МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Вятский государственный университет»

(ФГБОУ ВО «ВятГУ»)

Факультет автоматики и вычислительной техники

Кафедра ЭВМ

Отчёт

Лабораторная работа № 2 по дисциплине

«Исследование операций»

Выполнил студент группы ИВТб-2301-04-00 / Жеребцов К. А.

Проверил преподаватель / Коржавина А. С.

Киров 2022

1. Цель работы закрепить на практике знания о методе ветвей и границ решения задач целочисленного программирования и получить навыки его программной реализации.
2. Задание: реализовать метод ветвей и границ решения задачи целочисленного программирования.
3. Описание метода ветвей и границ:

Согласно алгоритму решения задачи целочисленного программирования методом ветвей и границ, на каждой *p*-й итерации требуется сделать 4 шага.

* **Шаг 1.** Если в списке решаемых задач нет ни одной задачи, то задача целочисленного программирования решена. Максимальное значение значение функции цели - то, которое было найдено на предыдущей итерации, оптимальный план - целочисленный план, найденный на предыдущей итерации. В противном случае следует выбрать одну из задач, имеющихся в списке.
* **Шаг 2.** Решается выбранная из списка задача линейного программирования. Если задача не имеет решения или для полученного на этом шаге оптимального плана https://function-x.ru/linprog/cp036.gif значение функции цели https://function-x.ru/linprog/cp082.gif, то следует принять https://function-x.ru/linprog/cp083.gif и выполнить шаг 1. В противном случае выполнять шаг 3.
* **Шаг 3.** Если найденный оптимальный план https://function-x.ru/linprog/cp036.gif является целочисленным, то следует принять, что https://function-x.ru/linprog/cp084.gif и выполнить шаг 1. В противном случае выполнить шаг 4.
* **Шаг 4.** Выбрать любую дробную координату оптимального плана https://function-x.ru/linprog/cp036.gif. Определить целую часть координаты, составить две новые задачи линейного программирования и включить их в список решаемых задач. Новые задачи отличаются от задачи, выбранной на шаге 1 только границами допустимых значений выбранной координаты. Принять, что https://function-x.ru/linprog/cp037.gif и выполнить шаг 1.

1. Листинг программы:

#include <iostream>

#include <cstdio>

#include <string>

#include "locale.h"

#include "math.h"

int SimplexMethod(int n1, int m1);

FILE\* stream;

float basicMatr[50][50];

float matr[50][50];

float matrResults[50][10];

float matrMatr[20][50][50];

int n, m, nResult;

double basicResult[15];

int matrTree[32][32];

float xx[1][5] = { 0, 0, 0, 0, 0 };

int FF;

int c = 0;

int main()

{

std::cout.precision(2);

setlocale(LC\_ALL, "Russian\_Russia.1251");

int err;

char ScanStr;

err = fopen\_s(&stream, "matrix.txt", "r");

if (err == 0 && stream != NULL)

{

int m1 = 0;

do

{

ScanStr = fgetc(stream);

if ((int)ScanStr != 10)

{

if ((int)ScanStr == 32)

{

m1 = m1 + 1;

}

}

else

{

if (m1 > m)

{

m = m1;

}

m1 = 0;

n = n + 1;

}

} while ((int)ScanStr != -1);

rewind(stream);

int h = NULL;

char bufChar[10] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

int n1 = 0;

m1 = 3;

bool singMoreFound = false;

do

{

ScanStr = fgetc(stream);

if ((int)ScanStr == 32)

{

basicMatr[n1][m1] = std::stod(bufChar);

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

{

bufChar[i] = NULL;

}

m1 = m1 + 1;

h = 0;

}

else if ((int)ScanStr == 10)

{

basicMatr[n1][m1] = std::stod(bufChar);

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

{

bufChar[i] = NULL;

}

if (singMoreFound == true)

{

for (int i = 3; i <= m1; i++)

{

basicMatr[n1][i] = -basicMatr[n1][i];

