Министерство образования и науки Российской Федерации Федеральное агентство по образованию Федеральное государственное бюджетное образовательное учреждение высшего образования «Вятский государственный университет»

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Лабораторная работа №3

по курсу «Исследование операций»

**Метод отсечений**

Выполнил студент группы ИВТ-31\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Птахова А.М/

Проверил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Коржавина А.С./

Киров 2022

1. Цель

Закрепить на практике знания о методе отсечений (Гомори) решения задач целочисленного программирования и получить навыки его программной реализации.

2. Задание

Реализовать метод Гомори решения задачи целочисленного программирования.

3. Описание метода

1. Решить исходную производственную задачу обычным симплекс-методом;
2. Убедиться, что ответ получился нецелочисленным. Если он целочисленный, то задача решена;
3. Умножить значения в последней строке (строка F) на -1;
4. Пока в ответе остались нецелочисленные переменные, делать следующее:
   1. Среди нецелочисленных значений в получившемся решении выбрать переменную, для которой необходимо составить дополнительное ограничение (как это сделать будет объяснено в нижеприведенном примере);
   2. К исходной задаче добавить специальным образом составленное ограничение для выбранной переменной (опять же, в примере будет показано, как именно это сделать). Это приведет к появлению еще одной вспомогательной переменной;
   3. К полученной задаче применить один этап симплекс-метода, причем убрать появившуюся вспомогательную переменную, и добавить какую-то из исходных (какую именно также будет объяснено в примере).
5. Предыдущий шаг (шаг 4) необходимо выполнять, пока все решение не станет целочисленным. Метод Гомори гарантирует, что на каком-либо шаге это точно произойдет.

4. Листинг кода

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

void input(vector <vector <double>>& mas, int s, int m)

{

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

{

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

{

if (i == m - 1)

{

mas[i][j] = 0;

}

else

{

if (j == 0)

{

mas[i][j] = 0;

}

else

{

cout << "Введите а[" << i << "," << j << "]: ";

cin >> mas[i][j];

}

}

}

}

}

void output(vector <vector <double>>& mas, int s, int m)

{

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

{

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

{

cout << mas[i][j] << ' ';

}

cout << '\n';

}

}

void bazic(vector <vector <double>>& mas, int s, int m)

{

int k = 0;

int ns;

int nr;

for (int j = 1; j < s - 1; j++)

{

for (int i = 1; i < m - 1; i++)

{

//cout << mas[i][j]<<' ';

if (abs(mas[i][j]) > 1)

{

//cout << "break" << mas[i][j] << '\n';

k = 0;

break;

}

if (mas[i][j] == 1)

{

//cout << "str: " << i << " row: " << j;

k++;

ns = i;

nr = j;

}

}

if (k == 1)

{

//cout << "str: " << ns << " row: " << nr<<'\n';

k = 0;

mas[ns][0] = mas[0][nr];

}

}

//cout << '\n';

}

void delta(vector <vector <double>>& mas, int s, int m, vector <double>& sum)

{

for (int j = 1; j < s; j++)

{

for (int i = 1; i < m - 1; i++)

{

sum[j] = sum[j] + mas[i][j] \* mas[i][0];

}

sum[j] = sum[j] - mas[0][j];

}

}

void zapic\_delta(vector <vector <double>>& mas, int s, int m, vector <double>& sum)

{

for (int j = 1; j < s; j++)

{

mas[m - 1][j] = sum[j];

}

}

int find\_opor(vector <vector <double>>& mas, int s, int m, int& ns, int& nr)

{

double min = 1000;

//int nr;

for (int j = 1; j < s - 1; j++)

{

if (mas[m - 1][j] < min && mas[m - 1][j] != 0)

{

min = mas[m - 1][j];

nr = j;

}

}

//cout << nr << '\n';

double min2 = 1000;

//int ns;

int c = 0;

double z;

for (int i = 1; i < m - 1; i++)

{

cout << mas[i][s - 1] << ' ';

cout << mas[i][nr] << '\n';

if (mas[i][nr] > 0)

{

z = mas[i][s - 1] / mas[i][nr];

if (z < min2)

{

min2 = z;

ns = i;

}

}

else

{

c++;

}

}

//cout <<"min2: "<< min2 << '\n';

//cout <<"str: " << ns << '\n';

return c;

}

void divided(vector <vector <double>>& mas, int str, int row, int m)

{

double l = mas[str][row];

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

{

mas[str][j] = mas[str][j] / l;

}

}

void guass(vector <vector <double>>& mas, int s, int m, int str, int row)

{

double p = 0;

for (int i = 1; i < m - 1; i++)

{

p = mas[i][row];

if (i != str)

{

for (int j = 1; j < s; j++)

{

mas[i][j] = mas[i][j] - mas[str][j] \* p;

}

}

}

}

int check(vector <vector <double>>& mas, int s, int m)

{

int kpol = 0;

for (int j = 1; j < s - 1; j++)

{

if (mas[m - 1][j] >= 0)

{

cout << mas[m - 1][j] << ' ';

kpol++;

}

}

return kpol;

}

void mov(vector <vector <double>>& mas1, vector <vector<double>>& mas, int s, int d, double koef)

