МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

дискретного аналізу та інтелектуальних систем
(повна назва кафедри)

Дипломна робота

ВЕБ-САЙТ ДЛЯ РОБОТИ З ГРАФОВИМИ МОДЕЛЯМИ

Виконав: студент IV курсу, групи ПМі-43с
спеціальності
122 «Комп'ютерні науки»
(шифр і назва спеціальності)

(підпис) Заречанський О. Р.
(прізвище та ініціали)

Керівник (підпис) Щербина Ю. М.
(прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

2023

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет Прикладної математики та інформатики
Кафедра Дискретного аналізу та інтелектуальних систем
Спеціальність 122 «Комп'ютерні науки»
(шифр і назва)

«ЗАТВЕРДЖУЮ»
Завідувач кафедри _____

"31 "серпня 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНУ У РОБОТУ СТУДЕНТУ

Заречанському Олексію Руслановичу
(прізвище, ім'я, по батькові)

1 .Тема роботи: Веб-сайт для роботи з графовими моделями

керівник роботи: Щербина Юрій Миколайович, професор кафедри дискретного аналізу та інтелектуальних систем,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені Вченою радою факультету від **" 13" вересня 2022 року № 15**

2. Строк подання студентом роботи **13.06.2023р.**

3. Вихідні дані до роботи Документація по мові програмування C# та .NET Core 6, документація бібліотеки D3, документація Azure, IDE Visual Studio 2022, .NET Core 6, бібліотека D3, веб-сервіси Azure, GitHub, інтернет ресурси, джерела про теорію графів.

4. Зміст дипломної роботи (перелік питань, які потрібно розробити) Ознайомитись з теорією графів, ознайомитись з можливостями бібліотеки D3, обрати мову програмування та спосіб імплементації графа, створити веб-сайт, створити граф на сайті, додати можливість модифікації графа, додати різні способи представлення, додати алгоритми найкоротшого шляху та алгоритми пошуку, опублікувати веб-сайт в загальний доступ.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Ілюстрації для ознайомлення з різними типами графів, ілюстрації можливих змін вигляду графу на веб-сайті, ілюстрації різних типів представлення графа на веб-сайті, ілюстрації роботи алгоритмів пошуку та найкоротшого шляху, ілюстрації з Azure та GitHub пов'язані з публікацією веб-сайту.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання **31 серпня 2022 р.**

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Дослідження теоретичних відомостей	10.03.2023	
2	Дослідження технологій	11.03.2023	
3	Написання першої версії веб-сайту, створення дизайну	12.03.2023	
4	Ознайомлення з бібліотекою D3	13.03.2023	
5	Додавання малювання графа під впливом сил	12.04.2023	
6	Додавання різноманітних представлень графа	13.04.2023	
7	Додавання алгоритмів найкоротшого шляху	14.04.2023	
8	Додавання алгоритмів пошуку по графу	15.04.2023	
9	Додавання можливості додавати нову вершину, ребра	22.04.2023	
10	Додавання можливості забрати вершину, очистити граф	04.05.2023	
11	Рефакторинг, переміщення коду у класи	05.05.2023	
12	Додавання можливості видалення ребро, змінити вагу	06.05.2023	
13	Дослідження доступних сервісів для публікації веб-сайту	20.05.2023	
14	Публікація веб-сайту на Azure	22.05.2023	
15	Додані теги для попереднього огляду веб-сторінки	28.05.2023	
16	Додавання веб-сайту до індексації Google	28.05.2023	

Студент _____ **Заречанський О.Р.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Щербина Ю.М.**
(підпис) (прізвище та ініціали)

Реферат

Дипломна робота на тему “Веб-сайт для роботи з графовими моделями” складається з вступу, 7 розділів, висновку та використаної літератури. В сумі робота має обсяг 34 сторінки, 35 рисунків, та 3 таблиці. В списку використаних джерел є 3 пункти.

Метою роботи була розробка веб-сайту, де користувач може в зручній та інтерактивній формі створювати та редагувати графи. При цьому сайт повинен мати приємний інтерфейс, а також найпопулярніші алгоритми які використовуються на графах. Сайт повинен бути опублікований в публічний доступ та бути доступним для будь якого пристрою в будь який час.

Перший розділ роботи включає короткі теоретичні відомості щодо графів, терміни звідти будуть використовуватись на протязі усієї роботи. Другий розділ розповідає про обраний технологічний стек веб-сайту та причини чому були обрані саме такі технології. Третій розділ описує як на веб-сайті створюється першочергова модель графу, описані виконані дії щоб граф мав його вигляд. Четвертий розділ містить можливі модифікації графа, як візуальні (розтягнути вершини на полотні), так і зі змінною даних (видалення вершини). П’ятий розділ описує різні способи представлення графа та показує результат кожного способу представлення отриманого щодо певного графа. Шостий розділ містить різноманітні алгоритми які можна застосовувати до графа, ці алгоритми включають пошук найкоротшого шляху та повне проходження графа. Останній розділ фокусується на публікації веб-сайту в загальний доступ та описує крок-за-кроком цей процес.

Як результат робота містить в собі весь процес створення сайту – від вибору технологій, до публікації в загальний доступ. Веб-сайт готовий і є можливим до використання в загальному доступі. В майбутньому функціонал сайту може бути значно розширений.

ЗМІСТ

ВСТУП

Дана робота присвячена написанню веб-сайту для роботи з графовими моделями. Актуальність даної роботи полягає у наданні усім людям в режимі реального часу, з будь якого пристрою, потребуючи лише доступ до інтернету, можливості працювати з графовими моделями. В цей же час застосувань в графових моделей може бути безліч, від деревовидної структури папок комп’ютера до знаходження найближчого маршруту до магазину в навігаційних системах, тому розробка такого веб-сайту дійсно принесе користь людям.

Мета дипломної роботи: написання зручного веб-сайту для роботи з графовими моделями, для здійснення мети веб-сайт повинен мати такі можливості:

- побудову та візуалізацію графа;
- представлення графа різними способами;
- надати користувачеві можливість використовувати декілька найпотрібніших алгоритмів при роботі з графами;
- опублікувати веб-сайт у публічний доступ.

1. КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Перед початком роботи над веб-сайтом пов’язаними з графовими моделями потрібно спочатку дати конкретні визначення термінам, оскільки в теорії графів термінологія часто відрізняється залежно від джерела інформації.

Граф – це пара $G = (V, E)$, де V – непорожня скінченна множина елементів, E – множина пар елементів з V. Елементи множини V називають *вершинами*, а кожну пару з E називають *ребрами*. Графи є основним поняттям вивчення теорії графів.

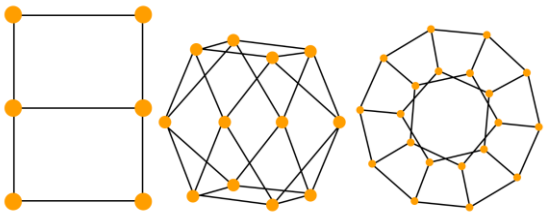

Рисунок 1.1 Різноманітні графи.

Вершини поєднанні одним ребром називають *суміжними*. В цьому ж випадку ці вершини є *інцидентними* до ребра, як і ребро є *інцидентним* до вершин. Декілька ребер, які інцидентні одній вершині теж називають *суміжними*.

Ребра графа можуть бути *орієнтовані* та *неорієнтовані*.

Неорієнтоване ребро не має напрямку і є *рівноцінним* для обох вершин які це ребро сполучає, тобто *обидві* вершини мають сполучення одна з одною Наприклад: за наявності вершин A і B та неорієнтованого ребра між ними, A має сполучення до B, а B має сполучення до A.

Відмінність орієнтованого ребра полягає у наявності у нього *напрямку* – іншими словами ребро має початок в одній вершині і кінець в іншій, при цьому сполучення відбувається *лише у напрямку ребра*. Наприклад: для вершин A і B та орієнтованого ребра між ними, яке має напрямок від A до B, A є сполученим з B за допомогою цього ребра, проте B з A – не є пов’язані. Орієнтовані ребра також називають *дугами*. Відповідно, залежно від наявності певного виду ребер графи поділяють на орієнтовані та неорієнтовані.

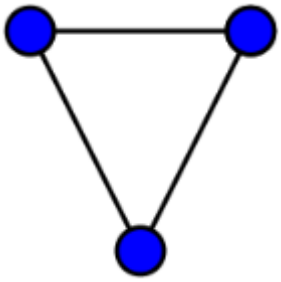

Рисунок 1.2 Неорієнтований граф, відсутні орієнтовані ребра (дуги).

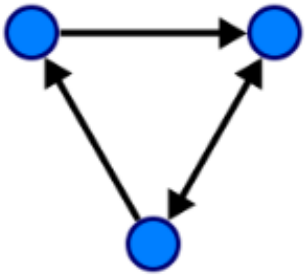

Рисунок 1.3 Орієнтований граф, вершини поєднані орієнтованими ребрами (дугами).

Слід відзначити, що орієнтований граф може виконувати функції неорієнтованого. Для цього достатньо кожну пару вершин поєднювати додатковою дугою у *протилежному* напрямку. В цей же час, зворотнього ефекту досягти неможливо.

Дуги та ребра, які поєднують ті самі вершини і мають однаковий напрямок називають *кратними* або *паралельними*.

У випадку коли дуга чи ребро сполучає вершину сама з собою, це ребро називають *петлею.*

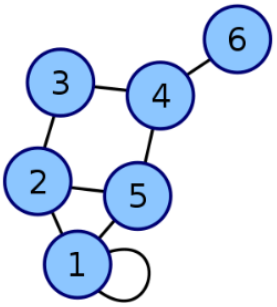

Рисунок 1.2 Граф з петлею.

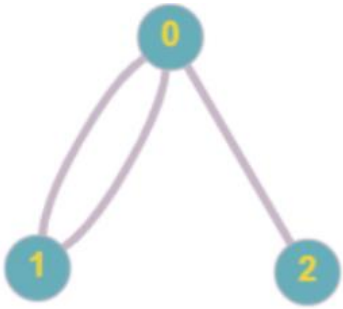

Рисунок 1.3 Граф з кратними ребрами.

Простим графом називають граф у якого відсутні петлі та кратні ребра.

Найчастіше графи використовують в поєднанні з додатковою інформацією – відстань між пунктами, вартість, час проїзду, тощо. Для використання цих додаткових параметрів на графі ввели поняття *зваженого графа.*

Зваженим називають граф, кожному ребру якого приписано дійсне число, яке називають *вагою* цього ребра. Відповідно, існують також *зважені орієнтовані* графи – де кожна дуга має вагу.

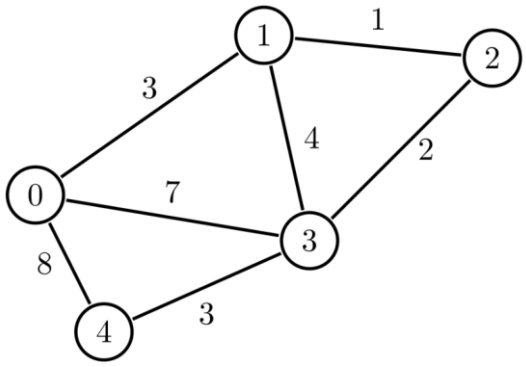

Рисунок 1.4 Зважений граф.

Подаймо дані вказані вище у вигляді таблиць для підсумку:

Таблиця 1.1 – Порівняння зважених і незважених графів

Граф	Ребра мають вагу
Незважений	Ні
Зважений	Так

Таблиця 1.2 – Порівняння орієнтованих та неорієнтованих графів

Граф	Ребра мають напрямок
Неорієнтований	Ні
Орієнтований	Так
Мішаний	Так, не всі

Таблиця 1.3 – Порівняння простих і складних графів

Граф	Має кратні ребра / петлі
Простий	Ні
Складний	Так

2. ТЕХНОЛОГІЧНИЙ СТЕК ВЕБ-САЙТУ

Для більшості дій над нашою графовою моделлю на веб-сайті нам не потрібні функції серверу, тому backend веб-сайту може бути абсолютно будь яким, мною була обрана мова C# та .Net Core, завдяки тому що IDE Visual Studio має MVC (model-view-controller) шаблон, який дозволяє за декілька кліків створити повністю готовий до роботи веб-сайт з backend та frontend частинами.

