**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЁТ**

**по лабораторной работе №3**

**по дисциплине «Построение и анализ алгоритмов»**

**Тема: Метод ветвей и границ.**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 5382 |  | Лисс Н.И. |
| Преподаватель |  | Шолохова О.М. |

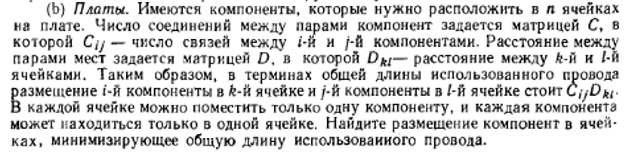
Санкт-Петербург

2017

**Цель работы.**

Изучение использования метода ветвей и границ.

**Постановка задачи.**



**Основные теоретические положения.**

**Метод ветвей и границ** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *branch and bound*) — общий алгоритмический [метод](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4) для нахождения оптимальных решений различных задач оптимизации, особенно дискретной и [комбинаторной оптимизации](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%B1%D0%B8%D0%BD%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F_%D0%BE%D0%BF%D1%82%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F). По существу, метод является вариацией [полного перебора](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%BD%D1%8B%D0%B9_%D0%BF%D0%B5%D1%80%D0%B5%D0%B1%D0%BE%D1%80) с отсевом подмножеств допустимых решений, заведомо не содержащих оптимальных решений.

Процедура *ветвления* состоит в разбиении множества допустимых значений переменной {\displaystyle x}х на подобласти (подмножества) меньших размеров. Процедуру можно рекурсивно применять к подобластям. Полученные подобласти образуют [дерево](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2)), называемое *деревом поиска* или *деревом ветвей и границ*. *Узлами* этого дерева являются построенные подобласти (подмножества множества значений переменной {\displaystyle x} х).

**Спецификация программы.**

Программа ищет такой вариант размещения компонент в ячейках, при котором общая длина использованного провода минимальна.

Программа написана на языке С++.

Входные данные: матрица С количества проводов между компонентами и матрица D расстояний между ячейками.

Выходные данные: Минимальная из получившихся разными размещениями общих длин использованного провода.

**Реализация программы.**

Используя рекурсию и вектор решений, который представляет собой список свободных ячеек, помещаем компоненту и после каждой компоненты считаем длину. После выполнения одной ветки полного решения записывается его результат. В дальнейшем на каждой вставке нового компонента в других решениях проверяется текущая длина. Если она больше чем записанная длина, то текущее решение обрывается. Иначе длина перезаписывается.

*Функции:*

T countS (const std::vector<std::vector<T>> &distance,const std::vector<std::vector<T>> &wires,std::vector<T> place) -подсчет длины на текущем решении

void spaceComponents(const std::vector<std::vector<T>> &distance,const std::vector<std::vector<T>> &wires,std::vector<T> options,std::vector<T> place,T &bestS) -В этой функции начинается ветвление решений ,которая начинается с выбора всех возможных кандидатов на первую ячейку .Все эти варианты входят в описанной ниже функции

void spaceComponentsT( T variant,T free,T choose,const std::vector<std::vector<T>> &distance,const std::vector<std::vector<T>> &wires,std::vector<T> options,std::vector<T> place,T &bestS)-продолжение выполнения решений ,функция заканчивает работу , когда текущее решение больше лучшего или это конец решения ,и оно оказывается лучше , чем предыдущее.

