

МІНІСТЕРСТВО ОСВІТИ І НАУКИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»



Лабораторна робота №7
З дисципліни «Математичні методи дослідження операцій»

Виконав:
студент групи КН-210
Бурак Марко

Мета роботи: Розглянути особливості застосування методу гілок та границь для розв'язування задач цілочисельного програмування.

Алгоритм методу гілок та границь для розв'язування задачі про комівояжера.

Завдання

№	4			
0	10	5	14	19
8	0	16	16	8
20	6	0	18	7
9	14	6	0	10
2	13	13	12	0

Для початку покажу реалізацію цієї задачі вручну, а опісля, кодову частину, та порівняю на правильність результатів.

Отож, для початку потрібно провести операцію редукцію. Це означає, що потрібно вибрати мінімальний аргумент по кожній стрічці, та відняти цей аргумент від кожного з елементів. Це саме повторю і для вертикальних ліній.

Тут я редуктую початкову матрицю.

Saleman problem

(4)

					min
∞	10	5	14	19	5
8	∞	16	16	8	8
20	6	∞	18	7	6
9	14	6	∞	10	6
2	13	13	12	∞	2

∞	5	0	9	14	5
0	∞	8	8	0	8
14	0	∞	12	1	6
3	8	0	∞	4	6
0	11	11	10	∞	2

0 0 0 8 0

∞	5	0	1	14	5
0	∞	8	0	0	8
14	0	∞	4	1	6
3	8	0	∞	4	6
0	11	11	2	∞	2

0 0 0 8 0

$$H = 5 + 8 + 6 + 6 + 2 + 8 = 35$$

Після цього потрібно знайти нижню границю цієї матриці, вона дорівнює сумі всіх мінімальних значень вертикальних та горизонтальних ліній, у моєму випадку це 35.

Наступним кроком є визначення ребра, для цього потрібно всі нулі перетворити на безкінечності, тоді знайти мінімальний елемент по горизонталі та вертикалі кожного рядка, опісля до кожного нуля записати суму мінімальних елементів горизонтального та вертикального рядка, відповідно.

Редування матриць з нижньою
гранницею 35

$$H = 35$$

∞	5	$O(1)$	1	14	1
$O(6)$	∞	8	$O(1)$	$O(1)$	0
14	$\boxed{O(6)}$	∞	4	1	1
3	8	$O(3)$	∞	4	3
$O(2)$	11	11	2	∞	2
0	5	0	1	1	

розділює на $(3,2)$ $(3^*,2^*)$

Отриманий максимальний елемент біля нуля і є вибраним ребром, у моєму випадку це ребро $(3,2)$.

Отже, далі розгалужую задачі на дві різні підмножини $(3,2)$ та $(3^*,2^*)$.

$(3^*, 2^*)$ буде мати вигляд такий же як попередня матриця, проте, з елементом ∞ замість елемента $(3,2)$. Для цієї матриці шукаю нижню границю.

$(3^*, 2^*)$

∞	5	0	1	14	0
0	∞	8	0	0	0
14	∞	∞	4	1	1
3	8	0	∞	4	0
0	11	11	2	∞	0
0	5	0	0	0	<u>6</u>

$35 + 6 = 41$

Для цієї матриці нижня межа 41.

Для побудови наступної матриці потрібно видалити 3 горизонтальний рядок та 2 вертикальний. Після порахувати нижню межу цієї матриці, якщо ж нижня межа менша, рівна нижній межі матриці, яка наведена зверху, тоді це ребро можна додати у маршрут.

$(3, 2)$

∞	0	1	14	0
0	8	0	0	0
3	0	∞	4	0
0	11	2	∞	0
0	0	0	0	

$35 + 0 = 35 \leq 41$

виключає $(3, 2)$

Нижня межа цієї матриці дорівнює 35, що менше 41, тоді додаю (3,2) до маршруту, якщо ж нерівність не виконується, тоді вибираємо ребро, базоване на найбільшому елементі, біля нуля.

