Київський національний університет імені Тараса Шевченка

Факультет комп'ютерних наук та кібернетики

Кафедра дослідження операцій

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА
за спеціальністю 113 «Прикладна математика»

на тему:

Алгоритми розв'язання задачі про найкоротший шлях

студентки 4 курсу

Білинської Олесі Василівни

Науковий керівник:

доцент, кандидат фізико-математичних наук

Якимів Р. Я.

Робота заслухана на засіданні кафедри дослідження операцій та рекомендована до захисту в ЕК, протокол № 9 від 23.05.2023 р.

Завідувач кафедри ДО  проф. Іксанов О.М

Київ – 2023

РЕФЕРАТ

Обсяг роботи 56 сторінок, 19 рисунків, 12 таблиць та 13 використаних джерел.

Робота присвячена дослідженню проблеми пошуку найкоротших шляхів на графах та алгоритмам, які здатні вирішувати дану задачу. У роботі досліджується також швидкість виконання комп'ютерної реалізації алгоритмів способом порівняння та прогнозу.

Об'єктом дослідження є задача про найкоротший шлях, алгоритми для розв'язання такої задачі, а саме: Дейкстри, Беллмана-Форда, Флойда-Уоршела та Джонсона, - а також швидкодія комп'ютерної реалізації алгоритмів при збільшенні кількості вершин графа.

Метою роботи є дослідити використання алгоритмів Дейкстри, Беллмана-Форда, Флойда-Уоршела та Джонсона для розв'язання задачі про найкоротший шлях. Виконати порівняльну оцінку реалізації алгоритмів на мові C++, щоб з'ясувати ефективність кожного. Розробити трендовий прогноз алгоритмів щодо їхньої тривалості виконання за критерієм збільшення кількості вершин. Проаналізувати отримані результати. Зробити висновки згідно виконаних досліджень.

Практична цінність: Розроблені дослідження можуть слугувати як дані для використання при виникненні проблеми пошуку найкоротшого шляху і це не тільки йдеться саме про шлях. Також робота може бути використана як посібник для вивчення, як базова аналітична інформація для сфер, що працюють з вищезазначеними алгоритмами.

ЗМІСТ

ВСТУП

Задача про пошук найкоротшого шляху найбільш асоціюється з дорожнім маршрутом чи маршрутом на карті. Це не єдині сфери виникнення цієї задачі чи навіть проблеми. На теперішній час актуальність даної задачі фігурує навіть у військовій сфері, наприклад, у визначенні оптимального комплекту робіт з мінімальними затратами часу у ході бойових дій, тому це доводить те, що така проблема постійно з'являється і необхідно чітко розуміти як саме її вирішувати.

Задача про найкоротший шлях зводиться до пошуку на графі. Графи, як виявилося, є математичною моделлю з широким класом об'єктів та процесів, що дає можливість розв'язати задачу про найкоротший шлях з різних сфер використання. Узагалі, теорія графів знаходить своє застосування у багатьох областях, наприклад, у фізиці, хімії, теорії зв'язку, проектуванні ЕОМ, електроніці, машинобудуванні, архітектурі, дослідженні операцій, генетиці, психології, соціології, економіці та антропології.

Задача про найкоротший шлях застосовується у інформатиці, економіці, географії та багато інших областях. Постановки цієї задачі також відрізняються і бувають дуже різноманітними, так як вага ребра може бути не тільки довжиною, але й часом, вартістю, витратами чи обсягом витрачених ресурсів(різних видів) або іншою характеристикою для проходження кожного ребра.

Таким чином, дослідження алгоритмів, які здатні розв'язувати задачу про найкоротший шлях є важливим для того, щоб зменшити витрачення часу, а також отримати точні результати у знаходженні маршрутів, оптимальних показників та інших об'єктів, що стосуватимуться даної проблеми.

Також важливим показником для таких алгоритмів є їхня стійкість у подальшому виконанні, тому прогнозування стабільного розвитку подій є невід'ємною частиною дослідження. Це дає гарантії для компаній розроблення маршрутів чи розподілу ресурсів, користувачів та різних сфер, де застосовують алгоритми розв'язання задачі про найкоротший шлях.

Робота структурована наступним чином:

Розділ 1 – теоретичний і включає у себе розгляд основних понять, необхідних для постановки та розробки використання алгоритмів для розв’язання задачі про найкоротший шлях, будуть сформульовані необхідні теореми, які є важливими для розв'язку задачі.

Розділ 2 – практичний. У ньому буде сформульовано задачі двох видів, а саме про пошук найкоротшого шляху та відстаней від однієї вершини до всіх інших та між усіма парами вершин. Буде розроблено покрокове розв’язання цих задач алгоритмами Дейкстри, Беллмана-Форда, Флойда-Уоршела та Джонсона з ілюстраціями та таблицями. Також у цьому розділі буде зроблено реалізацію алгоритмів, порівняльну оцінку та аналіз прогнозу результатів часу.

У висновках зроблено підсумовування всіх отриманих результатів та якості виконаного дослідження.

Мета роботи: дослідити використання алгоритмів Дейкстри, Беллмана-Форда, Флойда-Уоршела та Джонсона для розв’язання задачі про найкоротший шлях. Виконати порівняльну оцінку реалізації алгоритмів на мові C++, щоб з’ясувати ефективність кожного. Розробити трендовий прогноз алгоритмів щодо їхньої тривалості виконання за критерієм збільшення кількості вершин. Проаналізувати отримані результати. Зробити висновки згідно зроблених досліджень.

Завдання роботи:

1. Провести теоретичний огляд усіх видів графів, задачі про найкоротший шлях та алгоритмів, визначити поняття та сформулювати теореми, які будуть використовуватись надалі в роботі.
2. Розробити покрокове використання алгоритму Дейкстри до поставленої задачі про найкоротший шлях.
3. Розробити покрокове використання алгоритму Беллмана-Форда до поставленої задачі про найкоротший шлях.
4. Розробити покрокове використання алгоритму Флойда-Уоршела до поставленої задачі про найкоротший шлях.
5. Розробити покрокове використання алгоритму Джонсона до поставленої задачі про найкоротший шлях.

6. Зробити реалізацію розглянутих алгоритмів на мові С++, з можливістю визначення часу виконання розробки та зробити порівняльну оцінку за результатами часу.
7. Зробити трендовий аналіз в табличному редакторі Excel і проаналізувати його. Зробити підсумовування одержаних результатів.

РОЗДІЛ 1. ТЕОРЕТИЧНА ЧАСТИНА

1 Розгляд основних понять

1.1 Неорієнтовані графи

Задано непорожню скінченну множину V і множину E невпорядкованих пар різних елементів з множини V. *Простим графом* G називають пару множин V і E, тобто $G = (V, E)$ [5].

Елементи множини V називають *вершинами* графа G, а невпорядковані пари різних вершин – *ребрами*. Кількість вершин позначають як $|V|$, а кількість ребер – як $|G|$ [5]. Якщо пара вершин a і b є ребром, то її позначають $[a, b]$. Вершини a і b є *кінцями* ребра. Зауважимо, що $[a, b]$ і $[b , a]$ позначають одне і те ж ребро.

На рисунках ребра зображають прямою чи кривою лінією без стрілок, а вершини позначають літерами або числами. Оскільки *E – множина*, то у простому графі довільну пару вершин може з'єднувати не більше одного ребра.

У деяких випадках розглядають графи, у яких дві вершини може з'єднувати декілька ребер. Виникає поняття *мультиграфа* – пари (V, E), де V – непорожня скінченна множина, а *E – сім'я* невпорядкованих пар різних елементів з множини V.

Термін "сім'я" означає, що елементи E (ребра) можуть повторюватися. Ребра, які з'єднують одну й ту ж пару вершин, називають *кратними* (або *паралельними*) ребрами.

Наступне узагальнення полягає у тому, що окрім кратних ре- бер допускають наявність *петель* – ребер, які з'єднують вершину саму з собою. *Псевдографом* називають пару (V, E), де V – непорожня скінченна множина, а E – *сім'я* невпорядкованих пар елементів з множини V (не обов'язково різних) [5].

Три типи графів, які введені вище, називають *неорієнтованими графами*, причому:

- *псевдограф* – може мати петлі і кратні ребра;
- *мультиграф* – може мати кратні ребра (без петель);

- *простий граф* – без петель і кратних ребер[5].

Вершини a і b в неорієнтованому графі називають *суміжними*, якщо $[a, b]$ є ребром. Два ребра називають *суміжними*, якщо вони мають спільний кінець. Вершина v і ребро e називають *інцидентними*, якщо вершина v є кінцем ребра e. Отже, суміжність відображає зв'язок між однорідними елементами графа, а інцидентність – між неоднорідними елементами [5].

Степінь вершини v (позначають $deg(v)$) у неорієнтованому графі – це кількість ребер, інцидентних вершині v, причому петлю враховують двічі. Якщо $deg(v) = 0$, то вершину v називають *ізольованою*; якщо $deg(v) = 1$, то вершину v називають *висячою* [5].

Послідовність ребер $[v_0, v_1]$, $[v_1, v_2]$, $[v_2, v_3]$, …, $[v_{n-1}, v_n]$ неорієнтованого графа називають *шляхом* довжини n, що з'єднує вершини v_0 та v_n (ребро враховують стільки разів, скільки воно входить у шлях) [5]. Якщо при цьому $v_0 = v_n$, то шлях називають *циклом* [5].

Якщо усі ребра шляху/циклу *різні*, то його називають *простим* шляхом/циклом. Шлях з крайніми вершинами v_0 та v_n позначають $<v_0, v_n>$ і називають *шляхом*.

Граф $G' = (V', E')$ називають підграфом графа $G = (V, E)$, якщо $V' \subseteq V$ і $E' \subseteq E$.

Підграф неорієнтованого графа називають *зв'язним,* якщо у ньому для кожної пари вершин знайдеться хоча б один шлях, щоїх з'єднує. Відношення "з'єднані шляхом" є *відношенням еквівалентності* на множині вершин. Класи еквівалентності називають *компонентами зв'язності* графа (слово "*зв'язності*" у терміні іноді опускають) [5]. На рис. 1.1 зображено граф, що складається з двох компонент зв'язності: $\{a, b, f\}$ і $\{d, c, e, g\}$.

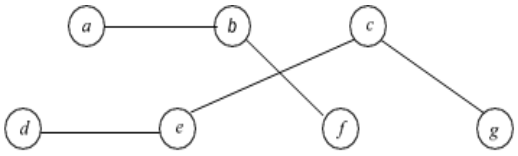

Рис. 1.1. Приклад двокомпонентного графа

Неорієнтований граф *зв'язний*, якщо він складається з єдиної компоненти зв'язності [5].

Граф називають *деревом*, якщо він зв'язний і не містить простих циклів. Граф, який не містить простих циклів і складається з k компонент, називають *лісом* з k дерев. З означень випливає, що дерева і ліси є простими графами. На рис. 1.1 зображено ліс, що складається з двох дерев.

Теорема 1.1.Для графа G, що має n вершин і m ребер (тобто $|V| = n; |G| = m$), такі твердження еквівалентні:

1. G – дерево.
2. G – зв'язний граф і $m = n - 1$.
3. G – не містить простих циклів і $m = n - 1$.
4. G – зв'язний граф, проте вилучення довільного ребра робитьйого незв'язним.
5. Будь-які дві вершини G з'єднані єдиним простим шляхом.
6. G – не містить простих циклів і при з'єднанні ребром двох не-суміжних вершин у новому графі буде єдиний простий цикл [5].

Наслідок 1.1 (з теореми 1.1). У лісі з k дерев: $m = n - k$ [5].

Каркасом (*з'єднувальним деревом*) простого зв'язного графа G називають його підграф, який є деревом і містить усі вершини G [5].

1.2 Орієнтовані графи

Орієнтованим графом (або *орграфом*) називають пару множин V і E, де V – непорожня скінченна множина, а E – множина *упорядкованих* пар елементів з множини V [5].

Елементи множини V називають *вершинами* (або *вузлами*), а множини E – *дугами* (або *орієнтованими ребрами*). Якщо пара вершин a і b є дугою, то її позначають (a, b). Вершину a називають *початковою*, а вершину b – *кінцевою*. Дугу (a, a) називають *петлею*. Зауважимо, що орграф може мати петлі [5].

На рисунках дугу позначають прямою чи кривою лінією зі стрілкою, яка вказує на кінцеву вершину. Дуга – це впорядкована пара вершин, причому дуги

(*a*, *b*) і (*b*, *a*) є *різними* дугами [5].

Орієнтованим мультиграфом називають пару множин *V* і *E*, де *V* – непорожня скінченна множина, а *E* – *сім'я* упорядкованих пар елементів з множини *V* [5].

Елементи *E* (дуги) в орієнтованому мультиграфі можуть повторюватися, тоді їх називають *кратними* дугами. Кратні дуги з'єднують одну й ту ж пару вершин та *однаково напрямлені* [5].

Граф, в якого є і ребра, і дуги, називають *змішаним* [5]. Його зводять до орграфа заміною кожного ребра [*a*, *b*] дугами (*a*, *b*) і (*b*, *a*).

В орграфі *напівстепенем входу* вершини *v* називають кількість дуг з *кінцевою* вершиною *v* (позначають $deg^-(v)$) [5]. *Напівстепенем виходу* вершини *v* називають кількість дуг, для яких вершина *v* є *початковою* (позначають $deg^+(v)$) [5]. Степінь вершини *v* визначають так: $deg(v) = deg^-(v) + deg^-(v)$.

Послідовність дуг (v_0, v_1), (v_1, v_2), (v_2, v_3), …, (v_{n-1}, v_n) орієнтованого графа називають *орієнтованим шляхом* (або просто *шляхом*), що з'єднує вузли v_0 та v_n (кінець довільної дуги шляху, окрім останньої, є початком наступної дуги). Довжиною шляху називають кількість дуг, з яких він складається [5].

В орграфі шлях однозначно визначається упорядкованою послідовністю вершин $p =< v_0, v_1, \dots, v_{n-1}, v_n >$. У цьому випадку кажуть, що для вершин v_0 і v_n існує шлях *p* від v_0 до v_n (або вершинаv_n є *досяжною* з вершини v_0) і позначають $v_0 \xrightarrow{p} v_n$ [5].

