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 №4
на тему: “Потокові алгоритми”

Мета роботи:

вивчення потокових алгоритмів.

Теоретичні відомості:

ПОНЯТТЯ ПРО ПОТОКИ

Потік-визначає спосіб пересилання деяких об’єктів з одного пункту в інший. Розв’язання задачі потоку зводиться до таких основних підзадач:

- Максимізація сумарного обсягу перевезень
- Мінімізація вартості пересилань предметів з одного пункту в інший
- Мінімізація часу перевезень в заданій системі

Сітка - це граф, в якому кожній дузі приписана деяка пропускна здатність. Введемо позначення: $c(x, y)$ - пропускна здатність дуги (x, y), $a(x, y)$ - вартість переміщення одиниці потоку по дузі (x, y), $T(x, y)$ - час проходження потоку, $k(x, y)$ - коефіцієнт підсилення потоку в дузі (x, y).

Припустимо, що є граф, в якому деяка кількість одиниць потоку проходить від джерела до стоку і для кожної одиниці потоку відомий маршрут руху. Назвемо кількість одиниць, що проходять по дузі (x, y), потоком в даній дузі. Будемо потік в дузі (x, y) позначати через $f(x, y)$ вочевидь $0 \leq f(x, y) \leq c(x, y)$.

Дуги графа можна віднести до трьох різних категорій:

1. дуги, в яких потік не може ні збільшуватись, ні зменшуватись (множина таких дуг позначається через - N);
2. дуги, в яких потік може збільшуватись (множина таких дуг позначається через - I);
3. дуги, в яких потік може зменшуватись (множина таких дуг позначається через - R);

Дуги, що мають нульову пропускну здатність або значну вартість проходження потоку, повинні належати множині N. Дуги, в яких потік менше пропускної здатності, повинні належати множині I. Дуги, по яких вже проходить деякий потік, повинні належати множині R. Дуги з множини I називають збільшуючими, а дуги з множини R - зменшуючими.

Будь-яка дуга графа належить хоча б одній з трьох введених множин - I, R або N. Можливо, що якась дуга належить як множині I, так і множині R. Це має місце в тому випадку, коли по дузі вже протікає деякий потік, який можна збільшувати чи зменшувати. Відповідні дуги називаються проміжними.

ПОНЯТТЯ ПОТОКОВИХ АЛГОРИТМІВ

Поняття потокові алгоритми включає в себе ряд алгоритмів

1) **Алгоритм пошуку збільшую чого ланцюга.**

Основна ідея алгоритму: побудова дерева, що росте з вершини "*s*" і складається з розфарбованих дуг, по яких з вершини "*s*" можуть пересилатись додаткові одиниці потоку. В процесі виконання алгоритму можуть виникнути дві різні ситуації:

a. стік "*t*" є розфарбованим???, тоді в побудованому з розфарбованих дуг дереві єдиний ланцюг з "*s*" в "*t*" є збільшуючим потік ланцюгом;
b. стік "*t*" не вдається розфарбувати, що означає: що у вихідній сітці не існує збільшуючого ланцюга між "*s*" і "*t*".

2) **Алгоритм пошуку максимального потоку.**

Основна задача алгоритму пошуку максимального потоку полягає в пошуку способів пересилання максимальної кількості одиниць потоку з витоку в стік при умові відсутності перевищення пропускних здатностей дуг вихідного графа. В основі алгоритму пошуку максимального потоку лежить наступна ідея: вибираємо початковий потік з витоку "*s*" в стік "*t*", потім використовуємо алгоритм пошуку збільшуючого ланцюга. Цей алгоритм дозволяє знайти єдиний збільшуючий ланцюг з "*s*" в "*t*", якщо той існує. Послідовність, в якій повинні розфарбовуватись вершини і дуги, конкретно не визначається. Тому можливі два варіанти розфарбування вершин і дуг:

1. вибирається яка-небудь вершина, а потім проводиться розфарбування максимальної кількості вершин;
2. проводиться розфарбування, виходячи з останньої розфарбованої вершини.

