**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №3**

**по дисциплине «Построение и Анализ Алгоритмов»**

**Тема: Метод ветвей и границ**

**Вариант 1b**

|  |  |  |
| --- | --- | --- |
| Студент гр. 5382 |  | Забалуев Н.Е. |
| Преподаватель |  | Шолохова О.М. |

Санкт-Петербург

2017

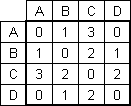
1. *Содержательная постановка задачи*

Платы. Имеются компоненты, которые нужно расположить в n ячейках на плате. Число соединений между парами компонент задается матрицей C, в которой Cij – число связей между *i*-й и *j*-й компонентами. Расстояние между парами мест задается матрицей D, в которой Dkl – расстояние между *k*-й и *l*-й ячейками. Таким образом, в терминах общей длины использованного провода, размещение *i*-й компоненты в *k*-й ячейке и *j*-й компоненты в *l*-й ячейке стоит C*ij\**D*kl*. В каждой ячейке можно поместить только одну компоненту, и каждая компонента может находиться только в одной ячейке. Найдите размещение компонент в ячейках, минимизирующее общую длину использованного провода.

1. *Анализ и пример решения задачи*

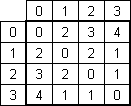
В данной лабораторной работе поставленную задачу необходимо решить с использованием метода ветвей и границ. Метод ветвей и границ ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) branch and bound) — общий алгоритмический [метод](http://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4) для нахождения оптимальных решений различных задач оптимизации, особенно дискретной и [комбинаторной оптимизации](http://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](http://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.

Так как метод ветвей и границ «базируется» на полном переборе (с частичным отсечением неудовлетворительных решений), рассмотрим возможность осуществления полного перебора в нашей задаче. Итак, в начале работы программы ей подаются две матрицы. Первая матрица – матрица смежности, т.е. какие вершины (транзисторы) с какими соединены и сколькими проводами. Например:



Для удобства транзисторы не пронумерованы, а имеют буквенные обозначения (A..D). Программе на вход подается область, выделенная черной обводкой, т.е. матрица смежности, размера N\*N.

Второй параметр – матрица «стоимостей», т.е. матрица, оценивающая расстояния между ячейками на плате. Ячеек ровно такое же количество, как и транзисторов, следовательно, вторая матрица так же имеет размер N\*N:



Каким образом можно произвести полный перебор при решении данной задачи? Полный перебор будет достигнут тогда, когда транзисторы побывают во всех возможных комбинациях. Эту задачу можно привести к нахождению количества перестановок для N элементов. Из курса дискретной математики известно, что для N элементов количество перестановок будет равно N!. Так, например, для N = 3:

A B C

A C B

B A C

B C A

C A B

C B A

Других комбинаций для трех элементов получить невозможно. Это и будет полным перебором в нашей задаче. Т.е. для того, чтобы найти наименьшую стоимость размещения N транзисторов в N ячейках, необходимо сделать вычисления стоимости для N! возможных наборов расстановок, после чего выделить среди них наименьшую, это и будет результатом.

Каким образом производить оценку стоимости при каждой расстановке транзисторов? Рассмотрим на примере. Данные возьмем из таблиц, представленных выше. Пусть, на некотором этапе мы получили перестановку вида A D B C. Как же подсчитать стоимость данной расстановки?



Выше показано, в каких ячейках расположены транзисторы при данной расстановке. Обратимся к таблице смежности C и посмотрим, какие транзисторы должны быть соединены.

1. Транзистор A должен быть соединен:

с транзистором B – 1 раз

с транзистором C – 3 раза

1. Транзистор B должен быть соединен:

с транзистором C – 2 раза

с транзистором D – 1 раз

1. Транзистор C должен быть соединен:

с транзистором D – 2 раза

При обработке данной таблицы достаточно рассматривать её как верхнее-треугольную матрицу, т.к. нижняя часть матрицы представляет собой симметричное отображение верхней. Это логично, т.к. если транзистор A должен быть соединен с транзистором B, то и транзистор B должен быть соединен с A.

Итак, мы увидели какие транзисторы с какими должны быть соединены. На следующем этапе смотрим, на каких позициях находятся в данной расстановке эти транзисторы. Необходимые соединения будем записывать парами букв:

AB = 02

AC = 03

BC = 23

BD = 21

CD = 31

У нас имеются все данные для того чтобы подсчитать вес данной перестановки. Для этого умножаем количество необходимых связей между компонентами X и Y на расстояние между ячейками, в которые установлены данные транзисторы.

