



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 7-2

Тема: «Графы: создание, алгоритмы обхода, важные задачи теории графов»

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент Фамилия И. О.
Группа AAAA-00-00

Москва 2024

СОДЕРЖАНИЕ

1 ВВЕДЕНИЕ.....	3
1.1 Постановка задачи.....	3
1.2 Индивидуальный вариант.....	3
2 ХОД РАБОТЫ.....	4
2.1 Анализ задачи.....	4
2.2 Результаты тестирования.....	8
3 ВЫВОД.....	9
4 ИНФОРМАЦИОННЫЕ ИСТОЧНИКИ.....	10

1 ВВЕДЕНИЕ

1.1 Постановка задачи

Составить программу создания графа и реализовать процедуру для работы с графом, определенную индивидуальным вариантом задания.

Самостоятельно выбрать и реализовать способ представления графа в памяти.

В программе предусмотреть ввод с клавиатуры произвольного графа. В вариантах построения остоного дерева также разработать доступный способ (форму) вывода результирующего дерева на экран монитора.

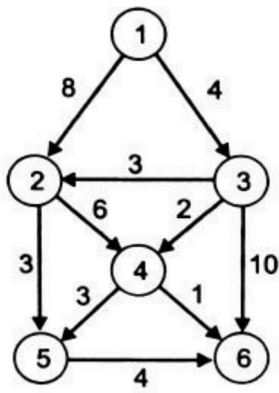
Провести тестовый прогон программы на предложенном в индивидуальном варианте задания графе. Результаты тестирования в виде скриншотов экранов включить в отчет по выполненной работе.

Сделать выводы о проделанной работе, основанные на полученных результатах.

Оформить отчет с подробным описанием рассматриваемого графа, принципов программной реализации алгоритмов работы с графом, описанием текста исходного кода и проведенного тестирования программы.

1.2 Индивидуальный вариант

Таблица 1 — Индивидуальный вариант

Вариант	Алгоритм	Предложенный граф
19	Нахождение кратчайшего пути методом построения дерева решений	

2 ХОД РАБОТЫ

2.1 Анализ задачи

Индивидуальным вариантом представлен взвешенный направленный граф. Предложенным алгоритмом является нахождение кратчайшего пути методом построения дерева решений. Алгоритм по сути является прямым перебором всех возможных путей с выбором наикратчайшего.

Для представления графа в памяти был выбран массив структур представляющих узлы графа. Код структуры представлен в листинге 1.

Представим узел АВЛ-дерева в виде структуры в соответствии с индивидуальным вариантом. Код структуры представлен в листинге 1.

Листинг 1 - Структура узла

```
struct Node
{
    int from;
    int to;
    int weight;
};
```

Само задание списка узлов представлено в листинге 2.

Листинг 2 - Задание списка узлов

```
vector<Node> graph = {};
graph = {
    {1, 2, 8},
    {1, 3, 4},
    {2, 4, 6},
    {2, 5, 3},
    {3, 2, 3},
    {3, 4, 2},
    {3, 6, 10},
    {4, 5, 3},
    {4, 6, 1},
    {5, 6, 4},
};
```

Алгоритм поиска кратчайшего пути представим рекурсивной функцией.

На вход функции поступают:

Ссылка на вектор graph — исходный граф,

Числа start и end — номера стартового узла и конечного соответственно,

Ссылка на вектор чисел path — вектор номеров узлов текущего пути,

Ссылка на значение числа minCost — число хранящее минимальный вес пути,

Ссылка на вектор число bestPath — вектор лучшего пути.

Функция проверяет, не равен ли номер стартового узла конечному.

Если равен, то функция считает вес пути path, и если он меньше минимального в minCost, то заменяет minCost на новое значение веса, и сохраняет текущий путь в bestPath а также завершает функцию.

Если стартовый номер не равен конечному, то необходимо выбрать следующий узел. Функция проходит по каждому следующему узлу в векторе graph, и рекурсивно вызывает функцию поиска кратчайшего пути.

В результате мы получим числа минимального веса пути, а также вектор номеров кратчайшего пути. Реализация функции представлена в листинге 4.