Для frontend частини будуть використовуватись стандартні для MVC представлення в форматі cshtml – які є модифікованими файлами розмітки html. Також ми використовуємо звичні CSS стилі та скрипти написані на мові JavaScript які збережені у файлах з розширенням JS.

Вся логіка буде відбуватись на frontend частині нашого веб-сайту. При тому вона буде розбита на дві частини, початкове налаштування нашого графа та власне користувацькі функції та модифікації наявного графа. Для виконання цієї логіки на мові JavaScript існує корисна бібліотека під назвою D3 яка спрощує та надає певні готові методи для відображення наборів даних на веб-сайтах, відповідно він і допоможе нам зобразити наш граф на нашій сторінці.

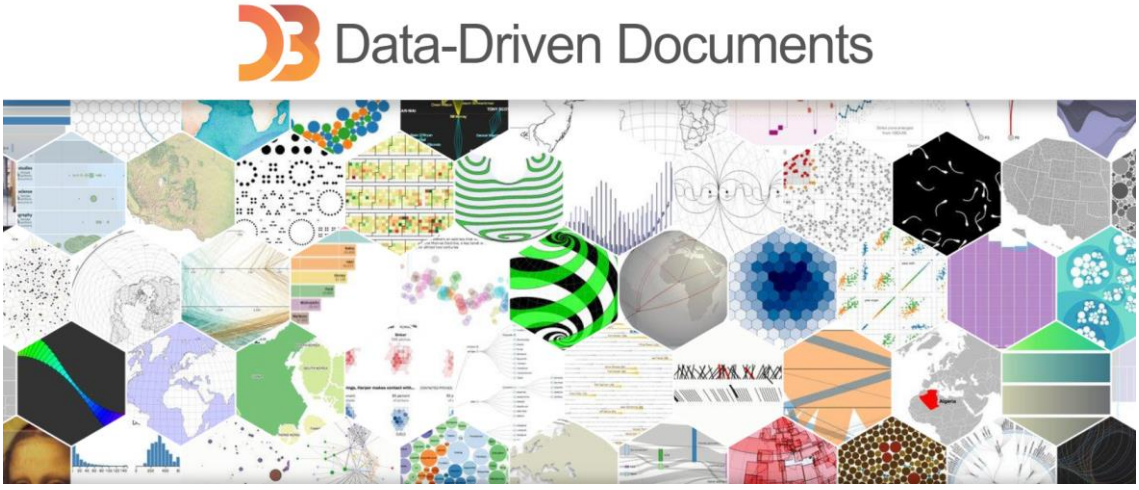

Рисунок 2.1 Бібліотека D3 - для допомоги у візуалізації будь яких даних.

3. СТВОРЕННЯ ГРАФА

Я вирішив створити орієнтований зважений граф, для надання більшої свободи та користі користувачеві, бо як ми з'ясували раніше це дає нам більше практичних застосувань графа. В цей же час через складність реалізації було вирішено обмежитись простим графом – без петель і без кратних дуг.

Розберемо створення графа в деталях.

Наші дані будуть збережені у двох списках, *списку вершин*, та *списку дуг*. Кожна вершина обов'язково має мати ідентифікатор, а кожна дуга обов'язково має початок (*source*) і кінець (*target*). Враховуючи що наш граф є також зваженим то кожен елемент в списку дуг також містить вагу цієї дуги (*weight*).

Збережені дані ми маємо, тепер потрібно створити для них граф, для цього нам потрібно зробити декілька речей:

1. Створити об'єкти векторної графіки, які будуть використовуватись для відображення наших дуг, вершин, вагів, та ваг. Для цього будемо можемо використати різні елементи векторної графіки які існують в html за замовчуванням, такі як шляхи (path), лінії (line), коло (circle) та інші, залежно від того як потрібно щоб виглядав кінцевий варіант.

   В роботі з векторною графікою в html потрібно пам'ятати що елементи додані в робочу область на початку будуть на нижчому рівні, ніж елементи додані пізніше. Тому важливо, щоб такі елементи як дуги були подані спочатку, а ідентифікатори вершин і ваги дуг пізніше. Таким чином вершини не будуть перекривати важливу інформацію, а дуги не будуть накладатись поверх вершин.

   Також під час створення цих елементів можемо задати їх поведінку у різних ситуаціях, наприклад переміщення їх мишею, фіксування позиції при кліку мишею тощо.

2. Створити функцію, яка дозволить нам визначити координати розміщення об’єктів створених в першому пункті в нашій робочій області. Ця функція буде викликатись кожен “тік” (дуже малий інтервал часу), що дасть нам неперервну зміну координат, завдяки чому ми отримуємо гладку анімацію переміщення об’єктів нашого графа, наприклад при перетягуванні вершини.

   Окрім цього, ці координати можна змінювати за нашим бажанням, наприклад в нашому випадку ми обмежимо за допомогою математичних функцій щоб координати наших вершин не виходили за межі робочої області. Але ця функція може задавати не тільки координати а і будь що інше, наприклад вигини ліній, зміну розмірів, вигляду та інше.

3. Створити “*симуляцію сил*”, який буде відповідальний за вплив на розміщення елементів на нашій робочій області за допомогою декількох факторів.