Так, для транзисторов A и B:

CAB = 1

AB = 02

D02 =3

Стоимость AB = CAB \* D02 = 1 \* 3 = 3

Аналогично для остальных:

AC = CAC \* D03 = 3 \* 4 = 12

BC = CBC \* D23 = 2 \* 1 = 2

BD = CBD \* D21 = 1 \* 2 = 2

CD = CCD \* D31 = 2 \* 1 = 2

Итоговая стоимость получается сложением полученных частичных сумм:

S = 3 + 12 + 2 + 2 + 2 = 21

Произведя приведенный выше набор операций для всех возможных перестановок, мы получим стоимость каждой из них. После чего можно произвести выбор минимальной это и будет решением. Так действует полный перебор, однако нам необходимо преобразовать данный способ, что бы он использовал метод ветвей и границ. Это можно сделать следующим способом.

Основная идея метода ветвей и границ состоит в том, чтобы для каждого подмножества решений, полученного на некотором уровне, найти оценку перспективности. В методе ветвей и границ эта оценка называется оценочной функцией. В качестве оценочной функции используется значение критерия эффективности для промежуточного решения – стоимость части затраченного провода от корневой вершины до текущей. Верхней границей будет являться уже найденное самое «дешевое» решение. В самом деле, если уже имеется решение, имеющее стоимость ниже, чем у какой-либо ветви на определенном этапе, значит эту ветвь можно отбросить, т.к. уменьшение стоимости с продвижением по ветви невозможно, лишь её увеличение.

Итак, на основе этого можно разработать следующую схему:

1. В начале работы метода верхняя граница принимается  (INT\_MAX)
2. Начинается вычисление стоимости всех перестановок, поочередно
3. Если на каком-либо этапе вычисления стоимости перестановки, её значение превышает значение верхней границы – данная ветвь отбрасывается, и программа переходит к вычислению стоимости следующей перестановки.
4. Если полученная стоимость перестановки оказалась равна верхней границе – значит найдено дополнительное решение, имеющее ту же стоимость. Необходимо сохранить данное решение и продолжить дальнейшую обработку перестановок.
5. Если полученная стоимость перестановки меньше верхней границы – значит найдено новое решение для задачи. Необходимо очистить список найденных ранее решений, сохранить на их место данное, переопределить значение верхней границы, заменив его стоимостью нового найденного решения и продолжить обработку остальных перестановок.
6. По окончании перебора всех перестановок необходимо вывести результаты работы алгоритма. Верхняя граница – будет являться требуемой наименьшей стоимостью среди всех решений, а в списке сохраненных решений будут находится все комбинации, при которых данное значение стоимости достигается.

При реализации программы, для реализации контейнера, хранящего решения поставленной задачи, был использован класс «vector» из стандартной библиотеки шаблонов STL.

1. *Описание функций*

void matrix\_input (int n, vector < vector <int> > &matrix, istream &in)

//Ввод матриц

void matrix\_output(int n, vector < vector <int> > matrix)

//Вывод матриц

void show\_solutions(int n, vector < vector <int> > solution)

//Вывод результатов

bool is\_a\_solution(vector<int> perm, int n, int current\_cost, int lowcost)

//Проверяет, является ли текущая перестановка решением

void process\_solution(vector<int> perm, int n, vector < vector <int> > &solution, int current\_cost, int &lowcost)

//Функция записывает очередное решение в вектор решений

void construct\_candidates (vector<bool> in\_perm, int n, vector<int> &candidates)

//Функция определения кандидатов на следующую позицию

int additional\_cost(int n, vector < vector <int> > C, vector < vector <int> > D, vector<int> perm, int current\_cost, int lowcost)

//Функция высчитывает добавочную стоимость на очередном шаге

void backtrack( vector<int> &perm, vector<bool> &in\_perm, int n, vector < vector <int> > C, vector < vector <int> > D, int &lowcost, int &current\_cost, vector < vector <int> > &solution)

//Функция, генерирующая перестановки

1. *Заключение*

В данной лабораторной мы познакомились с алгоритмом работы метода ветвей и границ. Воспользовались умениями использования стандартной библиотеки STL.

