

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)



ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №4.5

по дисциплине: Дискретная математика

тема: «Кратчайшие пути между каждой парой вершин во взвешенном орграфе»

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили:
ст. пр. Рязанов Юрий Дмитриевич
ст. пр. Бондаренко Татьяна Владими-
ровна

Белгород 2023 г.

Лабораторная работа №4.5

Кратчайшие пути между каждой парой вершин во взвешенном орграфе Вариант 10

Цель работы: изучить алгоритмы нахождения кратчайших путей между каждой парой вершин во взвешенном орграфе, научиться использовать их при решении различных задач.

1. Изучить алгоритмы нахождения кратчайших путей между каждой парой вершин во взвешенном орграфе.

В данной работе для реализации был выбран алгоритм Флойда.

```
template <typename E, typename EdgeValueType = typename E::NameType>
using ShortWay = std::pair<EdgeValueType, int>;
```

```
// Алгоритм Флойда
std::vector<std::vector<ShortWay<E>>> getShortestWayMatrix() {
    std::vector<std::vector<ShortWay<E>>> W(this->nodes.size(),
        std::vector<ShortWay<E>>(this->nodes.size()));

    // Формирование M
    for (int i = 0; i < this->nodes.size(); i++)
        for (int j = 0; j < this->nodes.size(); j++) {
            auto edge = getShortestEdge(i, j);
            if (edge == nullptr)
                W[i][j] = {std::numeric_limits<EdgeValueType>::max(), -1};
            else
                W[i][j] = {edge->name, (i == j ? 0 : i)};
        }

    for (int z = 0; z < this->nodes.size(); z++)
        for (int x = 0; x < this->nodes.size(); x++) {
            if (W[x][z].second == -1) continue;

            for (int y = 0; y < this->nodes.size(); y++) {
                if (W[z][y].second == -1) continue;

                EdgeValueType product = W[x][z].first + W[z][y].first;
                if (product < W[x][y].first) {
                    W[x][y].first = product;
                    W[x][y].second = W[z][y].second;
                }
            }
        }

    return W;
}

E* getShortestEdge(int i, int j) {
    if (this->edges[i][j] == nullptr) return nullptr;
```

```

    int minEdgeIndex = 0;
    for (int k = 0; k < (*this->edges[i][j]).size(); k++)
        if ((*this->edges[i][j])[minEdgeIndex]->current.name > (*this->edges[i][j])[k]->current.name)
            ↪ minEdgeIndex = k;

    return &(*this->edges[i][j])[minEdgeIndex]->current;
}

```

2. Разработать и реализовать алгоритм решения задачи.

Во взвешенном орграфе найти все пары вершин v_i и v_j , такие, что кратчайшее расстояние от v_i до v_j меньше кратчайшего расстояния от v_j до v_i . Вывести кратчайшие пути между найденными парами вершин.

```

template<typename G, typename W>
void printPath( G& graph, W shortestWayMatrix,
int begin, int end, bool isEnd) {
    if (begin == -1 || end == -1) return;

    if (begin != end)
        printPath(graph, shortestWayMatrix, begin, shortestWayMatrix[begin][end].second, false);

    std::cout << "(" << graph.nodes[end]->name << ")";
    if (!isEnd)
        std::cout << " ----> ";
}

template<typename T>
void analyzeTree(T& g, std::string graphName) {
    std::cout << "Graph " << graphName << ":\n";
    auto shortestWayMatrix = g.getShortestWayMatrix();

    for (int i = 0; i < g.nodes.size(); i++) {
        for (int j = i + 1; j < g.nodes.size(); j++) {
            if (j == i) continue;

            // Пути должны существовать
            if (shortestWayMatrix[i][j].second == -1 || shortestWayMatrix[j][i].second == -1) continue;

            // Длина путей должна быть равна
            if (shortestWayMatrix[i][j].first != shortestWayMatrix[j][i].first) continue;

            std::cout << "Found shortest paths with weight of " << shortestWayMatrix[i][j].first << ":\n";
            printPath(g, shortestWayMatrix, i, j, true);
            std::cout << "\n";
            printPath(g, shortestWayMatrix, j, i, true);
            std::cout << "\n" << std::endl;
        }
    }
}

