Politechnika Krakowska

Wydział Fizyki, Matematyki i Informatyki

ZADANIE 1

Zaawansowane techniki programowania

Prowadzący: dr inż. Jerzy Jaworowski

II stopień - rok 1, semestr 1

wykonanie:

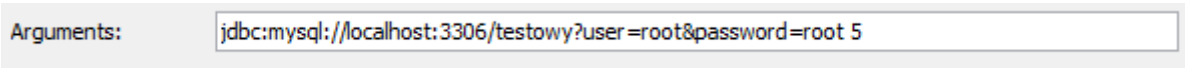
Piotr Adam Tomaszewski

Nr albumu: 104896

Wstęp

Celem zadania było znalezienie ścieżki o najmniejszym koszcie transportu łączącej wierzchołek o numerze 1 z wierzchołkiem określonym przez użytkownika. Dodatkowo należało wyznaczyć koszt takiej ścieżki. Koszt transportu dla każdego węzłów ścieżki został zdefiniowany jako wartość bezwzględna różnicy przepustowości krawędzi ścieżki o końcach w tym wierzchołku. Dodatkowo dla ścieżki koszt transportu zdefiniowano jako suma kosztu transportu wszystkich wierzchołków ścieżki.

Uruchomienie programu wymaga podania dwóch argumentów wejściowych. Pierwszy z nich jest to adres odnoszący się do sterownika bazy danych w celu poprawnego połączenia się z bazą danych w której przechowywane są, kolejno w kolumnach, numery wierzchołka początkowego, końcowego oraz koszt. Drugim argumentem jest numer wierzchołka końcowego do którego należy dojść.



Rysunek 1 Określenie w NetBeans danych wejściowych

Organizacja plików

Main.java

Klasa ta, przy użyciu metody main zawiera w sobie mechanizmy programu głównego. Przede wszystkim stworzenie instancji klasy Path w pliku Path.java. I wywoływanie metod tej klasy w celu rozwiązania problemu.

Metoda ***main*** będąca reprezentacyjną metodą projektu w sobie obiekt klasy Path. Dodatkowo implementowane są zmienne pomocnicze Do zmiennej typu List przypisujemy wynik metody getData, do której wcześniej przesyłamy adres sterownika do połączenia z BD. Po otrzymaniu listy wierzchołków i krawędzi wyliczamy maksymalną wartość wierzchołka oraz ilość krawędzi. Wywołana metoda calculateGraph przygotowuje dane to obliczeń związanych z rozwiązaniem problemu i zwraca ona liczbę w formacie double jako wynik.

Parametry:

- args - tablica wartości, które automatycznie są przekazywane podczas kompilacji programu.

```
public static void main(String[] args) {
      Path path = new Path();
      String driver = args[0];
      int index = Integer.parseInt(args[1]);

    List<Path.Edges> list;
    list = path.getData(driver);

    int v = path.maxVert(list);
    int e = list.size();

    double result = calculateGraph(v,e,index,list,path);

    System.out.format(Locale.ENGLISH, "%.3f", result);

}
```

Rysunek 2 Zawartość metody main

Metoda ***calculateGraph*** przygotowująca dane do przekazania ich do algorytmu rozwiązującego problem znalezienia najkrótszych ścieżek w grafie, w którym wagi krawędzi mogą być ujemne. Tworzona jest macierz VxV, gdzie V to ilość wierzchołków. Dodatkowo tworzona jest zmienna przechowująca najmniejsze wartości wag do każdego z wierzchołków. Dodatkowo w metodzie wypełniana jest macierz danymi z bazy. Wywoływana jest metoda implementująca algorytm Bellmana-Forda a po niej metoda obliczająca koszt transportu do konkretnego wierzchołka.