}

}

singMoreFound = false;

n1 = n1 + 1;

m1 = 3;

h = 0;

}

else if ((int)ScanStr == -1)

{

basicMatr[n1][m1] = std::stod(bufChar);

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

{

bufChar[i] = NULL;

}

if (singMoreFound == true)

{

for (int i = 3; i <= m1; i++)

{

basicMatr[n1][i] = -basicMatr[n1][i];

}

}

singMoreFound = false;

h = 0;

}

else if ((int)ScanStr == 62)

{

singMoreFound = true;

bufChar[0] = 48;

}

else if ((int)ScanStr == 60 || (int)ScanStr == 61)

{

bufChar[0] = 48;

}

else

{

bufChar[h] = ScanStr;

h = h + 1;

}

} while ((int)ScanStr != -1);

fclose(stream);

}

else

{

}

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

{

basicMatr[i][m + 2] = basicMatr[i][m + 3];

basicMatr[i][m + 3] = 0;

}

m = m - 1;

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

{

for (int j = 0; j <= m + 3; j++)

{

matr[i][j] = basicMatr[i][j];

}

}

std::cout << n + 1 << " " << m + 1 << "\n"; //вывод записанных данных

std::cout << std::endl;

int result;

result = SimplexMethod(n, m);

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

{

basicResult[i] = matr[i + 1][2];

}

int n1 = n;

int m1 = m;

if (result == 2) //начало целочисленных

{

bool notIntegerElementFound = false;

bool IntegerTreeFound = true;

nResult = 0;

int XTree = 0;

int YTree = 0;

matrTree[0][0] = 2;

/\*for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

std::cout << "|" << matrTree[i][j] << "\t";

}

std::cout << "\n";

}\*/

for (int i = 0; i <= n1; i++)

{

for (int j = 0; j <= m1 + n1 + 2; j++)

{

matrMatr[nResult][i][j] = basicMatr[i][j];

}

}

do

{

n1 = n1 + 1;

for (int i = 0; i <= 25; i++) //запись матрицы в матрицу для расчета

{

for (int j = 0; j <= 25; j++)

{

matr[i][j] = matrMatr[nResult][i][j];

}

}

for (int i = n + 1; i <= n1 + 2; i++)

{

matr[i - n][0] = i;

}

matr[n1][m1 + 3] = (int)matrResults[nResult][1];

matr[n1][(int)matrResults[nResult][0] + 2] = 1;

nResult = nResult + 1;

for (int i = 0; i <= n1; i++) //запись матрицы в массив матриц

{

for (int j = 0; j <= m1 + n1 + 2; j++)

{

matrMatr[nResult][i][j] = matr[i][j];

}

}

std::cout << std::endl;

std::cout << nResult << ")" << n1 + 1 << " " << m1 + 1 << "\n"; //вывод записанных данных

std::cout << std::endl;

result = SimplexMethod(n1, m1);

if (result == 2)

{

FF = matr[n1 + 1][2];

int v;

for (int i = 0; i <= 3; i++)

{

if (matr[i][0] <= m)

{

v = matr[i][0];

xx[0][v] = matr[i][2];

}

}

}

int sum = 2;

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

{

sum = sum \* 2;

}

if (sum > nResult + 1)

{

XTree = XTree + 1;

}

else

{

XTree = 0;

YTree = YTree + 1;

}

if (result == 2)

{

matrTree[YTree][XTree] = 2;

}

else if (result == 1)

{

matrTree[YTree][XTree] = 1;

}

else

{

matrTree[YTree][XTree] = -1;

}

/\*for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

std::cout << "|" << matrTree[i][j] << "\t";

}

std::cout << "\n";

}\*/

for (int i = 0; i <= 25; i++) //запись матрицы в матрицу для расчета

{

for (int j = 0; j <= 25; j++)

{

matr[i][j] = matrMatr[nResult - 1][i][j];

}

}

for (int i = n + 1; i <= n1 + 2; i++)

{

matr[i - n][0] = i;

