Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Алгоритми та складність

Алгоритм Джонсона

Виконав Мотрич Максим

Студент групи К-28

2022

**Постановка задачі:** Алгоритм Джонсона для розріджених графів (включає алгоритми Беллмана-Форда і Дейкстри). В алгоритмі Дейкстри використайте піраміду Фібоначчі. Вагидуг задаються дійсними числами. Вершини – раціональні числа(мая значення лише при побудові

**Теоретичні відомості:** Теорія Алгоритм Джонсона дозволяє знайти найкоротші шляхи між усіма парами вершин зваженого орієнтованого графа. Цей алгоритм працює, якщо у графі містяться ребра з додатною чи від'ємною вагою, але відсутні цикли з від'ємною вагою. В алгоритмі Джонсона використовують алгоритм Беллмана-Форда та алгоритм Дейкстри втілені у вигляді підпрограм. Ребра зберігають у вигляді переліків суміжних вершин. Алгоритм повертає звичайну матрицю 𝐷 = 𝑑𝑖𝑗 розміром 𝑛 × 𝑛 або видає повідомлення про те, що вхідний граф містить цикл із від'ємною вагою.

Алгоритм Дейкстри – алгоритм на графах, відкритий Дейкстрою. Знаходить найкоротший шлях від однієї вершини графа до всіх інших вершин. Класичний алгоритм Дейкстри працює тільки для графів без циклів від’ємної довжин.

1. Кожній вершині з V зіставимо мітку - мінімальне відоме відстань від цієї вершини до a. Алгоритм працює покроково - на кожному кроці він «відвідує» одну вершину і намагається зменшувати мітки. Робота алгоритму завершується, коли всі вершини відвідані.

2. Ініціалізація. Мітка самої вершини a покладається рівною 0, мітки інших вершин - нескінченності. Це відображає те, що відстані від a до інших вершин поки невідомі. Всі вершини графа позначаються як невідвіданих.

3. Крок алгоритму Дейкстри : Якщо все вершини відвідані, алгоритм завершується. В іншому випадку, з ще не відвіданих вершин вибирається вершина u, що має мінімальну позначку.

Ми розглядаємо різні маршрути, в яких u є передостаннім пунктом. Вершини, в які ведуть ребра з u, назвемо сусідами цієї вершини. Для кожного сусіда вершини u, крім позначених відвідані, розглянемо нову довжину шляху, що дорівнює сумі значень поточної мітки u і довжини ребра, що з'єднує u з цим сусідом. Якщо отримане значення довжини менше значення мітки сусіда, замінимо значення мітки отриманим значенням довжини. Розглянувши всіх сусідів, позначимо вершину u як відвіданих і повторимо крок алгоритму. Усі невідвідані вершини графу зберігаються у Піраміді Фібоначчі, яка з кожним кроком алгоритму зменшуються тобто в кожному кроці алгоритму Дейкстри з неї видаляються відвідані вершини, і так як видалення з піраміди Фібоначчі має середню складність 𝑂(log 𝑛) то час роботи алгоритму Дейкстри зменшується з 𝑂() до 𝑂(𝑛 log 𝑛 + 𝑚) що і зменшує складність самого алгоритму Джонсона

Алгоритм Беллмана-Форда – алгоритм пошуку найкоротшого шляху в зваженому графі. Знаходить найкоротші шляхи від однієї вершини графа до всіх інших. На відміну від алгоритму Дейкстри, алгоритм Беллмана-Форда допускає ребра з негативною вагою. Запропоновано незалежно Річардом Беллманом і Лестером Фордом. Для знаходження найкоротших шляхів від однієї вершини до всіх інших, скористаємося методом динамічного програмування.

**1** На першому кроці не започатковано відстані від вихідної вершини до всіх інших вершин, як нескінченні, а відстань до самого src приймається рівним 0. Створюється масив dist [] розміру | V | з усіма значеннями рівними нескінченності, за винятком елемента dist [src], де src - вихідна вершина.

