Київський національний університет імені Тараса Шевченка

Алгоритми та складність

Лаборатрна робота 2  
Виконав студент 2-го курсу Групи ПІ-22

Дідківський Володимир Вячеславович

## Завдання

Алгоритм Джонсона для розріджених графів (включає алгоритми Белмана-Форда і Дейкстри). В алгоритмі Дейкстри використайте піраміду Фібоначчі.

**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.

1. 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.

1. 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(𝑛2) 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.

**Крок 1**

На цьому 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.

### Крок 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

### Крок 3

На цьому 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 𝑉2)

## Мова програмування

С++

## Модулі програми

student.h

class Student{};

std::string getName();

void getStudent();

void setName(std::string name);

group.h

class Group {};

Group() : title("NULL");

Group(std::string title);

Group(std::string title, Student\* first\_student);

std::string getGroupTitle();  
std::vector<Student\*> getGroupStudents();  
void setGroupTitle(std::string title);

void setGroupStudents(std::vector<Student\*> students);

void addStudent(Student\* student);

void printStudents();

graph.h/.cpp

class Vertex;

class Edge;

class GraphFunctionality;

class AdjecentListBasedGraph;

class Graph : public GraphFunctionality;

JohnsonAlgorythm.h/.cpp

distanceVector belmanFord(GraphFunctionality& graph, size\_t fromVertex) throw(std::runtime\_error);

void relax(distanceVector& dist, GraphFunctionality& graph, size\_t u, size\_t v);

void relax(distanceVector& dist, edgesContainer& edgesWeight, size\_t u,

size\_t v);   
std::vector<distanceVector> johnsonAlgorithm(GraphFunctionality& g);

void initDistanceVector(distanceVector& distance, size\_t fromVertex);

std::pair<size\_t, int> findMin(std::vector<bool>& in, edgesContainer& edgesWeight);

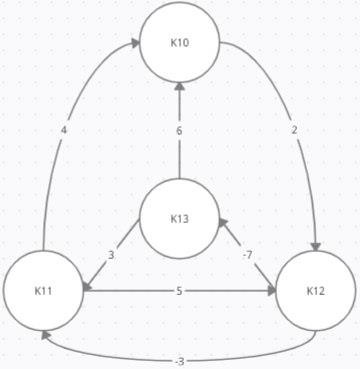
distanceVector dijkstra(GraphFunctionality& graph, edgesContainer newEdgesWeight, size\_t fromVertex);  
void initDistanceVector(distanceVector& distance, size\_t fromVertex);

## Інтерфейс користувача

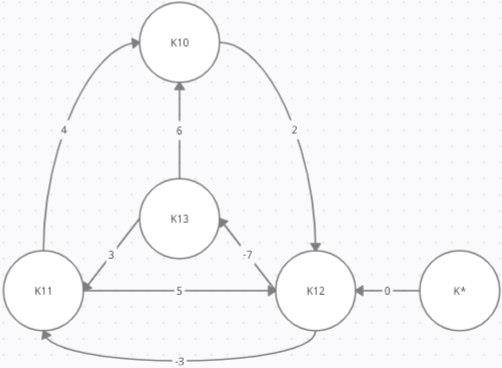
Вхідні дані назви груп прописані у програмі а довжини ребер графу беруться з файлу, вихідні виводяться у консоль.

## Тестовий приклад

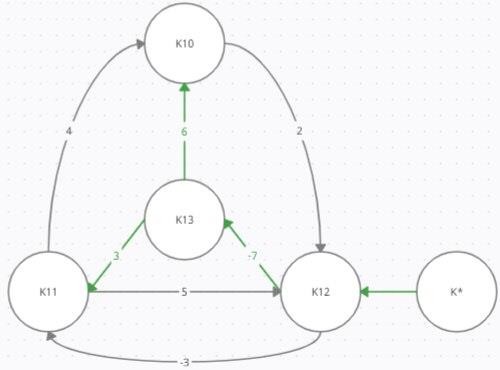
Нехай маємо множину Груп взаємне положення яких множина подати у вигляді орієнтованого графу



Додаємо уявну вершину К\*



Шукаємо для всіх вершин найкоротші шляхи до К\* за Алгоритмом Беллмана-Форда



h(К11) =3+(-7) = -4 h(К12)=0 h(К13)=-7

h(К10)=6+(-7)= -1

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

𝑤′(К11, К10) = 𝑤(К11, К10) + ℎ(К11) − ℎ(К10)

= 4 + (−4) − (−1) = 1

𝑤′(К10, К12) = 𝑤(К10, К12) + ℎ(К10) − ℎ(К12) =

2 + (−1) − 0 = 1

𝑤′(К12, К11) = 𝑤(К12, К11) + ℎ(К12) − ℎ(К11) =

−3 + 0 − (−4) = 1

𝑤′(К11, К12) = 𝑤(К11, К12) + ℎ(К11) − ℎ(К12) = 5 + (−4) − 0 = 1

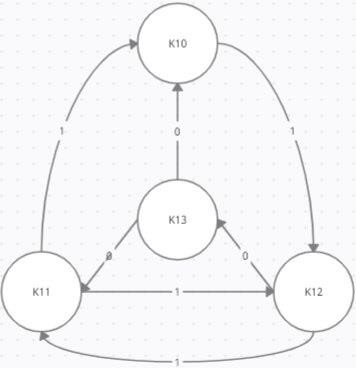
𝑤′(К13, К12) = 𝑤(К13, К12) + ℎ(К13) − ℎ(К12) =

−7 + 0 − (−7) = 0

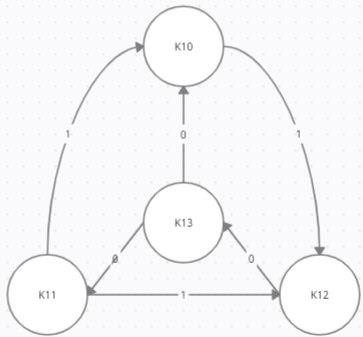
𝑤′(К11, К13) = 𝑤(К11, К13) + ℎ(К11) − ℎ(К13) = 3 + (−7) − (−4) = 0

𝑤′(К11, К10) = 𝑤(К11, К10) + ℎ(К11) − ℎ(К10) = 6 + (−7) − (−1) = 0

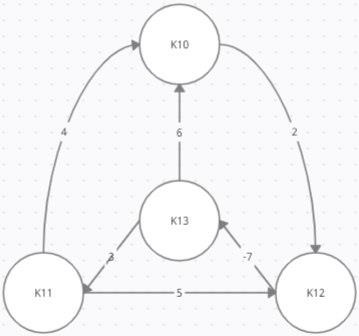
Далі граф має вигляд з новими вагами



Застосовуємо алгоритм Дейкстри для цього графу Вихідний граф



Повертаємо ребрам попередні визначення



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

## 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(𝑉2)

## Література:

* [https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D1%8B) [BE%D1%80%D0%B8%D1%82%D0%BC\_%D0%94%D0%B5%](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D1%8B) [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%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D1%8B)