Praca domowa 07 – ejb-graph

Szlachetka Miłosz
Nr albumu. 114014

1.Cel zadania

Na podstawie danych z pliku, którego ścieżka była parametrem programu, należało zbudować graf, a następnie wyznaczyć ilość cykli w nim występujących.

2.Struktura programu

Program składa się z pięciu plików: **IGraphRemote.java, Graph.java, ICycleRemote.java, Cycle.java, AppClient.java**

IGraphRemote.java zawiera:
@Remote
public interface IGraphRemote - interfejs typu Remote, który jest implementowany przez komponent Graph.Interfejs zawiera deklaracje metod: **public void addEdge(int u, int v), public List<Integer> getVertexAdjacenyList(int vertex), public void initGraph(), public Map<Integer, List<Integer>> getEdges().** Metody te zostały opisane poniżej.

Graph.java zawiera:
@Stateless
public class Graph implements IGraphRemote - klasa, będąca bezstanowym komponentem EJB. Umożliwia tworzenie i przechowywanie struktury grafu, a także dostęp do jego elementów.
private Map<Integer, List<Integer>> edges - pole przechowujące mapę krawędzi. Kluczem mapy jest wierzchołek początkowy krawędzi. Wartością dla danego klucz(wierzchołka) jest lista wierzchołków, które są końcami krawędzi wychodzących z wierzchołka będącego kluczem mapy.
@Override
public void addEdge(int u, int v) - dodaje krawędź do listy krawędzi. Parametr u jest początkiem krawędzi, natomiast v to jej koniec.
@Override
public void initGraph(int listSize) - metoda odpowiedzialna za stworzenie (inicjalizację) listy (**edges**), która będzie przechowywać krawędzie grafu.
@Override
public List<Integer> getVertexAdjacencyList(int vertex) - zwraca listę wierzchołków, które są połączone z wierzchołkiem o numerze zadanym parametrem “vertex”. Wierzchołki te są końcami krawędzi o początku w wierzchołku “vertex”.
@Override
public Map<Integer, List<Integer>> getEdges() - zwraca mapę krawędzi.

ICycleRemote.java zawiera:
@Remote
public interface ICycleRemote - interfejs typu Remote, który jest implementowany przez komponent Cycle. Interfejs zawiera deklaracje metody **public int getNumberOfCycles(IGraphRemote g),** która została opisana w ramach klasy Cycle.

Cycle.java zawiera:
@Stateless
public class Cycle implements ICycleRemote - klasa będąca bezstanowym komponentem EJB. Zawiera implementacje metody wyznaczającej ilość cykli dla zadanego grafu, a także zawiera implementacje prywatnych metod pomocniczych.
private boolean checkIfCyclesAreEqual(List<Integer> cycle1, List<Integer> cycle2) - sprawdza czy "cycle1" jest tym samym cyklem co "cycle2". Cykle są podawane w postaci list wierzchołków. Metoda sprawdza czy cykle zawierają tą samą liczbę wierzchołków. Następnie sprawdza czy wierzchołki w listach są w tej samej kolejności. Cykl jest wykrywany nawet wtedy, gdy jego początek zaczyna się w innym wierzchołku, natomiast kolejność wierzchołków po nim jest taka sama. Np. 1->2->3->1 będzie tym samym cyklem co: 2->3->1->2.
private int findCommonVertexIndex(List<Integer> list1, List<Integer> list2) - znajduje indeks w "list2", pod którym znajduje się wierzchołek będący na pierwszym miejscu w liście "list1". Jeśli wierzchołek nie zostanie znaleziony zwraca -1.
private boolean checkIfCycleIsInList(List<List<Integer>> cyclesList, List<Integer> cycle) - sprawdza czy cykl(w postaci listy wierzchołków) znajduje się już w liście "cyclesList". Metoda ta wykorzystuje metodę **checkIfCyclesAreEqual.**
private boolean findCycleDFS(IGraphRemote g, int v, int w, List<Integer> cycle, Map<Integer, Boolean> visitedVertices, List<List<Integer>> discoveredCycles) - znajduje cykl w grafie, którego początek znajduje się w wierzchołku określonym parametrem "v". Metoda ta jest metodą rekurencyjną. Do wyszukiwania cyklu został wykorzystany algorytm DFS, którego wyszukiwanie jest przerywane w momencie gdy aktualny wierzchołek jest wierzchołkiem, od którego zaczęto wyszukiwanie. Metoda w momencie gdy znajdzie cykl sprawdza czy nie został on już wcześniej znaleziony, jeśli nie to zwraca true. Jeśli nowy, unikalny cykl nie został znaleziony zwraca false.
@Override
public int getNumberOfCycles(IGraphRemote g) - zwraca liczbę cykli w grafie "g". Wyszukiwanie cyklu polega na sprawdzeniu wywołaniu dla każdego wierzchołka grafu "g" metody **findCycleDFS** opisanej powyżej. Jeśli metoda zwróci true, wtedy znaleziony cykl zostanie dodany do listy unikalnych cykli. Na końcu zwracany jest rozmiar listy unikalnych cykli.

