МІНІСТЕРСТВО ОСВІТИ І НАУКИ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»



Лабораторна робота №7 3 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

Nº	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.

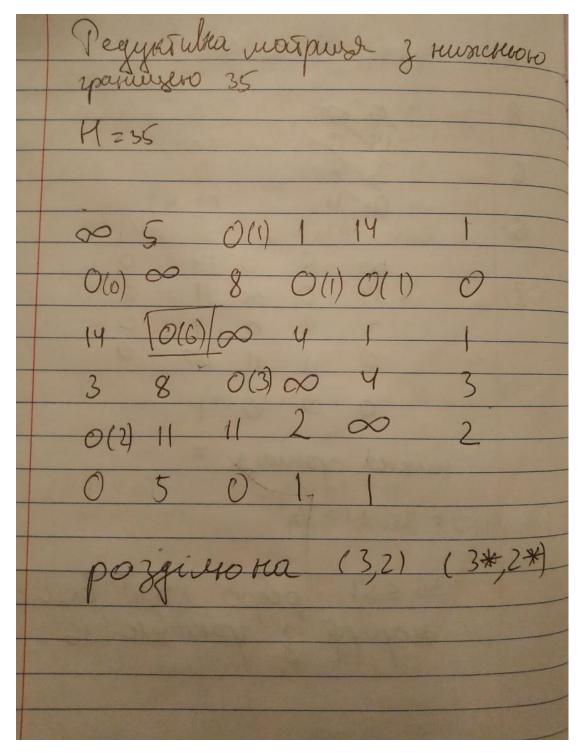
Отож, для початку потрібно провести операцію редукцію. Це означає, що потрібно вибрати мінімальний аргумент по кожній стрічці, та відняти цей аргумент від коженого з елементів. Це саме повторю і для вертикальних ліній.

Тут я редуктую початкову матрицю.

Saleman	L pro	oble	n		
(4)					
			min		-
2 10 3	5 14	19	5	The section of the se	
8 00	16 16	8	8		_
206	× 18	7	6	2	
9 14					
2 13	13 12	00	2	126	
					-
∞ 5 (9	14	5		
00			8	0	
1400	0 12	1	6		
200	7 00	4	6		
OIL	11 10	00	2	147 HE	19 5 10
A LATE OF	WY WY				
00	08	0			
				,	
005	01	14	5	13 7 90	
0 00	00	0	8	y Co 0	
140		1	6	20 O 11	
2 0	0 00	ч	6	0 2 5	
3 8		00	2	11 0	V
0 11	11 2			The state of the s	
00	0 8	0		H=5+8+6+6+	248=35

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

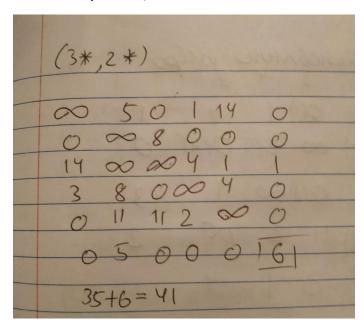
Наступним кроком ϵ визначення ребра, для цього потрібно всі нулі перетворити на безкінечності, тоді знайти мінімальний елемент по горизонталі та вертикалі кожного рядка, опісля до кожного нуля записати суму мінімальних елементів горизонтального та вертикального рядка, відповідно.



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

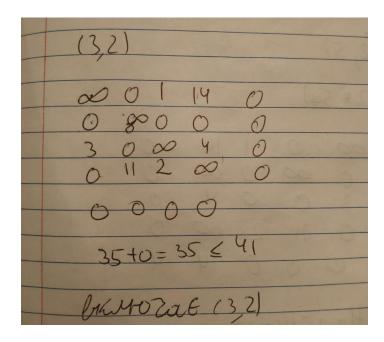
Отже, далі розгалужую задачі на дві різні підмножини (3,2) та (3*,2*).

