Лабораторная работа №4

Артамоновой Анастасии ПИн-24

**Конспект**

Метод ветвей и границ для решения задачи коммивояжера

Метод ветвей и границ применим в случае, когда выполняются специфические дополнительные условия на множество M и минимизируем на нем функцию.

Функция называется оценочной. Очевидно, что рекорд не обязан доставлять минимум функции f ; однако при благоприятных обстоятельствах возникает возможность сократить перебор. Описанный выше процесс построения рекорда состоял из последовательных этапов, на каждом из которых фиксировалось несколько множеств и затем выбиралось одно из них.

Рассмотрим пример использования метода ветвей и границ для решения задачи о коммивояжере. Пусть имеется несколько городов, соединенных некоторым образом дорогами с известной длиной. Орграф называется полным, если в нем имеются все возможные ребра. Требуется установить, имеется ли путь, двигаясь по которому можно побывать в каждом городе только один раз и при этом вернуться в город, откуда путь был начат («обход коммивояжера»), и, если таковой путь имеется, установить кратчайший из таких путей.

Формализуем условие в терминах теории графов. Города будут вершинами графа, а дороги между городами – ориентированными (направленными) ребрами графа, на каждом из которых задана весовая функция: вес ребра - это длина соответствующей дороги. Путь, который требуется найти, - это ориентированный остовный простой цикл минимального веса в орграфе, такие циклы называются также гамильтоновыми (цикл называется остовным, если он проходит по всем вершинам графа; цикл называется простым, если он проходит по каждой вершине только один раз; цикл называется ориентированным, если начало каждого последующего ребра совпадает с концом предыдущего; вес цикла ‒ это сумма весов его ребер).

Заметим, что вопрос о наличии в орграфе гамильтонова цикла достаточно рассмотреть как частный случай задачи о коммивояжере для полных орграфов. Если исходный орграф не является полным, то его можно дополнить недостающими ребрами и каждому из добавленных ребер приписать вес , т.е. максимальное значение из всех возможных в рассмотрениях чисел.

Если во вновь построенном полном орграфе найти гамильтоновый цикл, то при наличии у него ребер с весом ¥ можно будет говорить, что в данном, исходном графе «цикла коммивояжера» нет.

Если же в полном орграфе легчайший гамильтонов цикл окажется конечным по весу, то он и будет искомым циклом в исходном графе. Отсюда следует, что задачу о коммивояжере достаточно решить для полных орграфов с весовой функцией.

Основные определения

Пусть имеется некоторая числовая матрица. Привести строку этой матрицы означает выделить в строке минимальный элемент (его называют константой приведения) и вычесть его из всех элементов этой строки. Очевидно, в результате в этой строке на месте минимального элемента окажется нуль, а все остальные элементы будут неотрицательными. Аналогичный смысл имеет определение - привести столбец матрицы.

Привести матрицу по строкам означают, что все строки матрицы приводятся. Аналогичный смысл имеет фраза ‒ привести матрицу по столбцам. Привести матрицу означают, что матрица сначала приводится по строкам, а потом ‒ по столбцам.

Весом элемента матрицы называют сумму констант приведения матрицы, которая получается из данной матрицы заменой обсуждаемого элемента . Следовательно, словосочетание ‒ самый тяжелый нуль в матрице означает, что в матрице подсчитан вес каждого нуля, а затем фиксирован нуль с максимальным весом.

Начальное приведение матрицы стоимости.

Матрица стоимости называется приведенной, если она имеет в каждой строке и каждом столбце хотя бы один нуль. Операция приведения заключается в вычитании из элементов каждой строки (столбца)

минимального элемента этой строки (столбца) – константы приведения. Сумма констант приведения образует нижнюю граничную оценку стоимости любого возможного тура.

Вычисление функции штрафа. Функция штрафа – это множество чисел, вычисленных для каждого нуля приведенной матрицы посредством суммирования двух минимальных чисел, из той строки и того столбца, в которых расположен нулевой элемент.

Выбор ребра ветвления. Для ветвления необходимо выбирать ребро, которому соответствует максимальная функция штрафа. Если существует несколько одинаковых максимальных значений функции штрафа, то выбор среди них может быть произвольным.

Вычисление граничной оценки для ветви, соответствующей не включению ребра в тур. Эта оценка вычисляется как сумма граничной оценки, соответствующей предыдущему узлу дерева перебора, и выбранного значения функции штрафа.