Parametry:

- v - ilość wierzchołków w grafie
- e - ilość krawędzi w grafie
- index - numer docelowego wierzchołka
- list - lista wierzchołków początkowych i końcowych wraz z wartością krawędzi
- path - instancja klasy Path

Wartość zwracana:

- ostateczny koszt transportu od wierzchołka pierwszego do podanego w parametrze list

```
private static double calculateGraph
    (int v, int e, int index, List<Path.Edges> list, Path path)
    {
        double[][] matrix = new double[v][v];
        double[] distance = new double[v];

        for (Path.Edges k : list)
        {
            matrix[k.getX()-1][k.getY()-1]=k.getP();
        }
        distance=path.bellmanFordAlg(matrix, v);
        return path.getOneEdge(v,distance,matrix,index);
    }
```

Rysunek 3 Zawartość metody calculateGraph

Path.java

Klasa Path implementująca problem zawiera w sobie połączenie z bazą danych, pobranie danych z bazy i operacje polegające na wyznaczeniu grafu i kosztu transportu. Zawarta jest w niej również klasa wewnętrzna Edges pozwalająca przechowywać dane z bazy danych i odpowiednio je wykorzystywać do obliczeń. Tworzone są np. listy typu Edges w celu lepszej prezentacji danych.

Metoda ***getData*** łącząca się z bazą danych za pomocą adresu sterownika przekazanego przez parametr i pozwalająca wykonać działania na bazie danych – głównie w tym przypadku pobranie danych i zapisanie ich do listy.

Parametry:

- connect - przekazanie adresu sterownika JDBC pozwalającego na połączenie się z bazą danych.

Wartość zwracana:

- Lista par wierzchołków i odpowiadające im wagi.

```
protected List<Edges> getData(String connect){
    List<Edges> edge = new ArrayList<Edges>();
    try {
        Connection conn = DriverManager.getConnection(connect);
        Statement statement = conn.createStatement();
        edge = dataList(statement.executeQuery("SELECT * FROM GData"));
        statement.close();
        conn.close();
    }
    catch (SQLException e) {
    }
    return edge;
}
```

Rysunek 4 Zawartość metody getData

Metoda ***dataList*** dodająca dane pobrane z baz danych do listy typu Edge zawierającej parę wierzchołków i koszt przepływu pomiędzy nimi.

Parametry:

- result - dane uzyskane z bazy danych przy użyciu zapytania SQL

Wartość zwracana:

- Zwraca zestawienie połączenia krawędziami pomiędzy wierzchołkami w grafie

```
private List<Edges> dataList(final ResultSet result) throws SQLException {
    List<Edges> edges = new ArrayList<Edges>();

    while (result.next()) {
        Edges point = new Edges(
            result.getInt("x"), result.getInt("y"), result.getDouble("p")
        );
        edges.add(point);
    }

    return edges;
}
```

Rysunek 5 Zawartość metody dataList

Metoda ***maxVert*** pozwalająca wyznaczyć maksymalną wartość wierzchołka w grafie

Parametry:

- edges - lista wierzchołków grafu

Wartość zwracana:

- Maksymalny numer wierzchołka grafu

```
protected int maxVert(List<Edges> edges){
    int x=0;
    int y=0;
    int maxX = Integer.MIN_VALUE;
    int maxY = Integer.MIN_VALUE;

    return searchMaxEdge(x,y,maxX,maxY,edges);
}
```

Rysunek 6 Zawartość metody maxVert

W metodzie ***searchMaxEdge*** pobieramy kolejne wartości wierzchołka początkowego i końcowego i przesyłamy je do kolejnej funkcji w celu znalezienia maksimum.