AppClient.java zawiera:
@EJB
private static IGraphRemote graph - komponent EJB reprezentujący graf.
@EJB
private static ICycleRemote cycle - komponent EJB, który służy do wyznaczania liczby cykli w grafie.

private static List<Integer> getVerticesFromString(String str) - z zadanego napisu wyłuskuje za pomocą wyrażenia regularnego liczby reprezentujące numery wierzchołków.
private static List<Integer> getVerticesFromFile(String fileName) - metoda, która czyta plik (określony ścieżką “fileName”) linia po linii i pobiera z niego liczby naturalne reprezentujące wierzchołki. Zapisuje wierzchołki do listy wierzchołków zachowując ich kolejność. Wywołuje metodę getVerticesFromString dla każdej pobranej z pliku linii.
private static void buildGraph(List<Integer> verticesList) - inicjalizuje graf oraz dodaje do niego krawędzie. Krawędzie są dodawane do grafu w taki sposób, że z listy “verticesList” parami pobierane są wierzchołki, które tworzą krawędź i są dodawane do grafu. Jeśli liczba wierzchołków w liście jest nieparzysta to ostatni wierzchołek jest pomijany
public static void main(String[] args) - w metodzie tej pobierana jest nazwa pliku z linii komend, a następnie odczytywany jest plik, na podstawie którego budowany jest graf. Później wykonywany jest algorytm wyznaczający liczbę cykli w grafie, a otrzymana wartość jest wypisywana.

3.Struktura danych grafu wraz z metodami dostępowymi

Struktura danych przechowująca informacje o krawędziach grafu to mapa (Map<Integer, List<Integer>>), której indeksami są numery wierzchołków, które są początkami krawędzi. Kluczami mapy są natomiast listy wierzchołków(List<Integer>). W ten sposób dla danego wierzchołka (indeksu mapy) przetrzymywana jest lista wierzchołków z nim połączonych w taki sposób, że indeks mapy jest początkiem, a elementy listy dla danego indeksu mapy to końce krawędzi. Dane przechowywane w taki sposób tworzą niejako mapę sąsiedztwa.

Metody dostępowe :
public void initGraph() - służąca do inicjalizacji struktury danych, która będzie przechowywała krawędzie grafu. Po inicjalizacji struktura nie zawiera jeszcze żadnych krawędzi.
public void addEdge(int u, int v) - metoda dodająca krawędź (o początku w “u” i końcu w “v”) do grafu.
public List<Integer> getVertexAdjacencyList(int vertex) - metoda służąca do pobrania listy wierzchołków, które znajdują się na końcach krawędzi wychodzących z wierzchołka o numerze “vertex”.
public Map<Integer, List<Integer>> getEdges() - zwraca mapę krawędzi.
Metody te zostały opisane bardziej dokładnie w rozdziale 2.

4. Algorytm

Przed częścią programu odpowiedzialną za wyszukiwanie cykli w grafie następuje pobranie ścieżki do pliku z argumentu programu. W przypadku gdy plik nie istnieje program jest przerywany. Jeśli plik istnieje to jest czytany linia po linii. Z każdej linii wyłuskiwane są liczby naturalne dodatnie (za pomocą wyrażenia regularnego) następnie liczby te (wierzchołki) są umieszczane w liście. Jeśli lista posiada nieparzystą ilość elementów to jest skracana o 1. Później z listy parami pobierane są wierzchołki i dodawane do grafu. Następnie wykonywany jest algorytm szukający cykli w grafie.

Szukanie cykli w grafie:
Tworzone są pomocnicze struktury: Mapa odwiedzonych wierzchołków, lista przechowująca cykle, które zostały znalezione w grafie oraz lista przechowująca aktualną ścieżkę w grafie (lista wierzchołków, która może być cyklem).
Dla każdego wierzchołka grafu wykonywany jest rekurencyjny algorytm DFS, który przeszukuje graf w głąb. Począwszy od wierzchołka startowego odwiedzane są kolejne wierzchołki. Za każdym razem informacja o odwiedzinach jest zapisywana do mapy odwiedzonych wierzchołków. Dodatkowo do listy z aktualną ścieżką (lista wierzchołków) dodawany jest wierzchołek w którym się aktualnie znajdujemy. Dla aktualnie odwiedzonego wierzchołka sprawdzani są wszyscy sąsiedzi (wierzchołki, które są z nim połączone krawędzią). Jeśli któryś sąsiad jest równy pierwszemu elementowi na liście odwiedzonych wierzchołków to znaczy, że znaleziono cykl. Następnie sprawdzane jest czy dany cykl nie został już znaleziony poprzednio (np. z innego wierzchołka). Jeśli nie to cykl dodawany jest do listy znalezionych cykli, jeśli jednak cykl został znaleziony wcześniej to wyszukiwanie trwa dalej do momentu przejścia wszystkich osiągalnych wierzchołków. Następnie czyszczone są tablica odwiedzonych wierzchołków oraz lista zawierająca aktualną ścieżkę i ponownie wykonywany jest algorytm DFS dla kolejnego wierzchołka w grafie. Na końcu zwracany jest rozmiar listy zawierającej znalezione cykle w grafie.