**Київський національний університет імені Тараса Шевченка**

**Факультет комп’ютерних наук та кібернетики**

**Кафедра інформаційних систем**

**Алгоритми та складність**

**Лабораторна робота №9**

**«Алгоритм Джонсона»**

**ЗВІТ**

**Підготував студент**

**2 курсу**

**Групи К29**

**Опанюк Микита**

**Варіант 7**

**2018 рік**

**Опис предметної області :**

*Варіант 7*   
Предметная область : Продуктовый магазин   
Объекты : Категория продукта, Продукт   
Примечание : Продукты  в  магазине сгруппированы  по  категориям. Для  каждой  категории определено  множество продуктов.

**Теоретична частина:**

**Алгоритм Джонсона** дозволяє знайти найкоротші шляхи між всіма парами вершин за *O*(*V*2log(*V*)+*VE*). Для розріджених графів цей алгоритм веде себе асимптотично швидше ніж алгоритм Флойда. Цей алгоритм або повертає матрицю найкоротших відстаней між всіма парами вершин, або ж повідомлення про те, що в графі існує цикл від'ємної довжини.

В цьому алгоритмі використовується метод **зміни ваги ребра** (англ. *reweighting*). Суть його заключається в тому, що для заданного графа *G* будується нова вагова функція *ωφ*, невід'ємна для всіх ребер графа *G*, зі збереженням найкоротших відстаней. Така вагова функція будується з допомогою так званої потенціальної функції.

Нехай *φ*:*V*→R — відображення із множини вершин в натуральні числа. Тоді новою ваговою функцією буде *ωφ*(*u*,*v*)=*ω*(*u*,*v*)+*φ*(*u*)−*φ*(*v*).

Така потенціальна функція будується додаванням фіктивної вершини *s* в *G*, із якої лежать орієнтовані ребра нульової ваги у всі інші вершини графа, і запуском **алгоритма Форда-Беллмана** неї (*φ*(*v*) буде рівне довжині найкоротшого шляху із *s* в *v*). На цьому ж етапі ми зможемо виявити присутність від'ємного циклу в графі.

Тепер, коли ми знаємо, що у всіх ребер вага >= 0, і найкоротші шляхи зберігаються, можна запустити **алгоритм Дейкстри** із кожної вершини і таким чином знайти найкоротші відстані між всіма парами вершин.

**Алгоритм Форда-Беллмана**

**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.

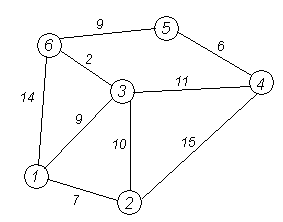
Дано орієнтований графз ваговою функцією Вагою шляхуназвемо суму ваг ребер, що входять в цей шлях: 

Вхідними даними для алгоритму є граф G G , {\displaystyle G,} вагова функція w w , {\displaystyle w,} та стартова вершина s. s . {\displaystyle s.} Потрібно знайти найкоротші шляхи від вершини s s {\displaystyle s} до всіх вершин графа. Алгоритм Беллмана-Форда повертає логічне значення, яке вказує на те, чи міститься в графі цикл з негативною вагою, досяжний з витоку. Якщо такий цикл існує у графі G G , {\displaystyle G,} алгоритм повідомляє, що найкоротших шляхів не існує. Якщо ж таких циклів немає, алгоритм видає найкоротші шляхи і їх вагу.

Сам алгоритм Форда-Беллмана представляє з себе кілька фаз. На кожній фазі проглядаються всі ребра графа, і алгоритм намагається справити релаксацію (relax, ослаблення) уздовж кожного ребра (u,v) ( u , v ) {\displaystyle (u,v)} ваги w(u,v)w ( u , v ) {\displaystyle w(u,v)} . Релаксація вздовж ребра — це спроба поліпшити значення v.d v . d {\displaystyle v.d} значенням v.u + w(u,v)v . u + w ( u , v ) {\displaystyle v.u+w(u,v)} . Фактично це означає, що ми намагаємося поліпшити значення для вершини v v , {\displaystyle v,} користуючись ребром (u,v) ( u , v ) {\displaystyle (u,v)} і поточним значенням для вершини uu {\displaystyle u} . Стверджується, що достатньо G.V – 1 | G . V | − 1 {\displaystyle |G.V|-1} фази алгоритму, щоб коректно порахувати довжини всіх найкоротших шляхів у графі (цикли негативної ваги відсутні). Для недосяжних вершин відстань v.d v . d {\displaystyle v.d} залишиться нескінченністю.

