ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И.Ульянова (Ленина)»

Кафедра МОЭВМ

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3**

**на тему:**

**«Метод ветвей и границ»**

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

Выполнила: студентка группы №1381

Лавринович Т. В.

Проверил: Казаков Б.Б.

Санкт-Петербург

2013 г.

**Цель работы**

Изучить метод ветвей и границ, принцип его работы. Составить программу с использованием данного метода.

**Задание**

Вариант 1а (задача о назначениях)

Имеется n человек, которых нужно назначить на n работ. Стоимость назначения i-ого человека на j-ую работу равна Cij. Задача состоит в отыскании назначения, при котором каждая работа выполняется некоторым человеком и которое минимизирует общую стоимость назначения.

**Описание алгоритма**

Шаг1. Инициализируем граф. Для каждого разработчика рассчитаем его максимальную «способность» по знакомым ему технологиям, и присвоим ему это число. О задачах пока ничего неизвестно, поэтому их «изученность» инициализируем нулями.

Шаг 2. При поиске «свободной задачи» для «незадействованного разработчика» мы ограничимся теперь только оптимальными ребрами графа, т.е. теми, для которых выполняется равенство: эффективность решения задачи (ребро) = способность разработчика (вершина) + изученность задачи (вершина). Берем по очереди незадействованных разработчиков и, подыскивая им свободные задачи, строим альтернирующее дерево (состоящее из чередующихся цепей), но уже только по оптимальным ребрам. Далее возможны 2 ситуации:

a) Удалось обнаружить свободную задачу. «Переназначаем» задачи, наращиваем паросочетание. Начинаем строить альтернирующее дерево заново, т.к. возможна ситуация, когда граф изменился.

б) Мы не нашли свободную задачу по оптимальным ребрам. А она есть, т.к. начинали ведь мы со свободного разработчика, а в графе у нас одинаковое количество задач и разработчиков. В данном случае нам нужно будет немного понизить наши требования к разработчикам и начать поиски заново.

**Контрольные примеры**

Таблица 1. Контрольные примеры

|  |  |  |
| --- | --- | --- |
| Исходные данные | | Результат |
| Размер матрицы (n) | Матрица(n\*n) | Назначение (i-ого человека на j-ую работу) |
| 2 | 1 5  6 3 | 1 – 1  2 – 2 |
| 3 | 10 20 7  5 30 11  50 10 8 | 1 – 3  2 – 1  3 – 2 |
| 4 | 1 5 6 7  8 2 9 10  11 12 3 13  14 15 16 4 | 1 – 1  2 – 2  3 – 3  4 ­­­­­– 4 |

**Исходная программа**

#include <iostream>

#include <vector>

#include <conio.h>

using namespace std;

int n;

vector < vector<int> > a; // Матрица эффективности a[разраб][работа]

vector<int> xy, yx; // Паросочетания: xy[разраб], yx[работа]

vector<char> vx, vy; // Альтернирующее дерево vx[разраб], vy[работа]

vector<int> minrow, mincol; // Стоимость назначения, выполненность

bool dotry (int i) { //Функция нахождения свободных задач по //оптимальным ребрам

if (vx[i]) return false;

vx[i] = true;

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

if (a[i][j]-minrow[i]-mincol[j] == 0)

vy[j] = true;

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

if (a[i][j]-minrow[i]-mincol[j] == 0 && yx[j] == -1) {

xy[i] = j;

yx[j] = i;

return true;

}

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

if (a[i][j]-minrow[i]-mincol[j] == 0 && dotry (yx[j])) {

xy[i] = j;

yx[j] = i;

return true;

}

return false;

}

int main() {

//------------Вводим размер и саму матрицу разработчиков и работ

cin>>n;

a.resize(n);

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

{

a[i].resize(n);

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

cin>>a[i][j];

}

//------------Изначально примем, что работы еще не изучена

mincol.assign (n, 0);

minrow.assign (n, 0);

for (int i=0; i<n; ++i) //и находим минимальную стоимость

// назначения по разработчику

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

minrow[i] = min (minrow[i], a[i][j]);

xy.assign (n, -1);

yx.assign (n, -1);

for (int c=0; c<n; ) {

vx.assign (n, 0);

vy.assign (n, 0);

int k = 0;

for (int i=0; i<n; ++i) //Пока не всем разработчикам нашли задачи.

if (xy[i] == -1 && dotry (i)) //Пока удается находить

//свободные задачи по оптимальным ребрам

++k;//«Переназначаем» задачи, увеличивая паросочетания

c += k;

if (k == 0) {//Не достигли свободной задачи. Венгерское дерево.

int z = INT\_MAX;//Понижаем способности разработчиков на min величину

for (int i=0; i<n; ++i) //Далее в цикле мы пробежим по «границе» дерева: по тем ребрам,

if (vx[i]) //которые соединяют незадействованных разработчиков или разработчиков,

for (int j=0; j<n; ++j) //достижимых из них со смежными задачами(по неоптимальным ребрам).

if (!vy[j])

z = min (z,a[i][j] - minrow[i] - mincol[j]); //и вычисляем минимальное на текущий момент «несоответствие»

//способностей разработчика

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

//разработчиков на z, чтобы присоеденить хотя бы одно

//ребро к альтернирующему дереву,

if (vx[i]) minrow[i] -= z; //по которому будем

//продолжать поиск оптимальной задачи

if (vy[i]) mincol[i] += z; //Повысим изученночть

//работ на z, чтобы альтернирнирующее дерево оставалось оптимальным

}

}

}

int ans = 0;

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

ans += a[i][xy[i]];

printf ("%d\n", ans);

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

{

cout<<i+1<<" - ";

printf ("%d ", xy[i]+1);

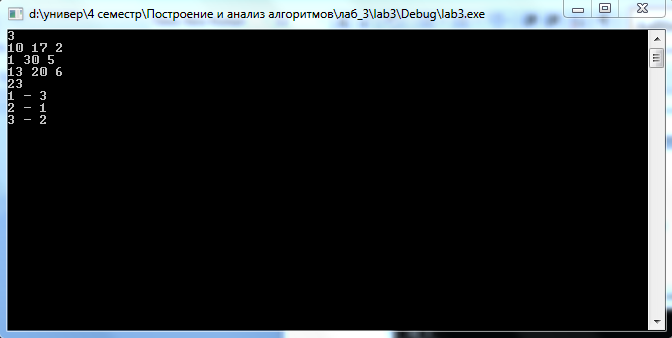
cout<<"\n";

}

\_getch();

}

**Результаты выполнения программы**

****

При выполнении программы полученные результаты совпадают с данными Таблицы 1. Ошибки не обнаружены.

**Вывод**

При выполнении лабораторной работы №3 «Метод ветвей и границ» был составлен и реализован алгоритм для поставленной задачи.