Parametry:

- x - wartość wierzchołka początkowego
- y - wartość wierzchołka końcowego
- maxX - wartość minimalna Integer
- maxY - wartość minimalna Integer
- edges - lista wszystkich wierzchołków początkowych i końcowych

Wartość zwracana:

- Maksymalny numer wierzchołka grafu

```
private int searchMaxEdge(
        int x, int y, int maxX, int maxY, List<Edges> edges
){
    int max = 0;
    for (Edges e : edges){
        x = e.getX();
        y = e.getY();
        max=searchMax(x,y,maxX,maxY);
    }
    return max;
}
```

Rysunek 7 Zawartość metody searchMaxEdge

Metoda ***searchMax*** wyznaczająca poprzez instrukcje warunkowe maksymalną wartość wierzchołka

Parametry:

- x - wartość wierzchołka początkowego
- y - wartość wierzchołka końcowego
- maxX - wartość minimalna Integer
- maxY - wartość minimalna Integer

Wartość zwracana:

- Maksymalny numer wierzchołka grafu

```
private int searchMax(int x, int y, int maxX, int maxY){
        if (maxX<=x) maxX=x;
        if (maxY<=y) maxY=y;
    return Integer.max(maxX, maxY);
}
```

Rysunek 8 Zawartość metody searchMax

Metoda ***bellmanFordAlg*** implementująca główny algorytm Bellmana-Forda do znalezienia ścieżki o najmniejszej wadze pomiędzy dwoma wierzchołkami w grafie ważonym. Na samym początku deklarujemy zmienną tablicową distance, która będzie przechowywać najmniejsze wagi do konkretnych wierzchołków. Następnie algorytm inicjuje odległości do wszystkich pozostałych wierzchołków jak INFINITE. Zaś korzeń ustawia jako wartość 0. W kolejnym kroku przechodzimy po kolejnych elementach macierzy i sprawdzamy sumę wag, czy jest mniejsza od już istniejącej. Dokładny opis tego algorytmu został podany w kolejnym dziale dokumentacji.

Parametry:

- matrix - macierz zawierająca zestawienie wag dla konkretnych wierzchołków, które posiadają krawędź.
- v - ilość wierzchołków w grafie.

Wartość zwracana:

- Tablica najmniejszych wag do każdego wierzchołka w grafie.

```
protected double[] bellmanFordAlg(double[][] matrix, int v)
{
    int i;
    int j;
    double[] distance = new double[v];
    for (i=0; i<v; ++i)
        distance[i] = Integer.MAX_VALUE;
    distance[0] = 0;

    return bellmanFordRoad(matrix,v,distance,0,0);
}
```

Rysunek 9 Zawartość metody bellmanFordAlg

Metoda ***bellmanFordRoad*** wyznaczająca najmniejszą drogę od wierzchołka początkowego do każdego następnego

Parametry:

- matrix - macierz zawierająca zestawienie wag dla konkretnych wierzchołków, które posiadają krawędź
- v - ilość wierzchołków w grafie
- distance - tablica zawierająca najmniejszą sumę wag od wierzchołka początkowego do każdego następnego.
- i - zmienna pomocnicza wykorzystywana jako iterator
- j - zmienna pomocnicza wykorzystywana jako iterator

Wartość zwracana:

- zaktualizowana tablica zawierająca najmniejszą sumę wag od wierzchołka początkowego do każdego następnego.

```
private double[] bellmanFordRoad(
        double[][] matrix, int v, double [] distance, int i, int j
){
    for (i=0; i<v; ++i){
        for (j=0; j< v; ++j){
            if (matrix[i][j]!=0.0){
                int u = i;
                int vv = j;
                double weight = matrix[i][j];
                if (distance[u]!=Integer.MAX_VALUE
                        && distance[u]+weight<distance[vv]){
                    distance[vv]=distance[u]+weight;
                }
            }
        }
    }
    return distance;
}
```

Rysunek 10 Zawartość metody bellmanFordRoad

Metoda ***stackEdges*** polegająca na wypełnienie stosu wierzchołkami będącymi na drodze od wierzchołka początkowego do końcowego określonego grafu. Metoda wywołuje metodę poprzednicy, która zwraca listę poprzednich wierzchołków grafu w zależności od wierzchołka docelowego. Następnie konstruowana jest droga od wierzchołka docelowego do startowego, która dodawana jest do stosu.