Вычисление граничной оценки для ветви, соответствующей включению ребра в тур. Для вычисления граничной оценки необходимо:

· вычеркнуть в матрице стоимости строку и столбец, соответствующие выбранному ребру;

· скорректировать полученную матрицу таким образом, чтобы устранить возможность досрочного завершения тура (устранить циклы);

· осуществить приведение (если необходимо) полученной матрицы, и если константа приведения отлична от нуля, то сложить эту константу с граничной оценкой предыдущего узла.

Проверка на окончание решения. Если скорректированная матрица имеет размер 2 × 2, и если узел дерева, которому соответствует эта матрица, имеет минимальную граничную оценку, то решение задачи заканчивается: два оставшихся нуля этой матрицы соответствуют двум последним ребрам,

которые включаются в тур непосредственно, при этом, очевидно, стоимость тура не изменяется.

Машина Тьюринга

В 1937 г. английский математик Алан Матисон Тьюринг опубликовал работу, в которой уточнил понятие алгоритма, прибегая к воображаемой вычислительной машине.

Машина Тьюринга – один из способов записи алгоритма. Машина Тьюринга – алгоритм, записью которого является функциональная таблица или функциональная диаграмма, а правилом выполнения – описание ее устройства.

Машина Тьюринга так же, как и конечный автомат, является дискретным устройством преобразования информации. Приведем ее точное определение, а затем интерпретацию ее работы.

Машиной Тьюринга называется частичное отображение М:

{Q0, Q1,...,Qn‒1} × {0, 1} {Q0, Q1,...,Qn‒1} × {L, R} × {0, 1},

где {Q0, Q1,...,Qn‒1} - множество состояний машины; L, R - обозначает

«влево», «вправо».

Тот факт, что отображение частичное, означает, что М может быть определено не для всех наборов аргументов. Машина Тьюринга работает с бесконечной в обе стороны лентой, разбитой на ячейки, в каждой из которых написан один из символов - 0 или 1.

Считывающая (записывающая) головка машины обозревает в каждый момент времени одну из ячеек и за один такт, сменяющий два последовательных момента времени, может перемещаться влево или вправо.

Машина Тьюринга в каждый момент времени находится в одном из состояний Q0, Q1, ..., Qn‒1, и в следующий момент времени переходит в другое состояние или остается в том же. Кроме того, машина может изменять символ, стоящий в обозреваемой ячейке. Все эти преобразования - изменения состояния, информации на ленте, направления движения полностью определяются отображением М, а именно, если М (i, ε) = (j, L, η), то в случае, когда машина находится в состоянии Qi, а на обозреваемой в данный момент ячейке написан символ ε, машина должна записать в эту ячейку вместо

символа ε символ η, перейти в состояние Qj и сдвинуться на одну ячейку влево. В случае, когда М (i, ε) = (j, R, η), те же действия будут сопровождаться сдвигом вправо.

Например, равенство М (2, 1) = (1, R, 1) означает, что, находясь в состоянии Q2 и обозревая ячейку, в которой написан символ 1, машина должна сохранить в этой ячейке символ 1, сдвинуться вправо и перейти в состояние Q1. Если же М (i, ε) не определено, то машина, находясь в состоянии Qi и обозревая ячейку с символом ε, прекращает свою работу, не изменяя своего состояния, информации на ленте и никуда не сдвигаясь.

Замечание. Существуют различные модификации машины Тьюринга (машина Поста, машина Минского и т.д.). Некоторые модификации предусматривают на ленте не символы 0 или 1, а буквы некоторого конечного алфавита А = {a1, a2,..., am}. В некоторых определениях разрешается не только сдвиг головки машины вправо или влево, но и оставление на прежней позиции. Однако различные модификации машины Тьюринга эквивалентны между собой в том смысле, что классы функций, вычислимых на этих машинах, совпадают.

Структура машины Тьюринга

В качестве исполнителя алгоритмов Тьюрингом был предложен

автомат, состоящий:

· из бесконечной ленты, разбитой на ячейки;

· головки (каретки), способной передвигаться над лентой, от

ячейки к ячейке, считывать символы, записанные на ленте, записывать

символы в ячейки.

