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.

1. Задание

Реализовать метод Гомори решения задачи целочисленного программирования.

1. 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.

1. 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')

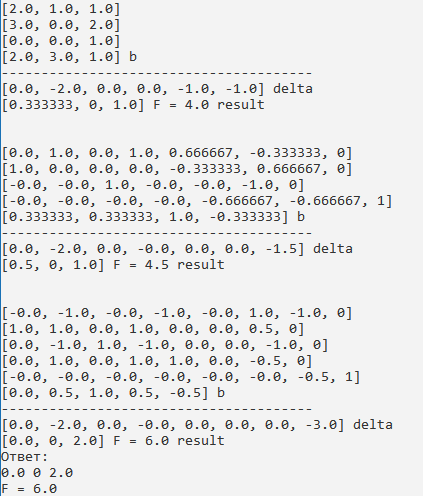
else:

print('Ответ:')

print(\*list(map(lambda x: round(x, pres), g\_res[0])))

print(f"F = {g\_res[1]}")

1. Экранные формы



1. Вывод

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