Прізвище: Пастернак
Ім'я: Вероніка
Група: КН-406
Варіант: 24
GitHub: https://github.com/veronikalpnu/8semest-kriviy
Кафедра: САПР
Дисципліна: Дискретні моделі САПР
Перевірив: Кривий Р.З.

ЗВІТ

до лабораторної роботи №3

на тему: "Алгоритм рішення задачі комівояжера"

Мета роботи:

вивчення і дослідження алгоритмів рішення задачі комівояжера.

Теоретичні відомості:

ФОРМУЛЮВАННЯ ТА ДЕЯКІ ВЛАСТИВОСТІ ЗАДАЧІ КОМІВОЯЖЕРА

Ми називаємо проблемою женця (оскільки це питання виникає в кожного листоноші, зокрема, її вирішують багато мандрівників) завдання віднайти найкоротший шлях між скінченною множиною місць, відстань між якими відома

Невдовзі з'явилась відома зараз назва задача мандруючого продавця. Для можливості застосування математичного апарату для розв'язання проблеми, її слід представити у вигляді математичної моделі. Проблему комівояжера можна представити у вигляді моделі на графі, тобто, використовуючи вершини та ребра між ними. Таким чином, вершини графу відповідають містам, а ребра *(i, j)* між вершинами *i* та *j* сполучення між цими містами. У відповідність кожному ребру *(i, j)* можна зіставити вагу $c_{ij} \geq 0$, яку можна розуміти як, наприклад, відстань між містами, час або вартість подорожі. ***Маршрутом (також гамільтоновим маршрутом)*** називається маршрут на цьому графі до якого входить по одному разу кожна вершина графа. Задача полягає у відшуканні найкоротшого маршруту.

З метою спрощення задачі та гарантії існування маршруту, зазвичай вважається, що модельний граф задачі є повністю зв'язним, тобто, що між довільною парою вершин існує ребро. Це можна досягти тим, що в тих випадках, коли між окремими містами не існує сполучення, вводити ребра з максимальною вагою (довжиною, вартістю тощо). Через велику довжину таке ребро ніколи не потрапить до оптимального маршруту, якщо він існує.

Загальною задачею комівояжера називають задачу пошуку маршруту найменшої довжини.

Задачею комівояжера називають задачу пошуку гамільторового контура найменшої довжини.

Контур комівояжера, який має найменшу довжину, називають ***оптимальним гамільтоновим контуром*** він є оптимальним рішенням задачі комівояжера. Оптимальний маршрут комівояжера не обов`язково є гамільтоновим контуром.

Виникає питання, в яких випадках гамільтоновий контур є рішенням загальної задачі комівояжера? Відповідь дає наступна теорема.

Теорема 1. Якщо для кожної пари вершин (x,y) виконується умова:

$$a(x,y) \leq a(x,z)+a(z,y) \text{ , для всіх } z \neq x \neq y. \quad (1)$$

то гамільтоновий контур є рішенням загальної задачі комівояжера на графі G (якщо рішення задачі взагалі існує).

Умова (1) говорить лише про те, що відстань безпосередньо від *x* до *y* ніколи не перевищує відстані від *x* до *y* через будь-яку іншу вершину *z*. Умову (1) називають ***нерівностю трикутника***.

УМОВИ ІСНУВАННЯ ГАМІЛЬТОНОВОГО КОНТУРУ. НИЖНІ ГРАНИЦІ.

Рішенням задачі комівояжера є оптимальний гамільтоновий контур. Нажаль, не всі графи містять гамільтоновий контур. Отже перед тим, ніж перейти до пошуку оптимального гамільтонового контура потрібно довести факт його існування в даному графі.

Граф називається ***сильно зв`язаним***, якщо в ньому для будь-яких двох вершин "*x*" і "*y*" існує шлях від "*x*" до "*y*". Підмножина вершин *X* деякого графа називається ***сильно зв`язаною***, якщо для будь-яких пар вершин *x - X* і *y - X* існує шлях з "*x*" в "*y*" і *X* не є підмножиною ніякої іншої множини вершин, які володіють тими ж властивостями. Загальна теорема Гуйя-Урі.