Який саме спосіб розфарбування буде вибраний, буде залежати від характеру більш загальної задачі, тобто від алгоритму пошуку максимального потоку, який в якості підалгоритму використовує алгоритм пошуку збільшуючого ланцюга. Якщо пошук збільшуючого ланцюга вдалий, тобто знайдено збільшуючий ланцюг з "*s*" в "*t*", то за допомогою алгоритму пошуку максимального потоку здійснюється максимально можливе збільшення потоку вздовж знайденого ланцюга. Потім повторюється пошук нового збільшуючого ланцюга і т.д. Виконання алгоритму завершується за скінчене число кроків, коли ланцюг, що збільшує потік, знайти не вдається: це означає, що біжучий потік з "*s*" в "*t*" є максимальним.

3) **Алгоритм пошуку потоку мінімальної вартості.**

Дана задача полягає в організації пересилання з мінімальними витратами заданої кількості v одиниць потоку з витоку в стік в графі з заданими на дугах пропускними здатностями і вартостями проходження одної одиниці потоку.

4) **Алгоритм дефекту.**

Основна ідея алгоритму: рішення задачі про потік мінімальної вартості, але на відміну від попереднього алгоритму алгоритм дефекту вирішує задачу про потік мінімальної вартості у випадку, коли найменша кількість одиниць потоку, яка повинна протікати по дузі, більша або рівна 0 для всіх дуг.

5) **Алгоритм пошуку динамічного потоку.**

Попередні алгоритми використовували потоки, які задовольняли деяким умовам, які визначались заданими на дугах пропускними здатностями і вартостями. Даний алгоритм використовує сітки, дуги яких характеризуються ще одним показником - часом проходження потоку (кожна одиниця в цих потоках проходить з витоку в стік за час, що не перевищує заданий).

6) **Потоки з підсиленням.**

Попередні розгляди потоку показували, що потік не змінювався: одиниця ввійшла - одиниця вийшла. При проходженні потоку через дугу нові одиниці не створювались, але і старі не щезали. Потоки з підсиленням усувають припущення, згідно якого при проходженні по дугам потік залишається незмінним. Висувається припущення, що кількість одиниць в потоці, що проходить по дузі, може збільшуватись або зменшуватись. Точніше, вважають, що якщо в будь-яку дугу *(x, y)* в вершині "*x*" входить *f(x, y)* одиниць потоку, то з цієї дуги в вершині "*y*" вийде *k(x ,y) * f(x, y)* одиниць потоку. Можна вважають, що кожна одиниця потоку, що проходить по дузі *(x, y)*, помножується на величину *k(x, y)* (яка називається підсиленням дуги *(x, y)*.