   Перший з них – це сили які будуть впливати на елементи графа, вони можуть бути довільними, але для веб-сайту було вибрано декілька з них які є доцільними:

   1) *Сила зв’язку* – не буде дозволяти пов’язаними дугами елементам наближатись надто близько або віддалятись, тобто вона буде намагатись тримати пов’язані елементи на певній відстані.
   2) *Центральна сила* – буде намагатись притягнути елементи до центру робочої області, щоб граф за можливості перебував в центрі а не біля країв.
   3) *Сила “зарядів”* – діє за принципом магнітів, надаючи вершинам певного заряду ми можемо досягти їх взаємодію між собою – в нашому випадку ми даємо кожній вершині однаковий негативний заряд що спричиняє відштовхування елементів один від одного, це зробить наш граф візуально більш розгалуженим і зручнішим у користуванні.

Другий фактор – це наша попередньо створена функція розміщення елементів, в якій ми задали певні обмеження, щоб не покладати все на тільки роботу сил.

Після всіх дій вище ми можемо запустити створену симуляцію, яка відобразить наш граф і почне кожен “тік” оновлювати його під дією заданих сил та функції, що зробить його динамічним, це дає можливість додати інтерактивність як переміщення вершин мишкою, додавання нових вершин та дуг в реальному часі, тощо. На цьому початкове налаштування графа завершено і користувач може почати його модифікувати або використовувати на ньому алгоритми.

На цьому етапі доцільно для зручності користувача дати йому можливість наближувати та переміщатись по робочій області. Також додаймо автоматичну зміну розмірів робочої залежно від розмірів вікна, це дає можливість користування веб-сайтом навіть на мобільних пристроях.

Кінцевий результат:

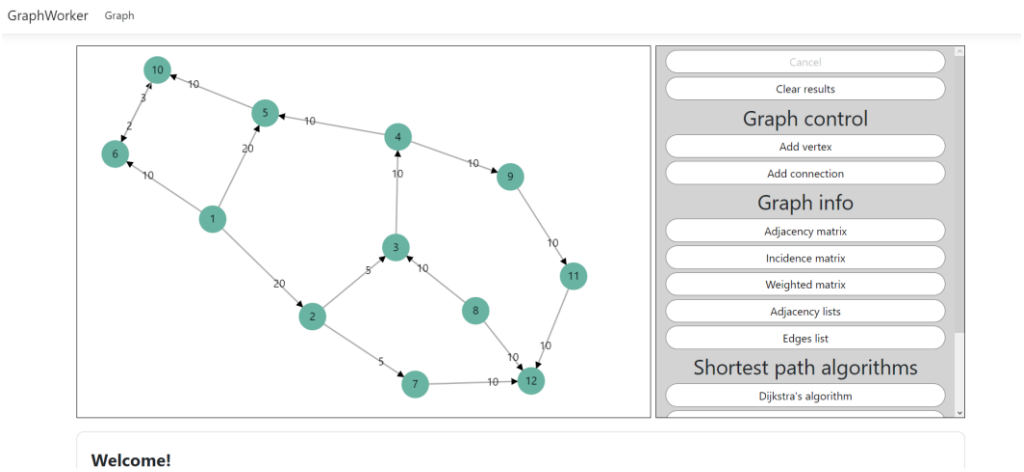

Рисунок 3.1 Вигляд веб-сайту після завантаження сторінки.

4. ІНСТРУМЕНТИ МОДИФІКАЦІЇ ГРАФА

Після повноцінного завантаження графа в користувача може виникнути потреба змінити його під свої потреби для цього на сайті передбачені спеціальні функції модифікації графа:

1. *Переміщення вершини*. За допомогою миші користувач може переміщати вершини на зручні йому позиції, де вони закріплять фіксовану позицію і не будуть переміщатись під дією будь яких сил, проте сама ця вершина буде досі впливати на інші незакріплені вершини. На натиск миші вершина повернеться до початкового стану і втратить фіксовану позицію. Ця модифікація лише візуальна і ніяк не впливає на наявні набори даних.
   Приклад:

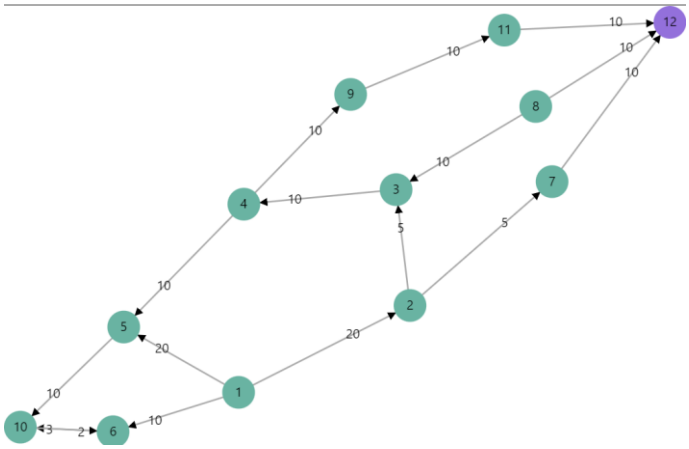

Рисунок 4.1 Граф з закріпленою точкою у верхньому правому куті.

Фіолетова точка є закріпленою в правому верхньому куті робочої області – як результат дії сили “заряду” для збалансування інша частина нашого графа розтягнулась в лівий нижній кут – протилежний закріпленій точці. При відкріпленні обидва кінці графа повернуться ближче до центру.

2. *Додавання нової вершини*. На панелі дій справа від графа в користувача наявна кнопка додавання нової вершини. При натиску на цю кнопку на робочу область додається нова вершина, з ідентифікатором, який більший за

максимальний наявний на 1. Ця дія змінить наші початкові дані, додавши в список вершин нову вершину. Як ми вже знаємо, елементи векторної графіки відображаються в порядку їх додавання, тому щоб ця точка не перекривала дані, які мали би бути над нею, ми спочатку забираємо з робочої області всі елементи які мають бути над нашою новою вершиною, тоді додаємо її і повертаємо видалені елементи назад. В нашому випадку ми забираємо ваги дуг, додаємо вершину і тоді знову додаємо ваги дуг.