Теорема 2. Якщо граф G(X,A) задовільняє умовам :

1) граф G силно зв'язаний ;

2) d(x) ≥ n для всіх x ∈ X, де d(x) - степінь вершини x, n – кількість вершин, то він містить гамільтоновий контур.

На практиці достатньо легко встановити, чи задовільняє деякий граф умовам (I.) і (II.) теореми 2.

Умова (1) перевіряється застосуванням до кожної пари вершин алгоритма Флойда або алгоритма Данцига для побудови найкоротшого шляху. Ця умова виконується, якщо для кожної пари вершин в графі існує зв'язуючий їх шлях скінченної довжини.

Умова (2) легко перевіряється підрахунком дуг, інцидентних кожній вершині графа.

Умови існування гамільтонового циклу на неорієнтованому графі сформульовані Квателем. Нехай, як і раніше, через n позначимо кількість вершин в графі. Пронумеруємо вершини так, щоб виконувалось відношення:

$d(x_i) \square d(x_i) \square \ldots \square d(x_n)$

Теорема 3. Граф G=(X,.A) містить гамільтоновий цикл, якщо:

$n \geq 3$ *і* $d(x_k) \square k \square 1/2\ n \square d(x_{n-k})\ n\text{-}k$ *(2)*

Умову (2) легко перевірити. Необхідно впорядкувати вершини по зростанню їх степеней і перевірити, чи виконується умова для перших n / 2 вершин.

Розглянемо тепер методи розрахунку нижніх границь довжини оптимальних гамільтонових контурів на орієнтованому графі *G = (X,A)*. Якщо від довжини довільного гамільтонового контура відняти значення нижньої границі, то отримана різниця рівна максимальному значенню, на яке довжина гамільтонового контура може перевищувати довжину оптимального гамільтонового контура. Значення цієї вершини потрібно для оцінки різниці довжин знайденого і оптимального гамільтонового контура. Для орграфа дуги, що входять в гамільтоновий контур, повинні володіти наступними двома властивостями:

1) Кожна вершина повинна бути інцидентна двом іншим дугам, одна з яких направлена до неї, а інша від неї;
2) Всі дуги повинні бути зв'язані. Дуги, які входять в будь-яку множину незв'язаних контурів, що містять всі вершини, володіють властивістю 1.

Будь-яка зв'язана множина дуг володіє властивістю 2. І тільки гамільтоновий контур володіє властивостями 1 і 2 одночасно.

МЕТОДИ РІШЕННЯ ЗАДАЧІ КОМІВОЯЖЕРА

Можна знайти точний розв'язок задачі комівояжера, тобто, обчислити довжини всіх можливих маршрутів та обрати маршрут з найменшою довжиною. Однак, навіть для невеликої кількості міст в такий спосіб задача практично нерозв'язна. Для простого варіанта, симетричної задачі з n містами, існує (n − 1)! / 2 можливих маршрутів, тобто, для 15 міст існує 43 мільярдів маршрутів та для 18 міст вже 177 більйонів. Те, як стрімко зростає тривалість

обчислень можна показати в наступному прикладі. Якщо існував би пристрій, що знаходив би розв'язок для 30 міст за годину, то для для двох додаткових міст в тисячу раз більше часу; тобто, більш ніж 40 діб.

Відомо багато різних методів рішення задачі комівояжера. Серед них можна виділити методи розроблені Белмором і Немхаузером, Гарфинкелем і Немхаузером, Хелдом і Карном, Стекханом. Всі ці методи відносяться до одного з двох класів: а) методи рішення, які завжди приводять до знаходження оптимального рішення, але потребують для цього, в найгіршому випадку, недопустимо великої кількості операцій(метод гілок та границь); б) методи, які не завжди приводять до находження оптимального результату, але потребують для цього допустимої великої кількості операцій (метод послідовного покращення рішення).