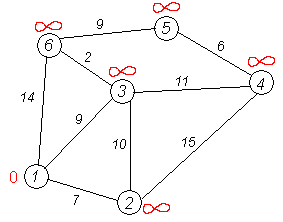
**Алгоритм Дейкстри** — [алгоритм](https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) на [графах](https://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), відкритий [Дейкстрою](https://uk.wikipedia.org/wiki/%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D0%B0_%D0%95%D0%B4%D1%81%D0%B3%D0%B5%D1%80). Знаходить [найкоротший шлях](https://uk.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B9%D0%BA%D0%BE%D1%80%D0%BE%D1%82%D1%88%D0%B8%D0%B9_%D1%88%D0%BB%D1%8F%D1%85) від однієї вершини графа до всіх інших вершин. Класичний алгоритм Дейкстри працює тільки для графів без циклів від'ємної довжини.

***Приклад :***

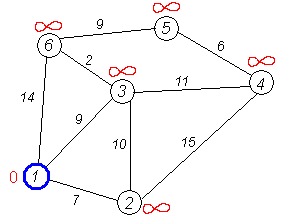
Зберігатимемо поточну мінімальну відстань до всіх вершин **V** (від даної вершини **a**) і на кожному кроці алгоритму намагатимемося зменшити цю відстань. Спочатку встановимо відстані до всіх вершин рівними нескінченості, а до вершини **а** — нулю. [](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph0.PNG)

Розглянемо виконання алгоритму на прикладі. Хай потрібно знайти відстані від 1-ї вершини до всіх інших. Кружечками позначені вершини, лініями — шляхи між ними («дуги»). Над дугами позначена їх «ціна» — довжина шляху. Надписом над кружечком позначена поточна найкоротша відстань до вершини.

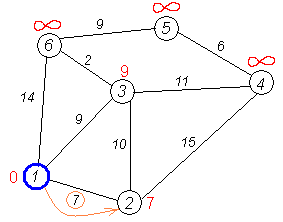
### Крок 1

Ініціалізація. Відстань до всіх вершин [графа](https://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) **V** = ∞ {\displaystyle \infty } . Відстань до **а** = 0. Жодна вершина графа ще не опрацьована.  
[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph1.PNG)

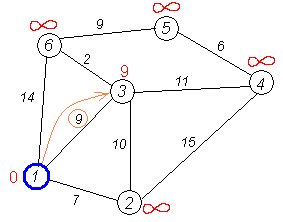
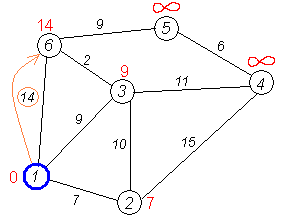
### Крок 2

Знаходимо таку вершину (із ще не оброблених), поточна найкоротша відстань до якої мінімальна. В нашому випадку це вершина 1. Обходимо всіх її сусідів і, якщо шлях в сусідню вершину через 1 менший за поточний мінімальний шлях в цю сусідню вершину, то запам'ятовуємо цей новий, коротший шлях як поточний найкоротший шлях до сусіда.  
[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph2.PNG)

### Крок 3

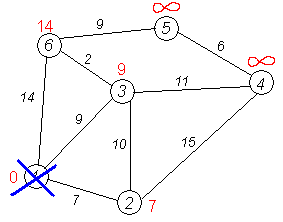
Перший по порядку сусід 1-ї вершини — 2-а вершина. Шлях до неї через 1-у вершину дорівнює найкоротшій відстані до 1-ї вершини + довжина дуги між 1-ю та 2-ю вершиною, тобто 0 + 7 = 7. Це менше поточного найкоротшого шляху до 2-ї вершини, тому найкоротший шлях до 2-ї вершини дорівнює 7.  
[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph3.PNG)

### Кроки 4, 5

Аналогічну операцію проробляємо з двома іншими сусідами 1-ї вершини — 3-ю та 6-ю.  
[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph4.PNG)[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph5.PNG)