Якщо в орієнтованому шляху $v_0 = v_n$, то цей шлях називають *орієнтованим циклом* (або просто *циклом*) [5]. Зауважимо, що петля є спеціальним типом циклу. Орграф, що не має циклів, називають *ациклічним* орграфом [5].

Якщо усі дуги орієнтованого шляху/циклу в орграфі – *різні*, то його називають *простим* орієнтованим шляхом/циклом [5].

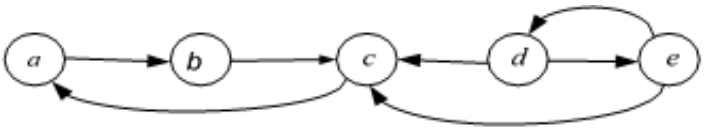

Рис. 1.2. Приклад орієнтованого графа

На рис. 1.2 можна вказати простий шлях <*a, b, c*>. Інші можливі прості шляхи: <*e, d, c, a, b*>, <*e, c, a, b*> і т. д. Шлях <*a, b, c, a*> є циклом.

Лема 1.1. В ациклічному орграфі є хоча б одна вершина, в яку не входить жодна дуга [5].

Доведення від *супротивного*: у будь-яку вершину входить дуга. Розглянемо довільну вершину i_1. Оскільки у цю вершину входить дуга, то перейдемо по ній до початку дуги – вершини i_2. У верши-ну i_2 теж входить дуга, отож перейдемо по ній до початку дуги – вершини i_3 і т.д. Однак орграф має скінченну кількість вершин, і в деякий момент ми знову потрапимо у вершину, в якій були раніше, тобто отримаємо цикл (*протиріччя*).

Теорема 1.2. Вершини ациклічного орграфа можна занумерувати так, що кінець кожної дуги матиме більший номер, ніж номер початку дуги [5].

За лемою 1.1 в ациклічному орграфі є хоча б одна вершина, в якуне входить жодна дуга [5]. Присвоїмо їй номер 1. Якщо є декілька таких вершин, то вибираємо довільну з них.

Видаляємо з орграфа цю вершину разом з дугами, що з неї виходять. В отриманому орграфі також відсутні цикли, а тому є хоча б одна вершина, в яку не входить жодна дуга. Присвоїмо їй номер 2. Якщо є декілька таких вершин, то вибираємо довільну з них. Цей процес виконуватиметься доти, доки не занумеруємо всівершини орграфа. Теорему доведено.

Для орграфа поняття зв’язності вводять по-різному, залежно від того, чи враховують напрям дуг [5]. Орієнтований підграф називають *сильнозв’язним,* якщо в ньому для кожної пари вершин *u* і *v* знайдеться шлях від *u* до *v* та шлях від *v* до *u* [5].

Будь-який орграф можна розбити на *компоненти сильної зв’язності*, які

визначають як класи еквівалентності відношення "v є досяжною з u і u є досяжною з v" [5]. Наприклад, орграф на рис. 1.2 має дві компоненти сильної зв'язності: $\{a, b, c\}$ і $\{e, d\}$.

Орієнтований граф *сильнозв'язний*, якщо він складається з єдиної компоненти сильної зв'язності [5].

Орієнтований граф називають *слабкозв'язним*, якщо існує шлях між будь-якими двома різними вершинами в його неорієнтованому варіанті (тобто *без урахування* напряму дуг). Очевидно, що сильнозв'язний орграф є водночас і слабкозв'язним [5].

Орієнтований граф називають *орієнтованим* (або *кореневим*) *деревом*, яке росте з вершини v_0 (*кореня* дерева), якщо:

- орграф утворює дерево в його неорієнтованому варіанті;
- v_0 – єдина вершина орграфа, в яку не заходить жодна дуга, а у всіінші вершини заходить тільки одна дуга [5].

Якщо (a, b) є дугою орієнтованого дерева, то вершину a називають *предком* (або *батьком*), а вершину b – *нащадком* (або *сином*). Вершини орієнтованого дерева, які не мають синів, називають *листками*. Вершини (окрім кореня), які мають синів, називають *внутрішніми* [5].

Орієнтований ліс – це ліс, що складається з орієнтованих дерев. *З'єднувальним орієнтованим деревом* (або *орієнтованим каркасом*) орграфа називають орієнтоване дерево, що містить усі вершини цього орграфа [5]. *З'єднувальним орієнтованим лісом* орграфа називають орієнтований ліс, що містить усі вершини цього графа [5].

1.3 Навантажені графи

Часто у реальних задачах ребро $[u, v]$ неорієнтованого графа чи дугу (u, v) орграфа *навантажують* дійсним числом – *вагою* $w(u, v)$, яка відображає кількісну характеристику ребра/дуги (наприклад: відстань, швидкість, ціна, пропускна здатність, дохід, прибуток тощо) [5]. Ваги, зазвичай, є невід'ємними

числами. У деяких випадках ваги мають від'ємні значення (витрати, штрафи тощо) [5].

Граф, у якому кожне ребро/дуга має вагу, називають *навантаженим (зваженим)* графом. Приклад орієнтованого навантаженого графа наведено на рис. 1.3 (поруч з кожною дугою проставлено її вагу).

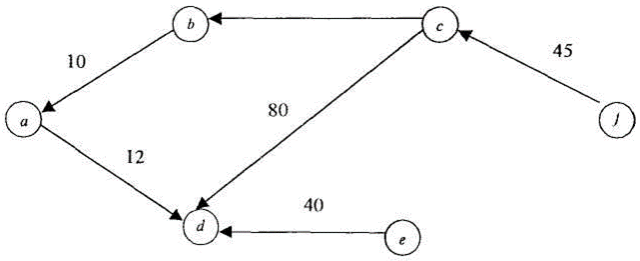

Рис. 1.3. Приклад орієнтованого навантаженого графа

Найчастіше для зберігання навантаженого графа $G = (V, E)$ у пам'яті комп'ютера використовують *матрицю ваг* [5]. Для цього нумерують вершини графа G числами 1, 2, …, n, де $n = |V|$, і розглядають матрицю $(w_{ij})_{n\times n}$, у якої $w_{ij} = w(i,j)$, якщо ребро $[i,j]$ чи дуга $(i,j) \in G$. Якщо ж ребро $[i, j]$ чи дуга $(i, j) \notin G$, то $w_{ij} = 0$ чи $w_{ij} = \infty$ (залежно від змісту задачі). Для простого навантаженого графа матриця ваг є симетричною відносно головної діагоналі.

Матриці ваг неефективно використовують пам'ять під час збереження розріджених графів, у яких $|E| \ll |V|^2$. Для зберігання розрідженого навантаженого графа $G = (V, E)$ використовують *списки суміжних вершин*.

2 Задача про найкоротший шлях

2.1 Постановка задачі

Задачею про найкоротший шлях називають таку задачу, де шукають короткий шлях(ланцюг) між двома вершинами на графі, у якому здійснюється мінімізація суми ваг ребер, які утворюють шлях.

Задача про найкоротший шлях являється однією з найважливіших задач теорії графів. Для вирішення цієї задачі існує безліч алгоритмів.

Ця задача містить різноманітні постановки. Три найбільш поширені описано нижче:

1. Задача про найкоротший шлях, коли задано пункт призначення. У даній задачі необхідно знайти найкоротший шлях у визначену вершину призначення s. Більш того, шлях може починатися з будь-якої вершини графа, окрім s. Якщо розглядати цю задачу навпаки, тобто змінити напрямок ребер, то отримаємо задачу пошуку найкоротших шляхів із заданої вершини в усі інші;
2. Задача про найкоротший шлях між визначеною парою вершин. Така задача зводиться до пошуку найкоротшого шляху із деякої вершини u в деяку вершину v;
3. Задача пошуку найкоротших шляхів між усіма парами вершин. У даному випадку шукають найкоротший шлях з кожної вершини u в кожну вершину v. Такий вид задачі можна розв’язати також алгоритмом для пошуку найкоротшого шляху з однієї вершини в усі інші, проте це займає більше часу.

Розглядаємо навантажений орієнтований граф, який може мати чи не мати циклів. Довжина довільної дуги (a, b) може не збігатися з довжиною дуги (b, a). Для неорієнтованих графів будь-яке ребро $[i, j]$ довжиною d_{ij} можна замінити парою дуг (i, j) та (j, i), кожна з яких матиме довжину d_{ij}.

Під оптимальними шляхами у графі, зазвичай, розуміють найкоротші шляхи між двома (або між усіма) вершинами графа. Проте у деяких задачах необхідно визначати і найдовші шляхи.

Уточнимо деякі визначення. У задачі про найкоротші шляхи дано орієнтований граф $G = (V, E)$ з ваговою функцією $w: E \to R$. *Довжиною* шляху $p =< v_0, v_1, \dots, v_{n-1}, v_n >$ називають суму довжин(ваг) дуг, які належать до цього шляху, тобто $w(p) = \sum_{i=1}^{n} w(v_{i-1}, v_i)$.

Довжину найкоротшого шляху з u в v визначають так:

$$\delta(u, v) = \begin{cases} min\left\{w(p): u \overset{p}{\to} v\right\}, & \text{якщо } \exists u \overset{p}{\to} v, \\ \infty, & \text{інакше.} \end{cases} \qquad (1.1)$$

Отже, якщо деякий шлях p з u до v є *найкоротшим* шляхом з u до v, то $w(p) = \delta(u, v)$. Визначимо такі задачі:

1. Від заданої *початкової* вершини графа s виокремити найкоротші шляхи до *всіх* інших вершин графа [5].
2. Від *усіх* вершин графа виокремити найкоротші шляхи до деякої *кінцевої* вершини графа t [5].
3. Від *заданої* вершини графа v_i виокремити найкоротший шлях до деякої *іншої* вершини графа v_j [5].
4. Виокремити найкоротші шляхи між *усіма парами* вершин графа [5].

Алгоритм, який розв'язує першу задачу, дає змогу розв'язати й інші три задачі. Розв'язування задачі 2 зводиться до розв'язування задачі 1, якщо умовно поміняти напрям усіх дуг [5]. Якщо відшукаємо найкоротші шляхи від початкової вершини v_i до *всіх* інших вершин графа, то одночасно відшукаємо і найкоротший шлях від вершини v_j до вершини v_j (задача 3) [5].

Задачу 4 можна розв'язати або багатократним застосуванням алгоритму задачі 1, де на кожнім кроці за початкову вершину s беруть інші вершини графа,

або однократним застосуванням *спеціального* алгоритму [5].

У деяких застосуваннях ваги дуг можуть бути *від'ємними*. У цьому випадку важливо, чи є цикли від'ємної довжини [5]. Якщо з вершини s можна добратися до такого циклу, то потім його можна обходити як завгодно довго, і вага шляху усе зменшуватиметься, отож для вершин цього циклу найкоротших шляхів не існує (рис. 1.4) [5]. У цьому випадку вважатимемо, що вага найкоротшого шляху дорівнює – ∞ [5].

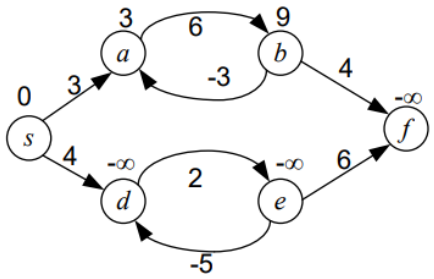

Рис.1.4. Цикл від'ємної ваги

На рис. 1.4 поблизу кожної вершини проставлено вагу найкоротшого шляху від вершини s. Якщо ж циклів від'ємної ваги немає, то будь-який цикл можна викинути, не подовжуючи шляху[5].

2.2 Найкоротші шляхи

Для зображення дуг найкоротших шляхів використовується такий же спосіб, як і в алгоритмі Пріма. Для кожної вершини $v \in V$ зберігається її *попередник* $\pi[v]$. Попередник вершини – це або інша вершина, або 0 (для початкової вершини s).

Під час роботи алгоритмів виокремлення найкоротших шляхів утворюється орієнтований підграф попередників $G_\pi = (V_\pi, E_\pi)$, де $V_\pi = \{v \in V : \pi[v] \neq 0\} \cup \{s\}$; $E_\pi = \{(\pi[v], v) \in E : v \in V_\pi \backslash \{s\}\}$. Після закінчення роботи цих алгоритмів граф G_π буде деревом найкоротших шляхів від кореня s до будь-якої, досяжної з кореня, вершини $v \in V$. Найкоротших шляхів від s до вершин, досяжних з s, а також дерев найкоротших шляхів у графі G, може бути декілька.

Усі алгоритми виокремлення найкоротшого (або найдовшого) шляху в графі

є багатокроковими процедурами ухвалення рішень в умовах визначеності, оскільки при досягненні певної вершини графа обирається наступна дуга цього шляху[5]. Під час реалізації цієї процедури користуються принципом оптимальності (або принципом Беллмана), який для навантаженого орграфа формулюють такою лемою [5].

Лема 1.2. *Відрізки найкоротших шляхів є найкоротшими*. Нехай $G = (V, E)$ – орієнтований граф з ваговою функцією $w: E \to R$. Якщо $p =< v_0, v_1, \dots, v_{n-1}, v_n >$ – найкоротший шлях з v_0 до v_n і $0 \le i \le j \le n$, то $p_{ij} =< v_i, v_{i+1}, \dots, v_{j-1}, v_j >$ є найкоротшим шляхом з v_i до v_j [5].

Доведення: Від супротивного. Нехай шлях p_{ij} не найкоротший. Замінивши шлях з v_i до v_j на коротший, ми тим самим зменшуємо довжину шляху з v_0 до v_n (*протиріччя*) [5].

Наслідок з леми 1.2. Нехай $G = (V, E)$ – орієнтований граф з ваговою функцією $w: E \to R$ і p – найкоротший шлях з s до v. Якщо (u, v) – остання дуга цього шляху, то $\delta(s, v) = \delta(s, u) + w(u, v)$ [5].