```

```
}  
}
```

3. Подобрать тестовые данные. Результат представить в виде диаграммы графа. Вывод в консоль:

```
Graph Wreath:  
Found shortest paths with weight of 4:  
(1) ----> (3) ----> (4) ----> (5) ----> (2)  
(2) ----> (1)  
  
Graph Chain:  
Found shortest paths with weight of 1:  
(1) ----> (2)  
(2) ----> (1)  
  
Found shortest paths with weight of 2:  
(1) ----> (2) ----> (3)  
(3) ----> (2) ----> (1)  
  
Found shortest paths with weight of 3:  
(1) ----> (2) ----> (3) ----> (4)  
(4) ----> (3) ----> (2) ----> (1)  
  
Found shortest paths with weight of 1:  
(2) ----> (3)  
(3) ----> (2)  
  
Found shortest paths with weight of 2:  
(2) ----> (3) ----> (4)  
(4) ----> (3) ----> (2)  
  
Found shortest paths with weight of 1:  
(3) ----> (4)  
(4) ----> (3)  
  
Graph Hard chain:  
Found shortest paths with weight of 7:  
(1) ----> (2) ----> (3) ----> (4)  
(4) ----> (3) ----> (2) ----> (1)  
  
Graph Whale:  
Found shortest paths with weight of 6:  
(1) ----> (6) ----> (4)  
(4) ----> (3) ----> (2) ----> (1)  
  
Found shortest paths with weight of 8:  
(1) ----> (6) ----> (4) ----> (5)  
(5) ----> (4) ----> (3) ----> (2) ----> (1)  
  
Found shortest paths with weight of 6:  
(2) ----> (1) ----> (6)
```

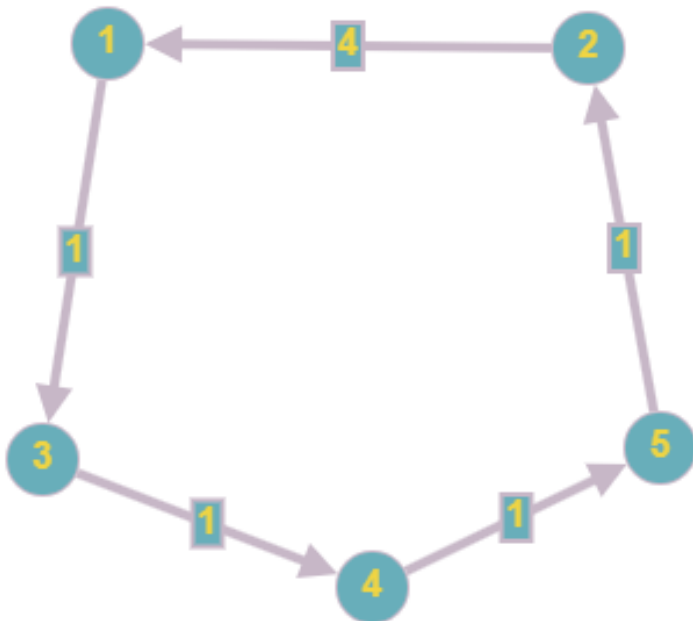
(6) ----> (4) ----> (3) ----> (2)

Found shortest paths with weight of 2:

(4) ----> (5)

(5) ----> (4)

Венок



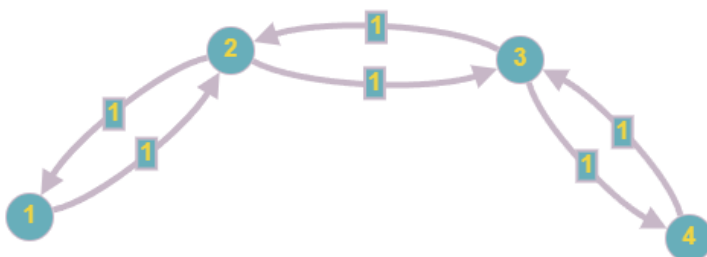
В данном графе есть две вершины 1 и 2, пути между которыми удовлетворяют условию задачи и их длина равна 4:

1 -> 3 -> 4 -> 5 -> 2

2 -> 1

Результат выполнения программы совпал с предположениями.