Опісля того, знову встановлюю нове ребро та виконую ті ж кроки.

Встановлюю ребро

	1	3	4	5	
1	∞	$O(1)$	1	14	1
2	$O(0)$	∞	$O(1)$	$O(4)$	O
4	3	$O(3)$	∞	4	3
5	$O(2)$	11	2	∞	2
	O	O	1	4	

(2, 5)

$(2*, 5*)$

∞	0	1	14	0
0	∞	0	∞	0
3	0	∞	4	0
0	11	2	∞	0
0	0	0	4	4

$$H = 35 + 4 = 39$$

	1	3	4	
1	∞	0	1	0
4	3	0	∞	0
5	0	11	2	0
	0	0	1	<u>1</u>

$$H = 35 + 1 = 36 \leq 39 \text{ годится в парижскую}$$

Редуцирую та вставляем ребро

1	∞	$O(4)$	$O(1)$	$\overset{0}{\circ}$
4	3	$O(3)$	∞	3
5	$O(4)$	11	1	1
	3	0	1	

(5,1)

∞	0	0	0
3	0	∞	0
∞	11	1	1
3	0	0	4

$H=40$

(5,1)

	3	4	
1	0	0	0
4	0	∞	0
	0	0	0

$H=36$

включая в маршрут

Маршрут (3,2)(2,5)(5,1)(1,4)(4,3)

Должна 36

Отже, таким чином я отримав маршрут (3,2)(2,5)(5,1)(1,4)(4,3)

Кодова частина задачі комівіяжера

Я написав код на мові python, використовував бубліотеку numpy, для видалення горизонтальних та вертикальних рядків, а також для транспонування матриці.

Код задачі гілок та границь.

```
import numpy as np
```

```
mylist = [[None, 10, 5, 14, 19], [8, None, 16, 16, 8], [20, 6, None, 18, 7], [9, 14, 6, None, 10],  
[2, 13, 13, 12, None]]
```

```
down_limit = 0
```

```
deleted_hor = []
```

```
deleted_ver = []
```

```
#горизонтальна редукція
```

```
def reduct_hor(A):
```

```
    global down_limit
```

```
    for i in range(len(A)):
```

```
        # перевірка на нан та ініціалізація мінімуму
```

```
        if (A[i][0] != None):
```

```
            minim = A[i][0]
```

```
        else:
```

```
            minim = A[i][1]
```

```
        for j in range(len(A[i])):
```

```
            if (A[i][j] != None and A[i][j] < minim):
```

```
                minim = A[i][j]
```

```
        down_limit += minim
```

```
        for count in range(len(A[i])):
```

```
            if (A[i][count] != None):
```

```
                A[i][count] -= minim
```

```
    return A
```

#вертикальна редукція

```
def reduct_ver(A):
```

```
    # нижня границя
```

```
    global down_limit
```

```
    for i in range(len(A)):
```

```
        # перевірка на нан та ініціалізація мінімуму
```

```
        if (A[0][i] != None):
```

```
            minim = A[0][i]
```

```
        else:
```

```
            minim = A[1][i]
```

```
        for j in range(len(A[i])):
```

```
            if (A[j][i] != None and A[j][i] < minim):
```

```
                minim = A[j][i]
```

```
        down_limit += minim
```

```
        for count in range(len(A[i])):
```

```
            if (A[count][i] != None):
```

```
                A[count][i] -= minim
```

```
    return A
```

мінімальна сума всіх значень по горизонталі

```
def sum_min_hor(A):
```

```
    sum = 0
```

```
    for i in range(len(A)):
```

```
        # перевірка на нан та ініціалізація мінімуму
```

```
        if (A[0][i] != None):
```

```
            minim = A[0][i]
```

```
        else:
```

```
            minim = A[1][i]
```

```
        for j in range(len(A[i])):
```

```
            if (A[j][i] != None and A[j][i] < minim):
```

```
        minim = A[j][i]
    sum += minim
return sum
```