{

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

{

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

{

if (i < d - 2 && j < s - 2)

{

mas1[i][j] = mas[i][j];

}

if (i == d - 2 && j < s - 2)

{

if (mas1[i - 1][j] != 0 && mas1[i - 1][j] != 1 && j != 0)

{

mas1[i][j] = -mas[i - 1][j];

}

}

if (i == d - 2 && j == s - 2)

{

mas1[i][j] = 1;

}

}

}

}

void mov1(vector <vector <double>>& mas1, vector <vector<double>>& mas, int s, int d, int &ns)

{

double intpart;

double factpart;

cout << ns << '\n';

cout << "\n";

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

{

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

{

factpart = modf(mas1[ns][j], &intpart);

if (i < d - 2 && j < s - 2)

{

mas1[i][j] = mas[i][j];

}

if (i == d - 2 && j < s - 2)

{

if (mas1[ns][j] >= 0)

{

if (mas1[ns][j] == 0)

{

mas1[i][j] = 0;

}

else

{

mas1[i][j] = -mas1[ns][j] + intpart;

}

}

else

{

intpart = intpart - 1;

mas1[i][j] = intpart-mas[ns][j];

}

}

if (i == d - 2 && j == s - 2)

{

mas1[i][j] = 1;

}

if (j == s - 1 && i < d - 1)

{

if (i == d - 2)

{

mas1[i][j] = -factpart;

}

else

{

mas1[i][j] = mas[i][j - 1];

}

}

}

}

}

int opor(vector<vector <double>>& mas, int s, int d, int& nr, int& ns)

{

int c = 0;

for (int i = 0; i < d - 1; i++)

{

if (mas[i][s - 1] < 0)

{

ns = i;

c = 1;

}

}

if (c != 0)

{

double min = 0;

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

{

if (mas[ns][j] < min)

{

min = mas[ns][j];

nr = j;

}

}

if (min == 0)

{

c = 2;

}

}

return c;

}

int check2(vector <vector <double>>& mas, int s, int d)

{

int kpol = 0;

double factpart, intpart;

for (int i = 1; i < d - 1; i++)

{

factpart = modf(mas[i][s - 1], &intpart);

if (factpart != 0.0)

{

cout << mas[i][s-1] << ' ';

kpol++;

}

}

return kpol;

}

int main()

{

setlocale(LC\_ALL, "rus");

int n;

cout << "Введите кол-во производимых товаров: ";

cin >> n;

int m;

cout << "Введите кол-во ограничений: ";

cin >> m;

int d = m + 2;//count string on the table

int s = n + m + 2; //count row

int w = s - 2;

vector <vector <double>> mas(d, vector<double>(s)); //массив на d строк, s столбцов

vector <double> sum(s);

input(mas, s, d); //ввели матрицу

bazic(mas, s, d);

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

{

sum[i] = 0;

}

delta(mas, s, d, sum); // посчитали дельты

zapic\_delta(mas, s, d, sum); // записали дельты

int k = 0;

int ns;

int nr;

int kpol = check(mas, s, d);

//cout << kpol << '\n';

while (kpol != w)

{

int c = find\_opor(mas, s, d, ns, nr); // нашли опорный

if (c == m)

{

cout << "Решений нет";

break;

}

else

{

divided(mas, ns, nr, s); //разделили строку на опорный

guass(mas, s, d, ns, nr); //недопреобразовани¤ гаусса

bazic(mas, s, d); // определили базиc

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

{

sum[i] = 0;

}

delta(mas, s, d, sum); // посчитали дельты

zapic\_delta(mas, s, d, sum); // записали дельты

kpol = check(mas, s, d);

}

}

cout << "\n ";

double max = 0.0;

double fractpart, intpart;

for (int i = 1; i < 3; i++)

{

fractpart = modf(mas[i][s - 1], &intpart);

if (fractpart>max)

{

//cout << fractpart;

max = fractpart;

ns = i;

}

}

cout << '\n';

cout << "x1= " << mas[1][s - 1] << '\n';

cout << "x2= " << mas[2][s - 1] << '\n';

cout << "F= " << mas[d - 1][s - 1] << '\n';

//cout << ns << '\n';

d++;

s++;

vector < vector <double>> mas1(d, vector <double>(s));

cout << '\n';

vector <double> sum1(s);

mov1(mas1, mas, s, d, ns);

ns = 0;

bazic(mas1, s, d);

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

{

sum1[i] = 0;

}

delta(mas1, s, d, sum1); // посчитали дельты

zapic\_delta(mas1, s, d, sum1); // записали дельты

output(mas1, s, d); // вывести матрицу

int ko = opor(mas1, s, d, nr, ns);

cout << ns << ' ' << nr<<' '<<mas1[ns][nr]<<'\n';

divided(mas1, ns, nr, s); //разделили строку на опорный

guass(mas1, s, d, ns, nr); //недопреобразовани¤ гаусса

bazic(mas1, s, d); // определили базиc

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

{

sum1[i] = 0;