**2** Другим кроком обчислюються найкоротші відстані. Наступні кроки потрібно виконувати | V | -1 раз, де | V | - число вершин в даному графі. Проведіть наступна дія для кожного ребра u-v: Якщо dist [v]> dist [u] + вага ребра uv, то поновіть dist [v] dist [v] = dist [u] + вага ребра uv

**3** На третьому кроці повідомляється, чи присутній в графі цикл негативного ваги. Для кожного ребра u-v необхідно виконати наступне: Якщо dist [v]> dist [u] + вага ребра uv, то в графі присутній цикл негативного ваги.

Алгоритм

• Перевірка графа на від’ємні цикли ,у разі виявлення алгоритм Повертає інформацію про те що алгоритм неможливий завершує свою роботу

• Спочатку до графу додається новий вузол q, пов'язаний ребрами з нульовим вагою з кожним з інших вузлів

• По-друге, алгоритм Беллмана - Форда використовується, починаючи з нової вершини q, для знаходження для кожної вершини v мінімальної ваги h (v) шляху з q в v.

• Потім ребра вихідного графа повторно зважуються з використанням значень, обчислених алгоритмом Беллмана - Форда: ребру від u до v, має довжину, дається нова довжина w (u, v) + h (u) - h (v).

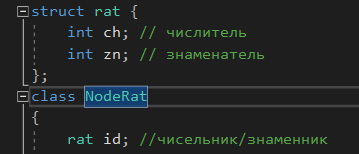
• Нарешті, q видаляється, і алгоритм Дейкстри використовується для пошуку найкоротших шляхів від кожного вузла s до кожної іншої вершини в переглянутому графі. Відстань у вихідному графі потім обчислюється для кожної відстані D (u, v) шляхом додавання h (v) - h (u) до відстані, що повертається алгоритмом Дейкстри

**Складність** Якщо в алгоритмі Дейкстри неспадну чергу з пріоритетами втілено у вигляді піраміди Фібоначчі, то тривалість роботи алгоритму Джонсона дорівнює 𝑂(𝑉𝐸 + log )

**Мова програмування** С++

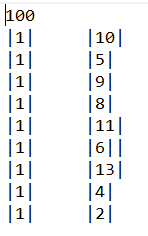
**Модулі програми** main.cpp, NodeRat.cpp, NodeRat.h, Graph.h, Graph.cpp, FibonacciHeap.h, FibonacciHeap.cpp

