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.

2. 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.

3. Разработка:

1. Структуры представления данных:

Представление графа:

```
class 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

```
}
```

2. 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
					}
				}

			}
		}

	}
```

3. 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
	}
```

4. Алгоритм добавления узла на псевдокоде:

```
void addNode(int iKeyNew) {
		Узел* tmp = new Узел(iKeyNew);
		vectNodes->push_back(tmp); //Запись в список узлов
		iSize++; //Увеличение кол-ва вершин
	}
```

5. 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 );
```

6. Код:

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

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

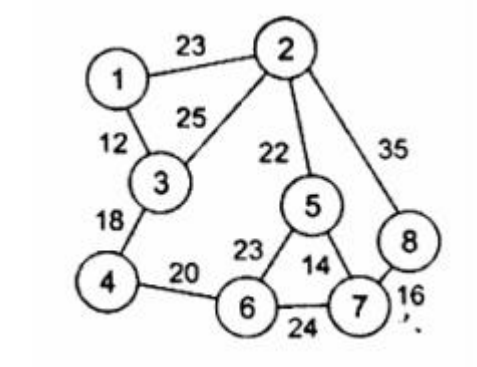
graph.h - класс Graph(граф) - файл graph.pdf

Source.cpp - основная программа - файл main.pdf

4. Тестирование:

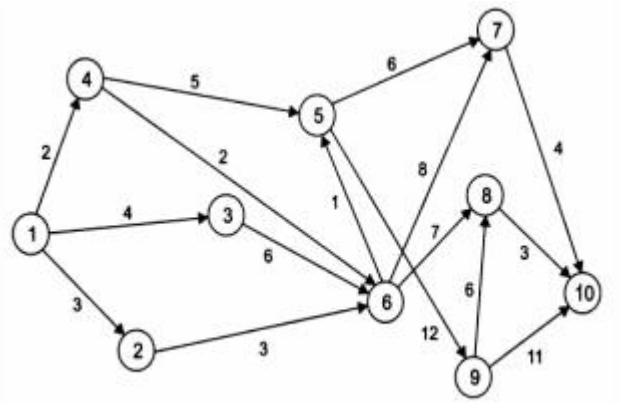
Для тестирования использовались графы представленные в задании:

Граф 10:



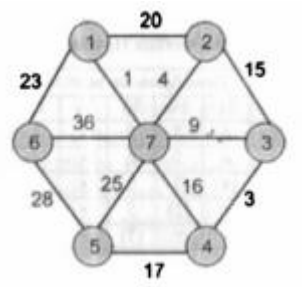

```
Graph 10
0
0 - 1
1 - 2
1 - 3
2 - 3
2 - 8
3 - 4
8 - 7
4 - 6
7 - 5
7 - 6
6 - 5
Way:
8
2
1
Way weighted: 58
```

Граф 7:




```
Graph 7
0
0 - 1
1 - 2
1 - 3
1 - 4
2 - 6
3 - 6
4 - 5
4 - 6
6 - 5
6 - 7
6 - 8
5 - 7
5 - 9
7 - 10
8 - 10
9 - 10
Way:
6
4
1
Way weighted: 4
```

Граф 3:




```
Graph 3
0
0 - 1
0 - 1
1 - 4
1 - 6
1 - 2
4 - 6
4 - 5
4 - 3
4 - 2
6 - 5
2 - 3
5 - 3
Way:
6
4
1
Way weighted: 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)