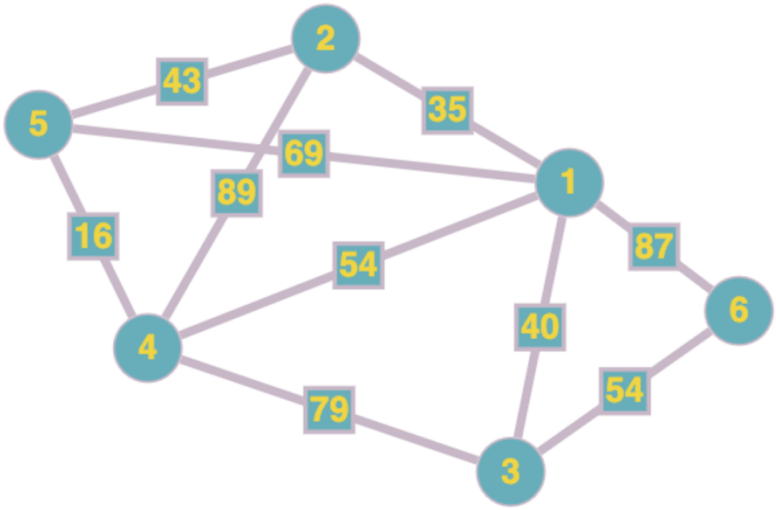
Індивідуальне завдання

1) Отримати у викладача індивідуальне завдання.
2) Підготувати програму для вирішення виданого завдання.
3) Запустити на виконання програму, що розв’язує задачу листоноші.
4) Проглянути результат роботи програми. Результат може бути позитивний (шлях знайдено) або негативний (шлях відсутній).
5) У випадку, коли шлях знайдено (не знайдено), необхідно модифікувати граф, коректуючи два або три зв’язки таким чином, щоб знайти граф, на якому задача комівояжера не вирішується (вирішується).
6) Здійснити перевірки роботи програм з результатами розрахунків, проведених вручну.
7) Зафіксувати результати роботи.
8) Оформити і захистити звіт

Ручні обчислення:

Дано матрицю суміжності у індивідуальному завданні (l3-3.txt):

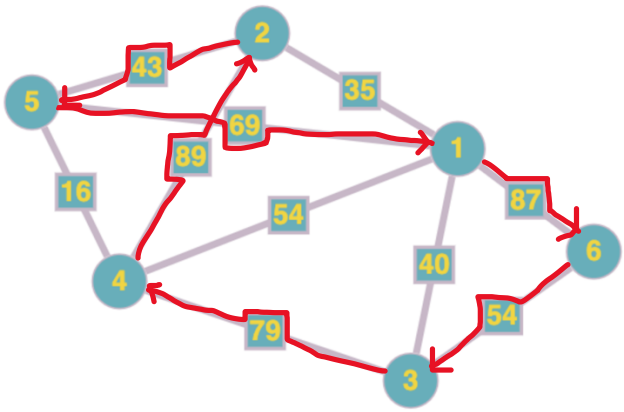
	1	2	3	4	5	6
1	0	35	40	54	69	87
2	35	0	0	89	43	0
3	40	0	0	79	0	54
4	54	89	79	0	16	0
5	69	43	0	16	0	0
6	87	0	54	0	0	0



1) Спочатку запишемо замість нулів матриці число максимально велике, для зручності та правильності обрахунків:

	1	2	3	4	5	6
1	999	35	40	54	69	87
2	35	999	999	89	43	999
3	40	999	999	79	999	54
4	54	89	79	999	16	999
5	69	43	999	16	999	999
6	87	999	54	999	999	999

2) Наступним кроком знайдемо випадковий Гамільтоновий цикл цього графа:
Ребра Гамільтонового циклу: *[1;6], [6,3], [3;4], [4;2], [2;5], [5;1].*
Отже, отриманий Гамільтоновий цикр: {1, 6, 3, 4, 2, 5}
Сума цього циклу: *69 + 87 + 54 + 79 + 89 + 35 = 413*



3) Для визначення нижньої межі множини скористаємося операцією редукції або приведення матриці рядками, для чого необхідно в кожному рядку матриці D знайти мінімальний елемент.

Сума отриманих елементів: 35 + 35 + 40 + 16 + 16 + 54 = 196

	1	2	3	4	5	6	
1	999	35	40	54	69	87	35
2	35	999	999	89	43	999	35
3	40	999	999	79	999	54	40
4	54	89	79	999	16	999	16
5	69	43	999	16	999	999	16
6	87	999	54	999	999	999	54

4) Потім віднімаємо мінімальні значення з елементів рядка, що розглядається. У зв'язку з цим у новоствореній матриці в кожному рядку буде щонайменше один нуль.
Таку ж операцію редукції проводимо по стовпцях, для чого в кожному стовпці знаходимо мінімальний елемент.

