Bartłomiej Barszczak

WEAIiIB Automatyka i Robotyka

Rok II semestr IV grupa 3

# Zadanie 1

|  |  |  |  |
| --- | --- | --- | --- |
| **Macierz sąsiedztwa** | | **Lista sąsiedztwa** | |
| Zalety: | Wady: | Zalety: | Wady: |
| * Łatwy i szybki dostęp do sprawdzenia czy dana krawędź istnieje. * Dodanie nowej krawędzi jest stosunkowo szybkie. | * Złożoność pamięciowa: O(n2). * Iteracja po wszystkich krawędziach jest stosunkowo wolna. * Dodanie lub usuniecie wierzchołka jest stosunkowo wolne. | * Złożoność pamięciowa zależy od rozmiaru grafu, z reguły O(W + K) * Iterowanie po wszystkich krawędziach jest stosunkowo szybkie. * Dodanie nowego wierzchołka i krawędzi jest stosunkowo szybkie | * Sprawdzenie czy dana krawędź istnieje jest nieznacznie wolniejsze niż w reprezentacji macierzy sąsiedztwa. |

# Zadanie 2

/\*  
 \* Funkcja przeszukuje graf za pomoca algorytmu bfs. Zwraca liste odwiedzonych wierzcholkow w kolejnosci z jaka  
 \* przeszukuje graf algorytm bfs oraz dwa slowa w postaci mapy ktore informuja czy graf jest spojny i czy posiada cykl  
 \*/  
std::map<std::vector<int>, std::map<std::string, std::string>> bfs(const std::map<int, std::vector<int>> &graph, int start) {  
 // inicjalizacja zmiennych  
 std::queue<int> queue = {};  
 std::map<int, int> visited = {};  
 std::vector<int> labels = **{**start + 1**}**;  
 int number = 1;  
 int v = start;  
 bool is\_consistent = false;  
 bool has\_cycle = false;  
  
 // wyzerowanie zmiennej visited, przypisanie kazdemu wierzcholkowi etykiety 0  
 for (const auto &elem: graph)  
 visited[elem.first] = 0;  
  
 // ustawienie poczatkowemu wierzcholkowi etykiety 1  
 visited[start] = number;  
  
 // dodanie do kolejki wszystkich sasiadow poczatkowego wierzcholka  
 for (auto value: graph.find(v)->second) {  
 queue.push(value);  
 }  
  
 // glowna petla realizujaca algorytm bfs  
 while (!queue.empty()) {  
 v = queue.front();  
 queue.pop();  
  
 if (visited[v] == 0) {  
 visited[v] = ++number;  
 labels.push\_back(v + 1);  
 }  
 for (auto elem: graph.find(v)->second) {  
 if (visited[elem] == 0) {  
 queue.push(elem);  
  
 }  
 if (visited[elem] == visited[v]) // warunek na sprawdzenie czy graf jest cykliczny  
 has\_cycle = true;  
 }  
 }  
  
 if (labels.size() == graph.size()) // Warunek na sprawdzenie czy graf jest spojny  
 is\_consistent = true;  
  
 // zwracanie wynikow  
 return **{**{labels, {{is\_consistent ? "TRUE" : "FALSE", has\_cycle ? "TRUE" : "FALSE"}}}**}**;  
}

# Zadanie 3

// deklaracja zmiennej result ktora przechowuje wynik zwrocony przez funkcje bfs  
std::map<std::vector<int>, std::map<std::string, std::string>> result;  
  
// graf spojny acykliczny  
std::map<int, std::vector<int>> g1 = **{** {0, {1, 2, 3, 6}},  
 {1, {2, 4, 7}},  
 {2, {3, 6, 7}},  
 {3, {5, 6, 9}},  
 {4, {7, 8}},  
 {5, {6, 9}},  
 {6, {9}},  
 {7, {6, 9}},  
 {8, {7}},  
 {9, {}}  
**}**;  
  
// graf spojny z cyklami  
std::map<int, std::vector<int>> g2 = **{** {0, {1, 4, 5}},  
 {1, {2, 5}},  
 {2, {2, 3, 8}},  
 {3, {4, 7, 9}},  
 {4, {5, 6}},  
 {5, {6}},  
 {6, {0, 7}},  
 {7, {8}},  
 {8, {3, 9}},  
 {9, {0, 4}}  
**}**;  
  
// graf niespojny z cyklami  
std::map<int, std::vector<int>> g3 = **{** {0, {1, 2}},  
 {1, {0, 1, 2, 4}},  
 {2, {0, 3}},  
 {3, {2, 4}},  
 {4, {1}},  
 {5, {6, 7, 9}},  
 {6, {5, 7}},  
 {7, {5, 6}},  
 {8, {7, 9}},  
 {9, {5, 6, 8}}  
**}**;



# Zadanie 4

**Wierzchołek rozpajający grafu** można znaleźć, gdy jest korzeniem i ma przynajmniej dwóch synów lub gdy nie jest korzeniem a dla przynajmniej jednego syna spełniony jest warunek: . Przed tym jednak należy wykonać algorytm DFS i określić czasy odwiedzenia danych wierzchołków jako właśnie funkcje . (Wikipedia)

**Centrum grafu** możemy znaleźć korzystając najpierw z algorytmu Floyd-Warshall’a, który wyznacza najkrótszą drogę pomiędzy dwoma wierzchołkami (jego złożoność obliczeniowa to O(n3)) a następnie wybrać te wierzchołki których najdłuższa droga jest nie większa od długości dróg łączących pozostałe wierzchołki.

Za pomocą BFS możemy odnaleźć wszystkich sąsiadów danego wierzchołka lub sąsiadów 2-rzędu, 3-rzędu itd.