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



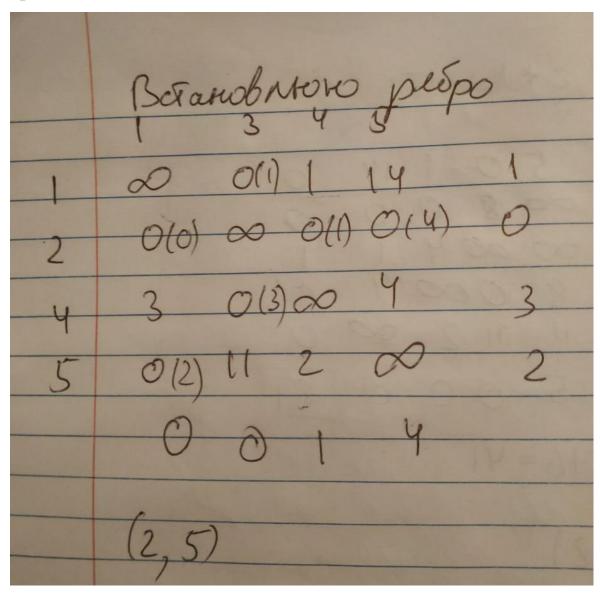
Для цієї матриці нижня межа 41.

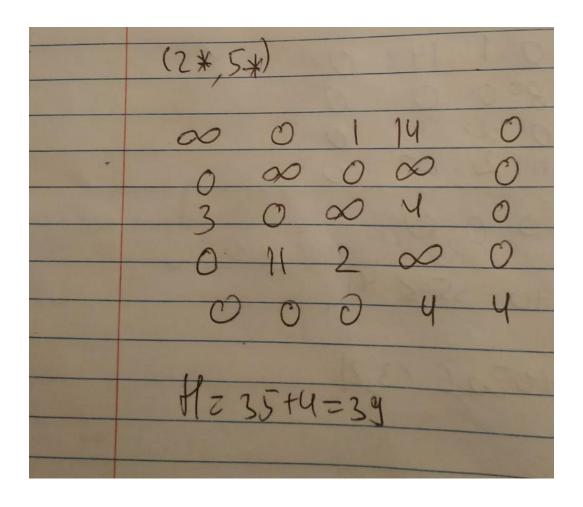
Для побудови наступної матриці потрібно видалити 3 горизонтальний рядок та 2 вертикальний. Опісля порахувати нижню межу цієї матриці, якщо ж нижня межа менша, рівна нижній межі матриці, яка наведена зверху, тоді це ребро можна додати у маршрут.



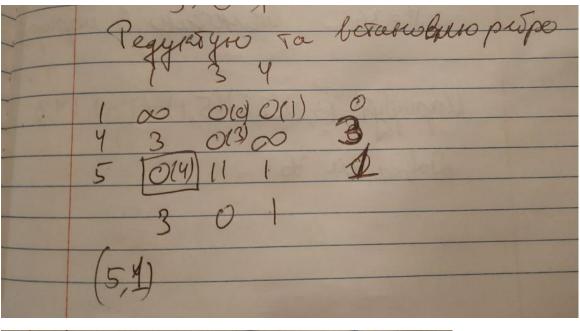
Нижня межа цієї матриці дорівнює 35, що менше 41, тоді додаю (3,2) до маршруту, якщо ж нерівність не виконується, тоді вибираємо ребро, базоване на найбільшому елементі, біля нуля.

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





100010 30000 501120 01120 11120 11120 11120 11120 11120 11120 11120 11120 11120 11120 11120 11120 11120 11120 11120 11120 11120 11120



	0000
	3 0 2 0
100	13004
	H=40
	(5,0) 3 4
	1000
	4000
	000
	9-36 By 1000 1
1	9=36 BRIGGAD 6 Magning
	Mapupy (3,2)(2,5)(5,1)(1,4)(4,3)
	Dobrayia 36
	77000000
1	
1	

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

Кодова частина задачі комовіяжера

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

Код задачі гілок та границь.

return A

```
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 мінімума
     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
```

```
#вертикальна редукція
def reduct_ver(A):
  # нижня границя
  global down_limit
  for i in range(len(A)):
     # перевірка на нан та ініціалізація мінімума
    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 мінімума
    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 мінімума
     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 мінімума
     for j in range(len(A[i])):
       if (i == j \text{ 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[i])
  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):</pre>
    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 комовіяжера методом гілок та границь, реалізував цю задачу на мові python та отримав оптимальну довжину подорожі мандрівника.