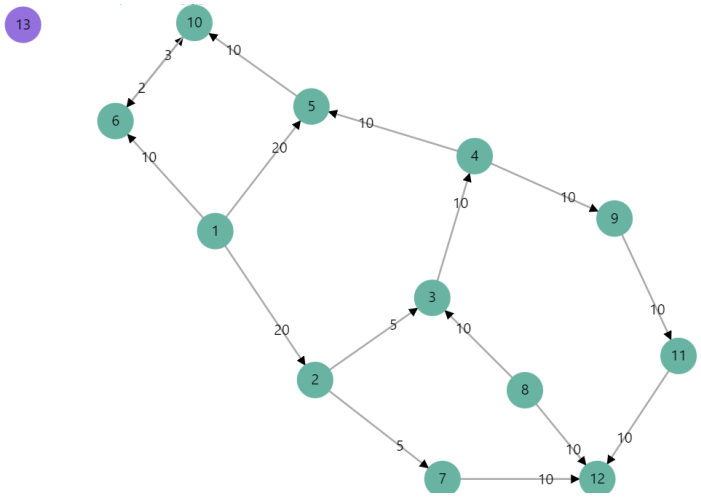

Рисунок 4.2 Граф в який була додана вершина 13.

3. *Додавання нової дуги.* На панелі дій також наявна кнопка додавання нової дуги, при натисканні якої нам потрібно вибрати 2 вершини. Між цими вершинами і буде сформована дуга, від тієї що була вибрана першою до тієї що була вибрана другою.

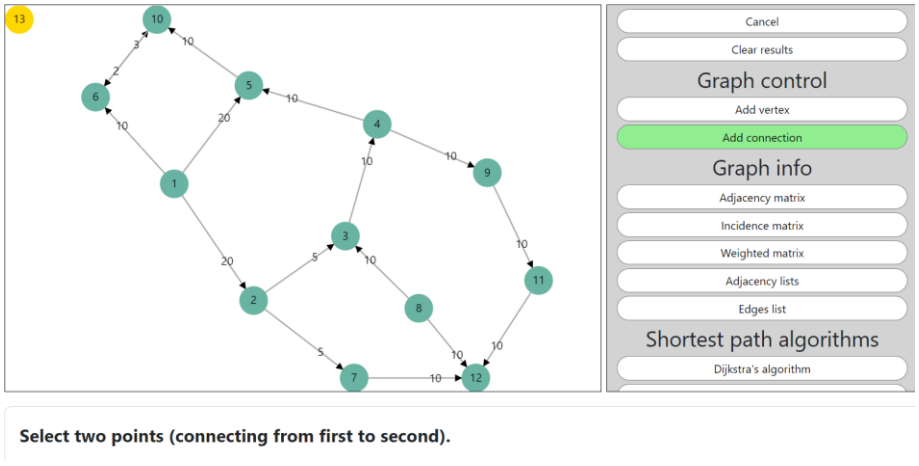

Рисунок 4.3 Граф з вибраною вершиною яка буде початком дуги.

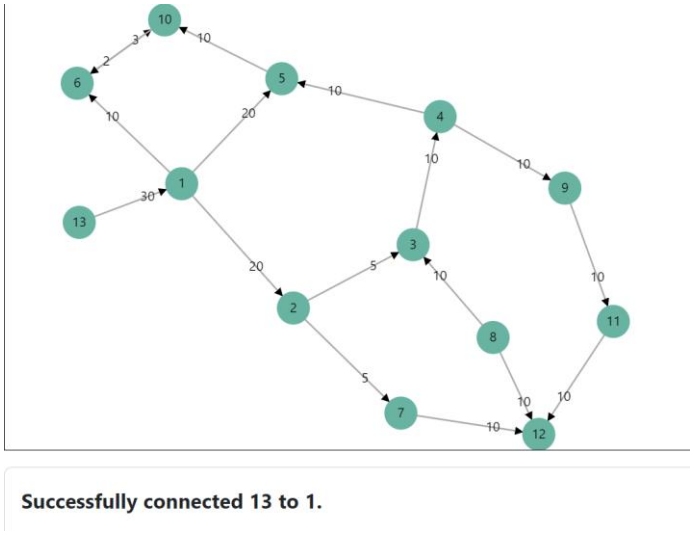

Рисунок 4.4 Після обрання другої вершини отримуємо нову дугу.

При виконанні цієї дії наш набір даних змінився – в список дуг була додана нова дуга. Також, як і випадку з новою вершиною, потрібно перемалювати наявні елементи щоб був правильний порядок відображення. У цьому випадку потрібно забрати все, крім дуг, оскільки вони є на найнижчому рівні, повернути решту елементів і в кінці додати вагу для нової дуги.

5. ПРЕДСТАВЛЕННЯ ГРАФІВ

Графічний спосіб представлення графів найзручніший для ока людини, проте існують також інші способи представлення графів, які найчастіше використовуються для зберігання даних графа в пам’яті комп’ютера. Для користувача надається можливість перевести дані графа, створеного ним графічно, в інший спосіб представлення.

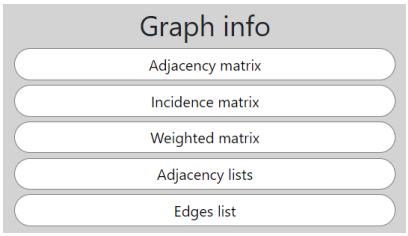

Рисунок 5.1 Отримання даних в іншому представлені натиском кнопки.

Розглянемо наявні способи представлення:

1. *Матриця суміжності* – булева квадратна матриця, кількість рядків та стовпців рівна кількості наявних вершин. Кожен рядок та стовпець відповідає за певну вершину, відповідно клітинки відповідають за наявність дуги між вершиною рядка та стовпця, якщо вона існує, записуємо в клітинку 1, в іншому випадку 0. В нашому випадку орієнтованого графа вона не обов’язково симетрична, на відміну від неорієнтованого графа.

Adjacency matrix:

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	1	0	0	1	1	0	0	0	0	0	0	0
2	0	0	1	0	0	0	1	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	1	0	0	0	0
5	0	0	0	0	0	0	0	0	0	1	0	0	0
6	0	0	0	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	1	0
8	0	0	1	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	0	1	0	0
10	0	0	0	0	0	1	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	1	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0
13	1	0	0	0	0	0	0	0	0	0	0	0	0

Рисунок 5.2 Матриця суміжності до графа на рис. 4.4.