В NodeRat задається вузол з раціональними числами

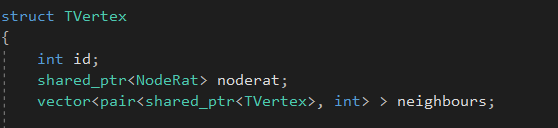


Дані зчитуються із текстового файлу

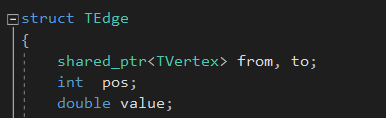




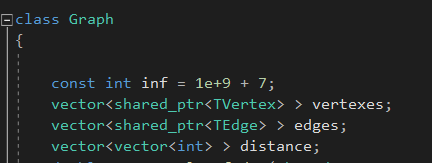
Далі в graph ідудь вершини з сусідами



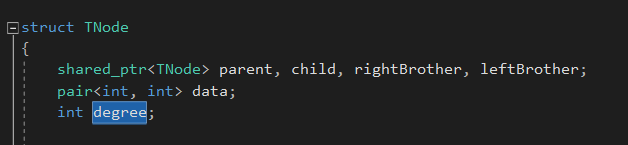
Структура ребер



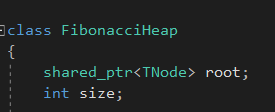
Й сам клас ребер з відстанню



Далі в FibonacciHeap знаходиться вузол

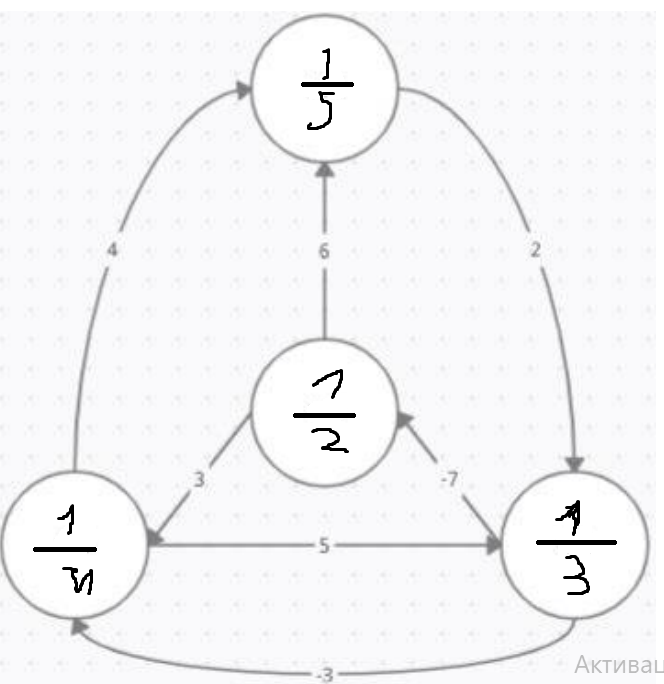


І клас

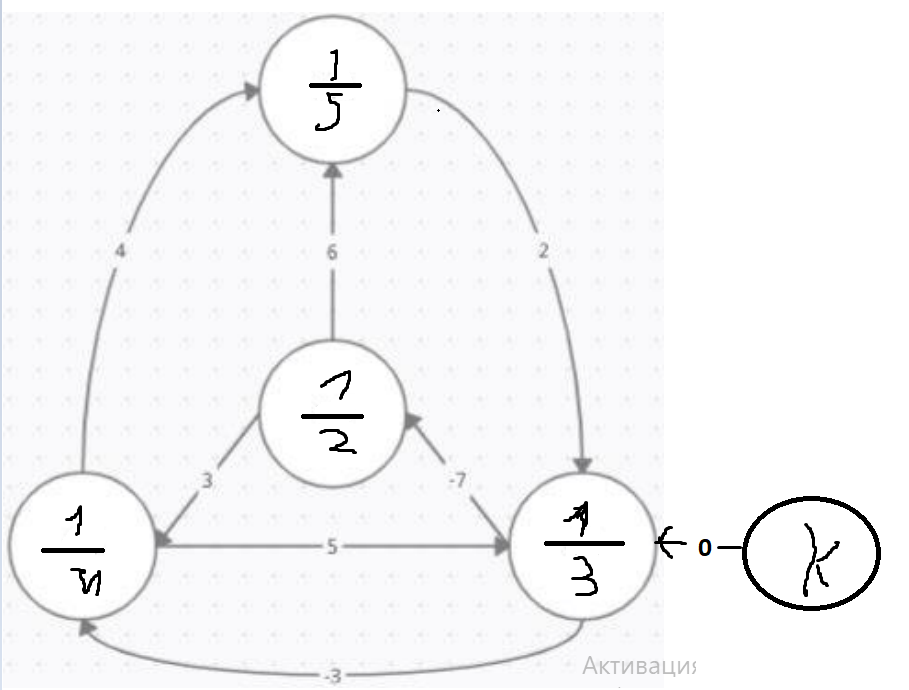


Тепер пройдемося по тому, як цей алгоритм працює на практиці:

Нехай маємо множину Груп взаємне положення яких множина подати у вигляді орієнтованого графу



Додаємо вершину к



Шукаємо для всіх вершин найкоротші шляхи до к за Алгоритмом Беллмана-Форда

h(1/4) =3+(-7) = -4

h(1/3)=0

h(1/2)=-7

h(1/5)=6+(-7)= -1

далі для кожного ребра w (u,v) даємо йому нове значення w (u,v) + h (u) - h (v)

