

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

Отчет по лабораторной работе №3 дисциплины
«Исследование операций»

Выполнил студент группы ИВТ-31 _____/Крючков И. С/
Проверил _____/Коржавина А. С./

Киров 2022

1. Цель работы

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

Идея метода Гомори заключается в следующем.

Отбрасывается условие целочисленности и полученная задача линейного программирования решается симплекс-методом. Если оптимальное решение задачи является целочисленным, то оно является и решением исходной задачи. Если оптимальное решение задачи не является целочисленным, то к основным ограничениям добавляется новое линейное ограничение, обладающее следующими свойствами:

- 1) оптимальный нецелочисленный план задачи ему не удовлетворяет;
- 2) любой целочисленный план задачи ему удовлетворяет.

Затем решается расширенная задача. Процесс повторяется до получения целочисленного решения. Способы построения дополнительного линейного ограничения различны для полностью и частично целочисленных задач линейного программирования. В силу свойств 1 и 2 дополнительное ограничение еще называют отсечением Гомори, а метод Гомори - методом отсечения.

Если задача разрешима в целых числах, то через конечное число итераций оптимальный целочисленный план будет найден.

Если в процессе решения появится строка с нецелым свободным членом и целыми остальными коэффициентами, то соответствующее уравнение не имеет решения в целых числах. В таком случае и исходная задача неразрешима в целых числах.

4. Листинг программы

```
import math
import copy

# Ввод данных
def input_data():

    v_num_in = int(input("Введите количество переменных: \n"))
    l_num_in = int(input("Введите количество ограничений: \n"))

    f_in = [0] * v_num_in

    f_raw = input(f"Введите коэффициенты целевой функции F(x) ({v_num_in} числ. через пробел): \n").split()

    f_in = list(map(float, f_raw))

    ok = 0
    while ok == 0:
        f_type_in = int(input(("Введите тип функции:\n"
                                "1. max\n"
                                "2. min\n")))
        if(f_type_in == 1 or f_type_in == 2):
            ok = 1

    limits_a_in = [[0] * v_num_in] * l_num_in
    limits_b_in = [0] * l_num_in
    limits_type_in = [1] * l_num_in

    for i in range(l_num_in):
        ok = 0
        while ok == 0:
            limits_raw = input(f"Ввод ограничений №{i+1} ({v_num_in+1} числ. через пробел): \n").split()
            if(len(limits_raw) == v_num_in+1):
                ok = 1

            limits_tmp = list(map(float, limits_raw))
            limits_a_in[i] = limits_tmp[:-1]
            limits_b_in[i] = limits_tmp[-1]

        ok = 0
        while ok == 0:
            l_type_in = int(input(("Введите тип ограничения:\n"
                                    "1. >=\n"
                                    "2. <=\n"
                                    "3. =\n")))
            if(l_type_in == 1 or l_type_in == 2 or l_type_in == 3):
                ok = 1

            limits_type_in[i] = l_type_in

    return f_in, limits_a_in, limits_b_in, limits_type_in, v_num_in, l_num_in, f_type_in

def comp(a, b, ft):
    if ft == 1:
        return a < b
    else:
        return a > b

def simplex(s_f, s_limits_a, s_limits_b, s_limits_type, v_num, l_num, f_type, nx):
    f = copy.deepcopy(s_f)
    limits_a = copy.deepcopy(s_limits_a)
```

```

limits_b = copy.deepcopy(s_limits_b)
limits_type = copy.deepcopy(s_limits_type)

sv_num = v_num
result_x = [0] * v_num

# Приведение к каноническому виду
# 1 >=
# 2 <=
k_type = 1 if f_type == 2 else 2

if nx == False:
    for i, lim in enumerate(limits_a):
        if limits_type[i] == 1:
            limits_a[i] = list(map(lambda x: -x, lim))
            limits_b[i] = -limits_b[i]
            limits_type[i] = k_type

        if limits_type[i] != 3:
            for ai, _ in enumerate(limits_a):
                limits_a[ai].append(0)

            v_num += 1

            limits_a[i][-1] = 1
            f.append(0)
            result_x.append(0)

limits_b.append(0)

min_max = min if f_type == 1 else max

def get_basis():

    vbasis = [-1]*l_num
    bs_zero = l_num

    cols_t = list(zip(*limits_a))
    cols = [list(sb) for sb in cols_t]

    z_cols_ids = []
    oe_cols_ids = []
    nz_cols_ids = []

    for cid, col in enumerate(cols):

        z_f = False
        cnt = 0
        el_id = 0

        oe_cnt = 0
        oe_id = 0

        nz_cnt = 0

        for i, v in enumerate(col):

            if v == 1:
                cnt += 1
                el_id = i

                oe_cnt += 1