2. *Зважена матриця* є аналогом матриці суміжності, проте розрахованою на зважений граф. Замість показу наявності дуги між двома вершинами в клітинці графа там буде відображена вага цієї дуги, в той же час відсутність дуги буде показана не нулем, а безмежністю.

Weighted matrix:

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	20	∞	∞	20	10	∞	∞	∞	∞	∞	∞	∞
2	∞	0	5	∞	∞	∞	5	∞	∞	∞	∞	∞	∞
3	∞	∞	0	10	∞	∞	∞	∞	∞	∞	∞	∞	∞
4	∞	∞	∞	0	10	∞	∞	∞	10	∞	∞	∞	∞
5	∞	∞	∞	∞	0	∞	∞	∞	∞	10	∞	∞	∞
6	∞	∞	∞	∞	∞	0	∞	∞	∞	3	∞	∞	∞
7	∞	∞	∞	∞	∞	∞	0	∞	∞	∞	∞	10	∞
8	∞	∞	10	∞	∞	∞	∞	0	∞	∞	∞	10	∞
9	∞	∞	∞	∞	∞	∞	∞	∞	0	∞	10	∞	∞
10	∞	∞	∞	∞	∞	2	∞	∞	∞	0	∞	∞	∞
11	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	10	∞
12	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	∞
13	30	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0

Рисунок 5.3 Зважена матриця до графа на рис.4.4.

3. *Матриця інцидентності.*

Матриця інцидентності містить кількість рядків рівну кількості вершин, а кількість стовпців рівну кількості дуг. Відповідно стовпці відповідають за дуги, а рядки за вершини.

Заповнення клітинок матриці:

- якщо дуга стовпця виходить з вершини рядка – записуємо -1;
- якщо дуга стовпця входить в вершину рядка – записуємо 1;
- якщо дуга стовпця не пов’язана з вершиною – записуємо 0.

Incidence matrix:

1	-1	-1	-1	0	0	0	0	0	0	0	0	0	0	0	0	1
2	1	0	0	-1	-1	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	1	0	-1	0	0	0	0	0	1	0	0	0	0
4	0	0	0	0	0	1	-1	-1	0	0	0	0	0	0	0	0
5	0	1	0	0	0	0	1	0	-1	0	0	0	0	0	0	0
6	0	0	1	0	0	0	0	0	0	-1	0	0	0	0	1	0
7	0	0	0	0	1	0	0	0	0	0	-1	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	-1	-1	0	0	0
9	0	0	0	0	0	0	0	1	0	0	0	0	0	-1	0	0
10	0	0	0	0	0	0	0	0	1	1	0	0	0	0	-1	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	-1
12	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1

Рисунок 5.4 Матриця інцидентності до графа на рис.4.4.

4. *Списки суміжності* – список суміжних вершин для кожної вершини графа. У випадку зваженого графа, як на веб-сайті, для кожної суміжної вершини потрібно також мати вагу інцидентної дуги.

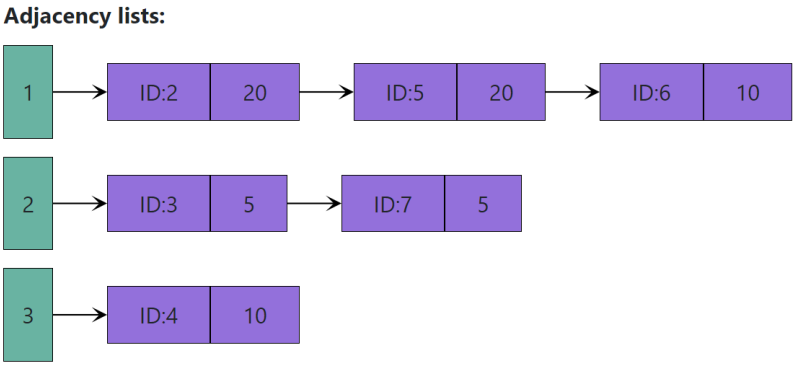

Рисунок 5.5 Списки суміжності для перших 3 вершин графа з рис.4.4.

5. *Список пар (список ребер)* – список ребер графа у вигляді двох вершин яким це ребро є інцидентним. У випадку орієнтованого графа – потрібно розрізняти які вершини є початком, а які кінцем дуги. У випадку зваженого графа крім збереження вершин потрібно також зберігати вагу інцидентної дуги. На веб-сайті граф зважений та орієнтований, тому в таблиці дуг наявні стовпчики з ідентифікаторами початку, кінця, а також вагою дуги.

Edges list:

From	To	Weight
1	2	20
1	5	20
1	6	10
2	3	5
2	7	5
3	4	10
4	5	10
4	9	10
5	10	10
6	10	3
7	12	10
8	3	10
8	12	10
9	11	10
10	6	2
11	12	10
13	1	30

Рисунок 5.6 Список пар для графа на рис. 4.4.

6. АЛГОРИТМИ ДЛЯ РОБОТИ З ГРАФАМИ

Окрім візуального представлення графа, користувачеві можуть знадобитись практичні застосування графа. Цими застосуваннями на веб-сайті представлено два типи алгоритмів для роботи з графами – пошук найкоротшого шляху та обхід графа.

Задача про найкоротший шлях полягає у знаходженні найкоротшого шляху від заданої початкової вершини до іншої заданої вершини. Відповідно ця задача актуальна лише в зважених графах, як і на веб-сайті. З цієї задачі випливає дві наступні:

1) Для обраної початкової вершини знайти найкоротший шлях від обраної вершини до решти вершин;
2) Знайти найкоротший шлях між усіма парами вершин.

Для надання користувачеві можливості розв’язування обидвох даних задач на сайті присутні два алгоритми: Дейкстри та Флойда.

Алгоритм Дейкстри – алгоритм для знаходження найкоротшого шляху від обраної початкової точки до всіх решти точок. Потрібно звернути увагу що він не працює для графів у яких можливі від’ємні ваги.