Сума отриманих елементів: 0 + 0 + 0 + 0 + 0 + 14 = 14

	1	2	3	4	5	6	
1	964	0	5	19	34	52	0
2	0	964	964	54	8	964	0
3	0	959	959	39	959	14	0
4	38	73	63	983	0	983	0
5	53	27	983	0	983	983	0
6	33	945	0	945	945	945	0
	0	0	0	0	0	14	

	1	2	3	4	5	6	
1	964	0	5	19	34	38	0
2	0	964	964	54	8	950	0
3	0	959	959	39	959	0	0
4	38	73	63	983	0	969	0
5	53	27	983	0	983	969	0
6	33	945	0	945	945	931	0
	0	0	0	0	0	0	

5) Після віднімання мінімальних елементів отримуємо повністю редуковану матрицю, де величини *196* та *14* називаються константами приведення.
Сума констант приведення визначає нижню межу: *196 + 14 = 210*

6) Визначаємо ребро розгалуження і розіб'ємо всі безліч маршрутів щодо цього ребра на дві підмножини *(i, j)* та *(i*, j*)*. З цією метою для всіх комірок матриці з нульовими елементами замінюємо по черзі нулі на якесь максимально велике число (в даному випадку *999*) і визначаємо для них суму констант приведення, що утворилися, вони наведені в комірках зеленого кольору.
d(1,2) = 5 + 27 = 32; d(2,1) = 8 + 0 = 8; d(3,1) = 0 + 0 = 0; d(3,6) = 0 + 38 = 38; d(4,5) = 38 + 8 = 46; d(5,4) = 27 + 19 = 46; d(6,3) = 33 + 5 = 38;

	1	2	3	4	5	6
1	964	0(32)	5	19	34	38
2	0(8)	964	964	54	8	950
3	0(0)	959	959	39	959	0(38)
4	38	73	63	983	0(46)	969
5	53	27	983	0(46)	983	969
6	33	945	0(38)	945	945	931

Найбільша сума констант приведення дорівнює *(38 + 8) = 46* для ребра *(4, 5)*, отже, безліч розбивається на два підмножини *(4, 5)* та *(4*, 5*)*.
Нижня межа гамільтонових циклів цієї підмножини:
H(4,5*) = 210 + 46 = 256*

7) Включення ребра *(4, 5)* проводиться шляхом виключення всіх елементів 4-го рядка та 5-го стовпця, в якому елемент *(5, 4)* замінюємо на *999*, для виключення утворення негамільтонова циклу.
В результаті отримаємо іншу скорочену матрицю (5 x 5), яка підлягає операції приведення.
Повторяємо кроки 1-4.

	1	2	3	4	6
1	964	0(32)	5	19	38
2	0(8)	964	964	54	950
3	0(0)	959	959	39	0(38)
5	53	27	983	999	969
6	33	945	0(38)	945	931

	1	2	3	4	6	
1	964	0(32)	5	19	38	0
2	0(8)	964	964	54	950	0
3	0(0)	959	959	39	0(38)	0
5	26	0	956	972	942	0
6	33	945	0(38)	945	931	0
	0	0	0	19	0	

	1	2	3	4	6	
1	964	0(32)	5	0	38	0
2	0(8)	964	964	35	950	0
3	0(0)	959	959	20	0(38)	0
5	26	0	956	953	942	0
6	33	945	0(38)	926	931	0
	0	0	0	0	0	

8) Сума констант приведення скороченої матриці: *27 + 19 = 46*
Нижня межа підмножини *(4, 5)* дорівнює:
H(4, 5) = 210 + 46 = 256 ≤ 256
Оскільки нижні межі підмножини *(4, 5)* та підмножини *(4*,5*)* рівні, то ребро *(4, 5)* включаємо в маршрут із новим кордоном *H = 256*