Доведення: Нехай p' – шлях з s до u. За лемою 1.2 p' – найкоротший шлях, отож $\delta(s, v) = w(p') + w(u, v) = \delta(s, u) + w(u, v)$ [5].

Лема 1.3. Нехай $G = (V, E)$ – орієнтований граф з ваговою функцією $w: E \to R$ і $s \in V$. Тоді для будь-якої дуги $(u, v) \in E$ вірна нерівність $\delta(s, v) \le \delta(s, u) + w(u, v)$ [5].

Доведення: Довжина найкоротшого шляху з s до v не перевищує довжину будь-якого шляху з s до v, у тім числі й шляху через дугу (u, v) [5].

Усі алгоритми виокремлення найкоротшого (або найдовшого) шляху, які описано у цьому підрозділі, базуються на принципі релаксації (від relaxation – “полегшення”) [5]. Згідно з цим принципом кожній вершині $v \in V$ приписують число $d[v]$ – верхню оцінку довжини найкоротшого шляху з s до v (або просто, *оцінку найкоротшого шляху*) [5].

3 Алгоритми розв'язання задачі про найкоротший шлях

3.1 Алгоритм Дейкстри

Алгоритм Дейкстри винайшов нідерландський вчений Едсгер Дейкстри в 1959 році. Цей алгоритм здатен знайти, наприклад, послідовність доріг, яку буде краще використати для поїздки з одного міста до багатьох інших, якщо наявна необхідна інформація, або ж він може вказати куди експортування нафти буде найвигіднішим тощо. Алгоритм знаходить своє застосування також у програмуванні та технологіях, наприклад, він використовується у протоколах маршрутизації OSPF та IS-IS.

Алгоритм Дейкстри містить недолік – він не здатен обробляти графи з від'ємною вагою. Якщо припустити, що деякій системі передбачено маршрути, які є збитковими для фірми, то доведеться шукати інший алгоритм.

Алгоритм Дейкстри розв'язує задачу про найкоротші шляхи для навантаженого графа $G = (V, E)$ від початкової вершини $s \in V$ до всіх інших вершин досяжних з s, у якому довжини усіх дуг *невід'ємні*.

Для кожної вершини v зберігаються довжина найкоротшого шляху від s до v, знайденого до цих пір та вершина, що є предком v на цьому шляху. Алгоритм будує оптимальні шляхи ітераційно і покращує рішення на кожному кроці. У результаті буде отримано найкоротший шлях від початкової вершини графа до всіх інших.

Щоб описати алгоритм необхідно ввести такі визначення:

- A – множина вершин v, для яких найкоротший шлях від s до v вже знайдено(множина опрацьованих вершин);
- X – множина вершин v, для яких шлях від s до v ще не визначений (множина вершин, що чекають своєї черги на опрацювання);
- $d[v]$ - довжина найкоротшого шляху від s до v, знайдена на момент поточної ітерації (це можна розглядати як оцінку або верхню межу для найкоротшого шляху від s до v);
- $d[s] = 0$, тобто немає ребра з s в s (алгоритм не передбачає петель).

На початку множина опрацьованих вершин $A = \emptyset$, а $X = \{s\}$, тобто вершина s буде на активному етапі дослідження.

Необхідно повторювати наступні кроки алгоритму, доки X не буде дорівнювати пустій множині:

1. Знайти вершину v з множини X таку, що

$$d[v] \coloneqq \min_{u \in X} d[u], \tag{1.2}$$

тоді видалити v з X та додати до A.

2. Для кожної вершини u, для якої виконується $(v, u) \in E$ (кожна сусідня вершина v):

а) Якщо A містить u, то не виконувати жодних дій.

б) Якщо X містить u, то якщо $d[v] + w(v,u) < d[u]$ то встановити $d[u] = d[v] + w(v,u)$ та $parent(u) = v$, де $w(v,u)$ – вага дуги (v, u).

в) Якщо u не міститься ані у A, ані у X, додати u до X та встановити $d[u] = d[v] + w(v,u)$ та $parent(u) = v$, $w(v,u)$ – вага дуги (v, u).

Коли алгоритм закінчується, всі вузли, які можуть бути досягнуті з s, будуть у A, і задачу буде вирішено.

Алгоритм не може бути завершеним лише знаходженням довжин найкоротших шляхів. Необхідно також визначати самі шляхи, які будуть найкоротшими. Для цього використовується масив "предків". Цей масив зберігає номери вершин, які є попередниками до деякої вершини v. Коли з вибраної вершини u відбувається покращення відстані до вершини v, то це означає, що предком для вершини v є вершина u $(parent(v) = u)$. Саме так і заповнюється масив “предків”.

3.2 Алгоритм Флойда Уоршела

Алгоритмом для знаходження найкоротших відстаней та шляхів між усіма парами вершин у зваженому орієнтованому графі являється алгоритм Флойда-Уоршела. Цей алгоритм розроблений в 1962 році Робертом Флойдом і Стівеном Уоршелом, але також відомо, що в 1959 році було опубліковано схожий алгоритм Бернардом Роєм, проте тоді цього ніхто не зауважив.

Умовою застосування алгоритму є відсутність в графі циклів з сумарною від’ємною вагою, але алгоритм може їх виявити [8]. Негативний цикл – це цикл, у

якому сума всіх ребер є меншою за нуль. Якщо існує така пара вершин (i, j), у якої довжина шляху постійно зменшується, то це означає, що вона входить до негативного циклу, тому між цією парою вершин немає найкоротшого шляху. Для отримання результату, у алгоритмі Флойда-Уоршела передбачається відсутність негативних циклів у графах. Якщо ж негативний цикл існує, то алгоритм Флойда-Уоршела зможе його виявити. Наступні міркування мають певний сенс:

- Алгоритм Флойда-Уоршела багаторазово змінює довжини шляху між усіма парами вершин (i, j), включаючи ті, де $i = j$;
- Початкова довжина шляху (i, i) рівна 0;
- Шлях $i \to j \to \cdots \to i$ може покращити початкову довжину, якщо він має довжину меншу за нуль, тобто позначає негативний цикл;
- Таким чином, після виконання алгоритму, найкоротший шлях (i, i) буде від'ємним, якщо існує негативний цикл — шлях від'ємної довжини від i до i.

Отже, для виявлення негативних циклів з використанням алгоритму Флойда-Уоршела необхідно перевіряти діагональ матриці шляхів. Якщо буде наявне від’ємне число, то це означає, що граф містить хоча б один негативний цикл. Наявність негативного циклу при виконанні алгоритму сприяє появі чисел, що будуть перевищувати модуль найменшої від’ємної ваги ребра.

Розглянемо орієнтований граф $G = (V, E)$, де V – вершини графа, E – ребра графа. Нехай $A^0 = \|a_{ij}\|$ – матриця довжин дуг цього графа [8]. Покладемо $a_{ij} = \infty$, якщо дуга (i, j) відсутня і $a_{ii} = 0$ для всіх $i \in V$. Вважаємо, що петель немає [8]. Алгоритм Флойда будує послідовність матриць $A^0, A^1, \dots, A^n$ таку, що елемент a_{ij}^n матриці A^n дорівнює довжині найкоротшого шляху від вершини i до вершини j в графі G [8]. Матриця A^k визначається за матрицею A^{k-1} такою формулою

$$a_{ij}^k = \min\{a_{ij}^{k-1}, a_{ik}^{k-1} + a_{kj}^{k-1}\} \qquad (1.3)$$

В алгоритмі Флойда одночасно з довжинами найкоротших шляхів шукаємо самі шляхи за допомогою матриць $P^0, P^1, \dots, P^n$, де елемент p_{ij}^k матриці P^k вказує на вершину, яка передує вершині j на шляху від вершини i до j [8]. Спочатку для всіх i

та j покладаємо $p_{ij}^0 = i$, якщо існує дуга (i, j) (тобто відповідний елемент матриці A^0 не дорівнює ∞) [8]. В іншому випадку $p_{ij}^0 = 0$. Далі на кожній ітерації матриця P^k перераховується разом з матрицею A^k за таким правилом [8]:

$$p_{ij}^k = \begin{cases} p_{kj}^{k-1}, \text{якщо } a_{ij}^k > a_{ik}^{k-1} + a_{kj}^{k-1} \\ p_{ij}^{k-1}, \text{якщо } a_{ij}^k \leq a_{ik}^{k-1} + a_{kj}^{k-1} \end{cases} \quad (1.4)$$

тобто, якщо на k-ій ітерації елемент a_{ij}^k міняється, то елемент p_{ij}^k отримує значення, що дорівнює елементу, який розташований в тому ж стовпці у k - му рядку [8].

3.3 Алгоритм Беллмана-Форда

Алгоритм Беллмана-Форда - алгоритм пошуку найкоротшого шляху у зваженому графі. Алгоритм здатен вирішувати задачу пошуку найкоротших шляхів від однієї вершини до всіх інших. Цей алгоритм застосовується також і до задач з графами, де наявні від'ємні ваги ребер, і це його перевага над алгоритмом Дейкстри. Його запропонували Річард Беллман і Лестер Форд.

Маємо орієнтований граф $G = (V, E)$ зі зваженими ребрами з n вершинами. Кожне ребро (u, v) має вагу w. Вага ребра може бути від'ємною або дорівнювати нулю. Довжиною шляху є сума ваг ребер, що входять в цей шлях. Потрібно знайти найкоротші шляхи від початкової вершини s до всіх інших вершин графа.

Слід зазначити, що найкоротших шляхів може і не бути. Граф, який містить цикл з сумарною вагою від'ємною, міститиме як завгодно короткий шлях від однієї вершини цього циклу до іншої, тобто кожен прохід циклу зменшуватиме довжину шляху. Цикл, сума ваг ребер якого негативна, називається негативним циклом і про нього вже згадувалось у підрозділі 3.2.

На початку алгоритму, кожній вершині призначається її відстань рівна ∞, окрім початкової вершини, якій призначається відстань рівна нулю: $d[s] = 0$, $d[v] = \infty$. Для кожного ребра $(u, v) \in E$ виконується перебір з умовою:

- Якщо $d[v] > d[u] + w(u, v)$, то $d[v] = d[u] + w(u, v)$. Іншими словами, $d[v] = \min\{d[v], d[u] + w(u, v)\}$ і предком вершини v буде вершина u.

Алгоритм здійснює такий перебір кілька разів, а саме не більше, ніж кількість вершин графа, поки не виявиться, що неможливо більше знайти ще менших відстаней. Вершина, у якої відстань є найменшою з усіх перерахованих, вважається “пройденою”, а ребро до цієї вершини - переглянутим. Якщо для деякої “пройденої” вершини v відбувається зменшення відстані $d[v]$, то ця вершина вже не буде “пройденою” і ребро не буде переглянутим (позначення знімаються).

3.4 Алгоритм Джонсона

Для знаходження найкоротших шляхів між усіма парами вершин у зваженому орієнтованому графі використовується алгоритм Джонсона. Як і попередньо розглянуті алгоритми Флойда-Уоршела та Беллмана-Форда, цей алгоритм може шукати найкоротші шляхи у графі, який містить ребра як з додатними вагами, так і з від’ємними, але без негативних циклів.

Нехай $G = (V, E)$ – граф з ваговою функцією $w: E \to R$. Якщо у даному графі не міститься ребер з від’ємними вагами, то для знаходження найкоротших відстаней між усіма парами вершин буде достатньо використати алгоритм Дейкстри по одному разу для кожної вершини. Якщо ж у графі будуть присутні ребра з від’мною вагою, але без негативних циклів, то можна знайти нові ваги ребер, які будуть додатні, та скористатись попереднім алгоритмом.

Нехай $w(u, v)$ – ваги ребер (u, v), а $\widehat{w}(u, v)$ – нові ваги ребер (u, v). Тоді для нових ваг виконуються наступні властивості:

1) $\widehat{w}(u, v) \geq 0$, для всіх ребер (u, v);
2) Шлях p буде найкоротшим шляхом з вершини u до вершини v з ваговою функцією w, для всіх таких пар вершин $u, v \in V$, тоді і тільки тоді, коли шлях p є найкоротшим шляхом з вершини u до вершини v з ваговою функцією $\widehat{w}$, тобто те, що $w(p) = \delta(u, v)$ є рівносильним тому, що $\widehat{w}(p) = \hat{\delta}(u, v)$.

Також слід зазначити, що граф G має негативний цикл з використанням вагової функції w тоді й лише тоді, коли він містить негативний цикл з ваговою функцією $\widehat{w}$.

Лема 1.4 (Про збереження найкоротших шляхів при зміні ваг). Нехай $h: V \to R$ – довільна функція, яка відображає вершини на дійсні числа. Тоді $\widehat{w}(u, v) = w(u, v) + h(u) - h(v)$ для всіх $(u, v) \in E$.

Зміна ваги відбувається наступним чином:

1) Створюється новий граф $G' = (V', E')$, де до множини вершин V додається вершина s, тобто $V' = V \cup \{s\}$, а $E' = E \cup \{(s, v): v \in V\}$;
2) Розширюється вагова функція w так, щоб для всіх вершин $v \in V$ виконувалося, що $w(s, v) = 0$;
3) Визначається для всіх вершин $v \in V'$ величина $h(v)$ – довжина найкоротшого шляху від деякої вершини s до v, а також визначаються нові ваги для всіх ребер $\widehat{w}(u, v) = w(u, v) + h(u) - h(v) \geq 0$.

Слід зазначити, що додання нової вершини s не змінює величину найкоротших шляхів між вершинами u та v у графі G.

Для знаходження найкоротших відстаней у графі з від'ємними вагами від вершини s до всі інших вершин у алгоритмі Джонсона використовується алгоритм Беллмана-Форда. Далі працює безпосередньо алгоритм Джонсона, шукаючи нові ваги всіх ребер. Алгоритм Дейкстри знаходить найкоротші шляхи у новому графі.

Останнім кроком є визначення реальної відстані, тобто $d[v] = d'[v] - h(u) + h(v)$.