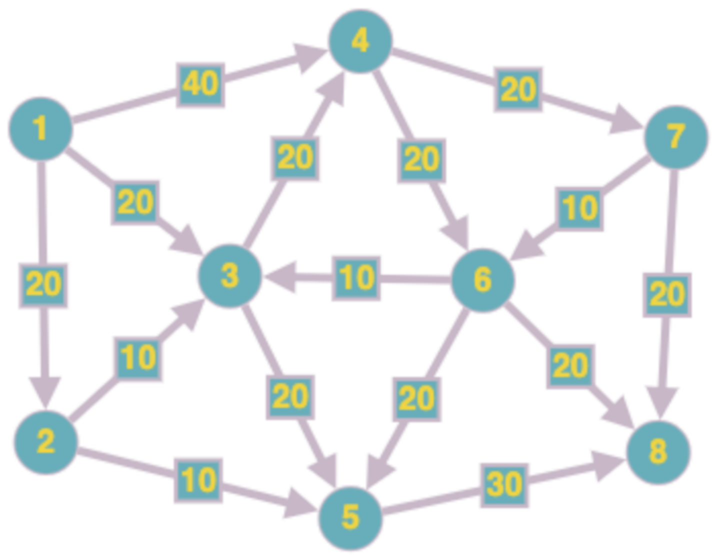
Індивідуальне завдання

1) Отримати у викладача індивідуальне завдання.
2) Підготувати програму для вирішення виданого завдання.
3) Запустити на виконання програму, що розв’язує задачу пошуку збільшуючого ланцюга.
4) Проглянути результат роботи програми. Результат роботи програми, що шукає збільшуючий ланцюг, може бути позитивний (стік є розфарбований) або негативний (стік не вдається розфарбувати).
5) У випадку, коли результат позитивний (або негативний) необхідно модифікувати граф (коректуючи два або три зв’язки), що дозволить знайти такий граф, на якому стік, відповідно, не вдається розфарбувати (розфарбовується).Здійснити перевірки роботи програм з результатами розрахунків, проведених вручну.
6) Зафіксувати результати роботи.
7) Оформити і захистити звіт

Ручні обчислення:

Дано матрицю суміжності індивідуальне завдання (l4-2.txt):

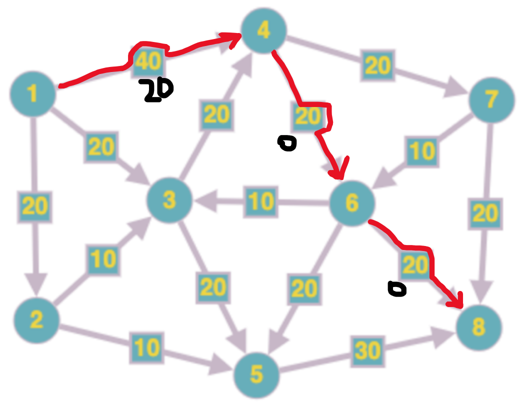
	1	2	3	4	5	6	7	8
1	0	20	20	40	0	0	0	0
2	0	0	10	0	10	0	0	0
3	0	0	0	20	20	0	0	0
4	0	0	0	0	0	20	20	0
5	0	0	0	0	0	0	0	30
6	0	0	10	0	20	0	0	20
7	0	0	0	0	0	10	0	20
8	0	0	0	0	0	0	0	0



Спочатку для всіх ребер відмітимо залишкову пропускну здатність. Потік усіх ребер на початку дорівнює 0.

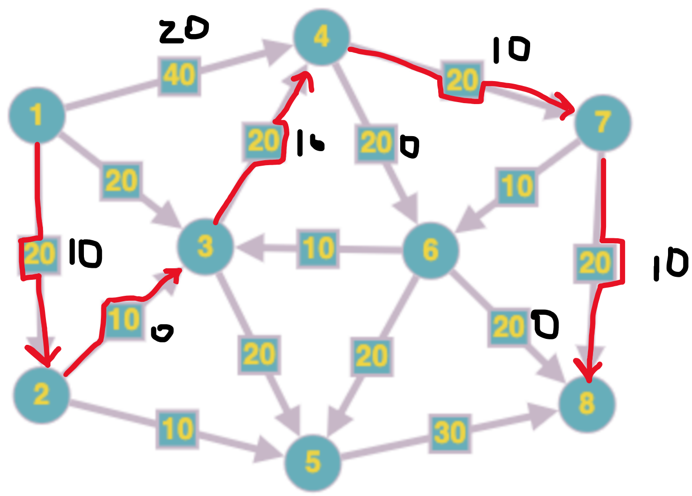
Крок 1.

Виберіть будь-який довільний шлях від 1 до 8. На цьому кроці ми вибрали шлях:
1 –(40)–> 4 –(20)–> 6 –(20)–> 8
Мінімальна ємність серед трьох ребер становить 20. На основі цього оновлюємо потік/ємність для кожного шляху.