Parametry:

- poprzedniki - lista zawierająca zestawienie wierzchołków od początkowego do docelowego
- distance - tablica zawierająca najmniejsze rozmiary dróg do wierzchołków
- v - ilość wierzchołków grafu
- u - numer poprzedniego wierzchołka na najmniejszej drodze od początkowego do końcowego wierzchołka
- matrix - macierz zawierająca zestawienie wag dla konkretnych wierzchołków, które posiadają krawędź

Wartość zwracana:

- kolejna wartość wierzchołka na najmniejszej drodze od początkowego do końcowego wierzchołka.

```
private int stackEdges(
        List<Integer> poprzedniki,double[] distance,
        int v, int u, double[][] matrix
){
    int size = poprzedniki.size();
    int i;
    for (i=0;i<size;i++)
        if (distance[v]==distance[poprzedniki.get(i)]
                +matrix[poprzedniki.get(i)][v])
            u=poprzedniki.get(i);
    return u;
}
```

Rysunek 11 Zawartość metody stackEdges

Metoda ***previous*** tworząca listę poprzednich wierzchołków grafu zależności od wierzchołka docelowego.

Parametry:

- x - wierzchołek dla którego aktualnie są szukani poprzednicy w macierzy
- v - ilość wierzchołków w grafie
- matrix - macierz zawierająca zestawienie wag dla konkretnych wierzchołków, które posiadają krawędź

Wartość zwracana:

- Lista poprzedników wierzchołka

```
private static List<Integer> previous(int x, int v, double[][] matrix) {
    List<Integer> result = new ArrayList<>();
    for (int i=0;i<v;i++)
        if (matrix[i][x]!=0.0)
            result.add(i);

    return result;
}
```

Rysunek 12 Zawartość metody previous

Metoda ***getOneEdge*** zwracająca wartość kosztu transportu dla każdego z węzłów ścieżki od wierzchołka początkowego do wierzchołka podanego przez użytkownika. Tworzony jest stos dla każdego wierzchołka grafu – przechowywane w nim będą pośrednie wierzchołki grafu. Za pomocą metody stackEdges wypełniamy stos wierzchołkami. W następnym kroku obliczany jest koszt transportu między wierzchołkiem początkowym a wierzchołkami końcowymi.

Parametry:

- vv - ilość wierzchołków w grafie
- distance - tablica zawierająca najmniejszą sumę wag od wierzchołka początkowego do każdego następnego.
- matrix - macierz zawierająca zestawienie wag dla konkretnych wierzchołków, które posiadają krawędź.
- index - numer docelowego wierzchołka grafu

Wartość zwracana:

- koszt transportu między wierzchołkiem początkowym a wierzchołkiem końcowym, który został podany przez użytkownika.

```
protected double getOneEdge(
        int vv, double[] distance, double[][] matrix, int index
){
     List<List<Integer>> stack = new ArrayList<List<Integer>>(vv);
     for (int i = 0; i < vv; i++) {
          stack.add(i,new ArrayList<Integer>());
     }

    stack=stackEdgesPrepare(vv,distance,matrix,stack,0,0);


   return getOneEdge2(stack.get(index-1).get(0), 0, 0, 0, 0, index-1, 0,
           stack, matrix, 0, stack.get(index-1).size());
}
```

Rysunek 13 Zawartość metody getOneEdge

Metoda ***stackEdgesPrepare*** polegająca na wypełnienie stosu wierzchołkami będącymi na drodze od wierzchołka początkowego do końcowego określonego grafu. Metoda wywołuje metodę poprzednicy, która zwraca listę poprzednich wierzchołków grafu w zależności od wierzchołka docelowego. Następnie konstruowana jest droga od wierzchołka docelowego do startowego która dodawana jest do stosu.