РОЗДІЛ 2. ПРАКТИЧНА ЧАСТИНА

1 Розв’язання задач про найкоротші шляхи за допомогою алгоритмів

1.1 Використання алгоритму Дейкстри

Постановка задачі

Нехай необхідно знайти відстані від першої вершини неорієнтованого навантаженого графа до всіх інших.

Існує розподілена система керування, що містить шість мікроконтролерів, пов’язаних так, як показано на рисунку 2.1

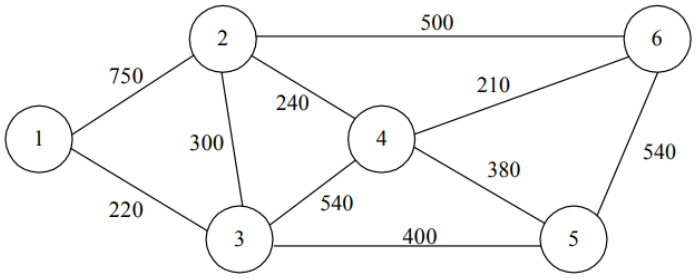

Рис. 2.1. Розподілена система керування

Задані також відстані між контролерами. Визначити найбільш ефективні маршрути пересилання повідомлень від першого контролера до будь-якого іншого.

Розв’язання задачі

На рис. 2.1 кожна вершина пронумерована у кружечку, а відстані між контролерами позначені надписом над ребром графа. У процесі розв’язання будемо позначати надписом над кружечком (вершиною) найкоротшу відстань до вершини.

Крок 1. Ініціалізація. Відстань до всіх вершин графа рівна ∞, а до вершини 1 рівна 0. Отже, множина опрацьованих вершин $A = \emptyset$, а множина вершин, що в черзі на опрацювання $X = \{1\}$(рис. 2.2):

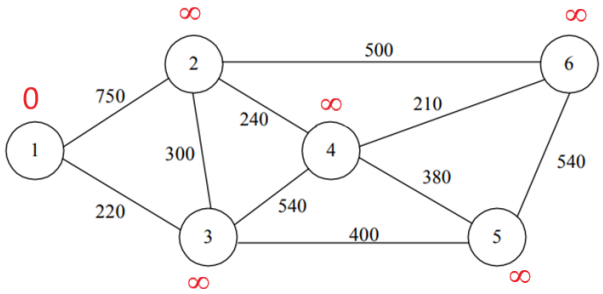

Рис. 2.2

Крок 2. Вершина 1 є першою на опрацюванні, $d[1] = 0$ і ця відстань є мінімальною . Тепер обходимо всі сусідні вершини до вершини 1. 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. Отже, вершини 2 та 3 є сусідні до вершини 1, тоді:

1) $d[1] + w(1,2) < d[2] \Rightarrow 0 + 750 < \infty$, тобто $d[2] = d[1] + w(1,2) = 0 + 750 = 750$ – найкоротший шлях вершини 2;
2) $d[1] + w(1,3) < d[3] \Rightarrow 0 + 220 < \infty$, тобто $d[3] = d[1] + w(1,3) = 0 + 220 = 220$ – найкоротший шлях вершини 3;

Усі сусіди вершини 1 перевірені (рис. 2.3).

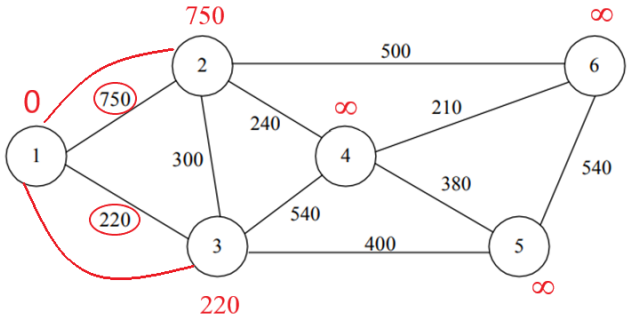

Рис. 2.3

Поточна мінімальна відстань до вершини 1 вважається остаточною і обговоренню не підлягає (те, що це дійсно так, вперше довів Дейкстра), тому в подальшому викреслимо її з графа, щоб відмітити цей факт (рис. 2.4).

Крок 3. Тепер $A = \{1\}$, а множина $X = \{3\}$, тому що ця вершина не належить множині A і є "найближчою", тобто її відстань є найменшою. Також відзначимо, що предком вершини 3 є вершина 1, тому цю вершину, як сусідню до вершини 3, розглядати вже не будемо. Тоді повторюємо крок 2, тобто обходимо сусідів вершини 3 і визначаємо їхній найкоротший шлях (рис. 2.4):

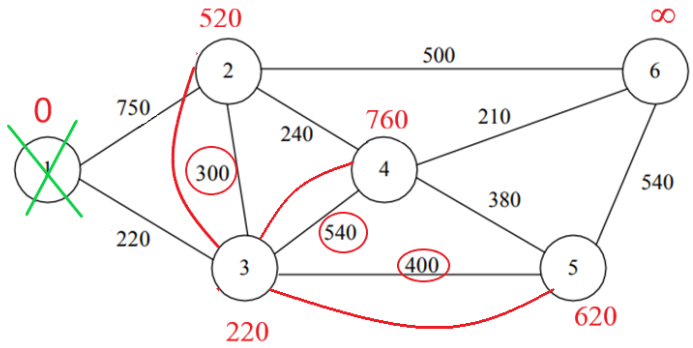

Рис.2.4

1) $d[3] + w(3,2) < d[2] \Rightarrow 220 + 300 < 750$, тобто $d[2] = d[3] + w(3,2) = 220 + 300 = 520$ – найкоротший шлях вершини 2;
2) $d[3] + w(3,4) < d[4] \Rightarrow 220 + 540 < \infty$, тобто $d[4] = d[3] + w(3,4) = 220 + 540 = 760$ – найкоротший шлях вершини 4;
3) $d[3] + w(3,5) < d[5] \Rightarrow 220 + 400 < \infty$, тобто $d[5] = d[3] + w(3,5) = 220 + 400 = 620$ – найкоротший шлях вершини 5;

Поточна мінімальна відстань до вершини 3 вважається остаточною і обговоренню не підлягає, тому викреслимо її з графа (рис. 2.5).

Крок 4. Множина $A = \{1,3\}$, $X = \{2\}$, тому що ця вершина не належить множині A і є "найближчою", тобто її відстань є найменшою. Також відзначимо, що предком вершини 2 є вершина 3, тому цю вершину, як сусідню до вершини 2, розглядати вже не будемо. Тоді обходимо сусідів вершини 2 і визначаємо їхній найкоротший шлях (рис.2.5):

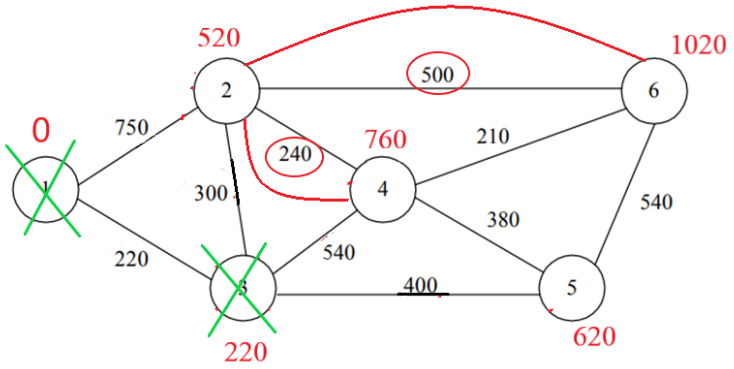

Рис.2.5

1) $d[2]+w(2,4)<d[4] \Rightarrow 520+240=760$, тобто $d[4]=760$ – найкоротший шлях вершини 4 і він залишається без змін;
2) $d[2]+w(2,6)<d[6] \Rightarrow 520+500<\infty$, тобто $d[6]=d[2]+w(2,6)=520+500=1020$ – найкоротший шлях вершини 6;

Поточна мінімальна відстань до вершини 2 вважається остаточною, тому викреслимо її з графа (рис.2.6).

Крок 5. Множина $A=\{1,3,2\}$, $X=\{4\}$, тому що ця вершина не належить множині A і є "найближчою", тобто її відстань є найменшою. Також відзначимо, що предком вершини 4 є вершина 2. Тоді обходимо сусідів вершини 4 і визначаємо їхній найкоротший шлях (рис.2.6):

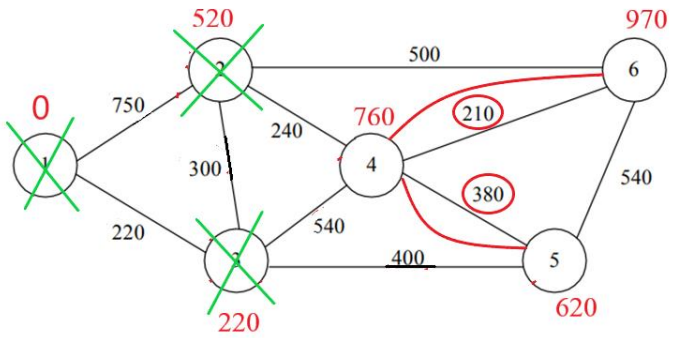

Рис.2.6

1) $d[4]+w(4,5)<d[5] \Rightarrow 760+380>620$, тобто $d[5]=620$ – найкоротший шлях вершини 5 і він залишається без змін;

2) $d[4]+w(4,6)<d[6] \Rightarrow 760+210<1020$, тобто $d[6]=d[4]+w(4,6)=760+210=970$ – найкоротший шлях вершини 6;

Поточна мінімальна відстань до вершини 4 вважається остаточною, тому викреслимо її з графа (рис.2.7).

Крок 6. Множина $A=\{1,3,2,4\}$, $X=\{5\}$, тому що ця вершина не належить множині A і є "найближчою", тобто її відстань є найменшою. Також відзначимо, що предком вершини 5 є вершина 4. Тоді обходимо сусідів вершини 5, а це тільки вершина 6: $d[5]+w(5,6)<d[6] \Rightarrow 620+540>970$, тобто $d[6]=970$ – найкоротший шлях вершини 6 і він залишається без змін.

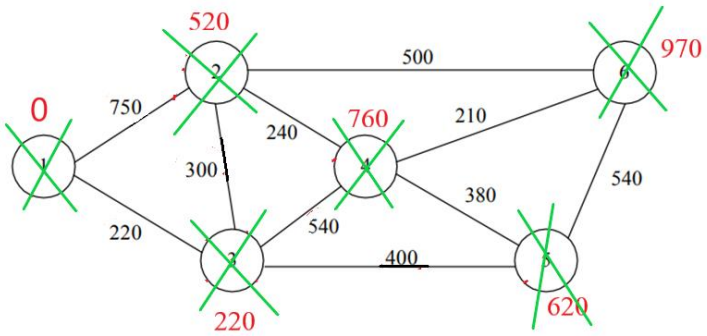

Рис.2.7

Поточна мінімальна відстань до вершини 6 вважається остаточною, тому викреслимо її з графа. Алгоритм закінчує роботу, коли викреслені всі вершини (рис.2.7). Тоді отримаємо остаточний результат:

$1 \to 2$: $d[2]=520$; $1 \to 3$: $d[3]=220$; $1 \to 4$: $d[4]=760$; $1 \to 5$: $d[5]=620$; $1 \to 6$: $d[6]=970$.

1.2 Використання алгоритму Флойда-Уоршела

Постановка задачі

Нехай необхідно знайти найкоротший шлях між всіма парами вершин та їх довжини у орієнтованому навантаженому графі.

Наведено схему мережі вулиць між п’ятьма факультетами університету, де рух автомобілів дозволений у напрямках, які показано на рисунку 2.8:

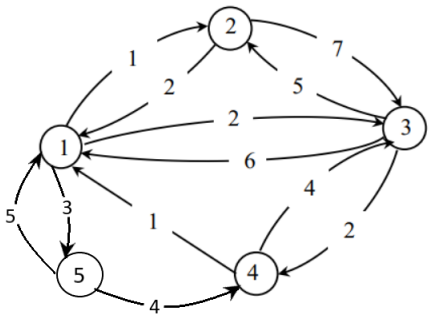

Рис.2.8. Мережа вулиць між факультетами

Задано також відстані між факультетами у кілометрах. Визначте найкоротший шлях між всіма факультетами та їх відстані.

Розв'язання задачі

На рис. 2.8 у кружечку вказано номери факультетів, а на ребрах графа – відстані між ними. Побудуємо матрицю довжин (рис. 2.9).

$$A^0 = \begin{array}{c|cccc} & 0 & 1 & 2 & \infty & 3 \\ \hline 2 & 0 & 7 & \infty & \infty \\ 6 & 5 & 0 & 2 & \infty \\ 1 & \infty & 4 & 0 & \infty \\ 5 & \infty & \infty & 4 & 0 \end{array}$$

Рис.2.9

Крок 1. Виділяємо перший стовпець та перший рядок матриці A^0. Виділені елементи не перераховуються. 4-ий елемент виділеного рядка дорівнює ∞, тому відповідний стовпчик (4-ий) також буде перераховуватися. Таке твердження аналогічно працює і для виділеного стовпця, проте на цьому етапі у виділеному стовпці немає ∞.

Оскільки довжини всіх дуг невід'ємні, то діагональні елементи матриці сталі й дорівнюють нулю. Таким чином, потрібно перерахувати лише елементи $a_{23}^0, a_{25}^0, a_{32}^0, a_{35}^0, a_{42}^0, a_{43}^0, a_{45}^0, a_{52}^0, a_{53}^0$(виділені на рисунку 2.9). Отримаємо такі елементи:

$a_{23}^1 = \min(7,2+2) = 4$; $a_{25}^1 = \min(\infty, 2+3) = 5$

$a_{32}^1 = \min(5,6+1) = 5$; $a_{35}^1 = \min(\infty, 6+3) = 9$

$a_{42}^1 = \min(\infty, 1+1) = 2$; $a_{43}^1 = \min(4,1+2) = 3$; $a_{45}^1 = \min(\infty, 1+3) = 4$

$a_{52}^1 = \min(\infty, 5+1) = 6$; $a_{53}^1 = \min(\infty, 5+2) = 7$.