Опис кроків алгоритму Дейкстри:

1) Створюємо список пройдених вершин, та надаємо кожній вершині мітки зі значенням безмежність. Мітка початкової вершини має значення 0. Встановлюємо початкову вершину як теперішню.
2) В кожній непройденій вершині, куди з теперішньої вершини прямує дуга, встановлюємо нові значення міток. Якщо теперішнє значення мітки є більшим за суму значення мітки теперішньої вершини та ваги інцидентної дуги, тоді значення мітки заміняється на цю суму.

3) Додаємо теперішню вершину в список пройдених та встановлюємо нову непройдену вершину з найменшим значенням мітки як теперішню. Повертаємось до другого пункту. Повторюємо поки довжина списку пройденого не стане рівною довжині списку всіх вершин (тобто всі вершини не будуть пройдені).
4) Як результат значення всіх міток і буде найкоротших шляхом від початкової точки. До вершин мітки яких після виконання алгоритму містять безмежність немає можливості дістатись з початкової точки.

На веб-сайті при виборі алгоритму потрібно буде обрати дві точки, після чого найкоротший шлях буде відмічений, а хід роботи алгоритму буде описаний нижче.

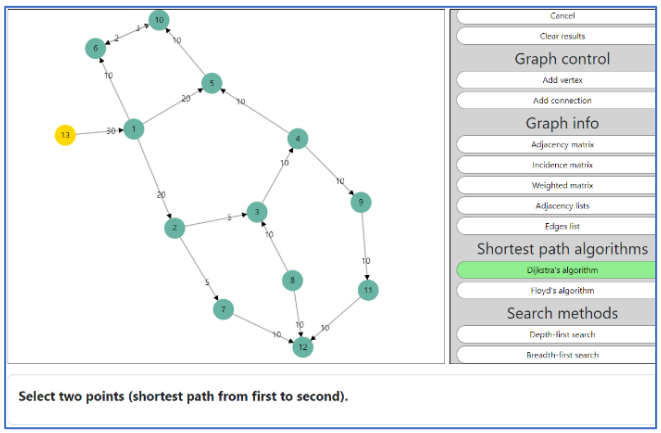

Рисунок 6.1 Обрана початкова точка для алгоритму Дейкстри.

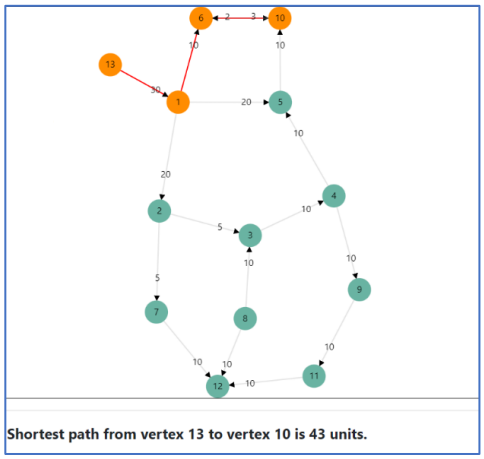

Рисунок 6.2 Показаний найкоротший шлях від вершини 13 до 10.

Algorithm steps:
Set weight of all points to infinity, starting from point 13, weight 0.

1. Setting new weights from point 13.
   1. Setting new weight for 1: from Infinity to 0 + 30 = 30
   New minimal point is 1, weight 30.
2. Setting new weights from point 1.
   1. Setting new weight for 2: from Infinity to 30 + 20 = 50
   2. Setting new weight for 5: from Infinity to 30 + 20 = 50
   3. Setting new weight for 6: from Infinity to 30 + 10 = 40
   New minimal point is 6, weight 40.

Рисунок 6.3 Описані перші кроки алгоритму.

12. Setting new weights from point 11.

All available points are passed

Entire path summary:

1. From 13 to 1: 0 + 30 = 30
2. From 1 to 6: 30 + 10 = 40
3. From 6 to 10: 40 + 3 = 43

Рисунок 6.4 Кінець роботи алгоритму та підсумок.

Алгоритм Флойда – алгоритм для знаходження найкоротших шляхів між усім вершинами графа. На відміну від алгоритму Дейкстри він може працювати і з від'ємними вагами дуг, проте граф не повинен містити циклів з від'ємною довжиною.

Опис кроків алгоритму Флойда:

1) Для початку роботи наш граф потрібно звести до вигляду зваженої матриці, яку ми вже розглядали, позначимо її як $W^{(0)}$.
2) Далі для визначимо ітераційні змінні:
   - i – змінна для ітерації по рядках матриці.
   - j – змінна для ітерації по стовпчиках матриці.
   - k – змінна для ітерації кроку виконання алгоритму.
3) В циклі k, яке набуває значень від 1 до кількості вершин графа, за допомогою матриці $W^{(k-1)}$ знаходимо матрицю $W^{(k)}$ використовуючи дане рекурентне співвідношення: $w_{ij}^{(k)} = min(w_{ik}^{(k-1)} + w_{kj}^{(k-1)};\ w_{ij}^{(k-1)})$.
4) Після виконання циклу, остання отримана матриця W і буде матрицею найкоротших шляхів між усіма вершинами графа. Клітинки де залишились знаки безмежності означають що шлях відсутній.

Final result (shortest distance matrix):

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	20	25	35	20	10	25	∞	45	13	55	35	∞
2	∞	0	5	15	25	37	5	∞	25	35	35	15	∞
3	∞	∞	0	10	20	32	∞	∞	20	30	30	40	∞
4	∞	∞	∞	0	10	22	∞	∞	10	20	20	30	∞
5	∞	∞	∞	∞	0	12	∞	∞	∞	10	∞	∞	∞
6	∞	∞	∞	∞	∞	0	∞	∞	∞	3	∞	∞	∞
7	∞	∞	∞	∞	∞	∞	0	∞	∞	∞	∞	10	∞
8	∞	∞	10	20	30	42	∞	0	30	40	40	10	∞
9	∞	∞	∞	∞	∞	∞	∞	∞	0	∞	10	20	∞
10	∞	∞	∞	∞	∞	2	∞	∞	∞	0	∞	∞	∞
11	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	10	∞
12	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	∞
13	30	50	55	65	50	40	55	∞	75	43	85	65	0

Рисунок 6.5 Матриця найкоротших шляхів графа з рис. 4.4, отримана за допомогою алгоритму Флойда.