}

delta(mas1, s, d, sum1); // посчитали дельты

zapic\_delta(mas1, s, d, sum1); // записали дельты

output(mas1, s, d); // вывести матрицу

ko = opor(mas1, s, d, nr, ns);

int y = check2(mas1, s, d);

cout << '\n';

cout << y << '\n';

cout << '\n';

mas.assign(d, vector <double>(s));

mas = mas1;

int o = 0;

while (o != 1)

{

d++;

s++;

mas1.assign(d, vector<double>(s));

sum1.assign(s, 0);

mov1(mas1, mas, s, d, ns);

ns = 0;

bazic(mas1, s, d);

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

{

sum1[i] = 0;

}

delta(mas1, s, d, sum1); // посчитали дельты

zapic\_delta(mas1, s, d, sum1); // записали дельты

output(mas1, s, d); // вывести матрицу

ko = opor(mas1, s, d, nr, ns);

y = check2(mas1, s, d);

cout << '\n';

cout << y << '\n';

cout << '\n';

cout << ns << ' ' << nr << ' ' << mas1[ns][nr] << '\n';

divided(mas1, ns, nr, s); //разделили строку на опорный

guass(mas1, s, d, ns, nr); //недопреобразовани¤ гаусса

bazic(mas1, s, d); // определили базиc

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

{

sum1[i] = 0;

}

delta(mas1, s, d, sum1); // посчитали дельты

zapic\_delta(mas1, s, d, sum1); // записали дельты

ko = opor(mas1, s, d, nr, ns);

mas.assign(d, vector <double>(s));

mas = mas1;

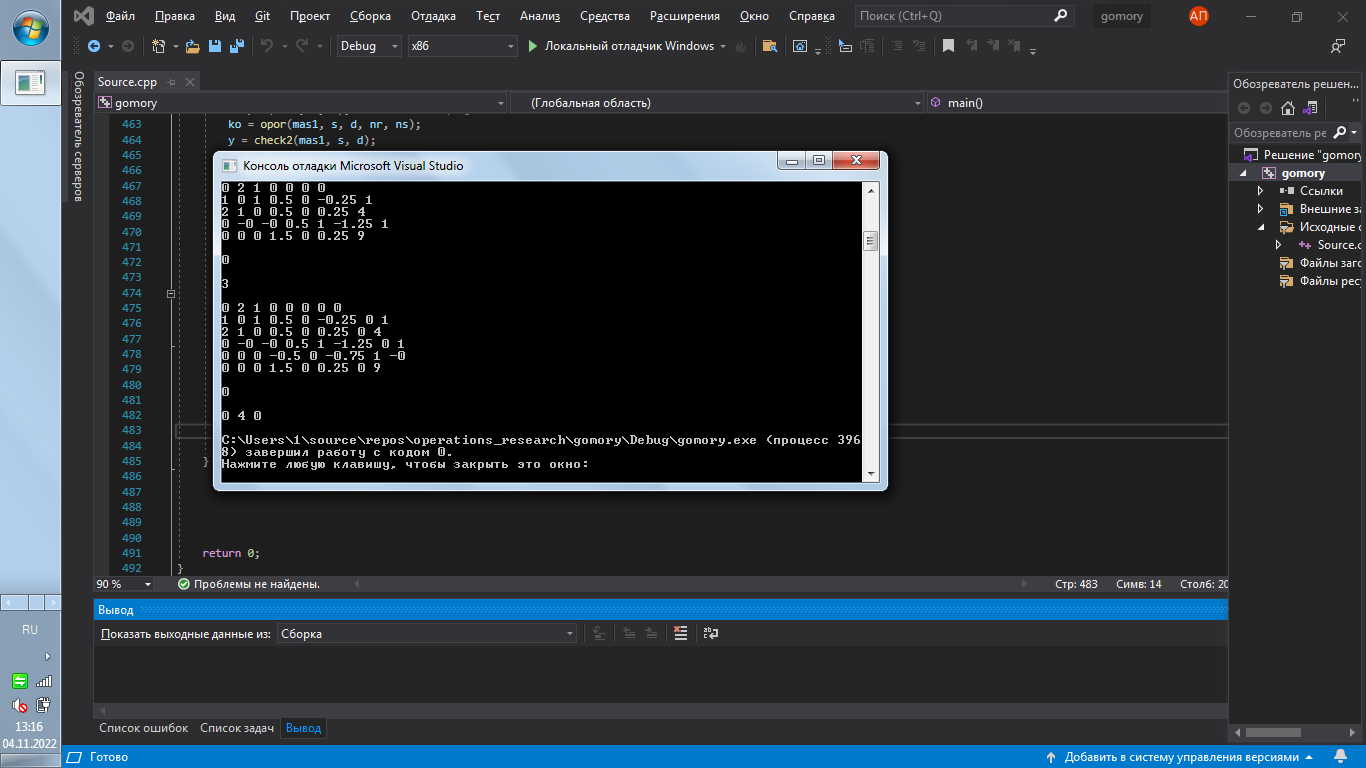
o++;

}

return 0;

}

5. Экранные формы



6. Выводы

В ходе выполнения лабораторной работы были получены навыки решения задач целочисленного программирования методом Гомори и реализации их на языке высокого уровня.