Звідси, отримуємо таку матрицю довжин:

$$A^1 = \begin{vmatrix} 0 & 1 & 2 & \infty & 3 \\ 2 & 0 & 4 & \infty & 5 \\ 6 & 5 & 0 & 2 & 9 \\ 1 & 2 & 3 & 0 & 4 \\ 5 & 6 & 7 & 4 & 0 \end{vmatrix}$$

Знайдемо відповідні шляхи. Матриця P^0 має вигляд

$$P^0 = \begin{vmatrix} 0 & 1 & 1 & 0 & 1 \\ 2 & 0 & 2 & 0 & 0 \\ 3 & 3 & 0 & 3 & 0 \\ 4 & 0 & 4 & 0 & 0 \\ 5 & 0 & 0 & 5 & 0 \end{vmatrix}$$

Матриця A^0 відрізняється від матриці A^1 елементами $a_{23}^0, a_{25}^0, a_{35}^0, a_{42}^0, a_{43}^0, a_{45}^0, a_{52}^0, a_{53}^0$. Відповідні елементи матриці P^1 будуть дорівнювати елементам першого рядка матриці P^0, які розташовані в тому ж стовпці, тобто

$$p_{23}^1 = p_{13}^0 = 1; p_{25}^1 = p_{15}^0 = 1;$$
$$p_{35}^1 = p_{15}^0 = 1; p_{42}^1 = p_{12}^0 = 1;$$
$$p_{43}^1 = p_{13}^0 = 1; p_{45}^1 = p_{15}^0 = 1;$$
$$p_{52}^1 = p_{12}^0 = 1; p_{53}^1 = p_{13}^0 = 1$$

$$P^1 = \begin{vmatrix} 0 & 1 & 1 & 0 & 1 \\ 2 & 0 & 1 & 0 & 1 \\ 3 & 3 & 0 & 3 & 1 \\ 4 & 1 & 1 & 0 & 1 \\ 5 & 1 & 1 & 5 & 0 \end{vmatrix}.$$

Крок 2. У матриці A^1 виділяємо другий рядок та другий стовпчик. 4-ий стовпець не перераховуємо, бо у виділеному рядку 4-ий елемент дорівнює ∞ (рис. 2.10).

$$A^1 = \begin{vmatrix} 0 & 1 & 2 & \infty & 3 \\ 2 & 0 & 4 & \infty & 5 \\ 6 & 5 & 0 & 2 & 9 \\ 1 & 2 & 3 & 0 & 4 \\ 5 & 6 & 7 & 4 & 0 \end{vmatrix}$$

Рис. 2.10

Виявляється, що усі виділені елементи матриці не зміняться, бо вони вже є мінімальними, тому матриця A^2 буде рівна матриці A^1 (рис.2.10). Отже, матриця P^2 також не зміниться і буде рівна матриці P^1.

Крок 3. У матриці A^2 виділяємо третій рядок і третій стовпець. Виділеними елементами, які потрібно перерахувати, будуть лише a_{14}^2, a_{24}^2, тому що всі інші вже є мінімальними (рис.2.11).

$$A^2 = \begin{vmatrix} 0 & 1 & 2 & \infty & 3 \\ 2 & 0 & 4 & \infty & 5 \\ 6 & 5 & 0 & 2 & 9 \\ 1 & 2 & 3 & 0 & 4 \\ 5 & 6 & 7 & 4 & 0 \end{vmatrix}$$

Рис. 2.11

$a_{14}^2 = \min(\infty, 2+2) = 4$; $a_{24}^2 = \min(\infty, 2+4) = 6$

Звідси, отримуємо таку матрицю довжин:

$$A^3 = \begin{vmatrix} 0 & 1 & 2 & 4 & 3 \\ 2 & 0 & 4 & 6 & 5 \\ 6 & 5 & 0 & 2 & 9 \\ 1 & 2 & 3 & 0 & 4 \\ 5 & 6 & 7 & 4 & 0 \end{vmatrix}$$

Знайдемо відповідні шляхи. Матриця P^2 має вигляд

$$P^2 = \begin{vmatrix} 0 & 1 & 1 & 0 & 1 \\ 2 & 0 & 1 & 0 & 1 \\ 3 & 3 & 0 & 3 & 1 \\ 4 & 1 & 1 & 0 & 1 \\ 5 & 1 & 1 & 5 & 0 \end{vmatrix}$$

Матриця A^2 відрізняється від матриці A^3 елементами a_{14}^2, a_{24}^2. Відповідні елементи матриці P^3 будуть дорівнювати елементам третього рядка матриці P^2, які розташовані в тому ж стовпці, тобто

$$p_{14}^3 = p_{34}^2 = 3; p_{24}^3 = p_{34}^2 = 3;$$

$$P^3 = \begin{vmatrix} 0 & 1 & 1 & 3 & 1 \\ 2 & 0 & 1 & 3 & 1 \\ 3 & 3 & 0 & 3 & 1 \\ 4 & 1 & 1 & 0 & 1 \\ 5 & 1 & 1 & 5 & 0 \end{vmatrix}.$$

Крок 4. У матриці A^3 виділяємо четвертий рядок і четвертий стовпець. Виділеними елементами, які потрібно перерахувати, будуть лише $a_{31}^3, a_{32}^3, a_{35}^3$, тому що всі інші вже є мінімальними (рис.2.12).

$$A^3 = \begin{vmatrix} 0 & 1 & 2 & 4 & 3 \\ 2 & 0 & 4 & 6 & 5 \\ 6 & 5 & 0 & 2 & 9 \\ 1 & 2 & 3 & 0 & 4 \\ 5 & 6 & 7 & 4 & 0 \end{vmatrix}$$

Рис. 2.12

$a_{31}^3 = \min(6,1+2) = 3$; $a_{32}^3 = \min(5,2+2) = 4$; $a_{35}^3 = \min(9,2+4) = 6$

Звідси, отримуємо таку матрицю довжин:

$$A^4 = \begin{vmatrix} 0 & 1 & 2 & 4 & 3 \\ 2 & 0 & 4 & 6 & 5 \\ 3 & 4 & 0 & 2 & 6 \\ 1 & 2 & 3 & 0 & 4 \\ 5 & 6 & 7 & 4 & 0 \end{vmatrix}$$

Знайдемо відповідні шляхи. Матриця P^3 має вигляд

$$P^3 = \begin{vmatrix} 0 & 1 & 1 & 3 & 1 \\ 2 & 0 & 1 & 3 & 1 \\ 3 & 3 & 0 & 3 & 1 \\ 4 & 1 & 1 & 0 & 1 \\ 5 & 1 & 1 & 5 & 0 \end{vmatrix}.$$

Матриця A^3 відрізняється від матриці A^4 елементами $a_{31}^3, a_{32}^3, a_{35}^3$. Відповідні елементи матриці P^4 будуть дорівнювати елементам четвертого рядка матриці P^3, які розташовані в тому ж стовпці, тобто

$$p_{31}^4 = p_{41}^3 = 4; p_{32}^4 = p_{42}^3 = 1; p_{35}^4 = p_{45}^3 = 1$$

$$P^4 = \begin{vmatrix} 0 & 1 & 1 & 3 & 1 \\ 2 & 0 & 1 & 3 & 1 \\ 4 & 1 & 0 & 3 & 1 \\ 4 & 1 & 1 & 0 & 1 \\ 5 & 1 & 1 & 5 & 0 \end{vmatrix}.$$

Крок 5. У матриці A^4 виділяємо п’ятий рядок і п’ятий стовпець (рис.2.13).

$$A^4 = \left|\begin{array}{cccc|c} 0 & 1 & 2 & 4 & 3 \\ 2 & 0 & 4 & 6 & 5 \\ 3 & 4 & 0 & 2 & 6 \\ 1 & 2 & 3 & 0 & 4 \\ \hline 5 & 6 & 7 & 4 & 0 \end{array}\right|$$

Рис. 2.13

Виявляється, що усі елементи матриці не зміняться, бо вони вже є мінімальними, тому матриця A^5 буде рівна матриці A^4. Отже, матриця P^5 також не зміниться і буде рівна матриці P^4.

Таким чином, отримали такий результат найкоротших шляхів та відстаней:

- $(1,2): 1 \to 2, d[2] = 1; (1,3): 1 \to 3, d[3] = 2; (1,4): 1 \to 3 \to 4, d[4] = 4; (1,5): 1 \to 5: d[5] = 3;$
- $(2,1): 2 \to 1, d[1] = 2; (2,3): 2 \to 1 \to 3, d[3] = 4; (2,4): 2 \to 1 \to 3 \to 4, d[4] = 6; (2,5): 2 \to 1 \to 5, d[5] = 5;$
- $(3,1): 3 \to 4 \to 1, d[1] = 3; (3,2): 3 \to 4 \to 1 \to 2, d[2] = 4; (3,4): 3 \to 4, d[4] = 2; (3,5): 3 \to 4 \to 1 \to 5, d[5] = 6;$
- $(4,1): 4 \to 1, d[1] = 1; (4,2): 4 \to 1 \to 2, d[2] = 2; (4,3): 4 \to 1 \to 3, d[3] = 3; (4,5): 4 \to 1 \to 5, d[5] = 4;$
- $(5,1): 5 \to 1, d[1] = 5; (5,2): 5 \to 1 \to 2, d[2] = 6; (5,3): 5 \to 1 \to 3, d[3] = 7; (5,4): 5 \to 4, d[4] = 4.$

1.3 Використання алгоритму Беллмана - Форда

Постановка задачі

Нехай необхідно знайти найкоротший шлях від однієї вершини до всіх інших та їх довжини у орієнтованому навантаженому графі, який не містить негативних циклів. Граф складається з п’яти вершин і також містить від’ємні ваги ребер (Рис.2.14). А – початкова вершина.

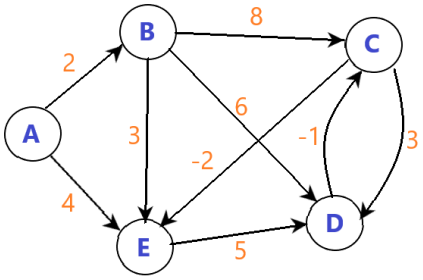

Рис. 2.14

Розв'язання задачі

Для розв'язання задачі алгоритмом Беллмана-Форда створимо таблицю, у якій будемо оновлювати дані про відстані(d) до вершин та про попередників(π) вершин (Таблиця 2.1).

Таблиця 2.1

	A	**B**	**C**	**D**	**E**
$\boldsymbol{d}$	0	∞	∞	∞	∞
$\boldsymbol{\pi}$	-	-	-	-	-

Алгоритм Беллмана-Форда проходить ітеративний шлях через всі ребра. Кількість ітерацій не повинна перевищувати кількість вершин у графі. Для початку усім відстаням до вершин присвоєно ∞, а початковій вершині – 0 (Таблиця 2.1).

Крок 1. Виконуємо перерахунок відстаней від вершини A до всіх інших вершин:

1) $d[B] > d[A] + w(A,B) \Rightarrow \infty > 0 + 2$, тобто $d[B] = d[A] + w(A,B) = 2$;
2) $d[C] > d[A] + w(A,C) \Rightarrow \infty \leq 0 + \infty$, тобто $d[C] = \infty$;
3) $d[D] > d[A] + w(A,D) \Rightarrow \infty \leq 0 + \infty$, тобто $d[D] = \infty$;
4) $d[E] > d[A] + w(A,E) \Rightarrow \infty > 0 + 4$, тобто $d[E] = d[A] + w(A,E) = 4$;

Оновлюємо дані в таблиці 2.2:

Таблиця 2.2

	A	**B**	**C**	**D**	**E**
$\boldsymbol{d}$	0	2	∞	∞	4
$\boldsymbol{\pi}$	-	**A**	-	-	**A**

Отже, на першому кроці було перераховано всі відстані і оновлено відстані від A до вершин B та E, а також зазначено, що вершина A є попередньою вершиною для них. Помічаємо, що відстань до вершини B є найменшою з усіх , тому позначаємо її як “пройдену”(виділено клітинку таблиці червоним) і відносно неї будемо перераховувати відстані на наступному кроці. Разом з цією вершиною переглянутим вважається і ребро (A, B).

Крок 2. Виконуємо знову перерахунок відстаней відносно вершини B до всіх інших:

1) $d[A] > d[B] + w(B, A) \Rightarrow 0 \leq 0 + \infty$, тобто $d[A] = 0$ – це підтверджує, що до вершини A з вершини B немає маршруту;
2) $d[C] > d[B] + w(B, C) \Rightarrow \infty > 2 + 8$, тобто $d[C] = d[B] + w(B, C) = 10$;
3) $d[D] > d[B] + w(B, D) \Rightarrow \infty > 2 + 6$, тобто $d[D] = d[B] + w(B, D) = 8$;
4) $d[E] > d[A] + w(A, E) \Rightarrow 4 \leq 2 + 3$, тобто $d[E] = 4$;

Оновлюємо дані в таблиці 2.3:

Таблиця 2.3

	A	**B**	**C**	**D**	**E**
$\boldsymbol{d}$	0	2	10	8	4
$\boldsymbol{\pi}$	-	**A**	**B**	**B**	**A**

На даному кроці оновилось ще кілька відстаней , а саме: $B \rightarrow C$ та $B \rightarrow D$. Відстань до E залишилась незмінною, так як вона є найкоротшою. Вершина E позначена як “пройдена” і ребро (B, E) є переглянутим, бо її відстань є найменшою з усіх перерахованих.

Крок 3. Виконуємо знову перерахунок відстаней відносно вершини E до всіх інших:

1) $d[A] > d[E] + w(E, A) \Rightarrow 0 \leq 4 + \infty$, тобто $d[A] = 0$ – це підтверджує, що до вершини A з вершини E немає маршруту;
2) $d[B] > d[E] + w(E, B) \Rightarrow 2 \leq 4 + \infty$, тобто $d[B] = 2$;
3) $d[\mathrm{C}] > d[E] + w(E, C) \Rightarrow 10 \leq 4 + \infty$, тобто $d[C] = 10$;

4) $d[D] > d[E] + w(E, D) \Rightarrow 8 \leq 4 + 5$, тобто $d[D] = 8$;

Таблиця 2.4

	A	B	C	D	E
$\boldsymbol{d}$	0	2	10	8	4
$\boldsymbol{\pi}$	-	**A**	**B**	**B**	**A**

Дані в таблиці не оновилися, тому що з вершини E виходить тільки одне ребро (E, D), а всі інші входять в цю вершину. Для вершини D виявилось, що її відстань вже є найкоротшою (Таблиця 2.4). На даному кроці ми можемо позначити як "пройдена" тільки вершину D, так як тільки у неї входить ребро з E. Тоді ребро (E, D) є переглянутим.