```

```

        oe_id = i
    elif v != 0:
        cnt = 0
        z_f = True

        oe_cnt += 1
        oe_id = i

    if v == 0:
        nz_cnt += 1

    if cnt == 1 and z_f == False:
        z_cols_ids.append((el_id, cid))

    if oe_cnt == 1:
        oe_cols_ids.append((oe_id, cid))

    if nz_cnt == 0:
        nz_cols_ids.append(cid)

for lavid, lav in enumerate(limits_a):
    added = 0

    for idld, ld in reversed(list(enumerate(lav))):
        if ld == 1:
            for zv in z_cols_ids:
                if idld == zv[1]:
                    vbasis[lavid] = idld
                    bs_zero -= 1
                    added = 1
                    break

            if added == 1:
                break

    if bs_zero == 0:
        return vbasis

for bsid, bs in enumerate(vbasis):
    if bs == -1:
        for ldiv, ldata in enumerate(limits_a[bsid]):
            added = 0
            for oe in oe_cols_ids:
                if bsid == oe[0] and ldiv == oe[1]:
                    vbasis[bsid] = ldiv
                    bs_zero -= 1
                    added = 1

            limits_b[bsid] /= limits_a[bsid][ldiv]

            for ldi, ldd in enumerate(limits_a[bsid]):
                limits_a[bsid][ldi] /= limits_a[bsid][ldiv]

            break

        if added == 1:
            break

    if bs_zero == 0:
        return vbasis

for bsid, bs in enumerate(vbasis):
    if bs == -1:

```

```

        for nz in nz_cols_ids:
            if nz not in vbasis:
                dtmp = limits_a[bsid][nz]

                limits_b[bsid] /= dtmp

            for ldi, ldd in enumerate(limits_a[bsid]):
                limits_a[bsid][ldi] /= dtmp

        for ltid, ltd in enumerate(limits_a):
            if ltid != bsid:
                mdt = ltd[nz]
                for ldi, ldd in enumerate(ltd):
                    limits_a[ltid][ldi] -= limits_a[bsid][ldi] * mdt

                limits_b[ltid] -= limits_b[bsid] * mdt

        vbasis[bsid] = nz
        bs_zero -= 1
        break

    return vbasis

def get_delta(bs):
    delta = [0]*v_num
    limits_b[-1] = 0

    cols_t = list(zip(*limits_a))
    cols = [list(sb) for sb in cols_t]

    for did, d in enumerate(delta):
        for bid, base in enumerate(bs):
            delta[did] += f[base] * cols[did][bid]
            delta[did] -= f[did]

    for bid, base in enumerate(bs):
        limits_b[-1] += f[base] * limits_b[bid]
    limits_b[-1] -= f[-1]

    return delta

def rem_nfc(bs):

    def num_negative(l):
        nn = 0
        for lb in l:
            if lb < 0:
                nn += 1
        return nn

    rne = False
    while rne == False and num_negative(limits_b[:-1]) > 0:

        nvs = [(i, v) for i, v in enumerate(limits_b[:-1]) if v < 0]
        mrow = max(nvs, key=lambda x: abs(x[1]))[0]

        if num_negative(limits_a[mrow]) == 0:
            rne = True
            break

```

```

nvs = [(i, v) for i, v in enumerate(limits_a[mrow]) if v < 0]

mcol = max(nvs, key=lambda x: abs(x[1]))[0]

dtmp = limits_a[mrow][mcol]

limits_b[mrow] /= dtmp

for ldi, ldd in enumerate(limits_a[mrow]):
    limits_a[mrow][ldi] /= dtmp

for ltid, ltd in enumerate(limits_a):
    if ltid != mrow:
        mdt = ltd[mcol]
        for ldi, ldd in enumerate(ltd):
            limits_a[ltid][ldi] -= limits_a[mrow][ldi] * mdt

        limits_b[ltid] -= limits_b[mrow] * mdt

bs[mrow] = mcol

return rne, bs

```

основной цикл вычислений

while True:

```

basis = get_basis()
err, basis = rem_nfc(basis)

```

```

if err:
    # Нет решений
    return None, 1

```

```

dlt = get_delta(basis)

```

```

norm = 0
for v in dlt:
    if comp(v, 0, f_type):
        norm += 1

```

```

if norm == 0:
    break

```

```

#разрешающий столбец
rc_id = min_max(range(len(dlt)), key=dlt.__getitem__)
rc = []
for i in range(l_num):
    rc.append(limits_a[i][rc_id])

```

```

bi_d_rc = []
for di, (bi, rci) in enumerate(zip(limits_b[:-1], rc)):
    if rci == 0:
        continue

