Praca domowa 1 – Path

Adam Kisielewski

1. Opis projektu

Celem projektu było dla podanych listy krawędzi (będących opisem grafu skierowanego, ważonego) znaleźć ścieżkę o najmniejszym koszcie (największej przepustowości).

2. Założenia

W bazie danych znajduje się lista krawędzi w postaci trójki liczb x, y, p, gdzie ‘x’ oznacza wierzchołek początkowy, ‘y’ wierzchołek końcowy, a ‘p’ przepustowość danej krawędzi. Dostęp do bazy danych zostaje przekazany jako jeden z parametrów wywołania programu, natomiast drugim parametrem jest numer wierzchołka, do którego trzeba znaleźć wymaganą ścieżkę (która jest poszukiwana z pierwszego wierzchołka);

Koszt transportu jest obliczany następująco:

- dla krawędzi jest to odwrotność przepustowości

- dla wierzchołka jest to wartość bezwzględna z różnicy przepustowości krawędzi, które te wierzchołek łączy.

3. Zastosowany algorytm

Algorytm z projekcie można podzielić na dwie fazy: znalezienie wszystkich ścieżek łączących wybrane wierzchołki i wybranie tej ścieżki, która ma najmniejszy koszt.

a) Znalezienie ścieżek

Kluczowym warunkiem poprawnego działania algorytmu było zapewnienie, że znajdzie on wszystkie unikatowe ścieżki łączące 2 wierzchołki. W tym celu zastosowano rekurencyjne sprawdzanie drzewa na zasadzie „jeśli mój sąsiad znalazł ścieżkę, to on mi powie jaka ona jest, a ja dopiszę do niej dojście to tego sąsiada”.

Inaczej mówiąc, aby znaleźć ścieżkę pomiędzy wierzchołkiem A, a wierzchołkiem B, szukano ścieżki dla wszystkich wierzchołków, z którymi A się łączy. Dla nich uruchamiano to samo, czyli dla połączonych z nimi szukano ścieżki do wierzchołka B. I tak dalej, i tak dalej, aż do przejścia całego grafu i znalezienia wszystkich list krawędzi.

b) Obliczenie kosztu

To już jest sprawa trywialnia – dla znalezionych kolekcji ścieżek obliczano ich koszt i szukano z nich najmniejszego poprzez porównanie z aktualnie obliczonym najmniejszym kosztem.

4. Szczegóły implementacji

Projekt został podzielony na 2 klasy: Main.java i Path.java. Pierwsza z nich sprawdza podane argumenty wywołania i uruchamia drugą, której zadaniem jest pobranie z bazy listy krawędzi i uruchomienie algorytmu opisanego wcześniej.

W celu ułatwienia implementacji, stworzono pomocniczą klasę Edge jako klasę wewnętrzną w klasie Path, mającą reprezentować daną krawędź – posiada ona pola takie jak numery wierzchołków i swoją przepustowość.