}

matr[n1][m1 + 3] = -((int)matrResults[nResult - 1][1] + 1);

matr[n1][(int)matrResults[nResult - 1][0] + 2] = -1;

nResult = nResult + 1;

for (int i = 0; i <= n1; i++) //запись матрицы в массив матриц

{

for (int j = 0; j <= m1 + n1 + 2; j++)

{

matrMatr[nResult][i][j] = matr[i][j];

}

}

std::cout << std::endl;

std::cout << nResult << ")" << n1 + 1 << " " << m1 + 1 << "\n"; //вывод записанных данных

std::cout << std::endl;

result = SimplexMethod(n1, m1);

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

{

sum = sum \* 2;

}

if (sum > nResult + 1)

{

XTree = XTree + 1;

}

else

{

XTree = 0;

YTree = YTree + 1;

}

if (result == 2)

{

matrTree[YTree][XTree] = 2;

}

else if (result == 1)

{

matrTree[YTree][XTree] = 1;

}

else

{

matrTree[YTree][XTree] = -1;

}

/\*for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

std::cout << "|" << matrTree[i][j] << "\t";

}

std::cout << "\n";

}\*/

IntegerTreeFound = false;

for (int i = 0; i < sum / 2; i++)

{

if (matrTree[YTree][i] == 2)

{

IntegerTreeFound = true;

int sum1 = 0;

for (int j = 0; j < sum / 2; j++)

{

sum1 = sum1 + 2;

if (sum1 == i || sum1 - 1 == i)

{

for (int h = 0; h <= n1; h++) //запись матрицы в матрицу для расчета

{

for (int k = 0; k <= m1 + n1 + 2; k++)

{

matr[h][k] = matrMatr[sum / 4 + j - 1][h][k];

}

}

}

}

}

}

} while (IntegerTreeFound == true);

std::cout << std::endl;

//std::cout << "Целочисленные X:";

std::cout << std::endl;

bool fullNumberFound = false;

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

{

if (matrResults[i][0] == 0)

{

std::cout << "F = " << matrResults[i][2] << " ";

for (int j = 3; j <= n + 2; j++)

{

fullNumberFound = true;

std::cout << "X" << j - 2 << " = " << matrResults[i][j] << " ";

}

std::cout << std::endl;

}

}

if (fullNumberFound == false)

{

/\*std::cout << "Целочисленные X не найдено";

std::cout << std::endl;

std::cout << "F = " << basicResult[n] << " ";

std::cout << std::endl;

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

{

std::cout << "X" << i + 1 << "=" << basicResult[i];

std::cout << std::endl;

}\*/

std::cout << "Конечный результат:\n";

std::cout << "\nF = " << FF << " ";

for (int j = 0; j < 5; j++)

{

if (xx[0][j] != 0)

{

std::cout << "X" << j << " = " << floor(xx[0][j]) << " ";

}

}

std::cout << "\n";

}

else

{

int resultString = 0;

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

{

if (matrResults[i][2] > matrResults[resultString][2] && matrResults[i][2] < basicResult[n])

{

resultString = i;

}

}

std::cout << std::endl;

std::cout << "Конечный результат:";

std::cout << std::endl;

std::cout << "F = " << matrResults[resultString][2] << " ";

for (int j = 3; j <= n + 2; j++)

{

std::cout << "X" << j - 2 << " = " << matrResults[resultString][j] << " ";

}

std::cout << std::endl;

std::cout << std::endl;

/\*std::cout << "Дерево:";

std::cout << std::endl;

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

std::cout << "|" << matrTree[i][j] << "\t";

}

std::cout << "\n";

}\*/

}

int a;

std::cin >> a;

}

int a;

std::cin >> a;

return 0;

}

int SimplexMethod(int n1, int m1)

{

for (int i = 0; i <= n1; i++)

{

for (int j = 3; j <= m1 + 3; j++)

{

std::cout << matr[i][j] << "\t";

}

std::cout << std::endl;

