Praca domowa 3 – highway

Przemysław Kleszcz

Zadanie

Cel pracy domowej: Określić czy planowana sieć połączeń jest możliwa do realizacji na podstawie danych z bazy danych.

Organizacja strukturalna

Projekt zawiera jeden plik: **Highway.java**

Highway.java:

public class Highway

protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
protected void calculate(HttpServletRequest request,HttpServletResponse response)
private Graph<Integer> loadData(String conn)

class PlanarityTest

private static Set<String> ctreateNis()
boolean isPlanar(Graph<Integer> graph)
private static <T> boolean isPlanar(Graph<T> graph, Graph<T> cycle)

class Graph

private void addVertex(T v)
void addEdge(T v1, T v2)
void deleteEdge(T v1, T v2)
int countVertices()
int countEdges()
Set<T> getBorderers(T v)
private int getDegree(T v)
Set<T> getVertices()
boolean hasVertex(T v)
boolean hasEdge(T v1, T v2)
static <T> boolean isPath(Graph<T> graph)
static <T> boolean isBipartite(Graph<T> graph)
static <T> Set<Graph<T>> divideToPieces(Graph<T> graph, Graph<T> cycle)
static <T> Graph<T> addGraphs(Graph<T> g1, Graph<T> g2)

class GraphProceeder

boolean isBipartite()
private boolean isBipartite(T v, boolean color)
T goThroughCycle()
Graph<T> getPath(T start, T end, Collection<T> banned)
private boolean getPath(T v)
Graph<T> findCycle()

```
private Graph<T> findCycle(T v)
Set<Graph<T>> divideToPieces(Graph<T> cycle)
private void makePiece(Graph<T> cycle, T v)
```

Highway - Klasa servletu realizująca podstawową funkcjonalność założoną w zadaniu.

PlanarityTest - Klasa zawierającą implementację algorytmu sprawdzania planarności, wykorzystanej w celu sprawdzenia możliwości realizacji połączeń.

Graph - Klasa struktury grafu, zawierająca wszystkie operacje manipulacji na grafie wykorzystane do realizacji zadania takie jak dodawanie, usuwanie krawędzi czy zliczanie wierzchołków.

GraphProceeder - Klasa zawierająca różne metody pozwalające na przechodzenie po grafie. Pomocna przy rekursywnym badaniu grafu.

Opis mechanizmu

Program jest zrealizowany w konwencji servletu. Zadaniem metod przetwarzania żądań jest odczyt parametru conn zawierającego connection string do połączenia z bazą danych. Jeśli connection string nie jest uzupełniony dalsza funkcjonalność jest pomijana. W dalszym etapie odbywa się połączenie z bazą danych za pomocą jdbc. Wykonywane jest zapytanie do bazy danych:

```
select i, j, id from hdata
```

Wyniki zapytania są przetwarzanie sekwencyjnie w pętli. Wartości pola i oraz j z bazy danych są dodawane do struktury grafu jako nowe krawędzie. Jednocześnie szukana jest wartość maksymalna n z i oraz j w obrębie całej tabeli. Do struktury grafu dodawane są istniejące połączenia drogi w przedziale od 1 do wartości maksymalnej n. Na koniec do struktury dodawane jest sztuczne połączenie pomiędzy wartością 1 oraz wartością maksymalną n. Dla tak przygotowanego grafu uruchamiana jest funkcjonalność sprawdzania planarności.

Funkcjonalność sprawdzania planarności w pierwszej kolejności wyodrębnia cykl grafu. Następnie sprawdza czy przekazany graf ma więcej niż 3n-6 krawędzi. Jeśli tak, od raz wiemy, że graf nie jest planarny. Jeśli poprzedni warunek nie zachodzi wyznaczane są części grafu, uwzględniając wcześniej wyodrębniony cykl. Dla każdej wyodrębnionej części sprawdzane jest czy to graf ścieżki. Jeśli nie, dodajemy część do cyklu. W uzyskanym cyklu zamieniamy odcinek między dwoma kolejnymi przyłączeniami z częścią. Powyższą funkcjonalność należy wykonywać rekursywnie aż do uzyskania wartości false. Wtedy graf jest nieplanarny. Jeśli wartość false nie zostanie uzyskana, z wyodrębnionych części konstruowany jest graf przecięć. Jeśli nowouzyskany graf nie jest dwudzielny, przekazany funkcjonalności na początku graf jest nieplanarny i zwracana jest wartość 0 w odpowiedzi Servletu. W przeciwnym wypadku jest planarny i zwracana jest wartość 1.