В каждой ячейке ленты может быть записан только один из определенного множества символов, называемого алфавитом. За одно срабатывание каретка способна выполнить следующие действия:

· считать символ из ячейки, над которой она находится;

· записать символ в ячейку, над которой она находится;

· переместиться либо влево, либо вправо на следующую ячейку,

либо остаться на месте.

· изменять свое внутреннее состояние.

Предполагается, что каретка может находиться в одном из состояний определенного множества состояний. Одним из ее действий, наряду с перечисленными выше, является переход из одного состояния в другое.

Машина Тьюринга работает следующим образом:

1) на ленту записывается исходная информация, и головка устанавливается в исходное положение. Затем через логический блок считывается информация с ленты, и по закону функционирования машины определяется символ, который нужно записать на ленту;

2) на пересечении строки qi и столбца aj указывается тройка символов

Композиция машин Тьюринга

Написать программу работы машины Тьюринга - значит составить диаграмму или таблицу ее функционирования. Программирование машины Тьюринга состоит в умении разделить процедуру решения задачи на последовательность трех шагов: запись символа, считывание символа, перемещение головки на одну позицию.

Если требуется написать достаточно сложную программу, то для упрощения процесса программирования используются операции композиции машин Тьюринга, т.е. построение сложной машины из более простых. Для этого выполняются две операции: умножения машин Тьюринга и итерации машин Тьюринга.

Умножение машин Тьюринга.

Пусть заданы две машины Тьюринга T1 и T2. В начальный момент времени t0 на ленте имеется некоторая конфигурация, которую начинает обрабатывать машина T1, отправляясь от некоторого начального состояния , причем головка машины в момент t0 находится напротив ячейки с номеромТогда к некоторому моменту машина T1 перейдет в состояние, а головка машины остановится напротив ячейки . В момент времени t машину T1 отключаем, и с этого момента начинает работать машина T2, отправляясь

от своего начального состояния . Причем в момент времени t1 головка будет расположена напротив ячейки

В момент времени t машина T2 закончит работу и остановится напротив ячейки . видно, что результатом последующей работы машин T1 и T2 будет некоторая машина T, функциональная таблица которой строится по правилам: верхняя часть таблицы описывает машину T1, а нижняя часть – работу машины T2, причем состояние останова машины T1 отождествляется с начальным состоянием машины T2.

Машина Поста. Конструктивно к машине Тьюринга близка машина Поста. Принципиально она отличается двоичным алфавитом входных и выходных данных (в машине Тьюринга алфавит не определен, а выбирается).

В машине Тьюринга следует сконструировать управляющий автомат, а в машине Поста - программу решения задачи. Как следствие -универсальность и простота машины Поста.

Система команд машины Поста:

· Shr - движение вправо к соседней позиции;

· Shl - движение влево к соседней позиции;

· Wr1 - читать текущую ячейку: если в ячейке символ 0, то запи-

сать 1; если в ячейке символ 1, то останов неприменимости;

· Wr0 - читать текущую ячейку: если в ячейке символ 1, то запи-

сать 0; если в ячейке символ 0, то останов неприменимости;

· Jmp j0, j1 - читать текущую ячейку: если в ячейке символ 0, то

перейти к команде j0; если в ячейке символ 1, то перейти к команде j1;

· Stop - останов.

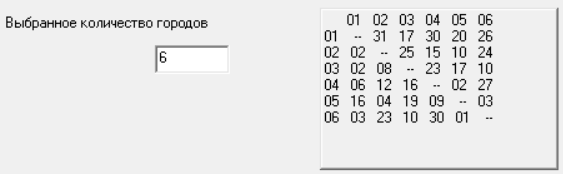
Упорядоченный набор команд (программа) применим к данной проблеме и является алгоритмом ее решения, если применение программы завершится командой Stор. Если произойдет останов по неприменимости, то программа не применима к данной проблеме и проблема алгоритмически не разрешима.

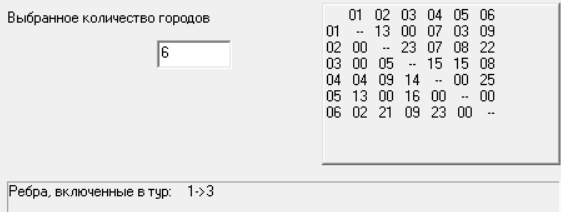
В теории вычислительных машин машина Поста представляет собой минимальную универсальную и полную систему команд и элементарную конструкцию.