**Приложение А. Исходный код.**

main.cpp

#include <iostream>

#include <vector>

#include <climits>

#include <fstream>

using namespace std;

bool finished=false;

void matrix\_input (int n, vector < vector <int> > &matrix, istream &in)

{

for (size\_t i=0; i<(n-1); i++)

{

for (size\_t j=(i+1); j<n; j++)

{

matrix[i][i]=0;

cout << "¬ведите элемент матрицы [" << i << "]" << "[" << j << "]:";

in >> matrix[i][j];

cout << matrix[i][j] << endl;

//cin>>matrix[i][j];

matrix[j][i]=matrix[i][j];

}

}

matrix[n-1][n-1]=0;

}

void matrix\_output(int n, vector < vector <int> > matrix)

{

for (size\_t i=0; i<n; i++)

{

for (size\_t j=0; j<n; j++)

cout<<matrix[i][j]<<'\t';

cout<<endl;

}

}

void show\_solutions(int n, vector < vector <int> > solution)

{

for (size\_t i=0; i<solution.size(); i++)

{

for (size\_t j=0; j<n; j++)

{

cout << solution[i][j]+1 << " ";

}

cout<<endl;

}

}

bool is\_a\_solution(vector<int> perm, int n, int current\_cost, int lowcost)

{

if ((perm.size()==n) && (current\_cost<=lowcost))

return true;

else

return false;

}

void process\_solution(vector<int> perm, int n, vector < vector <int> > &solution, int current\_cost, int &lowcost)

{

cout<<"\nЌайдено 'выгодное' решение!"<<endl<<"≈го 'стоимость' равна: "<<current\_cost<<endl;

if (current\_cost<lowcost)

{

solution.clear();

solution.push\_back(perm);

lowcost=current\_cost;

}

else

solution.push\_back(perm);

for (size\_t i=0; i<n-1; i++)

{

if (perm.at(i)>perm.at(i+1))

finished=true;

else

{

finished=false;

break;

}

}

}

void construct\_candidates (vector<bool> in\_perm, int n, vector<int> &candidates)

{

for (size\_t i=0; i<n; i++)

{

if (in\_perm.at(i)==0)

candidates.push\_back(i);

}

}

int additional\_cost(int n, vector < vector <int> > C, vector < vector <int> > D, vector<int> perm, int current\_cost, int lowcost)

{

int add=0;

for (size\_t i=0; i<(perm.size()-1); i++)

{

add+=D[i][perm.size()-1]\*C[perm.back()][perm.at(i)];

if (add+current\_cost>lowcost)

break;

}

return add;

}

void backtrack( vector<int> &perm, vector<bool> &in\_perm, int n, vector < vector <int> > C, vector < vector <int> > D, int &lowcost, int &current\_cost, vector < vector <int> > &solution)

{

vector<int> candidates;

if (is\_a\_solution(perm, n, current\_cost, lowcost))

{

process\_solution(perm, n, solution, current\_cost, lowcost);

}

else

{

construct\_candidates(in\_perm, n, candidates);

for (size\_t i=0; i<candidates.size(); i++)

{

perm.push\_back(candidates.at(i));

in\_perm.at(candidates.at(i))=1;

int added = additional\_cost(n, C, D, perm, current\_cost, lowcost);

current\_cost+=added;

if (current\_cost>lowcost)

{

current\_cost-=added;

in\_perm.at(candidates.at(i))=0;

perm.pop\_back();

break;

}

else

backtrack(perm, in\_perm, n, C, D, lowcost, current\_cost, solution);

current\_cost-=added;

in\_perm.at(candidates.at(i))=0;

perm.pop\_back();

}

}

if (finished) return;

}

int main()

{

setlocale( LC\_ALL,"Russian" );

int n=0;

int lowcost=INT\_MAX;

int current\_cost=0;

ifstream in ("C:/cpp/5382zabaluev/3laba/in.txt");

cout << "¬ведите количество ¤чеек на плате:";

in >> n;

cout<<n<<endl;

//cin >> n;

//while(n<3)

//{

// system("sls");

// cout << "¬ведите количество ¤чеек на плате (>1):";

// cin >> n;

//}

vector < vector <int> > C(n, vector <int> (n) );

vector < vector <int> > D(n, vector <int> (n) );

cout << "¬ведите матрицу смежности —:" << endl;

matrix\_input(n, C, in);

cout << "¬ведите матрицу 'стоимостей' D:" << endl;

matrix\_input(n, D, in);

vector < vector <int> > solution;

vector<int> perm;

vector<bool> in\_perm(n, 0);

backtrack(perm, in\_perm, n, C, D, lowcost, current\_cost, solution);

cout << "\nƒл¤ " << n << " плат, при следующих матрицах C и D:" << endl << "\nC:" << endl;

matrix\_output(n, C);

cout << "\nD:" << endl;

matrix\_output(n, D);

cout << "\nЌаименее затратными (со 'стоимостью' "<<lowcost<<") ¤вл¤ютс¤ следующие перестановки:" << endl;

show\_solutions(n, solution);

system("pause");

return 0;

}