|  |
| --- |
| https://www.mirea.ru/bitrix/templates/unlimtech/images/logo.png |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |

|  |  |
| --- | --- |
| Отчет по выполнению практического задания №5  По дисциплине «Структуры и алгоритмы обработки данных» | |
| **Тема:**  Основные алгоритмы работы с графами | |
|  | |
| Выполнил студент | Сидоров С.Д. |
| группа | ИКБО-20-21 |

Отчёт

1. Постановка задачи:

Выполнить разработку программы управления графом, в соответствии с вариантом, на основе класса Граф. Предусмотреть в качестве данных: количество вершин в графе, структура для хранения графа.

1. Задание варианта:

Вариант 6: Представление графа в памяти - список смежных вершин.

Задачи:

Ввод с клавиатуры графа (применение операции вставки ребра в граф). Вывод всей цепочки в графе, используя метод поиска в ширину.

Составить программу нахождения кратчайшего пути в графе от заданной вершины к другой заданной вершине методом «Дейкстры» и вывести этот путь.

1. Разработка:
2. Структуры представления данных:

**Представление графа:**

сlass Graph {

vector<Node\*>\* vectNodes; - список всех вершин графа.

Int iSize; - количество всех вершин в графе.

}

**Представление вершины:**

class Node {

Int iKey; - ключ вершины

Vector<Node\*>\* vectAdjacentNodes; - список смежных вершин

Vector<Edge\*>\* vectAdjacentEdges; - список смежных рёбер

}

**Представление ребра:**

class Edge{

int iKeyFrom; - ключ исходной вершины

int iKeyTo; - ключ конечной вершины

int iWeight; - вес ребра

}

1. Алгоритм вывода графа на псевдокоде:

|  |
| --- |
| void showGraph() {  если (iSize == 0) Выход;  Узел\* nextNode = vectNodes->at(0);  очередь<Ребро\*>\* qEdges = new очередь<Ребро\*>();  int\* visitedNodes = new int[iSize]; *//Список посещённых элементов*  цикл (от i = 0 до i = iSize) visitedNodes[i] = -1;  Вывод( "\n" , nextNode->getKey());  visitedNodes[nextNode->getKey()] = 1;  vector<Ребро\*>\* baseEdges = nextNode->getEdges();  если (baseEdges->size() == 0) Выход;  цикл (от i = 0 до i = кол-ву смежных вершин) qEdges->push(baseEdges->at(i));  пока (qEdges не пуста) {  Ребро\* nextEdge = qEdges->front(); *//достаем из очереди первый элемент*  qEdges->pop(); *// удаляем из очереди первый элемент*  Узел\* prevNode = getNode(nextEdge->getNodeFrom());  nextNode = getNode(nextEdge->getNodeTo());  baseEdges = nextNode->getEdges();  Вывод( "\n" , prevNode->getKey());  visitedNodes[prevNode->getKey()] = 1;  Вывод ( " - " , nextNode->getKey());  если (visitedNodes[nextNode->getKey()] == -1) {  visitedNodes[nextNode->getKey()] = 1;  цикл (int i = 0; i < baseEdges->size(); i++)  {  если (visitedNodes[baseEdges->at(i)->getNodeTo()] == -1) {  qEdges->push(baseEdges->at(i));  *//Запись смежных рёбер в очередь*  }  }    }  }  } |

1. Алгоритм добавления ребра на псевдокоде:

|  |
| --- |
| void addEdge(int iKeyFrom,int iKeyTo, int iWeight) {  Узел\* nodeFrom = getNode(iKeyFrom);  Узел\* nodeTo = getNode(iKeyTo);  Если(nodeTo не существует или nodeFrom не существует) Возврат;  nodeFrom->addEdge(nodeTo, iWeight);    }  void Node->addEdge(Узел\* tmp, int iWeight) {  vectAdjacentNodes->push\_back(tmp); *//Добавление вершины в список смежных*  vectAdjacentEdges->push\_back(new Ребро(iKey, tmp->getKey(), iWeight));  *//Добавление нового ребра в список смежных рёбер*  } |

1. Алгоритм добавления узла на псевдокоде:

|  |
| --- |
| void addNode(int iKeyNew) {  Узел\* tmp = new Узел(iKeyNew);  vectNodes->push\_back(tmp); *//Запись в список узлов*  iSize++; *//Увелечение кол-ва вершин*  } |

1. Алгоритм поиска кратчайшего пути:

|  |
| --- |
| void wayFinder(список\_вершин\_с\_весами\_и\_предыдущими\_вершинами wayWeighted, Узел\* root, int iKeyResult, список\_посещённых\_вершин alreadyVisited) {  Список\_смежны\_рёбер baseEdges = root->getEdges();  цикл(от i = 0 до i = кол-ва элементов baseEdges - 1) {  если(текущий узел уже посещён) продолжить;  если(узел не был добавлен) {  Добавление узла в wayWeighted с учетом веса ребра и веса родительской вершины.  }  иначе{  если(новый путь короче записанного) {  Обновление узла в wayWeighted с учётом веса ребра и веса родительской вершины.  }}  }  (\*alreadyVisited)[root->getKey()] = 1; *//Отметка о посещении узла*  int minNodeIndex = индекс вершины с минимальным весом;  wayFinder(wayWeighted, getNode(baseEdges->at(minNodeIndex)->getNodeTo()), iKeyResult, alreadyVisited);  } } |

По результатам работы **wayFinder**, получаем **map<int,pair<int,int>>**из которого можно получить длину пути и сам путь:

|  |
| --- |
| int iCounter = iKeyTo;  пока(iCounter != iKeyFrom) {  Вывод(iCounter);  iCounter = (\*wayMap)[iCounter].second;  }  Вывод ("\nWay weighted: " (\*wayMap)[iKeyTo].first ); |

1. Код:

node.h - класс Node (узел) - файл node.pdf

edge.h - класс Edge (ребро) - файл edge.pdf

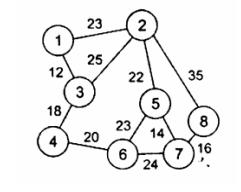
graph.h - класс Graph(граф) - файл graph.pdf

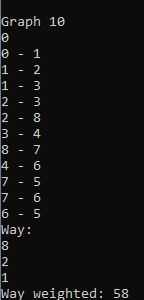
Source.cpp - основная программа - файл main.pdf

1. Тестирование:

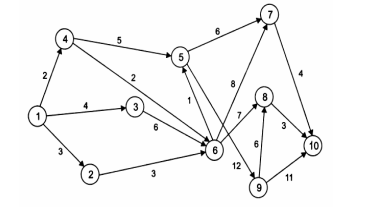
Для тестирования использовались графы представленные в задании:

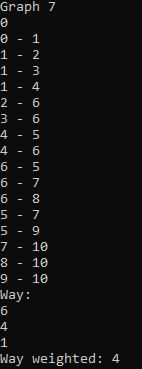
Граф 10:



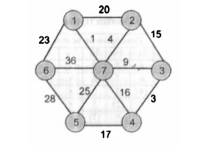


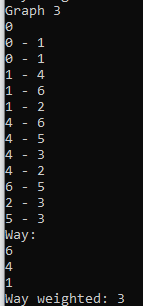
Граф 7:





Граф 3:





Выводы:

В ходе выполнения данной практической работы были получены навыки работы с графами, а также были получены знания об использовании различных алгоритмов для работы над графами.

**Список литературы:**

⦁ Лекции по структурам и алгоритмам обработки данных Рысин М.Л.

⦁ Методическое пособие по выполнению задания 1(битовые операции)