}

double buf;

for (int i = 1; i <= n1; i++) //перемещение последнего столбца

{

buf = matr[i][m1 + 3];

matr[i][m1 + 3] = matr[i][2];

matr[i][2] = buf;

}

for (int i = 1; i <= n1; i++) //заполнение единицами

{

matr[i][m1 + i + 2] = 1;

}

std::cout << std::endl;

std::cout << "|БП" << "\t" << "|Сб" << "\t" << "|B" << "\t"; //вывод верхней строки

for (int i = 1; i < m1 + n1 + 1; i++)

{

std::cout << "|x" << i << "(" << matr[0][i + 2] << ")" << "\t";

}

std::cout << std::endl;

for (int i = 1; i <= n1; i++) //вывод матрицы

{

std::cout << "|x";

for (int j = 0; j < m1 + n1 + 3; j++)

{

if (j == 0)

{

std::cout << matr[i][j] << "\t";

}

else

{

std::cout << "|" << matr[i][j] << "\t";

}

}

std::cout << std::endl;

}

std::cout << "|delta" << "\t" << "\t";

//for (int i = 2; i <= m1 + n1 + 2; i++) //вывод индекстной строки

//{

// std::cout << "|" << matr[n1 + 1][i] << "\t";

//}

for (int i = 2; i <= m1 + n1 + 2; i++) //вывод индекстной строки

{

matr[n1 + 1][i] = -matr[0][i];

std::cout << "|" << - matr[0][i] << "\t";

}

std::cout << std::endl;

std::cout << std::endl;

int n2 = 1;

int m2 = 0;

int basicM1 = 0;

double basicElement = 0;

double coefficient = 0;

bool negativeBasisFound = false;

while (n2 != n1 + 1)

{

if (matr[n2][2] < 0)

{

negativeBasisFound = true;

bool negativeElementFound = false;

for (int i = 3; i <= m1 + 4; i++)

{

if (matr[n2][i] < 0 && negativeElementFound == false)

{

negativeElementFound = true;

basicM1 = i;

}

}

if (negativeElementFound == true)

{

negativeBasisFound = false;

matr[n2][0] = basicM1 - 2; //запись номера x

matr[n2][1] = matr[0][basicM1]; //запись номера от F

basicElement = matr[n2][basicM1]; //запись опорного элемента

for (int i = 1; i <= n1; i++) //расчет симплекс таблицы относительно опорного элемента

{

coefficient = matr[i][basicM1] / basicElement;

if (i != n2)

{

for (int j = 2; j <= m1 + n1 + 2; j++)

{

matr[i][j] = matr[i][j] - matr[n2][j] \* coefficient;

}

}

}

for (int i = 2; i <= m1 + n1 + 2; i++) //деление базовой строки на опорный элемент

{

matr[n2][i] = matr[n2][i] / basicElement;

}

std::cout << "|БП" << "\t" << "|Сб" << "\t" << "|B" << "\t"; //вывод верхней строки

for (int i = 1; i < m1 + n1 + 1; i++)

{

std::cout << "|x" << i << "(" << matr[0][i + 2] << ")" << "\t";

}

std::cout << std::endl;

for (int i = 1; i <= n1; i++) //вывод матрицы

{

std::cout << "|x";

for (int j = 0; j < m1 + n1 + 3; j++)

{

if (j == 0)

{

std::cout << matr[i][j] << "\t";

}

else

{

std::cout << "|" << matr[i][j] << "\t";

}

}

std::cout << std::endl;

}

std::cout << "|delta" << "\t" << "\t";

for (int i = 2; i <= m1 + n1 + 2; i++) //вывод индекстной строки

{

std::cout << "|" << matr[n1 + 1][i] << "\t";

}

std::cout << std::endl;

std::cout << std::endl;

n2 = 0;

}

}

n2 = n2 + 1;

}

if (negativeBasisFound == false)