Крок 4. Виконуємо знову перерахунок відстаней відносно вершини D до всіх інших:

1) $d[A] > d[D] + w(D, A) \Rightarrow 0 \leq 8 + \infty$, тобто $d[A] = 0$ – це підтверджує, що до вершини A з вершини D немає маршруту;
2) $d[B] > d[D] + w(D, B) \Rightarrow 2 \leq 8 + \infty$, тобто $d[B] = 2$ – тобто відстань вершини B є найкоротшою
3) $d[C] > d[D] + w(D, C) \Rightarrow 10 > 8 - 1$, тобто $d[C] = d[D] + w(D, C) = 7$;
4) $d[E] > d[D] + w(D, E) \Rightarrow 4 \leq 8 + \infty$, тобто $d[E] = 4$ – тобто відстань вершини E є найкоротшою.

Оновлюємо дані в таблиці 2.5:

Таблиця 2.5

	A	B	C	D	E
$\boldsymbol{d}$	0	2	7	8	4
$\boldsymbol{\pi}$	-	**A**	**D**	**B**	**A**

На даному кроці оновилась відстань $D \rightarrow C$. Відстані до інших вершин залишились незмінними. Ми можемо позначити як "пройдена" тільки вершину C, так як тільки у неї входить ребро з D. Тоді ребро (D, C) є переглянутим.

Крок 5. Виконуємо перерахунок відстаней відносно вершини C до всіх інших:

1) $d[A] > d[C] + w(C, A) \Rightarrow 0 \leq 7 + \infty$, тобто $d[A] = 0$ – це підтверджує, що до вершини A з вершини C немає маршруту;
2) $d[B] > d[C] + w(C, B) \Rightarrow 2 \leq 7 + \infty$, тобто $d[B] = 2$ – тобто відстань вершини B є найкоротшою;
3) $d[D] > d[C] + w(C, D) \Rightarrow 8 \leq 7 + 3$, тобто $d[D] = 8$ – тобто відстань вершини D є найкоротшою;
4) $d[E] > d[C] + w(C, E) \Rightarrow 4 \leq 7 - 2$, тобто $d[E] = 4$ – тобто відстань вершини E є найкоротшою;

Усі вершини “пройдені” і нам не довелося знімати цю позначку, так як на цьому кроці дані не змінилися, тобто відбулась лише одна ітерація, на якій вдалося отримати наступний результат:

- $A \to B$: $d[B] = 2$;
- $A \to E$: $d[E] = 4$;
- $A \to B \to D$: $d[D] = 8$;
- $A \to B \to D \to C$: $d[C] = 7$.

1.4 Використання алгоритму Джонсона

Постановка задачі

Нехай необхідно знайти найкоротший шлях між усіма парами вершин та їх довжини у орієнтованому навантаженому графі, який не містить негативних циклів. Граф складається з чотирьох вершин і також містить від’ємні ваги ребер (Рис. 2.15).

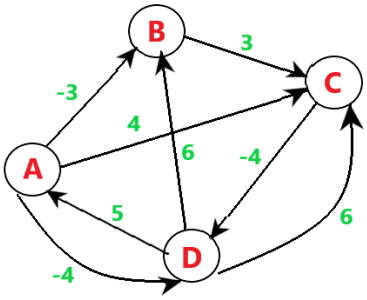

Рис. 2.15

Розв’язання задачі

Даний граф містить від’ємні ваги ребер, тому для початку потрібно знайти нові ваги, які будуть додатні, щоб використати алгоритм Дейкстри для знаходження найкоротших шляхів. Для цього додамо до даного графа нову вершину S і проведемо від неї ребра до всіх інших вершин з вагою рівною нулю. Нова вершина буде початковою з відстанню $d[S] = 0$, а для усіх інших вершин призначимо відстань рівну ∞ (Рис. 2.16).

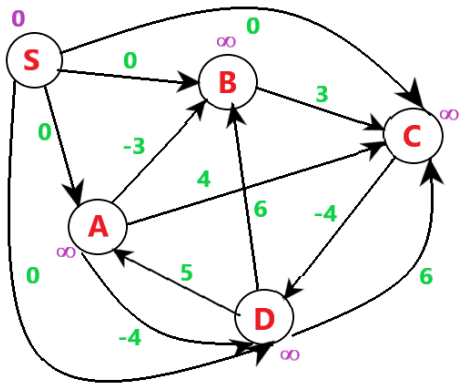

Рис. 2.16

Скористаємося алгоритмом Беллмана-Форда, щоб знайти нові ваги ребер для графа. Створимо таблицю, у якій будемо оновлювати дані про відстані(d) до вершин та про попередників(π) вершин (Таблиця 2.6).

Таблиця 2.6

	S	**A**	**B**	**C**	**D**
$\boldsymbol{d}$	0	∞	∞	∞	∞
$\boldsymbol{\pi}$	-	-	-	-	-

Ітерація 1. Виконуємо перерахунок відстаней від вершини S до всіх інших вершин:

1) $d[A] > d[S] + w(S, A) \Rightarrow \infty > 0 + 0$, тобто $d[A] = d[S] + w(S, A) = 0$;
2) $d[B] > d[S] + w(S, B) \Rightarrow \infty > 0 + 0$, тобто $d[B] = d[S] + w(S, B) = 0$;
3) $d[C] > d[S] + w(S, C) \Rightarrow \infty > 0 + 0$, тобто $d[C] = d[S] + w(S, C) = 0$;
4) $d[D] > d[S] + w(S, D) \Rightarrow \infty > 0 + 0$, тобто $d[D] = d[S] + w(S, D) = 0$.

Оновлюємо дані у таблиці 2.7:

Таблиця 2.7

	S	A	B	C	D
d	0	0	0	0	0
π	-	S	S	S	S

Ітерація 2. На даному етапі всі відстані до вершин однакові, тому можна обрати будь-яку. Обираємо вершину A і позначаємо її як “пройдена” (Таблиця 2.7).

Виконуємо перерахунок відстаней від вершини A до всіх інших вершин:

1) $d[B] > d[A] + w(A,B) \Rightarrow 0 > 0 - 3$, тобто $d[B] = d[A] + w(A,B) = -3$;
2) $d[C] > d[A] + w(A,C) \Rightarrow 0 \leq 0 + 4$, тобто $d[C] = 0$;
3) $d[D] > d[A] + w(A,D) \Rightarrow 0 > 0 - 4$, тобто $d[D] = d[A] + w(A,D) = -4$;
4) $d[S] > d[A] + w(A,S) \Rightarrow 0 \leq 0 + \infty$, тобто $d[S] = 0$.

Оновлюємо дані у таблиці 2.8:

Таблиця 2.8

	S	A	B	C	D
d	0	0	-3	0	-4
π	-	S	A	S	A

Позначаємо вершину D “пройденою”, так як на наступній ітерації будемо здійснювати перерахунок відносно неї, бо її відстань є найменшою.

Ітерація 3. Виконуємо перерахунок відстаней від вершини D до всіх інших вершин:

1) $d[A] > d[D] + w(D,A) \Rightarrow 0 \leq -4 + 5$, тобто $d[A] = 0$;
2) $d[B] > d[D] + w(D,B) \Rightarrow -3 \leq -4 + 6$, тобто $d[B] = -3$;
3) $d[C] > d[D] + w(D,C) \Rightarrow 0 \leq -4 + 6$, тобто $d[C] = 0$;
4) $d[S] > d[D] + w(D,S) \Rightarrow 0 \leq 0 + \infty$, тобто $d[S] = 0$.

Оновлюємо дані у таблиці 2.9:

Таблиця 2.9

	S	A	B	C	D
d	0	0	-3	0	-4

$\boldsymbol{\pi}$	-	S	A	S	A

Позначаємо вершину B “пройденою”, так як на наступній ітерації будемо здійснювати перерахунок відносно неї.

Ітерація 4. Виконуємо перерахунок відстаней від вершини B до всіх інших вершин:

1) $d[A] > d[B] + w(B,A) \Rightarrow 0 \leq -3 + \infty$, тобто $d[A] = 0$;
2) $d[C] > d[B] + w(B,C) \Rightarrow 0 \leq -3 + 3$, тобто $d[C] = 0$;
3) $d[D] > d[B] + w(B,D) \Rightarrow -4 \leq -3 + \infty$, тобто $d[D] = -4$;
4) $d[S] > d[B] + w(B,S) \Rightarrow 0 \leq 0 + \infty$, тобто $d[S] = 0$.

Оновлюємо дані у таблиці 2.10:

Таблиця 2.10

	S	**A**	**B**	**C**	**D**
$\boldsymbol{d}$	0	0	-3	0	-4
$\boldsymbol{\pi}$	-	S	A	B	S

Позначаємо вершину C “пройденою”, так як на наступній ітерації будемо здійснювати перерахунок відносно неї.

Ітерація 5. Виконуємо перерахунок відстаней від вершини C до всіх інших вершин:

1) $d[A] > d[C] + w(C,A) \Rightarrow 0 \leq 0 + \infty$, тобто $d[A] = 0$;
2) $d[B] > d[C] + w(C,B) \Rightarrow -3 \leq 0 + \infty$, тобто $d[B] = -3$;
3) $d[D] > d[C] + w(C,B) \Rightarrow -4 \leq 0 - 4$, тобто $d[D] = -4$;
4) $d[S] > d[C] + w(C,S) \Rightarrow 0 \leq 0 + \infty$, тобто $d[S] = 0$.

Оновлюємо дані у таблиці 2.11:

Таблиця 2.11

	S	**A**	**B**	**C**	**D**
$\boldsymbol{d}$	0	0	-3	0	-4
$\boldsymbol{\pi}$	-	S	A	B	C

На даній ітерації всі вершини “пройдені” і жодна відстань не змінилась, тому алгоритм Беллмана-Форда завершено. Таким чином маємо наступні довжини найкоротших шляхів: $h(A) = 0, h(B) = -3, h(C) = 0, h(D) = -4$.

Знайдемо нові додатні ваги для ребер даного графа:

1) $w'(A,B) = w(A,B) + h(A) - h(B) = -3 + 0 - (-3) = 0$;
2) $w'(A,C) = w(A,C) + h(A) - h(C) = 4 + 0 - 0 = 4$;
3) $w'(A,D) = w(A,D) + h(A) - h(D) = -4 + 0 - (-4) = 0$;
4) $w'(B,C) = w(B,C) + h(B) - h(C) = 3 - 3 - 0 = 0$;
5) $w'(C,D) = w(C,D) + h(C) - h(D) = -4 + 0 - (-4) = 0$;
6) $w'(D,A) = w(D,A) + h(D) - h(A) = 5 - 4 - 0 = 1$;
7) $w'(D,B) = w(D,B) + h(D) - h(B) = 6 - 4 - (-3) = 5$;
8) $w'(D,C) = w(D,C) + h(D) - h(C) = 6 - 4 - 0 = 2$;

Отримаємо поновлений граф (Рис. 2.17):

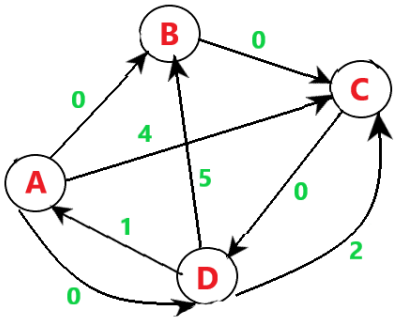

Рис. 2.17

Отже, можемо скористатися алгоритмом Дейкстри для знаходження найкоротших шляхів. Цей алгоритм потрібно застосувати чотири рази, щоб знайти найкоротші відстані між усіма вершинами.

Застосування 1. Нехай вершина A є початковою, тому $d'[A] = 0$, а для всіх інших відстань рівна ∞. Перерахуємо всі відстані від вершини A до всіх інших вершин:

Крок 1.

1) $d'[A] + w'(A,B) < d'[B] \Rightarrow 0 + 0 < \infty$, тобто $d'[B] = d'[A] + w'(A,B) = 0$;
2) $d'[A] + w'(A,C) < d'[C] \Rightarrow 0 + 4 < \infty$, тобто $d'[C] = d'[A] + w'(A,C) = 4$;
3) $d'[A] + w'(A,D) < d'[D] \Rightarrow 0 + 0 < \infty$, тобто $d'[D] = d'[A] + w'(A,D) = 0$;

Крок 2. Переходимо до вершини B:

$d'[B] + w'(B,C) < d'[C] \Rightarrow 0 + 0 < 4$, тобто $d'[C] = d'[B] + w'(B,C) = 0$;

Крок 3. Переходимо у вершину C:

$d'[C] + w'(C,D) < d'[D] \Rightarrow 0 + 0 = 0$, тобто $d'[D] = 0$;

Переходимо у вершину D і виходить, що всі вершини вже викреслені, тому для вершини A маємо: $A \to B: d'[B] = 0$; $A \to C: d'[C] = 0$; $A \to D: d'[D] = 0$.

Застосування 2. Нехай вершина B є початковою, тому $d'[B] = 0$, а для всіх інших відстань рівна ∞. Перерахуємо всі відстані від вершини B до всіх інших вершин:

Крок 1.

$d'[B] + w'(B,C) < d'[C] \Rightarrow 0 + 0 < \infty$, тобто $d'[C] = d'[B] + w'(B,C) = 0$;

Крок 2. Переходимо до вершини C:

$d'[C] + w'(C,D) < d'[D] \Rightarrow 0 + 0 < \infty$, тобто $d'[D] = d'[C] + w'(C,D) = 0$;

Крок 3. Переходимо у вершину D:

$d'[D] + w'(D,A) < d'[A] \Rightarrow 0 + 1 < \infty$, тобто $d'[A] = d'[D] + w'(D,A) = 1$;

Переходимо у вершину A і виходить, що всі вершини вже викреслені, тому для вершини B маємо: $B \to A: d'[A] = 1$; $B \to C: d'[C] = 0$; $B \to D: d'[D] = 0$.