Parametry:

- vv - ilość wierzchołków w grafie
- distance - tablica zawierająca najmniejszą sumę wag od wierzchołka początkowego do każdego następnego.
- matrix - macierz zawierająca zestawienie wag dla konkretnych wierzchołków, które posiadają krawędź
- stack - pusty stos przechowujący kolejne numery wierzchołków od wierzchołka docelowego do początkowego.
- v - ilość wierzchołków grafu

- u - numer poprzedniego wierzchołka na najmniejszej drodze od początkowego do końcowego wierzchołka

Wartość zwracana:

Wypełniony stos kolejnymi wierzchołkami grafu.

```
private List<List<Integer>> stackEdgesPrepare(
        int vv, double[] distance, double[][] matrix,
        List<List<Integer>> stack, int v,int u
){
    for (int x=0;x<vv;x++){
        stack.get(x).add(x);
        v=x;
        while (v!=0){
            u=stackEdges(previous(v,vv,matrix),distance,v,u,matrix);
            stack.get(x).add(u);
            v=u;
        }
    }
    return stack;
}
```

Rysunek 14 Zawartość metody stackEdgesPrepare

Metoda ***getOneEdge2*** rozszerzająca funkcjonalność metody getOneEdge

Parametry:

- px - zmienna przechowująca wartość wierzchołka x
- py - zmienna przechowująca wartość wierzchołka y
- pxy - zmienna przechowująca wartość krawędzi między x i y
- pyz - zmienna przechowująca wartość krawędzi między y i z
- b - zmienna będąca iteratorem w pętli
- index - numer wierzchołka docelowego
- k - wartość sumy odwrotności kolejnych wartości wag krawędzi
- stack - stos przechowujący kolejne drogi do wierzchołka docelowego
- matrix - macierz zawierająca zestawienie wag dla konkretnych wierzchołków, które posiadają krawędź
- abss - zmienna przechowująca wynik wartości bezwzględnej różnicy krawędzi.
- size - rozmiar stosu

Wartość zwracana:

- Suma wartości bezwzględnej z odwrotnościami kolejnych wartości wag krawędzi

```
private double getOneEdge2(int px, int py, double pxy, double pyz, int b,
        int index, double k, List<List<Integer>> stack, double[][] matrix,
        double abss,  int size
        ){

 for (b=1;b<size;b++){
     py=stack.get(index).get(b);
     pyz=matrix[py][px];

     k=getOneEdge3(k,pyz);

     if (pxy!=0){
         abss+=Math.abs(pxy-pyz);
     }
     pxy=pyz;
     px=py;
 }
 return  abss+k;
}
```

Rysunek 15 Zawartość metody getOneEdge2

Metoda ***getOneEdge3*** wyznaczająca z podanych danych wartość k, określającą oszt transportu definiowany dla każdego z węzłów ścieżki.

Parametry:

- k - wartość sumy odwrotności kolejnych wartości wag krawędzi
- pyz - waga krawędzi między drugim a trzecim wierzchołkiem

Wartość zwracana:

- odwrócona wartość wagi wierzchołka zsumowana z wartością początkową k.

```
private double getOneEdge3(double k, double pyz){
    if (pyz!=0)
        k+=1/(pyz);
    else
        k=0.0;
    return k;
}
```

Rysunek 16 Zawartość metody getOneEdge3

Ostatni element to klasa wewnętrzna ***Edges***, zawierająca dane dotyczące początku i końca krawędzi wraz z wartością określającą przepustowość krawędzi.

```
protected class Edges {
    private int x; //wierzcholek poczatkowy
    private int y; //wierzcholek koncowy
    private double p; //przepustowosc (waga) krawedzi

    public Edges(){}

    public Edges(int x, int y, double p) {
        this.x = x;
        this.y = y;
        this.p = p;
    }
```

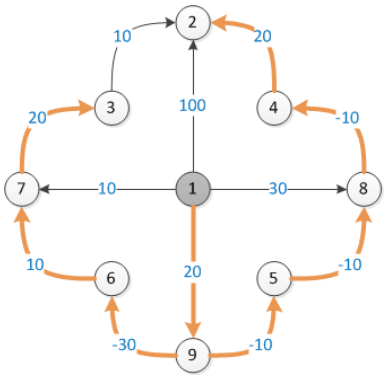
Rysunek 17 Zawartość klasy Edge

Klasa ta poza standardową implementacją pól i konstruktorów zawiera w sobie również tak zwane gettery i setery.