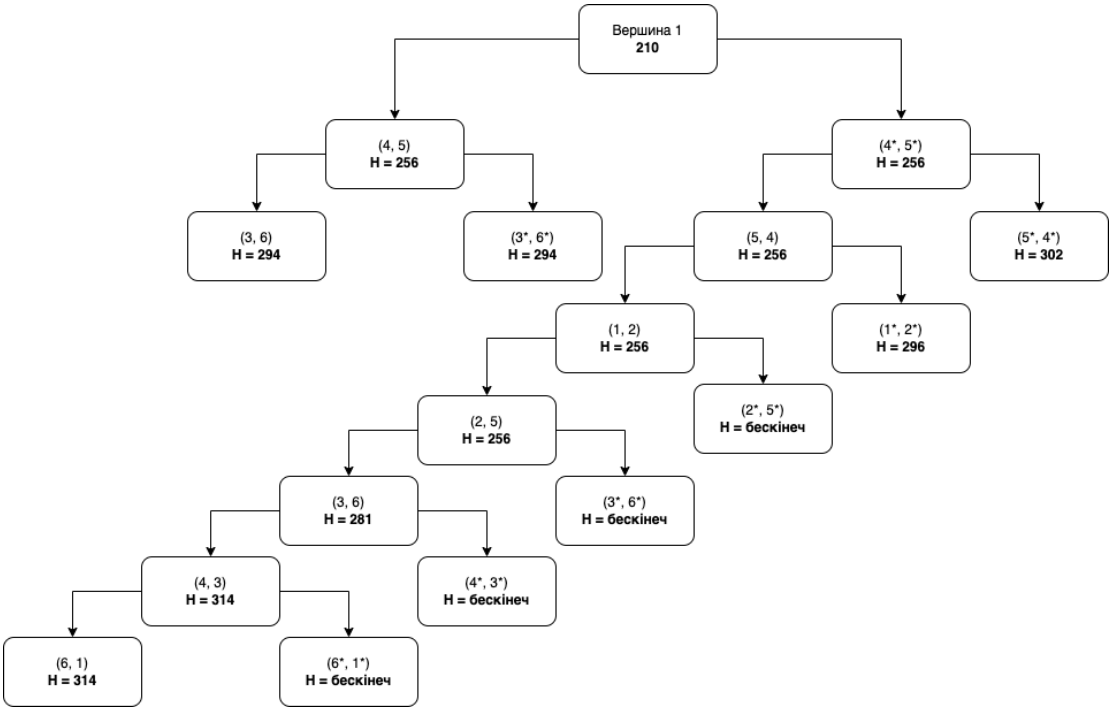
9) Визначаємо ребро розгалуження і розіб'ємо всі безліч маршрутів щодо цього ребра на два підмножини *(i, j)* та *(i*, j*)*. З цією метою для всіх комірок матриці з нульовими елементами замінюємо по черзі нулі на якесь максимально велике число (в даному випадку *999*) і визначаємо для них суму констант приведення, що утворилися, вони наведені в комірках зеленого кольору та з червоними значеннями. Найбільша сума констант приведення дорівнює *(0 + 38) = 38* для ребра *(3,6)*, отже, безліч розбивається на дві підмножини *(3, 6)* та *(3*, 6*)*.
Сума констант приведення скороченої матриці: 0 + 38 = 38
Нижня межа підмножини *(3, 6)* дорівнює:
H(3,6) = 256 + 38 = 294 ≤ 294
Оскільки *294 > 256*, виключаємо підмножину (4, 5) для подальшого розгалуження.
Повертаємось до колишнього плану.

	1	2	3	4	6
1	964	0(0)	5	0(20)	38
2	0(35)	964	964	35	950
3	0(0)	959	959	20	0(38)
5	26	0(26)	956	953	942
6	33	945	0(38)	926	931

10) Виключення ребра (3, 6) проводимо шляхом заміни елемента (6, 3) = 999, після чого здійснюємо чергове приведення матриці відстаней для підмножини (3*, 6*), що утворилося, в результаті отримаємо редуковану матрицю.