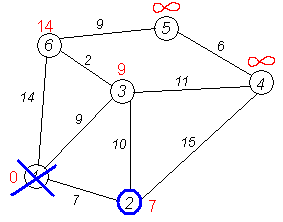
### Крок 6

Всі сусіди вершини 1 перевірені. Поточна мінімальна відстань до вершини 1 вважається остаточною і обговоренню не підлягає (те, що це дійсно так, вперше довів [Дейкстра](https://uk.wikipedia.org/wiki/%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D0%B0_%D0%95%D0%B4%D1%81%D0%B3%D0%B5%D1%80)). Тому викреслимо її з [графа](https://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), щоб відмітити цей факт.

[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph6.PNG)

### Крок 7

Практично відбувається повернення до кроку 2. Знову знаходимо «найближчу» необроблену (невикреслену) вершину. Це вершина 2 з поточною найкоротшою відстанню до неї = 7.

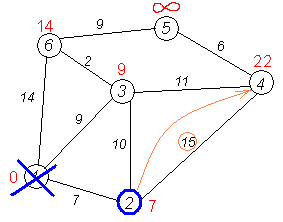
[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph7.PNG)

І знову намагаємося зменшити відстань до всіх сусідів 2-ї вершини, намагаючись пройти в них через 2-у. Сусідами 2-ї вершини є 1, 3, 4.

### Крок 8

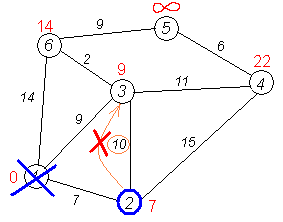
Перший (по порядку) сусід вершини № 2 — 1-ша вершина. Але вона вже оброблена (або викреслена — див. крок 6). Тому з 1-ю вершиною нічого не робимо.

### Крок 8 (з іншими вхідними даними)

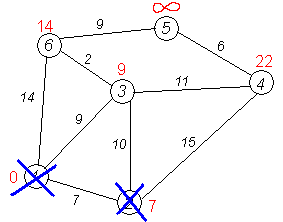
Інший сусід вершини 2 — вершина 4. Якщо йти в неї через 2-у, то шлях буде = найкоротша відстань до 2-ї + відстань між 2-ю і 4-ю вершинами = 7 + 15 = 22. Оскільки 22 < ∞, встановлюємо відстань до вершини № 4 рівним 22. [](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph8.PNG)

### Крок 9

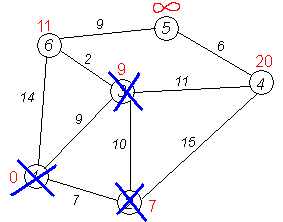
Ще один сусід вершини 2 — вершина 3. Якщо йти в неї через 2-у, то шлях буде = 7 + 10 = 17. Але 17 більше за відстань, що вже запам'ятали раніше до вершини № 3 і дорівнює 9, тому поточну відстань до 3-ї вершини не міняємо.

[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph9.PNG)

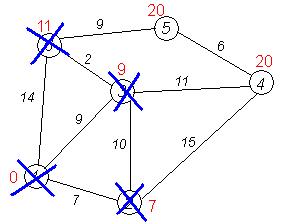
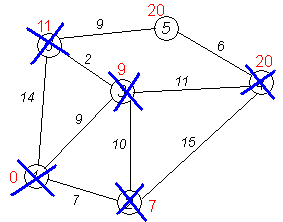
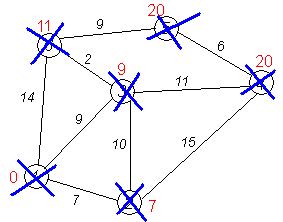
### Крок 10

Всі сусіди вершини 2 переглянуті, заморожуємо відстань до неї і викреслюємо її з [графа](https://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)). [](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph10.PNG)

### Кроки 11 — 15

По вже «відпрацьованій» схемі повторюємо кроки 2 — 6. Тепер «найближчою» виявляється вершина № 3. Після її «обробки» отримаємо такі результати:  
[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph11.PNG)

### Наступні кроки

Проробляємо те саме з вершинами, що залишилися (№ по порядку: 6, 4 і 5).  
[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph12.PNG)[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph13.PNG)[](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Dijkstra_graph14.PNG)

**Реалізація алгоритму (мова) :** Алгоритм реалізований з використанням мови програмування C++ з використанням інтегрованого середовища розробки Microsoft Visual Studio.