{

int m1End = 3; //номер опорного столбца

basicElement = 0; //опорный элемент

double basicElementBuf;

bool basicСolumnFound = true; //проверка найден ли опорный столбец

bool basicStringFound = false; //проверка найдена ли опорная строка

coefficient = 0; //коэффициент для приведение опорного столбцца к 0

int basicString = 0; //номер опорной строки

//const int INFINITY = 9999999;

do

{

basicElementBuf = -INFINITY;

basicСolumnFound = false;

for (int i = 3; i <= m1 + n1 + 2; i++) //поиск отрицательной переменной в индекстной строке

{

if (matr[n1 + 1][i] < 0)

{

if (basicElementBuf < abs(matr[n1 + 1][i]) && matr[n1 + 2][i] != 1)

{

basicElementBuf = abs(matr[n1 + 1][i]);

m1End = i;

basicСolumnFound = true;

}

}

}

if (basicСolumnFound == true) //если отрицательный элемент найден

{

matr[n1 + 2][m1End] = 1;

basicElementBuf = INFINITY;

basicStringFound = false;

for (int i = 1; i <= n1; i++) //поиск опорной строки

{

basicElement = matr[i][2] / matr[i][m1End];

if (basicElement < basicElementBuf && matr[i][m1End] > 0)

{

basicElementBuf = basicElement;

basicString = i;

basicStringFound = true;

//std::cout << counterEnd << " " << basicElement << " " << basicString;

}

}

if (basicStringFound == true) //если опорная строка найдена

{

matr[basicString][0] = m1End - 2; //запись номера x

matr[basicString][1] = matr[0][m1End]; //запись номера от F

basicElement = matr[basicString][m1End]; //запись опорного элемента

for (int i = 1; i <= n1 + 1; i++) //расчет симплекс таблицы относительно опорного элемента

{

coefficient = matr[i][m1End] / basicElement;

if (i != basicString)

{

for (int j = 2; j <= m1 + n1 + 2; j++)

{

matr[i][j] = matr[i][j] - matr[basicString][j] \* coefficient;

}

}

}

for (int i = 2; i <= m1 + n1 + 2; i++) //деление базовой строки на опорный элемент

{

matr[basicString][i] = matr[basicString][i] / basicElement;

}

matr[n1 + 2][m1End] = 0; //убирание индификатора проверенного элемента в индекстной строке

std::cout << "|БП" << "\t" << "|Сб" << "\t" << "|B" << "\t"; //вывод верхней строки

for (int i = 1; i < m1 + n1 + 1; i++)

{

std::cout << "|x" << i << "(" << matr[0][i + 2] << ")" << "\t";

}

std::cout << std::endl;

for (int i = 1; i <= n1; i++) //вывод матрицы

{

std::cout << "|x";

for (int j = 0; j < m1 + n1 + 3; j++)

{

if (j == 0)

{

std::cout << matr[i][j] << "\t";

}

else

{

std::cout << "|" << matr[i][j] << "\t";

}

}

std::cout << std::endl;

}

std::cout << "|delta" << "\t" << "\t";

for (int i = 2; i <= m1 + n1 + 2; i++) //вывод индекстной строки

{

std::cout << "|" << matr[n1 + 1][i] << "\t";

}

std::cout << std::endl;

std::cout << std::endl;

basicElementBuf = INFINITY;

}

}

else

{

int count = 0;

basicElementBuf = -INFINITY;

for (int i = 3; i <= m1 + n1 + 2; i++) //подсчет количества нулей в индекстной строке

{

if (matr[n1 + 1][i] == 0)

{

count = count + 1;

}

}

if (count == n1 && basicStringFound == true)

{

std::cout << "решение получено:" << std::endl;

std::cout << "F = " << matr[n1 + 1][2] << std::endl;

for (int i = 1; i <= n1; i++)

{

if (matr[i][0] <= n)

{

std::cout << "X" << matr[i][0] << " = " << matr[i][2] << std::endl;

}

}

bool notIntegerElementFound = false;

for (int i = 1; i <= n1; i++)

{

if (matr[i][2] != (int)matr[i][2] && notIntegerElementFound == false && matr[i][0] <= n)