Застосування 3. Нехай вершина C є початковою, тому $d'[C] = 0$, а для всіх інших відстань рівна ∞. Перерахуємо всі відстані від вершини C до всіх інших вершин:

Крок 1.

$d'[C] + w'(C,D) < d'[D] \Rightarrow 0 + 0 < \infty$, тобто $d'[D] = d'[C] + w'(C,D) = 0$;

Крок 2. Переходимо до вершини D:

1) $d'[D] + w'(D,A) < d'[A] \Rightarrow 0 + 1 < \infty$, тобто $d'[A] = d'[D] + w'(D,A) = 1$;
2) $d'[D] + w'(D,B) < d'[B] \Rightarrow 0 + 5 < \infty$, тобто $d'[B] = d'[D] + w'(D,B) = 5$;

Крок 3. Переходимо у вершину A:

$d'[A] + w'(A,B) < d'[B] \Rightarrow 1 + 0 < 5$, тобто $d'[B] = d'[A] + w'(A,B) = 1$;

Переходимо у вершину B і виходить, що всі вершини вже викреслені, тому для вершини C маємо: $C \to A: d'[A] = 1$; $C \to B: d'[B] = 1$; $C \to D: d'[D] = 0$.

Застосування 4. Нехай вершина D є початковою, тому $d'[D] = 0$, а для всіх інших відстань рівна ∞. Перерахуємо всі відстані від вершини D до всіх інших вершин:

Крок 1.

1) $d'[D] + w'(D,A) < d'[A] \Rightarrow 0 + 1 < \infty$, тобто $d'[A] = d'[D] + w'(D,A) = 1$;
2) $d'[D] + w'(D,B) < d'[B] \Rightarrow 0 + 5 < \infty$, тобто $d'[B] = d'[D] + w'(D,B) = 5$;
3) $d'[D] + w'(D,C) < d'[C] \Rightarrow 0 + 2 < \infty$, тобто $d'[C] = d'[D] + w'(D,C) = 2$;

Крок 2. Переходимо до вершини A:

1) $d'[A] + w'(A,B) < d'[B] \Rightarrow 1 + 0 < 5$, тобто $d'[B] = d'[A] + w'(A,B) = 1$;
2) $d'[A] + w'(A,C) < d'[C] \Rightarrow 1 + 4 > 2$, тобто $d'[C] = 2$;

Крок 3. Переходимо у вершину B:

$d'[B] + w'(B,C) < d'[C] \Rightarrow 1 + 0 < 2$, тобто $d'[C] = d'[B] + w'(B,C) = 1$;

Переходимо у вершину C і виходить, що всі вершини вже викреслені, тому для вершини D маємо: $D \to A: d'[A] = 1$; $D \to B: d'[B] = 1$; $D \to C: d'[C] = 1$.

Отже, у результаті складемо наступну матрицю H' найкоротших відстаней графа з новими вагами між усіма парами вершин:

$$H' = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Тепер необхідно знайти найкоротші відстані між усіма парами вершин для початкового графа:

1) Від вершини A до всіх інших:

   $d[B] = d'[B] - h(A) + h(B) = 0 - 0 - 3 = -3$

   $d[C] = d'[C] - h(A) + h(C) = 0 - 0 + 0 = 0$

   $d[D] = d'[D] - h(A) + h(D) = 0 - 0 - 4 = -4$

2) Від вершини B до всіх інших:

   $d[A] = d'[A] - h(B) + h(A) = 1 - (-3) + 0 = 4$

   $d[C] = d'[C] - h(B) + h(C) = 0 - (-3) + 0 = 3$

   $d[D] = d'[D] - h(B) + h(D) = 0 - (-3) - 4 = -1$

3) Від вершини C до всіх інших:

   $d[A] = d'[A] - h(C) + h(A) = 1 - 0 + 0 = 1$

   $d[B] = d'[B] - h(C) + h(B) = 1 - 0 - 3 = -2$

$d[D] = d'[D] - h(C) + h(D) = 0 - 0 - 4 = -4$

4) Від вершини D до всіх інших:

$d[A] = d'[A] - h(D) + h(A) = 1 - (-4) + 0 = 5$

$d[B] = d'[B] - h(D) + h(B) = 1 - (-4) - 3 = 2$

$d[C] = d'[C] - h(D) + h(C) = 1 - (-4) + 0 = 5$

Отримуємо остаточні результати у наступній матриці H:

$$H = \begin{bmatrix} 0 & -3 & 0 & -4 \\ 4 & 0 & 3 & -1 \\ 1 & -2 & 0 & -4 \\ 5 & 2 & 5 & 0 \end{bmatrix}$$

2 Аналіз прогнозу та порівняльна оцінка алгоритмів

Аналіз алгоритмів Дейкстри, Беллмана-Форда, Флойда-Уоршела та Джонсона було проведено щодо їхньої швидкості розв'язання задачі про найкоротший шлях. Було визначено часові показники виконання алгоритмів у комп'ютерній реалізації на мові програмування C++. Ця мова зручна у використанні та дає можливість отримати точні дані для аналізу. Таким чином, для порівняльної оцінки швидкодії алгоритмів та подальшого прогнозованого аналізу було розглянуто дані про кількість вершин графа та час, за який алгоритм зміг розрахувати результат.

Для того, щоб отримати дані для аналізу було реалізовано нескладну розробку кожного з алгоритмі на мові C++. Кожен алгоритм реалізований по-особливому щодо взаємодії з користувачем. У реалізації алгоритму Дейкстри було використано спосіб введення даних безпосередньо в коді у вигляді матриці суміжності, а виведення відображає шляхи з початкової вершини в усі інші та їхні найкоротші відстані. Реалізації алгоритмів Беллмана-Форда та Флойда-Уоршела вимагають ввід даних, а саме матриці суміжності та кількості вершин, користувачем. Вивід результату алгоритму Беллмана-Форда має вигляд рядка, а у алгоритмі Флойда-Уоршела – матриця найкоротших відстаней. У алгоритмі Джонсона ввід даних також виконується користувачем, але необхідно вводити кількості вершин та ребер і лише існуючі ребра графа з їхньою вагою. У результаті цієї реалізації виводиться матриця найкоротших відстаней.

Для кодування кожного з алгоритмів було використано цикл "for" та функцію умови "if", щоб робити прохід по вершинах та вагах графа і відбирати необхідні дані, а також функцію "clock()" для визначення часу виконання реалізації з бібліотеки "ctime".

Порівняльна оцінка досліджуваних алгоритмів щодо швидкості виконання надасть конкретики у тому, який алгоритм ефективніший та має більше переваг для того, щоб користувачі його застосовували для задач про найкоротший шлях, які у теперішній час виникають ще у більшій кількості ситуацій. Для кожного алгоритму було обрано

графи з кількостями вершин 4, 8, 12 та 16 і визначено час їхнього виконання (Таблиця 2.12).

Таблиця 2.12

Кількість вершин графа	Час виконання(у секундах)			
	Дейкстри	Беллмана-Форда	Флойда-Уоршела	Джонсона
4	0,001597	0,001932	0,002178	0,001901
8	0,001992	0,002238	0,002410	0,002071
12	0,002160	0,002406	0,002505	0,002238
16	0,002342	0,002626	0,002687	0,002577

У даній роботі об’єктами дослідження є задачі про найкоротший шлях двох видів, а саме знаходження найкоротших шляхів та відстаней від однієї вершини до всіх інших та між усіма парами вершин. У такому випадку є сенс здійснювати порівняльну оцінку саме між алгоритмами, які розв’язують дані види задач.

Отже, алгоритми Дейкстри та Беллмана-Форда здатні розв’язувати задачу знаходження найкоротших шляхів від однієї вершини до всіх інших, тому слід порівнювати дані між ними двома. Із таблиці 2.12 очевидно, що алгоритм Дейкстри у швидкості обійшов алгоритм Беллмана-Форда, тобто він виявляється більш ефективним, проте це тільки у тому випадку, якщо вагові показники ребер графа виключно додатні. Так як другий алгоритм здатний знаходити результат з від’ємними ваговими показниками, то таке триваліше виконання розрахунків стає зрозумілим і виправданим. Можна зробити висновок, що алгоритм Дейкстри є ефективним для роботи з графами з додатніми вагами, проте алгоритм Беллмана-Форда також є досить ефективним, щоб справлятися з від’ємними показниками.

Наступні алгоритми Флойда-Уоршела та Джонсона здатні розв’язувати задачу знаходження найкоротших шляхів між усіма парами вершин, тому їх варто порівняти між собою. Результати таблиці 2.12 дають зрозуміти, що алгоритм Джонсона ефективніше справляється зі своєю роботою, незважаючи на те, що у ньому використовується ще два попередні алгоритми.

Під терміном аналізу прогнозу мається на увазі трендовий аналіз прогнозу виконання даних алгоритмів. Узагалі, трендовий аналіз використовується для вивчення та аналізу фінансового стану підприємств та організацій, щоб дослідити фінансову стійкість та ефективність використання капіталу підприємства чи організації. Таким чином, можна спрогнозувати подальшу поведінку терміну виконання кожного з алгоритмів за критерієм збільшення кількості вершин у графі. Результати цього прогнозу надають кращих показників для підтвердження стійкості та ефективності алгоритму.

На рисунках 2.18 та 2.19 представлено трендовий прогноз алгоритмів Дейкстри та Беллмана-Форда, Флойда-Уоршела та Джонсона, який здійснено за допомогою табличного редактора Excel. Вісь абсцис графіка це кількість вершин, а вісь ординат – показники часу. Час виконання алгоритмів взято у форматі $t \cdot 10^{-4}$секунд.

Графіки даних були апроксимовані функціями кубічного поліному. На вибір формату лінії тренду впливає коефіцієнт апроксимації R^2, який дає зрозуміти на скільки відсотків лінія тренду описує вхідні дані. Необхідно обирати такий вид лінії тренду, щоб відсоток апроксимації був вищим за 80%. У даному випадку кубічний поліном це найкраще рішення, так як апроксимація становить 100%.

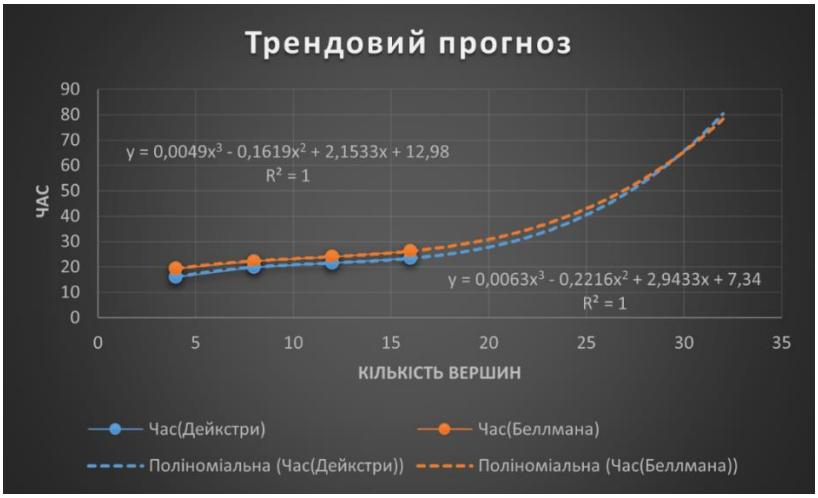

Рис. 2.18

У даному аналізі було використано 16 періодів прогнозу, тобто дані спрогнозувалися до кількості вершин 32. На рисунку 2.18 видно, що прогноз подальших показників часу виконання алгоритмів Дейкстри та Беллмана-Форда дає зрозуміти, що надалі тривалість виконання алгоритмів зростатиме зі зростанням

кількості вершин графа, і при наближенні до кількості вершин 28 алгоритм Беллмана-Форда розпочне наздоганяти алгоритм Дейкстри і надалі може навіть виконуватись швидше за нього. На рисунку 2.19 прогнозується час виконання алгоритмів Флойда-Уоршела та Джонсона. Видно, що після кількості вершин 16 алгоритм Флойда-Уоршела розпочинає наздоганяти алгоритм Джонсона і навіть починає швидше виконуватись.

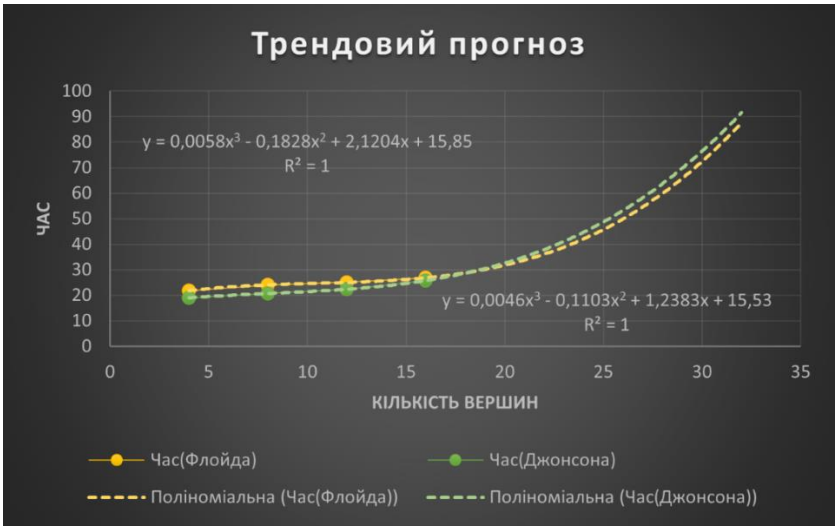

Рис. 2.19

Така поведінка тривалості виконання може пояснюватися тим, що було взято невелику кількість вершин для графів, відповідно для цих даних найкращою лінією тренду була саме поліноміальна, що може змінитися при взятті більшої кількості вершин. Тоді найкращий результат може бути представлений логарифмічною або ж експоненціальною лінією тренду.