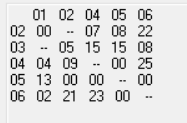
**Задание 2.2**

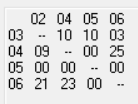
Решить задачу коммивояжера для трех графов, содержащих по 6 вершин.

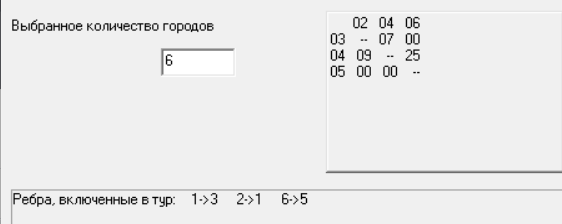
1 Граф

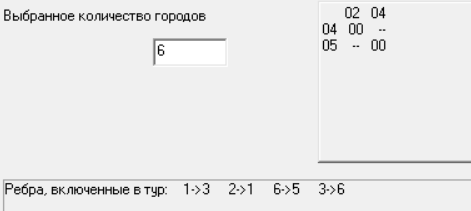


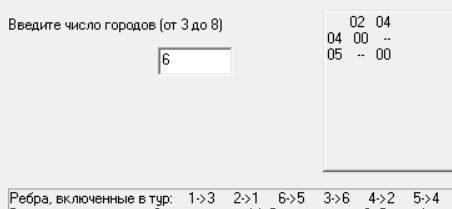




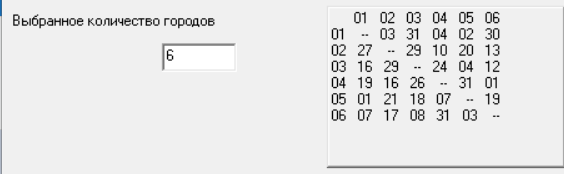


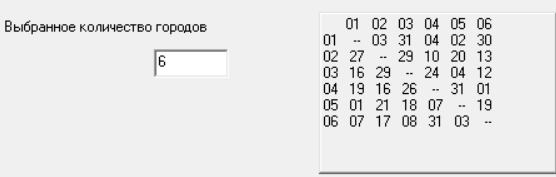


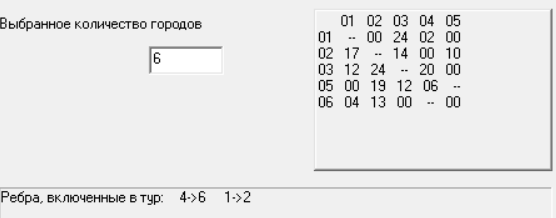


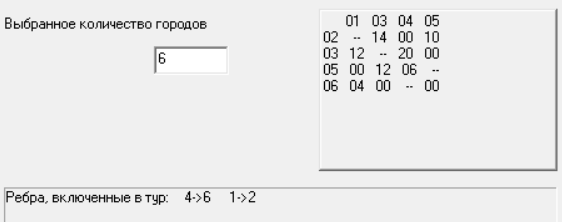


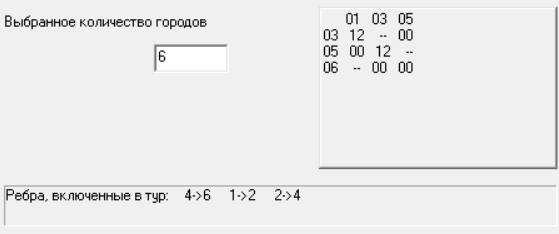
2 Граф

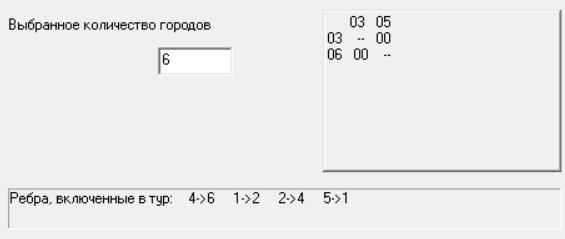




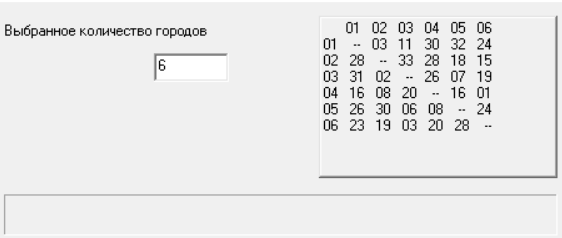


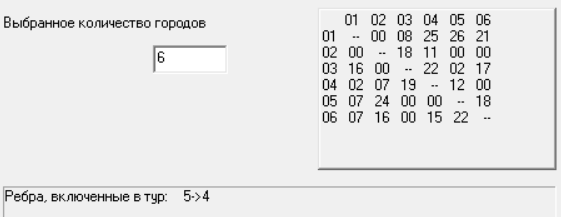


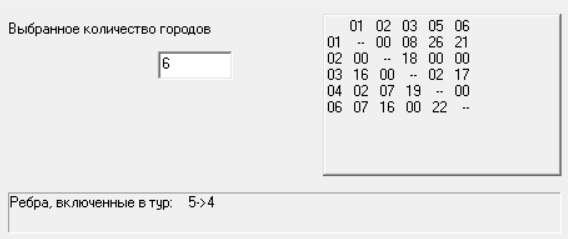


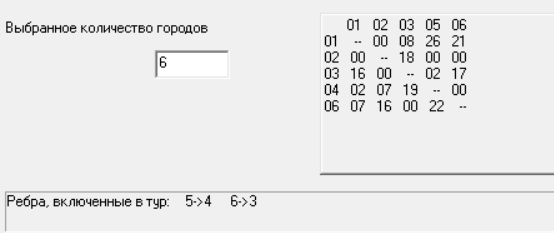


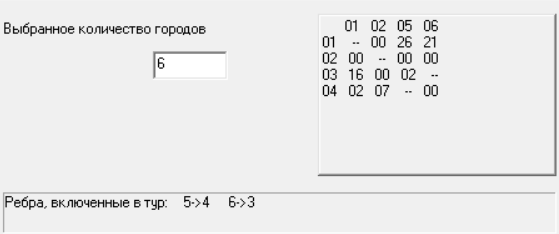
3 Граф

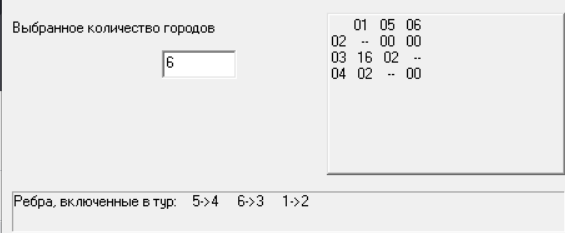


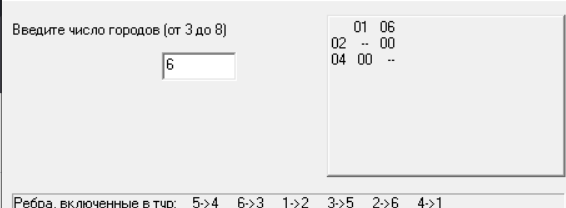






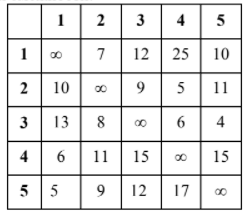






**Задание 2.3**

Составить программу решения задачи коммивояжера для графа, заданного матрицей смежности:



#include <iostream>

#include <list>

#include <vector>

#define TEMPSIZE 5

using namespace std;

const int inf = INT\_MAX;

struct Penalty

{

int value;

int i;

int j;

};

struct Matrix

{

int size;

int\*\* elem;

vector<int> trueI;

vector<int> trueJ;

};

class Node

{

public:

int a;

int b;

int weight;

list<int> way;

Matrix matrix;

Node(int a, int b, int weight, list<int> way, Matrix matrix)

{

this->a = a;

this->b = b;

this->weight = weight;

//this->way.swap(way);

this->way.clear();

for (list<int>::iterator i = way.begin(); i != way.end(); ++i)

{

this->way.push\_back(\*i);

}

this->matrix.size = matrix.size;

this->matrix.trueI.clear();

this->matrix.trueJ.clear();

for (vector<int>::iterator i = matrix.trueI.begin(); i != matrix.trueI.end(); ++i)

{

this->matrix.trueI.push\_back(\*i);

}

for (vector<int>::iterator i = matrix.trueJ.begin(); i != matrix.trueJ.end(); ++i)

{

this->matrix.trueJ.push\_back(\*i);

