МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**
(БГТУ им. В.Г. Шухова)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №7

по дисциплине: Исследование операций
тема: «Решение полностью целочисленных задач с помощью первого алгоритма Гомори, а также методом ветвей и границ»

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили:
проф. Вирченко Юрий Петрович

Белгород 2024 г.

Лабораторная работа №7

Решение полностью целочисленных задач с помощью первого алгоритма Гомори, а также методом ветвей и границ

Цель работы: освоить метод отсечения Гомори для полностью целочисленных задач. Изучить алгоритм этого метода. Программно реализовать этот алгоритм.

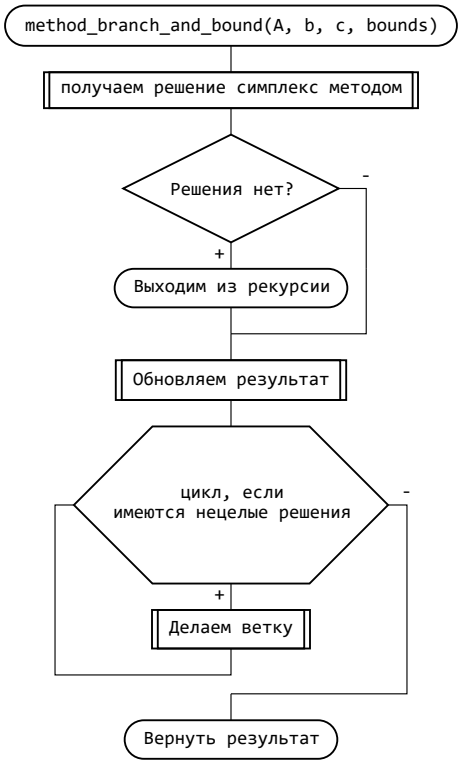
Задание: выяснить для каких задач применяется первый алгоритм Гомори. Изучить этот алгоритм и написать реализующую его программу для ПЭВМ. Изучить и программно реализовать алгоритм метода ветвей и границ. В качестве тестовых данных использовать, решенную вручную одну из нижеследующих задач.

Вариант 10

$$z = 9x_1 + x_2 + 3x_5 \to max;$$

$$\begin{cases} 10x_1 + 3x_2 + x_3 = 40, \\ 9x_1 - 4x_2 + x_4 = 7, \\ -4x_1 + x_2 + x_5 = 14, \\ x_i \geq 0 (i = \overline{1,5}). \end{cases}$$

Блок-схемы:



Листинг программы:

```
#include <vector>
#include <array>

#include "../libs/alg/alg.h"

template <std::size_t T, std::size_t MatrixLines>
std::tuple<Fraction, std::vector<Fraction>> solveInInt(std::vector<std::array<Fraction, T>> matrix, std::array<Fraction,
↪  T> function);

template <>
std::tuple<Fraction, std::vector<Fraction>> solveInInt<15ULL, 12ULL>(std::vector<std::array<Fraction, 15ULL>> matrix,
↪  std::array<Fraction, 15ULL> function) {
    throw std::invalid_argument("Too much!");
}

template <std::size_t T, std::size_t MatrixLines>
std::tuple<Fraction, std::vector<Fraction>> solveInInt(std::vector<std::array<Fraction, T>> matrix, std::array<Fraction,
↪  T> function) {
    std::cout << "Simplex" << std::endl;
    // Применяем симплекс метод
    auto res = solveSimplexMethodMaxRaw(matrix, function, Fraction());

    // Ищем нецелые свободные члены с максимальным остатком
    int maxFracIndex = -1;
    std::vector<int> freeIndices;
    for (int i = 0; i < matrix.size(); i++) {
        if (!matrix[i].back().isInt()) {
            if (maxFracIndex == -1) {
                maxFracIndex = i;
                continue;
            }
```

```
            auto newDouble = matrix[i].back().getFrac();
            auto oldDouble = matrix[maxFracIndex].back().getFrac();
            if (newDouble > oldDouble) maxFracIndex = i;
        }
    }

    // Определяем свободные переменные
    for (int j = 0; j < T; j++) {
        bool metOne = false;
        bool onlyOnesAndZeros = true;
        for (int i = 0; i < MatrixLines; i++) {
            if (!(matrix[i][j] == Fraction() || matrix[i][j] == Fraction(1))) {
                onlyOnesAndZeros = false;
                break;
            }

            if (matrix[i][j] == Fraction(1)) {
                if (metOne) {
                    onlyOnesAndZeros = false;
                    break;
                } else {
                    metOne = true;
                }
            }
        }

        if (!onlyOnesAndZeros) freeIndices.push_back(j);
    }

    // Если все свободные члены целые
    if (maxFracIndex == -1) {
        // Возвращаем ответ
        std::vector<Fraction> ansB;
        for (int i = 0; i < MatrixLines; i++) {
            ansB.push_back(matrix[i].back());
        }

        return {res, ansB};
    }

    // Формируем сечение
    std::array<Fraction, T + 1> newLine;
    newLine[T] = matrix[maxFracIndex].back().getFrac() * Fraction(-1);
    newLine[T - 1] = Fraction(1);
    for (int i = 0; i < T - 1; i++) {
        if (std::find(freeIndices.begin(), freeIndices.end(), i) != freeIndices.end()) {
            newLine[i] = Fraction(-1) * matrix[maxFracIndex][i].getFrac();
        }
    }

    std::vector<std::array<Fraction, T + 1>> simplexMatrix;
    for (int i = 0; i < matrix.size(); i++) {
        simplexMatrix.push_back({});
```

```
    for (int j = 0; j < T - 1; j++) {
        simplexMatrix[i][j] = matrix[i][j];
    }

    simplexMatrix[i][T] = matrix[i].back();
}
simplexMatrix.push_back(newLine);
std::array<Fraction, T + 1> funcLine = {};
for (int i = 0; i < T - 1; i++) {
    funcLine[i] = function[i];
}
funcLine[T] = function.back();
simplexMatrix.push_back(funcLine);


std::cout << "Marks" << std::endl;
// Используем метод последовательного уточнения оценок
while (true) {
    for (int i = 0; i < simplexMatrix.size(); i++) {
        for (int j = 0; j < T + 1; j++) {
            std::cout << std::setw(10) << simplexMatrix[i][j] << " ";
        }
        std::cout << std::endl;
    }
    std::cout << std::endl;

    int minRowIndex = -1;
    for (int i = 0; i < MatrixLines + 1; i++)
        if (simplexMatrix[i].back() < Fraction() && (minRowIndex == -1 || simplexMatrix[i].back() <
        ↪  simplexMatrix[minRowIndex].back()))
            minRowIndex = i;

    if (minRowIndex == -1) break;

    int minColumnIndex = -1;
    for (int i = 0; i < T + 1; i++)
        if (simplexMatrix[minRowIndex][i] < Fraction() && (minColumnIndex == -1 ||
        (Fraction(-1) * simplexMatrix.back()[i] / simplexMatrix[minRowIndex][i]) < (Fraction(-1) *
        ↪  simplexMatrix.back()[minColumnIndex] / simplexMatrix[minRowIndex][minColumnIndex])))
            minColumnIndex = i;

    if (minColumnIndex == -1) throw std::invalid_argument("No solution");

    subtractLineFromOther(simplexMatrix, minRowIndex, minColumnIndex, Fraction());
}

std::array<Fraction, T + 1> newFunction = {};
for (int i = 0; i < T + 1; i++) {
    newFunction[i] = simplexMatrix.back()[i] * Fraction(-1);
}

simplexMatrix.pop_back();

// Решаем следующую задачу
```

```
    return solveInInt<T + 1, MatrixLines + 1>(simplexMatrix, newFunction);
}

int main() {
    std::vector<std::array<Fraction, 6>> matrix;
    matrix.push_back({{{10}, {3}, {1}, {0}, {0}, {40}}});
    matrix.push_back({{{9}, {-4}, {0}, {1}, {0}, {7}}});
    matrix.push_back({{{-4}, {1}, {0}, {0}, {1}, {14}}});
    std::array<Fraction, 6> function{{{9}, {1}, {0}, {0}, {3}, {0}}};
    Fraction EPS;

    auto res = solveInInt<6, 3>(matrix, function);
    std::cout << std::get<0>(res) << std::endl;
}
```

```
from scipy.optimize import linprog
import numpy as np
import math as m

A = np.array([[7, 5, 1, 0, 0],
              [4, -6, 0, 3, 0],
              [-3, 4, 0, 0, 1]])
b = np.array([28, 14, 6])
c = np.array([3, 1, 0, 0, - 1])


max_res_simplex_method = None
def method_branch_and_bound(A, b, c, bounds):
    global max_res_simplex_method
    # получаем решение симплекс методом
    res = linprog(-c, A_ub=A, b_ub=b, bounds=bounds, method='highs')
    # если решения нет выходим из рекурсии
    if not res.success:
        return None

    print("P:", res.x)
    print("Макс ф:", -res.fun)

    # обновляем результат
    if all(is_int(res.x)) and (max_res_simplex_method is None or -res.fun > -max_res_simplex_method.fun):
        max_res_simplex_method = res

    # цикл, если имеются нецелые решения делаем ветку
    for index, x in zip(range(len(res.x)), res.x):
        if not is_int(x):
            bounds_for_bigger_than = bounds.copy()
            bounds_for_bigger_than[index] = (m.ceil(x), None)
            bounds_for_less_than = bounds.copy()
            bounds_for_less_than[index] = (0, m.floor(x))

            method_branch_and_bound(A, b, c, bounds_for_bigger_than)
            method_branch_and_bound(A, b, c, bounds_for_less_than)
```

```
    # выводим результат
    return max_res_simplex_method


def is_int(n, epsilon=0.001):
    return abs(n % 1) <= epsilon


if __name__ == '__main__':
    print("Промежуточные решения и целевые функции:")
    res = method_branch_and_bound(A, b, c, [(0, None)]*len(c))
    print()
    print("Решение:", res.x)
    print("Максимизированная функция:", -res.fun)
```