𝑤′ (1/4, К10) = 𝑤(1/4, К10) + ℎ(1/4) − ℎ(К10) = 4 + (−4) − (−1) = 1

𝑤 ′ (1/5, 1/3) = 𝑤(1/5, 1/3) + ℎ(1/5) − ℎ(1/3) = 2 + (−1) − 0 = 1

𝑤 ′ (1/3, 1/4) = 𝑤(1/3, 1/4) + ℎ(1/3) − ℎ(1/4) = −3 + 0 − (−4) = 1

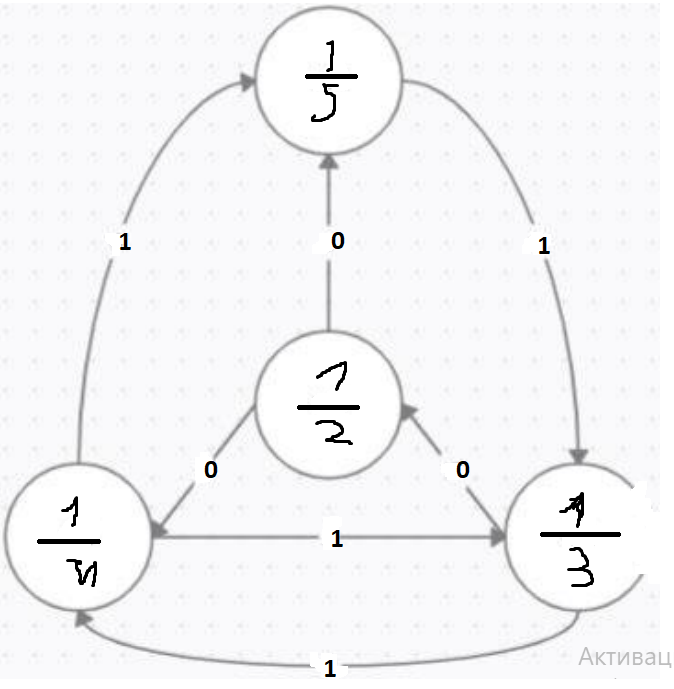
𝑤 ′ (1/4, 1/3) = 𝑤(1/4, 1/3) + ℎ(1/4) − ℎ(1/3) = 5 + (−4) − 0 = 1

𝑤 ′ (1/2, 1/3) = 𝑤(1/2, 1/3) + ℎ(1/2) − ℎ(1/3) = −7 + 0 − (−7) = 0

𝑤 ′ (1/4, К13) = 𝑤(1/4, К13) + ℎ(1/4) − ℎ(К13) = 3 + (−7) − (−4) = 0

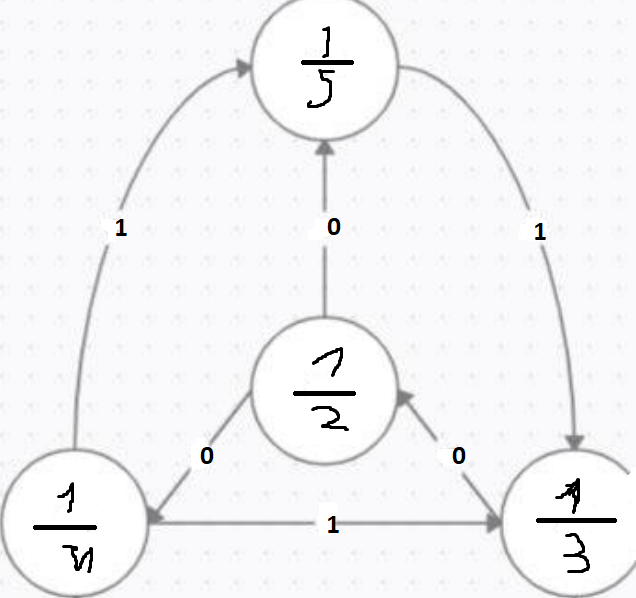
𝑤 ′ (1/4, К10) = 𝑤(1/4, К10) + ℎ(1/4) − ℎ(К10) = 6 + (−7) − (−1) = 0