{

notIntegerElementFound = true;

matrResults[nResult][0] = matr[i][0];

matrResults[nResult][1] = matr[i][2];

}

}

if (notIntegerElementFound == false)

{

for (int i = 1; i <= n1; i++)

{

if (matr[i][0] <= n)

{

matrResults[nResult][(int)matr[i][0] + 2] = matr[i][2];

}

}

matrResults[nResult][2] = matr[n1 + 1][2];

return 1;

}

else

{

return 2;

}

}

else if (basicStringFound == true)

{

std::cout << "множество решений" << std::endl;

matrResults[nResult][0] = -1;

return 0;

}

else

{

std::cout << "решений нет" << std::endl;

matrResults[nResult][0] = -1;

return 0;

}

}

} while (basicСolumnFound != false);

}

else

{

std::cout << "решений нет" << std::endl;

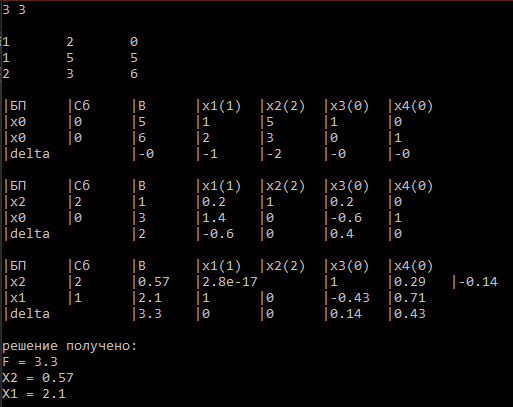
matrResults[nResult][0] = -1;

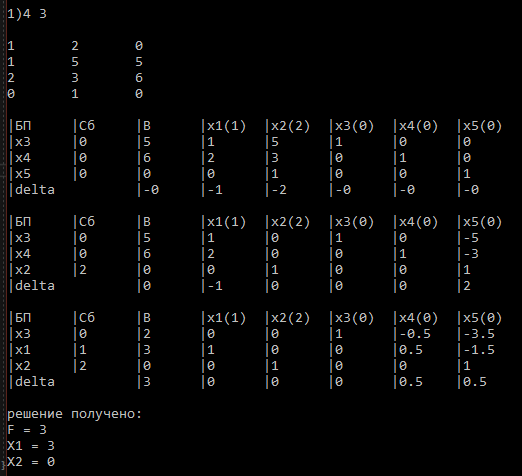
return 0;

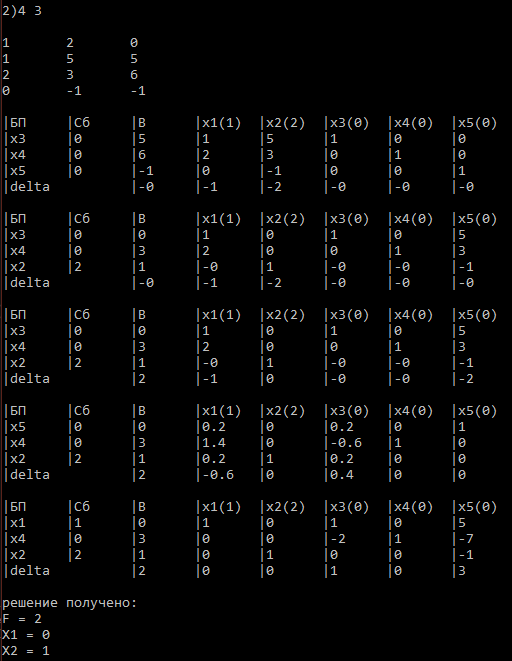
}

}

1. Экранные формы:







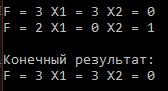


Рисунок 1 – Успешное выполнение программы.

1. Вывод:

В ходе выполнения лабораторной работы были закреплены на практике знания о методе ветвей и границ решения целочисленных задач линейного программирования и получены навыки его программной реализации. Была написана программа, реализующая метод ветвей и границ.