Листинг 3 - Функция поиска кратчайшего пути

```
void findShortestPath(const vector<Node> &graph, int start, int
end, vector<int> &path, int &minCost, vector<int> &bestPath)
{
    if (start == end)
    {
        int currentCost = 0;
        for (int i = 0; i < path.size() - 1; ++i)
        {
            auto it = find_if(graph.begin(), graph.end(), [&
(const Node &node) { return node.from == path[i] &&
node.to == path[i + 1]; }]);
            if (it != graph.end())
            {
                currentCost += it->weight;
            }
        }
        if (currentCost < minCost)
        {
            minCost = currentCost;
            bestPath = path;
        }
        return;
    }

    for (auto &edge : graph)
    {
```

```

        if (edge.from == start && find(path.begin(),
path.end(), edge.to) == path.end())
        {
            path.push_back(edge.to);
            findShortestPath(graph, edge.to, end, path,
minCost, bestPath);
            path.pop_back();
        }
    }
}

```

Для взаимодействия с графом и данной функцией пользователю предоставлен текстовый интерфейс. Реализация функции интерфейса представлена в листинге 4.

Листинг 4 - Функция вычисления высоты для заданного узла

```

void menu()
{
    cout << "Программа для работы с графом. Команды:" << endl;
    cout << "1 - Добавить узел" << endl;
    cout << "2 - Найти кратчайший путь" << endl;
    cout << "3 - Использовать предустановленный граф" << endl;
    cout << "4 - Выход" << endl;

    string command;
    int start = 1, end = 6;
    vector<Node> graph = {};

    while (true)
    {
        cout << "Введите номер команды: ";
        cin >> command;
        if (command == "1")
        {
            cout << "Формат ввода: от до вес. Пример: 1 2 5" <<
endl;

            cout << "Введите узел: ";
            int a, b, c;
            cin >> a >> b >> c;
            graph.push_back({a, b, c});
        }
        else if (command == "2")
        {
            int start, end;
            cout << "Введите номер узла откуда искать: ";
            cin >> start;
            cout << "Введите номер узла до куда искать: ";
            cin >> end;
            vector<int> path = {start};
            vector<int> bestPath;

```

```

        int minCost = numeric_limits<int>::max();
        findShortestPath(graph, start, end, path, minCost,
bestPath);

        cout << "Кратчайший путь: ";
        for (int node : bestPath)
        {
            cout << node << " ";
        }
        cout << endl;
        cout << "Всё всего пути: " << minCost;
    }
    else if (command == "3")
    {
        graph = {
            {1, 2, 8},
            {1, 3, 4},
            {2, 4, 6},
            {2, 5, 3},
            {3, 2, 3},
            {3, 4, 2},
            {3, 6, 10},
            {4, 5, 3},
            {4, 6, 1},
            {5, 6, 4},
        };
    }
    else if(command == "4"){
        break;
    }
    else {
        cout << "Неизвестная команда";
    }
    cout << endl;
}
}

```

Код основной программы представлен в листинге 5.

Листинг 5 - Код основной программы

```

int main()
{
    menu();
    return 0;
}

```

2.2 Результаты тестирования

Результаты тестирования представлены на рис. 1. Для тестирования был использован предложенный граф, и добавлена одна вершина к концу графа.

```
● [rubicus@rubicus output]$ ./"main"
Программа для работы с графом. Команды:
1 - Добавить узел
2 - Найти кратчайший путь
3 - Использовать предустановленный граф
4 - Выход
Введите номер команды: 3

Введите номер команды: 1
Формат ввода: от до вес. Пример: 1 2 5
Введите узел: 6 7 2

Введите номер команды: 2
Введите номер узла откуда искать: 1
Введите номер узла до куда искать: 7
Кратчайший путь: 1 3 4 6 7
Вес всего пути: 9
Введите номер команды: 4
○ [rubicus@rubicus output]$
```

Рисунок 1 - Результаты тестирования

3 ВЫВОД

Были освоены приёмы по представлению графов в программе, реализации функции поиска кратчайшего пути методом построения дерева решений.

4 ИНФОРМАЦИОННЫЕ ИСТОЧНИКИ

1. Структуры и алгоритмы обработки данных (часть 2): Лекционные материалы / Рысин М. Л. МИРЭА — Российский технологический университет, 2022/23. – 82 с.