}

this->matrix.elem = new int\* [TEMPSIZE];

for (int i = 0; i < TEMPSIZE; i++)

{

this->matrix.elem[i] = new int[TEMPSIZE];

for (int j = 0; j < TEMPSIZE; j++)

{

this->matrix.elem[i][j] = matrix.elem[i][j];

}

}

}

};

void showMatrix(Matrix matrix)

{

for (int i = 0; i < TEMPSIZE; i++)

{

for (int j = 0; j < TEMPSIZE; j++)

{

if (matrix.elem[i][j] != inf)

{

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

}

else

{

cout << "inf\t";

}

}

cout << '\n';

}

}

int\* reductByRow(Matrix& matrix)

{

int\* hi = new int[matrix.size];

for (int i = 0; i < matrix.size; i++)

{

int minElem = INT\_MAX;

for (int j = 0; j < matrix.size; j++)

{

if (matrix.elem[matrix.trueI[i]][matrix.trueJ[j]] < minElem)

{

minElem = matrix.elem[matrix.trueI[i]][matrix.trueJ[j]];

}

}

hi[i] = minElem;

for (int j = 0; j < matrix.size; j++)

{

if (matrix.elem[matrix.trueI[i]][matrix.trueJ[j]] != inf)

{

matrix.elem[matrix.trueI[i]][matrix.trueJ[j]] -= hi[i];

}

}

}

return hi;

}

int\* reductByColumn(Matrix& matrix)

{

int\* hj = new int[matrix.size];

for (int j = 0; j < matrix.size; j++)

{

int minElem = INT\_MAX;

for (int i = 0; i < matrix.size; i++)

{

if (matrix.elem[matrix.trueI[i]][matrix.trueJ[j]] < minElem)

{

minElem = matrix.elem[matrix.trueI[i]][matrix.trueJ[j]];

}

}

hj[j] = minElem;

for (int i = 0; i < matrix.size; i++)

{

if (matrix.elem[matrix.trueI[i]][matrix.trueJ[j]] != inf)

{

matrix.elem[matrix.trueI[i]][matrix.trueJ[j]] -= hj[j];

}

}

}

return hj;

}

Penalty penaltyFunction(Matrix matrix)

{

list<Penalty> penalties;

for (int i = 0; i < matrix.size; i++)

{

for (int j = 0; j < matrix.size; j++)

{

Penalty D;

if (matrix.elem[matrix.trueI[i]][matrix.trueJ[j]] == 0)

{

D.value = 0;

int min = INT\_MAX;

for (int ki = 0; ki < matrix.size; ki++)

{

if (ki != i)

{

if (matrix.elem[matrix.trueI[ki]][matrix.trueJ[j]] < min)

{

min = matrix.elem[matrix.trueI[ki]][matrix.trueJ[j]];

}

}

}

D.value += min;

min = INT\_MAX;

for (int kj = 0; kj < matrix.size; kj++)

{

if (kj != j)

{

if (matrix.elem[matrix.trueI[i]][matrix.trueJ[kj]] < min)

{

min = matrix.elem[matrix.trueI[i]][matrix.trueJ[kj]];

}

}

}

D.value += min;

D.i = matrix.trueI[i];

D.j = matrix.trueJ[j];

penalties.push\_back(D);

}

}

}

Penalty maxD;

maxD.value = -1000;

for (list<Penalty>::iterator k = penalties.begin(); k != penalties.end(); ++k)

{

if ((\*k).value > maxD.value)

{

maxD.i = (\*k).i;

maxD.j = (\*k).j;

maxD.value = (\*k).value;

}

}

return maxD;

}

int firstElem(Matrix& matrix)

{

int R = 0;

int\* hi = new int[matrix.size];

int\* hj = new int[matrix.size];

hi = reductByRow(matrix);

hj = reductByColumn(matrix);

for (int k = 0; k < matrix.size; k++)

{

R += hi[k];

R += hj[k];

}

return R;

}

Node currNode(list<Node>& nodes)

{

Node currentNode = nodes.back();

list<Node>::iterator currIter = nodes.end();

for (list<Node>::iterator i = nodes.begin(); i != nodes.end(); ++i)

{

if ((\*i).weight <= currentNode.weight)

{

currentNode = (\*i);

currIter = i;

}

}

nodes.erase(currIter);

return currentNode;