Отже, трендовий аналіз прогнозу дав зрозуміти як поводитиме себе тривалість виконання алгоритмів при подальших збільшеннях кількості вершин. Має сенс розробити прогнозування цих алгоритмів, використавши більше даних, у майбутньому продовженні цього дослідження.

ВИСНОВКИ

У роботі були досліджені алгоритми для розв’язання задачі про найкоротші шляхи та було розроблено порівняльну оцінку та трендовий прогноз щодо швидкості кожного. Алгоритмами, які досліджувалися були: Дейкстри, Беллмана-Форда, Флойда-Уоршела та Джонсона.

У теоретичній частині було розглянуто усі основні теоретичні відомості про графи та задачу про найкоротший шлях, які необхідні для подальших досліджень. Також зроблено огляд алгоритмів з теоретичної сторони.

У дослідженні з використання кожного з алгоритмів було розроблено покрокові розрахунки та рисунки або таблиці для конкретизації та демонстрації того, як саме алгоритм знаходить найкоротші шляхи та відстані на графі. Було подано кінцеві результати, тобто алгоритми чітко і точно виконують розрахунки та дійсно здатні розв’язати дану задачу, яка у поточному дослідженні була у двох виглядах, а саме: пошук найкоротших шляхів від однієї вершини до всіх інших та між усіма парами вершин.

У дослідженні аналізу прогнозу та порівняльної оцінки було розроблено програмну реалізацію алгоритмів, яка знаходить час виконання. Після розрахованих даних для аналізу було побудовано таблицю з часовими показниками та кількостями вершин 4, 8, 12 та 16. Порівняльна оцінка показала, що алгоритм Дейкстри ефективніший за Беллмана-Форда, проте останній здатен шукати найкоротші відстані та шляхи з від’ємними вхідними даними. Щодо двох інших, алгоритм Джонсона виявився ефективнішим за Флойда-Уоршела, хоча у ньому використовуються два попередні алгоритми.

Трендовий прогноз був здійснений за допомогою табличного редактора Excel. Лінія тренду була у поліноміальному форматі третього степеня, тобто графіки даних були апроксимовані функціями кубічного поліному. На вибір такої лінії тренду вплинув коєфіцієнт апроксимації R^2, який становив 100%. Із результатів прогнозу виявилося, що алгоритми стійкі у тривалості виконання, проте при збільшенні

кількості вершин ефективність кожного змінюється, тому є сенс продовжити дослідження цього прогнозу на більшій кількості даних.

Отже, проведена порівняльна оцінка роботи алгоритмів, їхньої ефективності та розроблено прогноз часу роботи алгоритмів з використанням лінії тренду поліноміального виду. Це дає змогу рекомендувати вибір конкретного алгоритму серед розглянутих для розв’язання задачі про найкоротший шлях .

ЛІТЕРАТУРА

1. Dijkstra E. A note on two problems in connexion with graphs. // In.: Numerische Mathematik. - 1959. - V. 1.— pp. 269-271.
2. Hamdy Taha. Operations Research: An Introduction 10th Edition. — Pearson; 2016. — 848 с. Електронний ресурс: http://zalamsyah.staff.unja.ac.id/wp-content/uploads/sites/286/2019/11/9-Operations-Research-An-Introduction-10th-Ed.-Hamdy-A-Taha.pdf
3. Labeling Algorithm for Shortest Paths on Road Networks. / [Abraham I., Delling D., Goldberg A., Werneck R.]. - Philadelphia. - Symposium on Experimental Algorithms, 2011. — pp. 230-241. Електронний ресурс: https://www.microsoft.com/en-us/research/wp-content/uploads/2010/12/HL-TR.pdf
4. Williams R. Faster All-Pairs Shortest Paths Via Circuit Complexity// Proceedings of the 2014 ACM Symposium on Theory of Computing, 2014. – p. 664-673. Електронний ресурс: https://dspace.mit.edu/bitstream/handle/1721.1/136331/15m1024524.pdf?sequence=2&isAllowed=y
5. Бартіш М.Я., Дудзяний І.М. Дослідження операцій. Частина 2. Алгоритми оптимізації на графах. Навчальний посібник для студентів. – Л.: Видавничий центр ЛНУ ім. Івана Франка, 2007. – 118 с. Електронний ресурс: https://ami.lnu.edu.ua/wp-content/uploads/2014/02/DO_CH2_Alhorytmy-optymizatsii-na-hrafakh.pdf
6. Вступ до алгоритмів, Кормен, Томас; Лейзерсон, Чарльз; Рівест, Рональд; Стайн, Кліфорд. К.І.С. 2019 - 1296 с.. ISBN 978-617- 684-239-2.
7. Вступ до програмування мовою С++. Структури даних: навч. посіб. Р. А. Веклич, Т. О. Карнаух, А. Б. Ставровський – К. : ВПЦ "Київський університет", 2018. – 99 с. Електронний ресурс: http://csc.knu.ua/media/filer_public/69/0f/690f16f1-513d-4538-b438-82f1d15cafce/kkpsverstka02.pdf

8. Дослідження операцій [Текст] : [навчальний посібник] / Меньшикова О.В., Чмир О.Ю., Карабин О.О. –Львів : ЛДУ БЖД, 2019. –196с. Електронний ресурс: https://books.ldubgd.edu.ua/index.php/ed/catalog/download/124/87/384-1?inline=1
9. Катренко А. В. Дослідження операцій: підручник / А. В. Катренко. – Львів: Магнолія, 2014. – 352с.
10. Мелешко Є.В., Якименко М.С., Поліщук Л.І. Алгоритми та структури даних: Навчальний посібник для студентів технічних спеціальностей денної та заочної форми навчання. – Кропивн.: Видавець – Лисенко В.Ф., 2019. – 156 с. Електронний ресурс: http://dspace.kntu.kr.ua/jspui/bitstream/123456789/8944/1/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%B8%20%D1%82%D0%B0%20%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B8%20%D0%B4%D0%B0%D0%BD%D0%B8%D1%85.pdf
11. Попов Ю.Д., Тюптя В.І., Шевченко В.І. Методи оптимізації. Навчальний посібник для студентів спеціальностей «Прикладна математика», «Інформатика», «Соціальна інформатика». − К.: Абрис, 1999. − 217 с. Електронний ресурс: http://www.cyb.univ.kiev.ua/library/books/popov-30.pdf
12. Тарануха В.Ю. Задачі на графах для курсу Алгориміка: Навчальний посібник для студентів факультету комп'ютерних наук та кібернетики / В. Ю.Тарануха. –Київ: електронна публікація на сайті факультету, 2021. – 69 с. Електронний ресурс: http://csc.knu.ua/media/filer_public/85/05/850574f0-fe7a-4eb4-9a3f-1f99e18dd8bf/algorithmicsgraph.pdf
13. Теорія графів у задачах, Карнаух Т.О., Ставровський А.Б.–Київ:-90с. Електронний ресурс: http://www.cyb.univ.kiev.ua/library/books/karnaukh-23.pdf

ДОДАТОК А. Програмний код

Реалізація алгоритму Дейкстри:

```cpp

#include <iomanip>
#include <climits>
#include <iostream>
#include  <ctime>
using namespace std;

const unsigned START = 0;
const unsigned VERTICES = 6;      //кількість вершин
const unsigned AD_MATRIX[VERTICES][VERTICES] = {
    {0, 750, 220, 0, 0, 0},
    {750, 0, 300, 240, 0, 500},
    {220, 300, 0, 540, 400, 0},
    {0, 240, 540, 0, 380, 210},
    {0, 0, 400, 380, 0, 540},
    {0, 500, 0, 210, 540, 0}};          //матриця суміжності

void Algoritm_Deikstra(const unsigned AD_MATRIX[VERTICES][VERTICES], const unsigned &START){
    bool visited[VERTICES];
    unsigned distances[VERTICES];
    unsigned minimalWeight, minimalIndex;
    for (unsigned i = 0; i < VERTICES; ++i){
        visited[i] = false;
        distances[i] = INT_MAX;
    }
    distances[START] = 0;
    do{
        minimalIndex = INT_MAX;
        minimalWeight = INT_MAX;

        for (unsigned i = 0; i < VERTICES; ++i){
            if (!visited[i] && distances[i] < minimalWeight){
                minimalIndex = i;
                minimalWeight = distances[i];
            }
        }
```

```cpp
        if (minimalIndex != INT_MAX){
            for (unsigned i = 0; i < VERTICES; ++i){
                if (AD_MATRIX[minimalIndex][i]){
                    unsigned temp = minimalWeight + AD_MATRIX[minimalIndex][i];
                    if (temp < distances[i])
                        distances[i] = temp;
                }
            }
            visited[minimalIndex] = true;
        }
    }
    while (minimalIndex < INT_MAX);
    for (unsigned i = 0; i < VERTICES; ++i){
        if (distances[i] != INT_MAX){
            cout << "Вага: " << START << " ~> " << i << " = " << setw(6) << left << distances[i] << "\t";
            unsigned end = i;
            unsigned weight = distances[end];
            string way = to_string(end) + " >~ ";
            while (end != START){
                for (unsigned j = 0; j < VERTICES; ++j){
                    if (AD_MATRIX[j][end]){
                        int temp = weight - AD_MATRIX[j][end];
                        if (temp == distances[j]) {
                            end = j;
                            weight = temp;
                            way += to_string(j) + " >~ ";
                        }
                    }
                }
            }
            cout << "Шлях: ";
            for (int j = way.length() - 5; j >= 0; --j)
                cout << way[j];
            cout << endl;
```

```
        }
        else
            cout << "Вага: " << START << " ~ " << i << " = " << "маршрут недоступний" << endl;
    }
}
int main() {
    int a = clock();
    Algoritm_Deikstra(AD_MATRIX, START);
    cout <<"Час виконання "<< ((float)a/CLOCKS_PER_SEC) <<" "<<"секунд"<< endl;
}
```

Реалізація алгоритму Беллмана-Форда:

```
#include <iostream>
#define inf 100500
#include  <ctime>
using namespace std;
struct Edges {
    int u, v, w;
    };
const int max_V = 1000;
const int max_E = max_V*(max_V-1)/2;
int i, j, n, e, start;
Edges edge[max_E];
int d[max_V];
void Bellman_Ford(int n, int s) {
    int i, j;
    for(i=0; i<n; i++)
        d[i] = inf;
    d[s] = 0;
    for(i=0; i<n-1; i++)
       for(j=0; j<e; j++)
          if(d[edge[j].v] + edge[j].w < d[edge[j].u])
             d[edge[j].u] = d[edge[j].v] + edge[j].w;
    for(i=0; i<n; i++)
        if (d[i]==inf)
          cout << " " << "-1";
        else
          cout << " " << d[i];
        cout << endl;

}
int main() {
    int w;
    cout << "Введіть кількість вершин: "; cin>>n;
    e = 0;
    for(i=0; i<n; i++)
       for(j=0; j<n; j++) {
           cin >> w;
           if(w!=0) {
```

```
                edge[e].v = i;
                edge[e].u = j;
                edge[e].w = w;
                e++;
            }
        }
    int a = clock();
    Bellman_Ford(n, 0);
    cout <<"Час виконання "<< ((float)a/CLOCKS_PER_SEC) <<" "<<"секунд"<< endl;
```

Реалізація алгоритму Флойда-Уоршела:

```
#include <cstdlib>
#include <iostream>
#define B 1000
#include  <ctime>

using namespace std;

int main() {
    int n, i, j, k;
    cout<<"Введіть кількість вершин: ";
    cin >> n;
    int a[n][n],c[n][n];
    for(i=0; i<n; i++) {
        for(j=0; j<n; j++) {

            cin >> a[i][j];
            if (a[i][j]==0) c[i][j]=B;
            else c[i][j]=a[i][j];
        }
    }
    int s = clock();
    for(k=0; k<n; k++) {
        for(i=0; i<n; i++) {
            for(j=0; j<n; j++) {
                if(c[i][j]>c[i][k]+c[k][j]) c[i][j]=c[i][k]+c[k][j];
                }
            }
        }
    cout << "\n Матриця суміжності:\n";
    cout << endl;
    for(i=0; i<n; i++) {
        for(j=0; j<n; j++) {
            if(i == j) c[i][j] = 0;
            cout <<" "<< c[i][j];
        }
        cout<<endl;
    }
    cout <<"Час виконання "<< ((float)s/CLOCKS_PER_SEC) <<" "<<"секунд"<< endl;
    return 0;
}
```

Реалізація алгоритму Джонсона:

```
#include<iostream>
#include<ctime>
#define INF 9999
using namespace std;
int min(int a, int b);
int cost[10][10], adj[10][10];
inline int min(int a, int b){
   return (a<b)?a:b;
}
int main() {
   int vert, edge, i, j, k, c;
   cout << "Введіть кількість вершин: ";
   cin >> vert;
   cout << "Введіть кількість ребер: ";
   cin >> edge;
   cout << "Введіть всі ребра:\n";
   int t = clock();
   for (k = 1; k <= edge; k++) { //беремо вхідні дані та зберігаємо у матриці cost та adj
      cin >> i >> j >> c;
      adj[i][j] = cost[i][j] = c;
   }
   for (i = 1; i <= vert; i++)
      for (j = 1; j <= vert; j++) {
         if (adj[i][j] == 0 && i != j)
            adj[i][j] = INF;
      }
```

```
   for (k = 1; k <= vert; k++)
      for (i = 1; i <= vert; i++)
         for (j = 1; j <= vert; j++)
            adj[i][j] = min(adj[i][j], adj[i][k] + adj[k][j]); //знайти мінімалний шлях від i до j через k
   cout << "Result adj matrix\n";
   for (i = 1; i <= vert; i++) {
      for (j = 1; j <= vert; j++) {
         if (adj[i][j] != INF)
            cout << adj[i][j] << " ";
      }
      cout << "\n";
   }
   cout <<"Час виконання "<< ((float)t/CLOCKS_PER_SEC) <<" "<<"секунд"<< endl; }
```