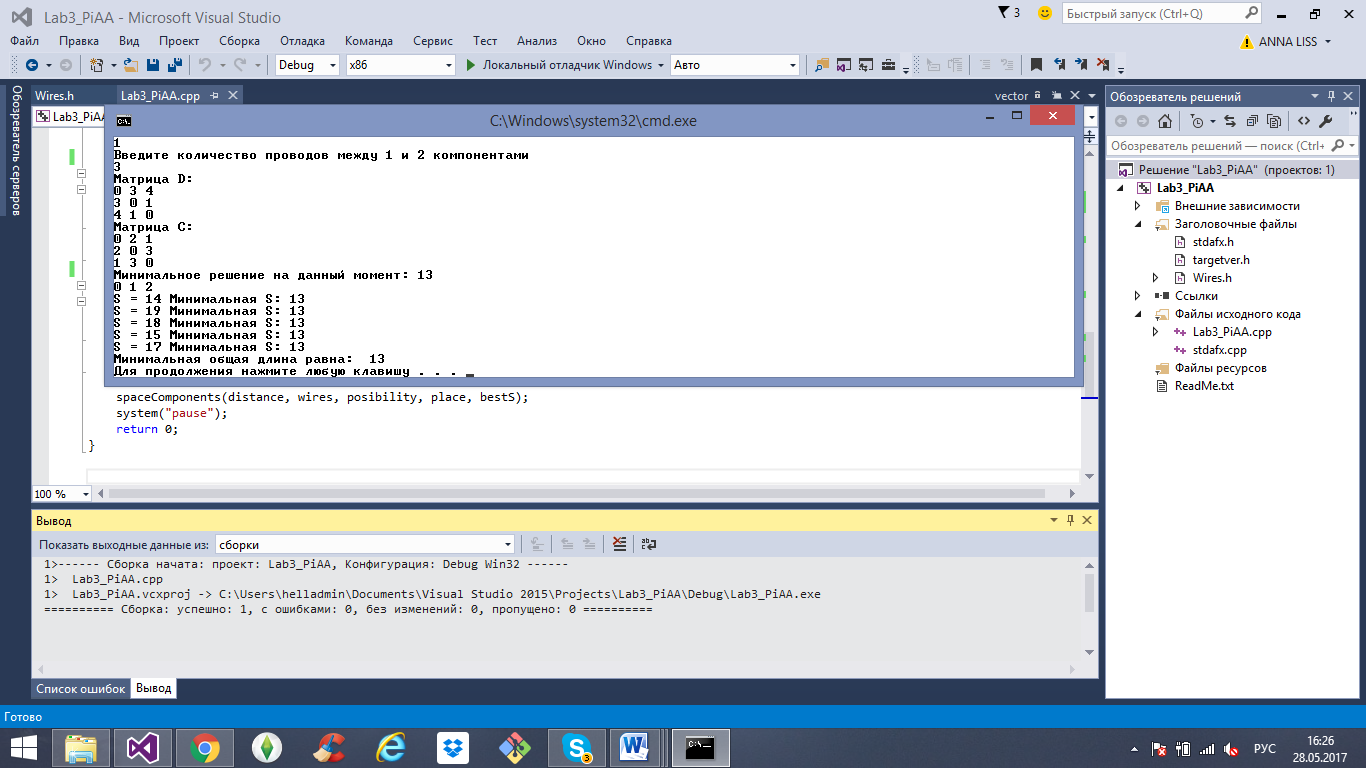
**Тестирование.**

1.Рассмотрим случай, когда дана матрица C и D:

C =; D =

Возьмем первый вариант расположения: 123, тогда S = 2\*3+1\*3+1\*4 = 13.  
Возьмем второй вариант расположения: 231, тогда S = 3\*3+1\*1+4\*2 = 18.  
Возьмем третий вариант расположения: 321, тогда S = 3\*3+1\*4+ 2\*1 = 15.  
И тд.

Проверим правильность выполнения работы программы:



Результаты одинаковы, следовательно, программа работает корректно.

**Код программы.**

Файл Lab3\_PiAA:  
// Lab3\_PiAA.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include <iostream>

#include <vector>

#include "Wires.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

cout << "Лабораторная работа №3 ст.гр. 5382 Лисс Натальи. " << endl;

cout << "Введите количество ячеек: " << endl;

int n;

cin >> n;

vector <vector<int>> distance(n);

for (int j = 0; j < n; ++j) {//Resizes the container so that it contains n elements.

distance[j].resize(n);

}

int count = 0;

for (int i = 0; i<n; i++) {

for (int q = 0; (q + count)<n; q++) {

if (i != (q + count)) {

cout << "Введите расстояние между " << i << " и " << count + q << " ячейкой:" << endl;

cin >> distance[i][count + q];//fill the matrix D

distance[count + q][i] = distance[i][count + q];

}

else

distance[i][count + q] = 0;

}

++count;

};

count = 0;

vector <vector<int>> wires(n);

for (int j = 0; j < n; ++j) {

wires[j].resize(n);

}

for (int i = 0; i<n; i++) {

for (int q = 0; (q + count)<n; q++) {

if (i != (q + count)) {

cout << "Введите количество проводов между " << i << " и " << count + q << " компонентами" << endl;

cin >> wires[i][count + q];//matrix C(wires)

wires[count + q][i] = wires[i][count + q];

}

else

wires[i][count + q] = 0;

}

++count;

};

vector<int> posibility;

for (int p = 0; p<n; p++)

posibility.push\_back(p);

vector<int> place;

int bestS = 0;

cout << "Матрица D:" << endl;

for (int i = 0; i<n; ++i) {

for (int q = 0; q<n; ++q) {

cout << distance[i][q] << " ";

}

cout << endl;

}

cout << "Матрица C:" << endl;

for (int i = 0; i<n; ++i) {

for (int q = 0; q<n; ++q) {

cout << wires[i][q] << " ";

}

cout << endl;

}

spaceComponents(distance, wires, posibility, place, bestS);

system("pause");

return 0;

}

Файл Wires.h:  
#pragma once

#include <iostream>

#include <vector>

#include <math.h>

template <class T> //counting the sum distance

T countS(const std::vector<std::vector<T>> &distance, const std::vector<std::vector<T>> &wires, std::vector<T> place)

{

T bS = 0;

for (unsigned int count1 = 0; count1<place.size(); count1++) {

for (unsigned int count2 = 1; count2 + count1<place.size(); count2++) {

bS = bS + distance[count1][count1 + count2] \* wires[place[count1]][place[count1 + count2]];

}

}

return bS;

}

template <class T>

void spaceComponentsT(T variant, T free, T choose, const std::vector<std::vector<T>> &distance, const std::vector<std::vector<T>> &wires, std::vector<T> options, std::vector<T> place, T &bestS)

{

std::swap(options[0 + free - 1], options[choose]);

place.push\_back(variant);

T S = countS(distance, wires, place);

if ((S>bestS) && (bestS != 0)) {

std::cout << "S = " << S << " Минимальная S: " << bestS << std::endl;

return;

}

if (place.size() == options.size()) {

bestS = S;

std::cout << "Минимальное решение на данный момент: " << bestS << std::endl;

for (unsigned int p = 0; p<place.size(); p++)

std::cout << place[p] << " ";

std::cout << std::endl;

return;

}

for (unsigned int var = 0; var + free < options.size(); var++)

{

spaceComponentsT(options[var + free], (free + 1), (int)(var + free), distance, wires, options, place, bestS);

}

return;

}

template <class T>//for first

void spaceComponents(const std::vector<std::vector<T>> &distance, const std::vector<std::vector<T>> &wires, std::vector<T> options, std::vector<T> place, T &bestS)

{

for (unsigned int var = 0; var < options.size(); var++)

{

spaceComponentsT(options[var], 1, (int)var, distance, wires, options, place, bestS);

}

std::cout << "Минимальная общая длина равна: " << bestS << std::endl;

return;

}

**Вывод.**

В данной лабораторной работе был использован метод ветвей и границ, который по сути напоминает поиск с возвратом.