**Інтерфейс програми :** консоль.

**Використання структур даних :**

**1)** В задачці використовується class market{}; - саме в цьому класі йде основна реалізація предметної області. А саме :

Клас працює з 2 типами структур :

struct product\_element {

string group; - назва різновиду продуктів, до якого відноситься вказаний.

string name; - відповідно ім’я.

double price; - ціна (для інформативності).

int key; - для зручності роботи з B+ деревом.

};

**vector<type \*> array\_of\_products; - масив продуктів в магазині, це основні дані, з якими працює B+ дерево.**

int check\_is\_product(string product); - перевіряє наявність вказаного продукту в магазині.

void new\_data(); - додавання нового продукту з перевіркою того чи існує вже вказаний продукт.

void delete\_data(product\_element \*name); - видаляє з каталогу вказаний продукт.

void save\_changing(); - зберігає всі зміни каталогу, перезаписуючи файл, на якому зберігаються дані що до продуктів.

void read\_data\_from\_file(); - зчитує данні що до продуктів з файлу.

Для роботи з алгоритмом Джонсона структура:

struct Graph { - граф є орієнтованим зваженим

struct Vertex { - вершина графа

int d; - найкоротша відстань

Vertex \*pi; - попередник вершини

string name; - назва продукту

unsigned int number; індекс вершини в списку суміжних вершин

string group; - тип продукту

// New vertex

Vertex(string name, unsigned int number, string group) : d(0), name(name), number(number), group(group) {} – ініціалізація вершини

};

struct Edge { - ребро графа

Vertex \*v; - до якої вершини (вершина, з якої виходить ребро – зв’язана з ребром в окремому масиві)

int w; - вага

// New edge

Edge(Vertex \*v\_, int w\_) : v(v\_), w(w\_) {} – ініціалізація ребра

};

vector<forward\_list<Edge>> Adj; - лист суміжних вершин

vector<Vertex> gr; - масив, що зберігає в собі всі вершини графа

vector<vector<unsigned int>> matr; - матриця найкоротших шляхів між вершинами.

Graph(unsigned int n) { - функція ініціалізації графа в залежності від кількості вершин (в моєму випадку кількість вершин = кількості продуктів, зчитаних з файлу)

Adj.resize(n);

matr.resize(n - 1, vector<unsigned int>(n - 1));

}

Допоміжна функція, що задає початкові (нульові) значення найкоротшої відстані для кореня та попередників, використовується в алгоритмі Дейкстри Белмана-Форда.

void Initialize\_Single\_Sourse(Graph&, Vertex&);

Релаксація, про яку я розповідав у алгоритмі Белмана-Форда, що при можливості покращує значення найменшої відстані.

void Relax(Vertex, Vertex\*&, int);

Алгоритм Белмана-Форда, що визначає вагу найкоротших шляхів від спеціальної вершини, що додається до набору вершин графа, отримуючи новий граф, якщо немає циклів з негативною вагою. Завдяки цьому отримуємо позитивні значення ваги у кожного ребра оригінального графа

bool Belman\_Ford(Graph&, Vertex&);

Алгоритм Дейкстри, що знаходить всі найкоротші відстані між вершинами

vector<unsigned int> Dijkstra(Graph&, Vertex&);

Алгоритм Джонсона, що використовує 2 функції вище та будує матрицю розвязків

vector<vector<unsigned int>> \*Johnson(Graph&);

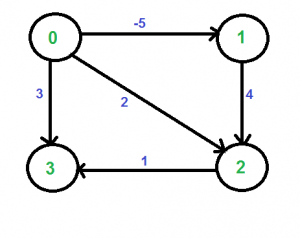
Виведення графа

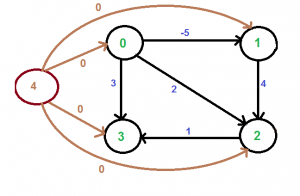
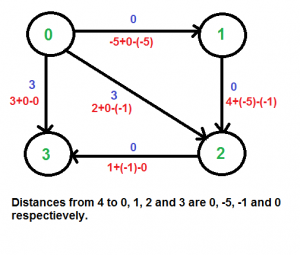
void print(Graph);