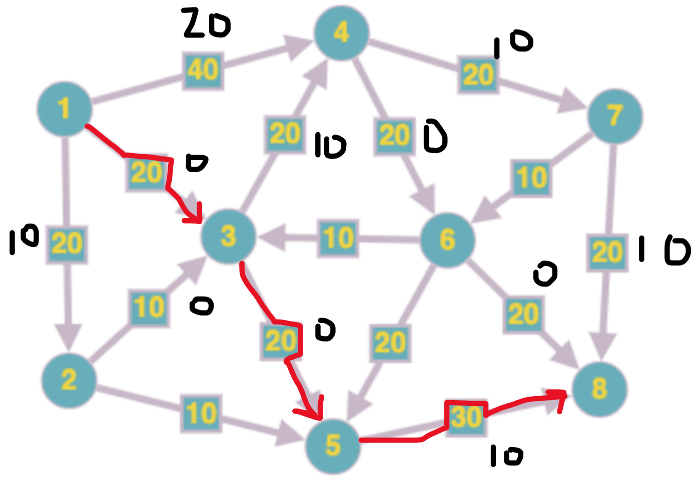
Крок 2.

Вибираємо інший шлях:
1 –(20)–> 2 –(10)–> 3 –(20)–> 4 –(20)–> 7 –(20)– 8
Мінімальна ємність серед цих ребер становить 10. Знову ж таки оновлюємо потік/ємність для кожного шляху.



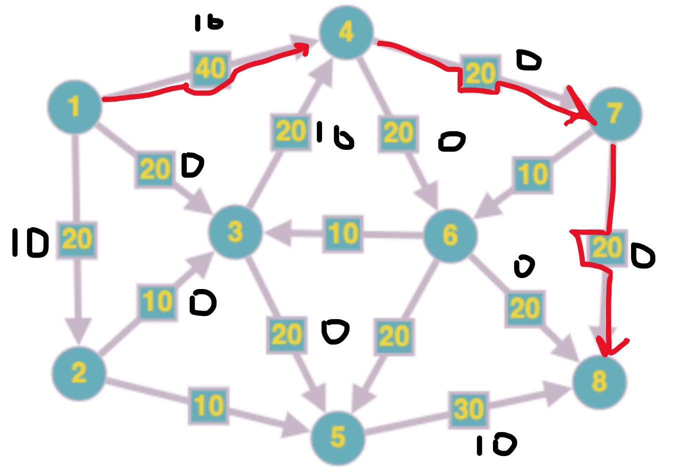
Крок 3.

Вибираємо наступний шлях:
1 –(20)–> 3 –(20)–> 5 –(30)–> 8
Мінімальна ємність серед цих ребер становить 20. Знову оновлюємо потік/ємність для кожного шляху.



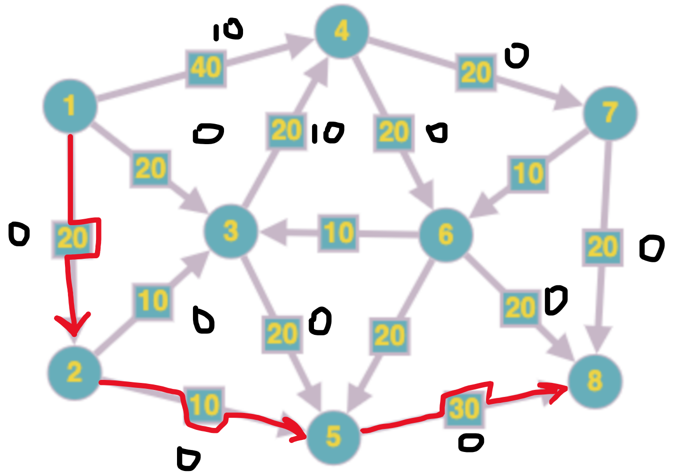
Крок 4.

Вибираємо наступний шлях:
1 –(20)–> 4 –(10)–> 7 –(10)–> 8
Мінімальна ємність серед цих ребер становить 10. Знову оновлюємо потік/ємність для кожного шляху.



Крок 5.

Вибираємо наступний шлях:
1 –(10)– >2 –(10)–> 5 –(10)–> 8
Мінімальна ємність серед цих ребер становить 10. Знову оновлюємо потік/ємність для кожного шляху.



Крок 6.