	1	2	3	4
1	964	0	5	0
2	0	964	964	35
5	26	0(26)	956	953
6	33	945	999	926

11) Обчислення кожної гілки здійснюємо доти, доки не утвориться перший цикл. В результаті отримаємо дерево рішень:



Програмна реалізація (python):

```
#Функція знаходження мінімального елемента, крім поточного елемента
def Min(lst,myindex):
    return min(x for idx, x in enumerate(lst) if idx != myindex)

#Функція видалення потрібного рядка та стовпчиків
def Delete(matrix,index1,index2):
    del matrix[index1]
    for i in matrix:
        del i[index2]
    return matrix

#Функція виведення матриці
def PrintMatrix(matrix):
    print("---------------")
    for i in range(len(matrix)):
        print(matrix[i])
    print("---------------")

n=int(input())
matrix=[]
H=0
PathLenght=0
Str=[]
Stb=[]
res=[]
result=[]
StartMatrix=[]

#Ініціалізуємо масиви для збереження індексів
for i in range(n):
```

```
    Str.append(i)
    Stb.append(i)

#Вводимо матрицю
for i in range(n): matrix.append(list(map(int, input().split())))

#Зберігаємо початкову матрицю
for i in range(n):StartMatrix.append(matrix[i].copy())

#Привласнюємо головній діагоналі float(inf)
for i in range(n): matrix[i][i]=float('inf')

while True:
     #Редукуємо
     #--------------------------------------
     #Віднімаємо мінімальний елемент у рядках
    for i in range(len(matrix)):
        temp=min(matrix[i])
        H+=temp
        for j in range(len(matrix)):
            matrix[i][j]-=temp

    #Віднімаємо мінімальний елемент у стовпцях
    for i in range(len(matrix)):
        temp = min(row[i] for row in matrix)
        H+=temp
        for j in range(len(matrix)):
            matrix[j][i]-=temp
    #--------------------------------------

    #Оцінюємо нульові клітини та шукаємо нульову клітину з максимальною оцінкою
    #--------------------------------------
    NullMax=0
    index1=0
    index2=0
    tmp=0
    for i in range(len(matrix)):
        for j in range(len(matrix)):
            if matrix[i][j]==0:
                tmp=Min(matrix[i],j)+Min((row[j] for row in matrix),i)
                if tmp>=NullMax:
                    NullMax=tmp
                    index1=i
                    index2=j
    #--------------------------------------

    #Знаходимо потрібний нам шлях, записуємо його в res і видаляємо все непотрібне
    res.append(Str[index1]+1)
    res.append(Stb[index2]+1)

    oldIndex1=Str[index1]
    oldIndex2=Stb[index2]
    if oldIndex2 in Str and oldIndex1 in Stb:
        NewIndex1=Str.index(oldIndex2)
```

```
        NewIndex2=Stb.index(oldIndex1)
        matrix[NewIndex1][NewIndex2]=float('inf')
    del Str[index1]
    del Stb[index2]
    matrix=Delete(matrix,index1,index2)
    if len(matrix)==1:break

#Формуємо порядок шляху
for i in range(0,len(res)-1,2):
    if res.count(res[i])<2:
        result.append(res[i])
        result.append(res[i+1])
for i in range(0,len(res)-1,2):
    for j in range(0,len(res)-1,2):
        if result[len(result)-1]==res[j]:
            result.append(res[j])
            result.append(res[j+1])
print("----------------------------------")
print(result)

#Рахуємо довжину колії
for i in range(0,len(result)-1,2):
    if i==len(result)-2:
        PathLenght+=StartMatrix[result[i]-1][result[i+1]-1]
        PathLenght+=StartMatrix[result[i+1]-1][result[0]-1]
    else: PathLenght+=StartMatrix[result[i]-1][result[i+1]-1]
print(PathLenght)
print("----------------------------------")
input()
```

Результати виконання програми:

```
MacBook-Pro-Verik:Lab-3 verikpaster$  cd
y/launcher 60388 -- /Users/verikpaster/D
6
999 35 4999 54 69 87
35 999 999 89 43 999
4999 999 999 79 999 54
54 89 79 999 16 999
69 43 999 16 999 999
87 999 54 999 999 999
----------------------------------
[1, 6, 6, 3, 3, 4, 4, 5, 5, 2]
314
----------------------------------
```

Висновок

В ході даної лабораторної роботи я ознайомлась з поняттями Гамільтонів граф, Гамільтонів маршрут та Гамільтонів цикл, також на основі цих понять відбулося вивчення алгоритму комівояжера. Було проведені ручні обчислення для обходу графа, які відображені в звіті, а також реалізовано програму, яка буде виконувати аналогічні обрахунки. Результати програми і ручних обчислень зійшлися.