Далі граф має вигляд з новими вагами

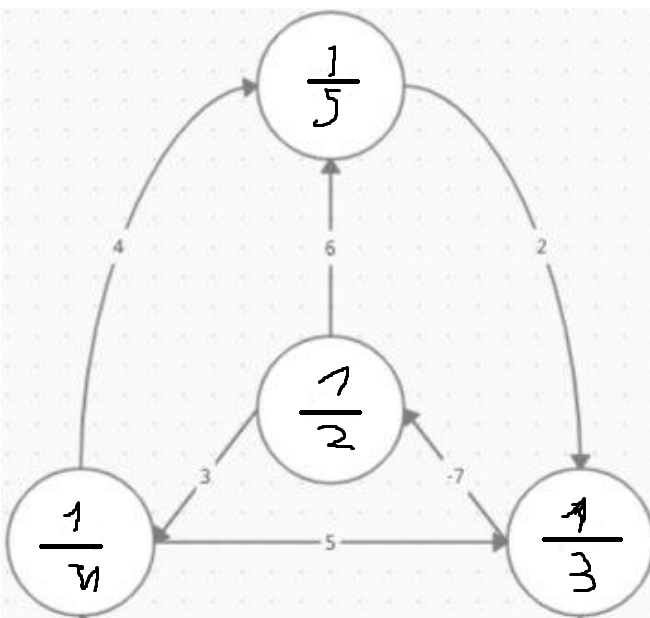


Застосовуємо алгоритм Дейкстри для цього графу

Вихідний граф



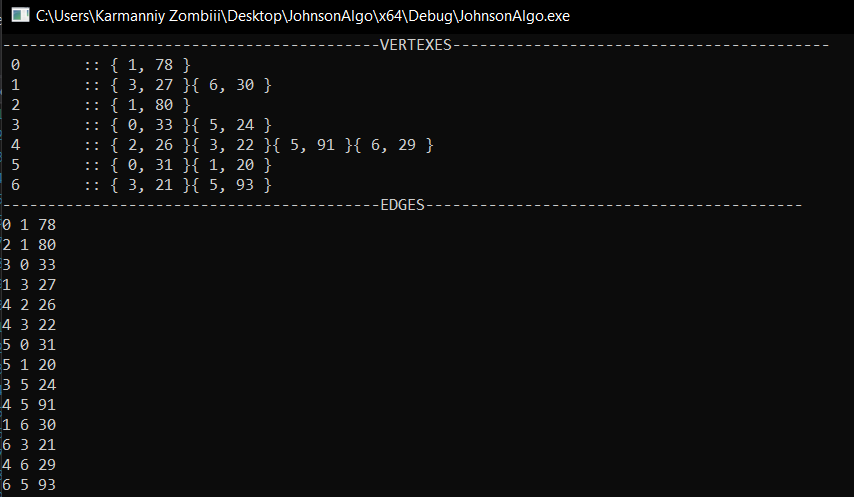
Повертаємо ребрам попередні значення



Це остаточний граф з мінімальним маршрутами і Алгоритм завершив свою роботу.