}

Node currNode(list<Node>& nodes, bool e)

{

Node currentNode = nodes.back();

list<Node>::iterator currIter = nodes.end();

for (list<Node>::iterator i = nodes.begin(); i != nodes.end(); ++i)

{

if ((\*i).weight <= currentNode.weight)

{

currentNode = (\*i);

currIter = i;

}

}

return currentNode;

}

void matrixResize(int di, int dj, Node& currentNode)

{

vector<int>::iterator k;

for (vector<int>::iterator i = currentNode.matrix.trueI.begin(); i != currentNode.matrix.trueI.end(); ++i)

{

if ((\*i) == di)

{

k = i;

break;

}

}

currentNode.matrix.trueI.erase(k);

for (vector<int>::iterator j = currentNode.matrix.trueJ.begin(); j != currentNode.matrix.trueJ.end(); ++j)

{

if ((\*j) == dj)

{

k = j;

break;

}

}

currentNode.matrix.trueJ.erase(k);

currentNode.matrix.size--;

currentNode.matrix.elem[dj][di] = inf;

}

void newNodes(list<Node>& nodes)

{

Node currentNode = currNode(nodes);

//work with left node

Penalty D;

D = penaltyFunction(currentNode.matrix);

list<int> tempWay = currentNode.way;

tempWay.push\_back(D.i);

tempWay.push\_back(D.j);

Node tempNode = Node(D.i, D.j, currentNode.weight + D.value, tempWay, currentNode.matrix);

nodes.push\_back(tempNode);

matrixResize(D.i, D.j, currentNode);

//work with right node

int H = 0;

int\* hi = new int[currentNode.matrix.size];

int\* hj = new int[currentNode.matrix.size];

hi = reductByRow(currentNode.matrix);

hj = reductByColumn(currentNode.matrix);

for (int k = 0; k < currentNode.matrix.size; k++)

{

H += hi[k];

H += hj[k];

}

tempWay.clear();

tempWay = currentNode.way;

tempWay.push\_back(D.i);

tempWay.push\_back(D.j);

tempNode = Node(D.i, D.j, currentNode.weight + H, tempWay, currentNode.matrix);

nodes.push\_back(tempNode);

}

int main()

{

Matrix matrix;

int size = 5;

matrix.size = size;

matrix.elem = new int\* [size];

matrix.trueI = vector<int>(size);

matrix.trueJ = vector<int>(size);

for (int k = 0; k < size; k++)

{

matrix.elem[k] = new int[size];

matrix.trueI[k] = k;

matrix.trueJ[k] = k;

}

int tempMatrix[5][5] =

{

{inf, 7, 12, 25, 10},

{10, inf, 9, 5, 11,},

{13, 8, inf, 6, 4, },

{6, 11, 15, inf, 15},

{5, 9, 12, 17, inf,}

};

for (int i = 0; i < matrix.size; i++)

{

for (int j = 0; j < matrix.size; j++)

{

matrix.elem[matrix.trueI[i]][matrix.trueJ[j]] = tempMatrix[i][j];

}

}

showMatrix(matrix); cout << endl;

int R = firstElem(matrix);

list<int> tempList;

tempList.push\_back(-8);

list<Node> nodes;

Node tempNode = Node(-8, -8, R, tempList, matrix);

nodes.push\_back(tempNode);

while (currNode(nodes, 1).matrix.size != 2)

{

newNodes(nodes);

}

Node finNode = currNode(nodes, 1);

int finWay = -7;

cout << "Way:: ";

for (list<int>::iterator i = finNode.way.begin(); i != finNode.way.end(); ++i)

{

if (finWay != \*i + 1)

{

finWay = \*i + 1;

cout << finWay << " ";

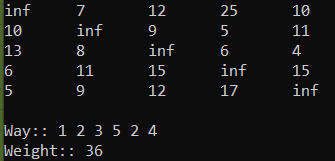
}

}

cout << endl << "Weight:: " << finNode.weight;

return 0;

}

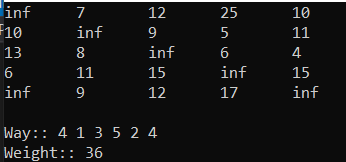


**Задание 2.4**

Решить задачу коммивояжера для задач, представленных в следующем разделе.



Аналогичный код, только удалим одно ребро.



**Задание 5.2**

