Федеральное агентство связи

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра ПМиК

**РАСЧЁТНО-ГРАФИЧЕСКОЕ ЗАДАНИЕ**

по дисциплине «Теория сложности вычислительных процессов и структур»

Выполнил:

студент гр. ИП-713\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Михеев Н.А./

подпись

Проверил:

Доцент кафедры ПМиК

к.т.н. /Разинкина Т.Э./

ОЦЕНКА, подпись

Новосибирск, 2019

**Содержание**

1. Постановка задачи …..........................................................................................3

2. Основные идеи используемых методов…… ...................................................3

3. Код программы..............…..................................................................................4

4. Результаты работы...............................................................................................6

**Постановка задачи**

Написать программу для решения задачи Коммивояжера. Найти оптимальный путь (с минимальной стоимостью) при этом посетив все города и вернуться в изначальный город.

**Основные идеи используемых методов**

**Жадный метод решения задачи Коммивояжера.**

Для решения задачи Коммивояжера мною был выбран жадный алгоритм – алгоритм ближайшего соседа. Далее подробнее о нем:

**Алгоритм ближайшего соседа** — один из простейших алгоритмов решения задачи коммивояжёра. Относится к категории «жадных» алгоритмов.

Формулируется следующим образом:

Пункты обхода плана последовательно включаются в маршрут, причем каждый очередной включаемый пункт должен быть ближайшим к последнему выбранному пункту среди всех остальных, ещё не включенных в состав маршрута.

Алгоритм прост в реализации, быстро выполняется, но, как и другие «жадные» алгоритмы, может выдавать неоптимальные решения. Одним из эвристических критериев оценки решения является правило: если путь, пройденный на последних шагах алгоритма, сравним с путём, пройденным на начальных шагах, то можно условно считать найденный маршрут приемлемым, иначе, вероятно, существуют более оптимальные решения. Другой вариант оценки решения заключается в использовании алгоритма нижней граничной оценки (lower bound algorithm).

Но, к сожалению, у данного алгоритма есть свои недостатки: для любого количества городов, большего трех, в задаче коммивояжёра можно подобрать такое расположение городов (значение расстояний между вершинами графа и указание начальной вершины), что алгоритм ближайшего соседа будет выдавать наихудшее решение.

**Код программы**

#include <bits/stdc++.h>

using namespace std;

const int N = 6;

int final\_path[N+1];

bool visited[N];

int path\_cost = INT\_MAX;

int O = 0;

void copyToFinal(int *curr\_path*[])

{

    for (int i=0; i<N; i++)

        final\_path[i] = curr\_path[i];

    final\_path[N] = curr\_path[0];

}

int firstMin(int *matrix*[N][N], int *i*)

{

    int min = INT\_MAX;

    for (int k=0; k<N; k++)

        if (matrix[i][k] < min && i != k)

            min = matrix[i][k];

    return min;

}

int secondMin(int *matrix*[N][N], int *i*)

{

    int first = INT\_MAX, second = INT\_MAX;

    for (int j = 0; j < N; j++)

    {

        if (i == j)

            continue;

        if (matrix[i][j] <= first)

        {

            second = first;

            first = matrix[i][j];

        }

        else if (matrix[i][j] <= second && matrix[i][j] != first)

            second = matrix[i][j];

    }

    return second;

}

void TSPRec(int *matrix*[N][N], int *curr\_bound*, int *curr\_weight*,  int *level*, int *curr\_path*[])

{

    if (level == N)

    {

        if (matrix[curr\_path[level-1]][curr\_path[0]] != 0)

        {

            int curr\_res = curr\_weight + matrix[curr\_path[level-1]][curr\_path[0]];

*//O++;*

            if (curr\_res < path\_cost)

            {

                copyToFinal(curr\_path);

                path\_cost = curr\_res;

            }

        }

        return;

    }

    for (int i = 0; i < N; i++)

    {

        if (matrix[curr\_path[level - 1]][i] != 0 && visited[i] == false)

        {

            int temp = curr\_bound;

            curr\_weight += matrix[curr\_path[level - 1]][i];

*//O++;*

            if(level == 1)

              curr\_bound -= ((firstMin(matrix, curr\_path[level-1]) + firstMin(matrix, i))/2);

            else

              curr\_bound -= ((secondMin(matrix, curr\_path[level-1]) + firstMin(matrix, i))/2);

            if(curr\_bound + curr\_weight < path\_cost)

            {

                O++;

                curr\_path[level] = i;

                visited[i] = true;

                TSPRec(matrix, curr\_bound, curr\_weight, level + 1, curr\_path);

            }

            curr\_weight -= matrix[curr\_path[level - 1]][i];

            curr\_bound = temp;

            memset(visited, false, sizeof(visited));

            for (int j = 0; j <= level - 1; j++)

            {

                visited[curr\_path[j]] = true;

            }

        }

    }

}

void TSP(int *matrix*[N][N])

{

    int curr\_path[N+1];

    int curr\_bound = 0;

    memset(curr\_path, -1, sizeof(curr\_path));

    memset(visited, 0, sizeof(curr\_path));

    for (int i = 0; i < N; i++)

    {

        curr\_bound += (firstMin(matrix, i) + secondMin(matrix, i));

    }

    curr\_bound = (curr\_bound & 1)? curr\_bound/2 + 1 : curr\_bound/2;

    visited[0] = true;

    curr\_path[0] = 0;

    TSPRec(matrix, curr\_bound, 0, 1, curr\_path);

}

int main()

{

    int matrix[N][N];

    cout << "Enter villages matrix: " << endl;

    for(int i = 0; i < N; i++)

    {

        for(int j = 0; j < N; j++)

        {

            cin >> matrix[i][j];

        }

    }

    TSP(matrix);

    if(path\_cost != INT\_MAX)

    {

        cout << "Minimal cost: " << path\_cost << endl;

        cout << "Path of jorney:" << endl;

        for (int i = 0; i <= N; i++)

        {

            if(i != N)

                cout << final\_path[i] << " -> ";

            else

            {

                cout << final\_path[i] << endl;

            }

        }

        cout << "Difficulty: " << O << endl;

    }

    else

    {

        cout << "Couldn't go through all villages." << endl;

    }

    return 0;

}

**Результаты работы**

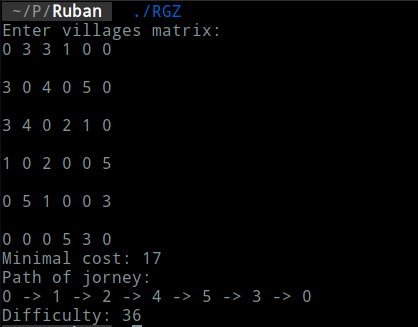


Рисунок 1 – Результат работы программы для 6 деревень со своей матрицей расстояний, при проходе из 0 деревни.

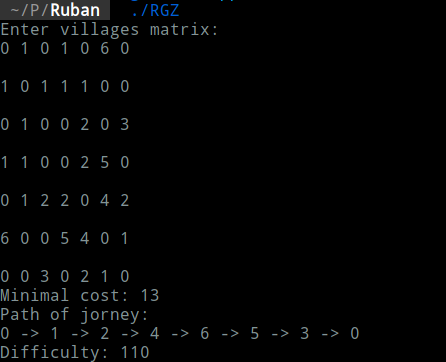


Рисунок 2 – Результат работы программы для 7 деревень со своей матрицей расстояний, при проходе из 0 деревни.