**Далі поглянемо що ми отримуємо при запуску мого коду:**

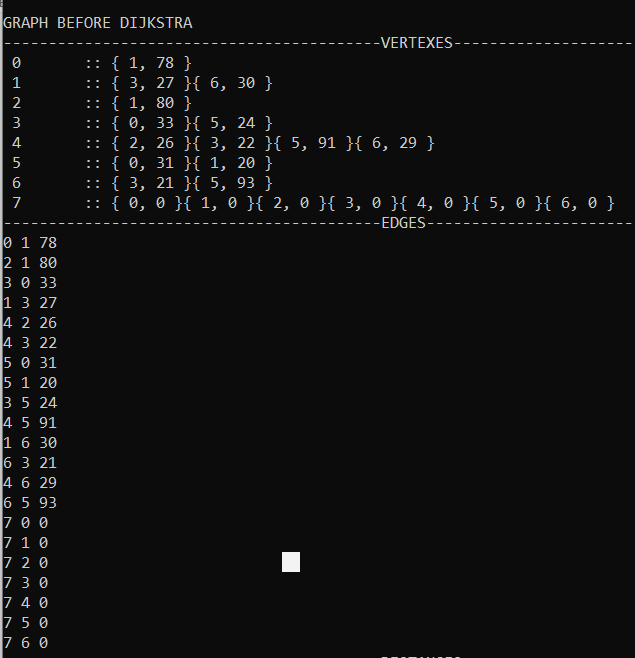
Граф в такому вигляді



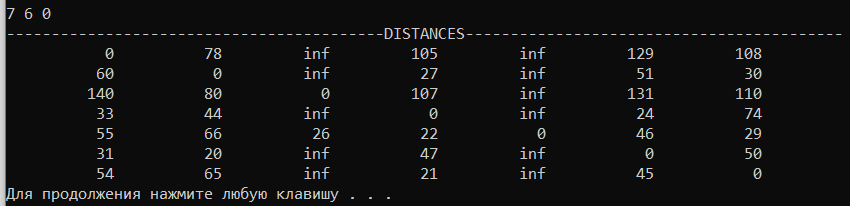
Далі результат алгоритму беллман форда



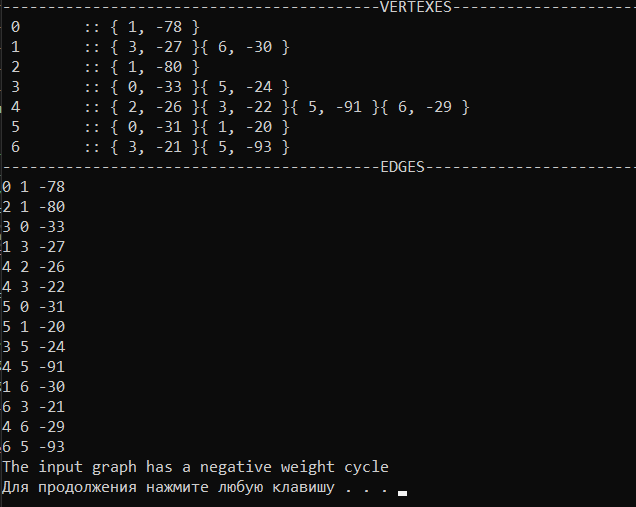
З доданою к вершиною



Й результат в вигляді матриці



В випадку якщо існують відємні цикли виведеться таке повідомлення:



**Висновок**: Так як ми для зберігання невідвіданих вершин використовували купу Фібоначчі то ми значно зменшили складність алгоритму з тої яка б була при наївній реалізації. При розріджених графах складність алгоритму стає меншою чи складність алгоритму Флойда-Уоршала який виконує цю задачу за 𝑂()

**Джерела:**

https://habr.com/ru/company/otus/blog/484382/ [https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D%B8%D1%82%D0%BC\_%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D1%8B](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%25D%B8%D1%82%D0%BC_%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D1%8B)

https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC\_%D0%94%D0%B6%D0%BE%D0%BD%D1%81%D0%BE%D0%BD%D0%B0#:~:text=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%20%D0%94%D0%B6%D0%BE%D0%BD%D1%81%D0%BE%D0%BD%D0%B0%20%E2%80%94%20%D0%BF%D0%BE%D0%B7%D0%B2%D0%BE%D0%BB%D1%8F%D0%B5%D1%82%20%D0%BD%D0%B0%D0%B9%D1%82%D0%B8%20%D0%BA%D1%80%D0%B0%D1%82%D1%87%D0%B0%D0%B9%D1%88%D0%B8%D0%B5,%D0%BE%D1%82%D1%81%D1%83%D1%82%D1%81%D1%82%D0%B2%D1%83%D1%8E%D1%82%20%D1%86%D0%B8%D0%BA%D0%BB%D1%8B%20%D1%81%20%D0%BE%D1%82%D1%80%D0%B8%D1%86%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%BC%20%D0%B2%D0%B5%D1%81%D0%BE%D0%BC.