Opis algorytmu

Algorytm Bellmana-Forda pozwala na wyznaczenie najmniejszej odległości od ustalonego wierzchołka początkowego do wszystkich pozostałych w grafie skierowanym posiadającym ujemne długości krawędzi. Często grafy tego typu mogą reprezentować pewne połączenie, np. między miejscowościami, gdzie wagi oznaczają np. dystans , przy czym dystans nie jest ujemny. Jeśli jesteśmy pewni że graf nie posiada wartości ujemnych, to można zastosować algorytm Dijkstry. W omawianym problemie zaimplementowano algorytm, mogący wykorzystywać ujemne wagi.

Na przykład, dla grafu z powyższego rysunku i źródła w wierzchołku 1, najkrótsze ścieżki można przedstawić w następujący sposób:



Rysunek 18 Najkrótsze drogi grafu z ujemnymi wagami

Niech $D[u]$ będzie długością najkrótszej ścieżki z wierzchołka początkowego do u. Na samym początku należy wyznaczyć wartości $D[u]$, gdzie $u \in V$. W algorytmie tym głównym założeniem jest fakt, że dla każdej krawędzi $u \to v$ zachodzi nierówność $D[u] + f(u \to u) \geq D[v]$. Można to zrozumieć tak, że najkrótsza droga do wierzchołka v nie może być dłuższa niż przejście najkrótszą drogą do wierzchołka u, a następnie krawędzią $u \to v$.

Pseudokod algorytmu:

1. Dla każdego wierzchołka $u \in V$ przypisz $D[u] = \infty$
2. $D[s] = 0$
3. Dopóki istnieje krawędź $u \to v$, dla której $D[u] + f(u \to v) < D[v]$, wykonaj
    1. $D[v] = D[u] + f(u > v)$

Wykonanie jednego kroku 3 sprawia, że co najmniej jeden wierzchołek u ma prawidłowo obliczoną wartość $D[u]$. Kolejne wykonanie tego samego kroku sprawi że co najmniej dwa wierchołki będą mieć obliczoną długość najkrótszej ścieżki od wierzchołka początkowego s. Dlatego jeśli w grafie mamy określoną ilość wierzchołków (np. n), krok 3 będzie wykonywał się $n - 1$ razy.

Złożoność obliczeniowa takiego algorytmu to $O(n \cdot m)$, gdzie n i m to odpowiednio liczba wierzchołków i krawędzi w grafie.

Dodatkowo w ramach rozwiązywania problemu zadania wykorzystywano algorytm wyznaczania najkrótszej drogi w grafie między wierzchołkiem początkowym a docelowym podanym przez użytkownika dla znanej odległości, którą wyznaczono przy użyciu algorytmu Belmana-Forda.

Dla grafu opisanego za pomocą macierzy wag wywołano algorytm wyznaczający najkrótszą ścieżkę od wierzchołka początkowego do pozostałych i w wyniku otrzymano tablicę odległości. Należy odczytać najmniejszą odległość między wierzchołkiem początkowym a docelowym.

Do pomocy w wyznaczaniu drogi stosujemy stos na którym otrzymano ciąg wierzchołków będących drogą między wierzchołkiem początkowym a docelowym.

```
Zainicjuj stos S, odłóż na stos wierzchołek końcowy,
podstaw v=ut
while v<>us do
    znajdź wierzchołek u taki, że D(v)=D(u)+A[u,v]
    odłóż u na stosie S
    podstaw v=u
```