq O(n^{(16/15)-\epsilon})$ while KRT solve it in $O(VE)$ for $m > n^{1+\epsilon}$