Результаты выполнения программы:

```
78
```

```
Решение: [ 2.  3.  0.  0. 19.]
Максимизированная функция: 78.0
```

Результаты вычислений:

$$z = 9x_1 + x_2 + 3x_5 \to max;$$
$$\begin{cases} 10x_1 + 3x_2 + x_3 = 40, \\ 9x_1 - 4x_2 + x_4 = 7, \\ -4x_1 + x_2 + x_5 = 14, \\ x_i \geq 0 (i = \overline{1,5}). \end{cases}$$

Сформируем задачу ЗО:

Баз. пер.	Св. чл.	x_1	x_2	x_3	x_4	x_5
x_3	40	10	3	1	0	0
x_4	7	9	-4	0	1	0
x_5	14	-4	1	0	0	1
z	0	-9	-1	0	0	-3

Баз. пер.	Св. чл.	x_1	x_2	x_3	x_4	x_5
x_3	290/9	0	67/9	1	-10/9	0
x_1	7/9	1	-4/9	0	1/9	0
x_5	154/9	0	-7/9	0	4/9	1
z	7	0	-5	0	1	-3

Баз. пер.	Св. чл.	x_1	x_2	x_3	x_4	x_5
x_2	290/67	0	1	9/67	-10/67	0
x_1	181/67	1	0	4/67	3/67	0
x_5	1372/67	0	0	7/67	22/67	1
z	1919/67	0	0	45/67	17/67	-3

Баз. пер.	Св. чл.	x_1	x_2	x_3	x_4	x_5
x_2	290/67	0	1	9/67	-10/67	0
x_1	181/67	1	0	4/67	3/67	0
x_5	1372/67	0	0	7/67	22/67	1
z	6035/67	0	0	66/67	83/67	0

Так как план содержит нецелые числа, необходимо сделать новое сечение И получить задачу З1.

Баз. пер.	Св. чл.	x_1	x_2	x_3	x_4	x_5	v_1
x_2	290/67	0	1	9/67	-10/67	0	0
x_1	181/67	1	0	4/67	3/67	0	0
x_5	1372/67	0	0	7/67	22/67	1	0
v_1	-47/67	0	0	-4/67	-3/67	0	1
z	6035/67	0	0	66/67	83/67	0	0

Баз. пер.	Св. чл.	x_1	x_2	x_3	x_4	x_5	v_1
x_2	11/4	0	1	0	-1/4	0	9/4
x_1	2	1	0	0	0	0	1
x_5	77/4	0	0	0	1/4	1	7/4
x_3	47/4	0	0	1	3/4	0	-67/4
z	157/2	0	0	0	1/2	0	33/2

Симплекс метод неприменим. Так как план содержит нецелые числа, необходимо сделать новое сечение И получить задачу З2.

Баз. пер.	Св. чл.	x_1	x_2	x_3	x_4	x_5	v_1	v_2
x_2	11/4	0	1	0	-1/4	0	9/4	0
x_1	2	1	0	0	0	0	1	0
x_5	77/4	0	0	0	1/4	1	7/4	0
x_3	47/4	0	0	1	3/4	0	-67/4	0
v_2	-3/4	0	0	0	-3/4	0	-1/4	1
z	157/2	0	0	0	1/2	0	33/2	0

x_2	3	0	1	0	0	0	7/3	-1/3
x_1	2	1	0	0	0	0	1	0
x_5	19	0	0	0	0	1	5/3	1/3
x_3	11	0	0	1	0	0	-17	1
x_5	1	0	0	0	1	0	1/3	-4/3
z	78	0	0	0	0	0	49/3	2/3

Симплекс метод неприменим.
План содержит только целые числа.
Ответ: [2, 3, 0, 0, 19], $z_{max} = 78$

Вывод: в ходе лабораторной работы освоили метод отсечения Гомори для полностью целочисленных задач. Изучить алгоритм этого метода. Программно реализовали и отладили программу, реализующую этот алгоритм.