```

```

    q = bi/rci

```

```

    if q < 0:
        continue

```

```

    if bi == 0 and rci < 0:
        continue

```

```

        bi_d_rc.append((di, q))

    if len(bi_d_rc) == 0:
        # Оптимальное решение отсутствует
        return None, 2

    row, _ = min(bi_d_rc, key=lambda x: x[1])

    limits_b[row] /= limits_a[row][rc_id]
    limits_a[row] = list(map(lambda x: x / limits_a[row][rc_id], limits_a[row]))

    av = limits_a[row][rc_id]

    for i, lim in enumerate(limits_a):
        if i != row:

            b_bv = limits_a[i][rc_id]
            bv = limits_a[i][rc_id]

            mab = bv/av
            for idl, _ in enumerate(lim):
                limits_a[i][idl] -= limits_a[row][idl]*mab

            limits_b[i] -= limits_b[row]*(b_bv/av)

# получение результата

limits_a.append(dlt)

cols_t = list(zip(*limits_a))
cols = [list(sb) for sb in cols_t]

nd_cols_ids = []
for cid, col in enumerate(cols):
    cnt = 0
    el_id = 0
    for i, v in enumerate(col):
        if v != 0:
            cnt += 1
            el_id = i

    if cnt == 1:
        nd_cols_ids.append((el_id, cid))

nd_cols_ids.sort()

for col in nd_cols_ids:
    result_x[col[1]] = limits_b[col[0]]/limits_a[col[0]][col[1]]

f_val = limits_b[-1]

return (result_x, f_val, basis, limits_a, limits_b, limits_type, v_num, l_num, f, err), 0

def getFraction(v):
    return v - math.floor(v)

pres = 6

def print_arr(a):
    for v in a:
        print(list(map(lambda x: round(x, pres), v)))

```



```

# Метод гомори
def gomori_method(b_f, b_lts_a, b_limits_b, b_limits_type, v_num, l_num, f_type):
    lf = copy.deepcopy(b_f)
    la = copy.deepcopy(b_lts_a)
    lb = copy.deepcopy(b_limits_b)
    lt = copy.deepcopy(b_limits_type)

    vn = v_num
    ln = l_num
    lnx = False

    while True:
        print()
        print()
        print_arr(la)
        print(f'{list(map(lambda x: round(x, pres), lb))} b')
        data, code = simplex(lf, la, lb, lt, vn, ln, f_type, lnx)

        if data != None:
            (res, f_res, basis, la, lb, lt, vn, ln, lf, er) = data
            print('-----')
            print(f'{list(map(lambda x: round(x, pres), la[-1]))} delta')
            print(f'{list(map(lambda x: round(x, pres), res[:v_num]))} F = {f_res} result')

            if all([float(round(x, pres)).is_integer() for x in res]) == False:

                a_b = [(aid, res[x]) for aid, x in enumerate(basis)]

                mv, md = max(a_b, key=lambda x: getFraction(x[1]))

                la.pop(-1)
                lb.pop(-1)

                la.append(list(map(lambda x: - getFraction(x), la[mv])))

                for ai, _ in enumerate(la):
                    la[ai].append(0)
                la[-1][-1] = 1

                lb.append(-getFraction(lb[mv]))
                lt.append(2)
                lf.append(0)

                vn += 1
                ln += 1
                lnx = True

            else:
                return [res[:v_num], f_res]

        else:
            return None, 1

if __name__ == '__main__':
    f_in, limits_a_in, limits_b_in, limits_type_in, v_num_in, l_num_in, f_type_in =
input_data()
    g_res = gomori_method(f_in, limits_a_in, limits_b_in, limits_type_in, v_num_in, l_num_in,
f_type_in)

    if g_res[0] == None:
        print('Нет решений')
    else:
        print('Ответ:')
        print(*list(map(lambda x: round(x, pres), g_res[0])))
        print(f"F = {g_res[1]}")

```

5. Экранные формы

```
[2.0, 1.0, 1.0]
[3.0, 0.0, 2.0]
[0.0, 0.0, 1.0]
[2.0, 3.0, 1.0] b
-----
[0.0, -2.0, 0.0, 0.0, -1.0, -1.0] delta
[0.333333, 0, 1.0] F = 4.0 result

[0.0, 1.0, 0.0, 1.0, 0.666667, -0.333333, 0]
[1.0, 0.0, 0.0, 0.0, -0.333333, 0.666667, 0]
[-0.0, -0.0, 1.0, -0.0, -0.0, -1.0, 0]
[-0.0, -0.0, -0.0, -0.0, -0.666667, -0.666667, 1]
[0.333333, 0.333333, 1.0, -0.333333] b
-----
[0.0, -2.0, 0.0, -0.0, 0.0, 0.0, -1.5] delta
[0.5, 0, 1.0] F = 4.5 result

[-0.0, -1.0, -0.0, -1.0, -0.0, 1.0, -1.0, 0]
[1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.5, 0]
[0.0, -1.0, 1.0, -1.0, 0.0, 0.0, -1.0, 0]
[0.0, 1.0, 0.0, 1.0, 1.0, 0.0, -0.5, 0]
[-0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.5, 1]
[0.0, 0.5, 1.0, 0.5, -0.5] b
-----
[0.0, -2.0, 0.0, -0.0, 0.0, 0.0, 0.0, -3.0] delta
[0.0, 0, 2.0] F = 6.0 result
Ответ:
0.0 0 2.0
F = 6.0
```

6. Вывод

В ходе выполнения лабораторной работы был изучен метод отсечений (Гомори) для решения задач целочисленного программирования, получены навыки его программной реализации.