На цьому кроці ми й зупинимося, так як ребра, які входять у вершину 8 мають повну ємність. Отже максимальна ємність потоку становить:
<u>20 + 10 + 20 + 10 + 10 = 70</u>

Програмна реалізація (python):

```
import math

def get_max_vertex(k, V, S):
    m = 0   # найменше допустими значення
    v = -1
    for i, w in enumerate(V[k]):
        if i in S:
            continue

        if w[2] == 1:   # рух за стрілкою
            if m < w[0]:
                m = w[0]
                v = i
        else:           # рух проти стрілки
            if m < w[1]:
                m = w[1]
                v = i

    return v


def get_max_flow(T):
    w = [x[0] for x in T]
    return min(*w)


def updateV(V, T, f):
    for t in T:
        if t[1] == -1:  # це граф
            continue
```

```
        sgn = V[t[2]][t[1]][2]  # напрямок руху

        # міняємо ваги в таблиці для (i,j) і (j,i)
        V[t[1]][t[2]][0] -= f * sgn
        V[t[1]][t[2]][1] += f * sgn

        V[t[2]][t[1]][0] -= f * sgn
        V[t[2]][t[1]][1] += f * sgn


V = [[[0,0,1], [20,0,1], [20,0,1], [40,0,1], [0,0,1], [0,0,1], [0,0,1], [0,0,1]],
     [[20,0,-1], [0,0,1], [10,0,1], [0,0,1], [10,0,1], [0,0,1], [0,0,1], [0,0,1]],
     [[20,0,-1], [10,0,-1], [0,0,1], [20,0,1], [20,0,1], [10,0,-1], [0,0,1], [0,0,1]],
     [[40,0,-1], [0,0,1], [20,0,-1], [0,0,1], [0,0,1], [20,0,1], [20,0,1], [0,0,1]],
     [[0,0,1], [10,0,-1], [20,0,-1], [0,0,1], [0,0,1], [20,0,-1], [0,0,1], [30,0,1]],
     [[0,0,1], [0,0,1], [10,0,1], [20,0,-1], [20,0,1], [0,0,1], [10,0,-1], [20,0,1]],
     [[0,0,1], [0,0,1], [0,0,1], [20,0,-1], [0,0,1], [10,0,1], [0,0,1], [20,0,1]],
     [[0,0,1], [0,0,1], [0,0,1], [0,0,1], [30,0,-1], [20,0,-1], [20,0,-1], [0,0,1]],
]

N = len(V)  # число вершин в графі
init = 0    # вершина початку(нумерация с нуля)
end = 7     # вершина кінця
Tinit = (math.inf, -1, init)      # первая мітка (a, from, vertex)
f = []      # максимальні потоки знайдених маршрутів

j = init
while j != -1:
    k = init  # стартова вершина (нумерация с нуля)
    T = [Tinit]  # мітка маршрута
    S = {init}  # множина пройдених вершин

    while k != end:     # поки не дішли до кінця
        j = get_max_vertex(k, V, S)  # вибираємо вершину з найбільшой пропускною
здібністю
        if j == -1:         # якшо наступних вершин нема
            if k == init:      # і ми на початку, то
                break          # завершуємо пошук маршрутів
            else:           # чи, переходимо до попередньої вершини
                k = T.pop()[2]
                continue

        c = V[k][j][0] if V[k][j][2] == 1 else V[k][j][1]   # визначаємо теперішній
поток
        T.append((c, j, k))    # добавляєм мітку маршрута
        S.add(j)            # запамятовуємо вершину як пройдену

        if j == end:    # якшо дійшли до кінця
            f.append(get_max_flow(T))     # знаходимо максимальну пропускну здібність
маршрута
            updateV(V, T, f[-1])        # обновляєм веги дуг
            break

        k = j
```

```python
F = sum(f)
print(f"Максимальний потік рівний: {F}")
```

Результати виконання програми:

```
cher 61644 -- /Users/verikpaster/Docum
Максимальний потік рівний: 70
MacBook-Pro-Verik:Lab-4 verikpaster$
```

Висновок

В ході даної лабораторної роботи я ознайомилась з поняттями потоків в орієнтованих графах. Було проведені ручні обчислення для заданого графа, які відображені в звіті, а також реалізовано програму, яка буде виконувати аналогічні обрахунки. Результати програми і ручних обчислень зійшлися.