Цепочка



В данном графе можно составить пути между смежными вершинами, длина которых равна 1, например для вершин 1 и 2:

1 -> 2

2 -> 1

Аналогичные пути можно составить для вершин 2 и 3, 3 и 4.

Пути длиной 2 можно составить для вершин 1 и 3:

1 -> 2 -> 3

3 -> 2 -> 1

Аналогичные пути можно составить для вершин 2 и 4:

2 -> 3 -> 4

4 -> 3 -> 2

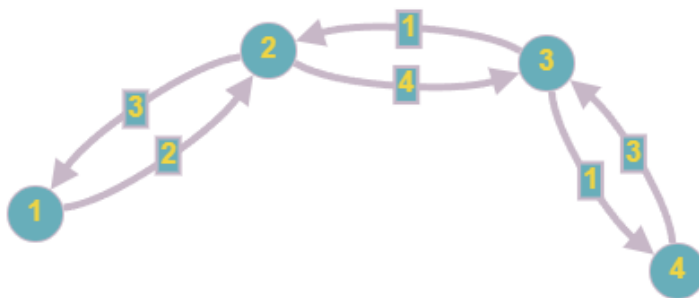
Пути длиной 3 можно составить для вершин 1 и 4:

1 -> 2 -> 3 -> 4

4 -> 3 -> 2 -> 1

Результат выполнения программы совпал с предположениями.

Сложная цепочка



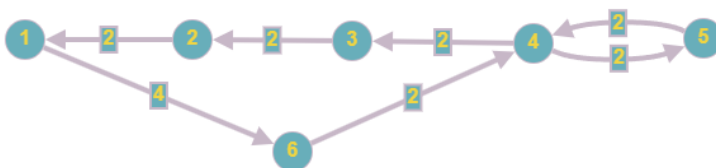
В данном графе уже не получится составить много путей с одинаковой длиной. В этом графе возможны пути только между вершинами 1 и 4 длиной 7:

1 -> 2 -> 3 -> 4

4 -> 3 -> 2 -> 1

Результат выполнения программы совпал с предположениями.

Кит



В данном графе возможны пути между вершинами 1 и 5 длиной 8:

5 -> 4 -> 3 -> 2 -> 1

1 -> 6 -> 4 -> 5

Результат выполнения программы содержит ожидаемый граф, однако также содержит пути, которые не были обнаружены при составлении графа вручную:

Между вершинами 1 и 4 с длиной 6:

1 -> 6 -> 4

4 -> 3 -> 2 -> 1

Между вершинами 2 и 6 длиной 6:

2 -> 1 -> 6

6 -> 4 -> 3 -> 2

Между вершинами 4 и 5 длиной 2:

4 -> 5

5 -> 4

Программа:

```
#include "../libs/alg/alg.h"

template<typename G, typename W>
void printPath( G& graph, W shortestWayMatrix,
int begin, int end, bool isEnd) {
    if (begin == -1 || end == -1) return;

    if (begin != end)
        printPath(graph, shortestWayMatrix, begin, shortestWayMatrix[begin][end].second, false);

    std::cout << "(" << graph.nodes[end]->name << ")";
    if (!isEnd)
        std::cout << " ----> ";
}

template<typename T>
void analyzeTree(T& g, std::string graphName) {
    std::cout << "Graph " << graphName << ":\n";
    auto shortestWayMatrix = g.getShortestWayMatrix();

    for (int i = 0; i < g.nodes.size(); i++) {
        for (int j = i + 1; j < g.nodes.size(); j++) {
            if (j == i) continue;

            // Пути должны существовать
            if (shortestWayMatrix[i][j].second == -1 || shortestWayMatrix[j][i].second == -1) continue;

            // Длина путей должна быть равна
            if (shortestWayMatrix[i][j].first != shortestWayMatrix[j][i].first) continue;

            std::cout << "Found shortest paths with weight of " << shortestWayMatrix[i][j].first << ":\n";
            printPath(g, shortestWayMatrix, i, j, true);
            std::cout << "\n";
            printPath(g, shortestWayMatrix, j, i, true);
            std::cout << "\n" << std::endl;
        }
    }
}

// Венок
void testWreath() {
    AdjacencyMatrixGraph<NamedEdge<Node<int>, unsigned long long>> g;
```

```

Node<int> N1(1);
Node<int> N2(2);
Node<int> N3(3);
Node<int> N4(4);
Node<int> N5(5);

g.addNode(N1);
g.addNode(N2);
g.addNode(N3);
g.addNode(N4);
g.addNode(N5);

g.addEdge({&N1, &N3}, 1, true);
g.addEdge({&N3, &N4}, 1, true);
g.addEdge({&N4, &N5}, 1, true);
g.addEdge({&N5, &N2}, 1, true);
g.addEdge({&N2, &N1}, 4, true);

analyzeTree(g, "Wreath");
}

// Цепочка
void testChain() {
    AdjacencyMatrixGraph<NamedEdge<Node<int>, unsigned long long>> g;

    Node<int> N1(1);
    Node<int> N2(2);
    Node<int> N3(3);
    Node<int> N4(4);

    g.addNode(N1);
    g.addNode(N2);
    g.addNode(N3);
    g.addNode(N4);

    g.addEdge({&N1, &N2}, 1, true);
    g.addEdge({&N2, &N1}, 1, true);
    g.addEdge({&N3, &N2}, 1, true);
    g.addEdge({&N2, &N3}, 1, true);
    g.addEdge({&N3, &N4}, 1, true);
    g.addEdge({&N4, &N3}, 1, true);
    analyzeTree(g, "Chain");
}

// Сложная цепочка
void testHardChain() {
    AdjacencyMatrixGraph<NamedEdge<Node<int>, unsigned long long>> g;

    Node<int> N1(1);
    Node<int> N2(2);
    Node<int> N3(3);
    Node<int> N4(4);

```



```

    g.addNode(N1);
    g.addNode(N2);
    g.addNode(N3);
    g.addNode(N4);

    g.addEdge({&N1, &N2}, 2, true);
    g.addEdge({&N2, &N1}, 3, true);
    g.addEdge({&N3, &N2}, 1, true);
    g.addEdge({&N2, &N3}, 4, true);
    g.addEdge({&N3, &N4}, 1, true);
    g.addEdge({&N4, &N3}, 3, true);
    analyzeTree(g, "Hard chain");
}

// Kum
void testWhale() {
    AdjacencyMatrixGraph<NamedEdge<Node<int>, unsigned long long>> g;

    Node<int> N1(1);
    Node<int> N2(2);
    Node<int> N3(3);
    Node<int> N4(4);
    Node<int> N5(5);
    Node<int> N6(6);

    g.addNode(N1);
    g.addNode(N2);
    g.addNode(N3);
    g.addNode(N4);
    g.addNode(N5);
    g.addNode(N6);

    g.addEdge({&N5, &N4}, 2, true);
    g.addEdge({&N4, &N5}, 2, true);
    g.addEdge({&N4, &N3}, 2, true);
    g.addEdge({&N3, &N2}, 2, true);
    g.addEdge({&N2, &N1}, 2, true);
    g.addEdge({&N1, &N6}, 4, true);
    g.addEdge({&N6, &N4}, 2, true);
    analyzeTree(g, "Whale");
}

void test() {
    testWreath();
    testChain();
    testHardChain();
    testWhale();
}

int main() {
    test();
}

```

Вывод: в ходе лабораторной работы изучили алгоритм Дейкстры нахождения кратчайших путей между вершинами взвешенного орграфа, научились рационально использовать его при решении различных задач.