мінімальна сума всіх значень по вертикалі

```
def sum_min_ver(A):
    sum = 0
    for i in range(len(A)):
        # перевірка на нан та ініціалізація мінімуму
        if (A[i][0] != None):
            minim = A[i][0]
        else:
            minim = A[i][1]
        for j in range(len(A[i])):
            if (A[i][j] != None and A[i][j] < minim):
                minim = A[i][j]
        sum += minim
    return sum
```

знаходження ребра

```
def find_branch(A):
    # horizontal zeroes
    hor_arr = []
    for i in range(len(A)):
        # перевірка на нан та ініціалізація мінімуму
        for j in range(len(A[i])):
            if (i == j and A[i][j] == None):
                A[i][j] = 100000
    for i in range(len(A)):
        arr = sorted(A[i])
        zeroes = (len(A[i]) - np.count_nonzero(arr))
```

```

if (zeroes == 0):
    hor_arr.append(arr[0])
elif (zeroes == 1):
    hor_arr.append(arr[1])
else:
    hor_arr.append(0)
# vertical zeroes
ver_arr = []
A = np.array(A)
Transpose_A = A.transpose()
for i in range(len(A)):
    arr = sorted(Transpose_A[i])
    zeroes = (len(Transpose_A[i]) - np.count_nonzero(arr))
    if (zeroes == 0):
        ver_arr.append(arr[0])
    elif (zeroes == 1):
        ver_arr.append(arr[1])
    else:
        ver_arr.append(0)
whole_zeroes = []
X = 0
Y = 0
max = 0
for i in range(len(A)):
    for j in range(len(A)):
        if (A[i][j] == 0):
            temp = hor_arr[i] + ver_arr[j]
            if (temp >= max):
                max = temp
                X = i
                Y = j

```



```

        whole_zeroes.append(hor_arr[i]+ver_arr[j])
deleted_hor.append(X)
deleted_ver.append(Y)
return (X, Y)

```

порівняння нижніх границь

```
def compare_limit(A_star,shrink,branch):
```

```
    A_star[branch[0]][branch[1]] = 10000
```

```
    A_star_sum = (sum_min_hor(A_star) + sum_min_ver(A_star))
```

```
    shrink = np.delete(shrink, branch[0], axis=0)
```

```
    shrink = np.delete(shrink, branch[1], axis=1)
```

```
    shrink_sum = (sum_min_hor(shrink)+ sum_min_ver(shrink))
```

```
    if(shrink_sum<= A_star_sum):
```

```
        hor_count =0
```

```
        ver_count =0
```

```
        for i in range(len(deleted_hor)):
```

```
            if(branch[0]>=deleted_hor[i]):
```

```
                hor_count+=1
```

```
        for i in range(len(deleted_ver)):
```

```
            if (branch[1] >= deleted_ver[i]):
```

```
                ver_count += 1
```

```
        print('(' + (str(branch[0] + hor_count)) + ',' + (str(branch[1] + ver_count)) + ')')
```

```
        return shrink
```

```
for i in range(5):
```

```
    mylist = (reduct_hor(mylist))
```

```
    mylist = reduct_ver(mylist)
```

```
    branch = (find_branch(mylist))
```

```
    A_star = mylist.copy()
```

```
    shrink = np.asarray(mylist.copy())
```

```
    mylist = compare_limit(A_star,shrink,branch)
```

```
print("Довжина: " +str(down_limit))
```

Апробація програми:

```
(3,2)
(2,5)
(5,1)
(1,4)
(4,3)
Довжина: 36
```

Висновок: на цій лабораторній роботі я навчився розв'язувати задачу комів'язера методом гілок та границь, реалізував цю задачу на мові python та отримав оптимальну довжину подорожі мандрівника.