};

**Крім того алгоритм Дейкстри повністю базується на основі Фібоначчієвої купи, що зберігається в файлі Fibonacci\_heap.h. Це практично та сама купа, що використовувалась в 8 завдані, проте була перероблена структура даних для роботи з графом. Кожна вершина графа – є елементом Фібоначієвої купи. Фібоначієва купа дозволяє прискорити роботу всієї задачі, якщо граф є розрідженим.**

**Алгоритм на пальцях:**  
1) Даний граф G. Додаємо до графа нову вершину (в моєму випадку це перша вершина в масиві вершин), проводимо ребра з цієї вершини у всі інші вершини графа G. Нехай модифікований граф буде G '.  
2) Запустіть алгоритм Беллмана-Форда на G ', в якому вказаний корінь s. Нехай відстані, розраховані Беллманом-Фордом, будуть h [0], h [1], .. h [V-1]. Якщо ми знайдемо негативний ваговий цикл, то повернемося.  
3) Змінюємо вагу ребер оригінального графу відповідно до алгоритму. Для кожного ребра (u, v) призначити нову вагу як "початковий вага + h [u] - h [v]".  
4) Вилучаємо додану вершину s і запускаємо алгоритм Дейкстри для кожної вершини.

Давайте розглянемо наступний графік.  
[](https://www.geeksforgeeks.org/wp-content/uploads/Johnson1.png)  
Ми додаємо додаткову вершину s і додаємо ребра з s до всіх вершин оригінального графа. На наступній діаграмі s має номер 4.

[](https://www.geeksforgeeks.org/wp-content/uploads/Johnson2.png)  
  
Ми обчислюємо найкоротші відстані від 4 до всіх інших вершин за допомогою алгоритму Беллмана-Форда. Найкоротші відстані від 4 до 0, 1, 2 і 3 відповідно 0, -5, -1 та 0, тобто h [] = {0, -5, -1, 0}. Як тільки ми отримаємо ці відстані, ми видаляємо вихідну вершину 4 і змінюємо вагу ребер за допомогою наступної формули. w (u, v) = w (u, v) + h [u] - h [v].  
[](https://www.geeksforgeeks.org/wp-content/uploads/Johnson3.png)  
Оскільки всі ваги зараз позитивні, ми можемо запустити алгоритм Дейкстра для кожної вершини.  
**Опис алгоритму (предметна область):**

Ініціалізуємо market.h. Зчитуємо з файлу market.h :

6

candies;white bar;16.99

candies;chocoshaker;20.55

drinks;shaker;25.55

meat;chicken;45.99

fruit;orange;27.95

fruit;apple;15.55

Починаємо обробку бази даних, після завершення зберігаємо зміни до файлу і переходимо до побудови графа.

If file is not empty, reading data from file!

white bar; chocoshaker; chicken; orange; apple; shaker;

1)Add new data.

2)Delete data.

3)Use this data.

3 – переходимо зразу до роботи з алгоритмом Джонсона

== Johnson algorithm ==

1. white bar; 2. chocoshaker; 3. chicken; 4. orange; 5. apple; 6. shaker;

Add new edge? 1 - Yes, 2 - no

2

Use example? 1 - yes; 2 - no

1 – користуюсь вже фіксованими значеннями ваги для певних ребер щоб прискорити демонстрацію програми (для інших випадків (щоб отримати інші результати) треба використовувати ручне введення даних)

Відбувається процес обробки графа

Matrix with shortest paths' weights between all vertices of graph 3: - виведення матриці мінімальних відстаней

| 1 2 3 4 5 6

1| 0 2 2 2 0 2147483647

2| 2 0 0 0 2 2147483647

3| 2 0 0 0 2 2147483647

4| 2 0 0 0 2 2147483647

5| 4 2 2 2 0 2147483647

6| 2147483647 2147483647 2147483647 2147483647 2147483647

**Використані джерела :**

https://uk.wikipedia.org/wiki/Фібоначчієва\_купа

Лекції Оксани Степанівни (Фібоначчієва купа)