Обхід графів використовується для систематичному переборі вершин, при якому кожна вершина отримує унікальний порядковий номер. Алгоритми обходу вершин графа називають *методами пошуку*. На сайті представлено два алгоритми обходу вершин графа – пошук углиб та пошук вшир.

При *пошуку углиб*, або *DFS-методі* (від англ. depth-first search) користувач повинен вибрати довільну вершину з якої і почнеться обхід графа. При роботі цього алгоритму ми будемо проходити як можна далі по ребрах графа, після чого повертатись до попередніх вершин і йти якомога далі по них, починаючи з довільної обраної вершини.

Кроки DFS-методу:

1) Створюємо порожній стек (stack – структура даних, останній елемент стеку – перший в черзі на вилучення з нього). В стек додаємо довільну обрану вершину, даємо їй DFS-номер 1.
2) Візьмемо останню вершину в стеку.
    - Якщо всі суміжні їй вершини отримали DFS-номер то вилучаємо її зі стеку і повертаємось на початок цього кроку, якщо стек виявився порожній то переходимо до пункту 3.
    - Якщо є суміжна вершина яка не має DFS-номеру тоді додаємо другу вершину у стек, давши їй наступний DFS-номер. Позначимо інцидентне ребро. Повертаємось на початок кроку.
3) На сайті граф є орієнтованим та може бути незв’язним, тому для гарантії проходження всього графа після пункту 2, додаємо вершину яка не отримала DFS-номер до стеку, даємо їй номер та повертаємось до пункту 2. Так робимо поки усі вершин не отримають DFS-номер. Після цього всі вершини графа будуть пройдені і виконання алгоритму буде закінчено.

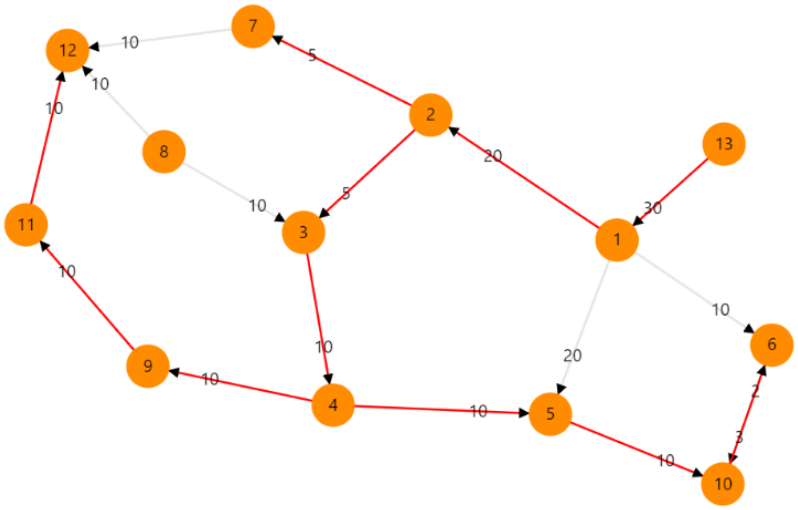

Рисунок 6.6 Обхід графа з рис. 4.4 за допомогою DFS-методу з початком у вершині 13.

DFS:

Vertex	DFS-Number	Stack
13	1	13
1	2	13, 1
2	3	13, 1, 2
3	4	13, 1, 2, 3
4	5	13, 1, 2, 3, 4
5	6	13, 1, 2, 3, 4, 5
10	7	13, 1, 2, 3, 4, 5, 10
6	8	13, 1, 2, 3, 4, 5, 10, 6
-	-	13, 1, 2, 3, 4, 5, 10
-	-	13, 1, 2, 3, 4, 5
-	-	13, 1, 2, 3, 4
9	9	13, 1, 2, 3, 4, 9
11	10	13, 1, 2, 3, 4, 9, 11
12	11	13, 1, 2, 3, 4, 9, 11, 12
-	-	13, 1, 2, 3, 4, 9, 11
-	-	13, 1, 2, 3, 4, 9
-	-	13, 1, 2, 3, 4
-	-	13, 1, 2, 3
-	-	13, 1, 2
7	12	13, 1, 2, 7
-	-	13, 1, 2
-	-	13, 1
-	-	13
-	-	∅
8	13	8
-	-	∅

Рисунок 6.7 Протокол обходу графа пошуком вглиб.

На відміну від пошуку вглиб – *пошук вшир* або *BFS-метод* (від англ. breadth-first search) при проходженні графу спочатку буде проходити по всіх можливих суміжних вершинах і тільки після цього буде рухатись далі вглиб, розглядаючи інші вершини.

Кроки BFS-методу:

1) Створюємо порожню чергу (queue – структура даних, перший елемент черги – перший в черзі на вилучення з неї). В чергу додаємо довільну обрану вершину, даємо їй BFS-номер 1.
2) Візьмемо першу вершину в черзі.
    - Якщо всі суміжні їй вершини отримали BFS-номер то вилучаємо її з черги і повертаємось на початок цього кроку, якщо черга виявилась порожньою то переходимо до пункту 3.

- Якщо є суміжна вершина яка не має BFS-номеру тоді додаємо другу вершину у чергу, давши їй наступний BFS-номер. Позначимо інцидентне ребро. Повертаємось на початок кроку.

3) Так само як і для пошуку вглиб, для гарантії проходження всього графа, додаємо вершину яка не отримала BFS-номер до черги, даємо їй номер та повертаємось до пункту 2. Так робимо поки усі вершин не отримають BFS-номер. Після цього всі вершини графа будуть пройдені і виконання алгоритму буде закінчено.

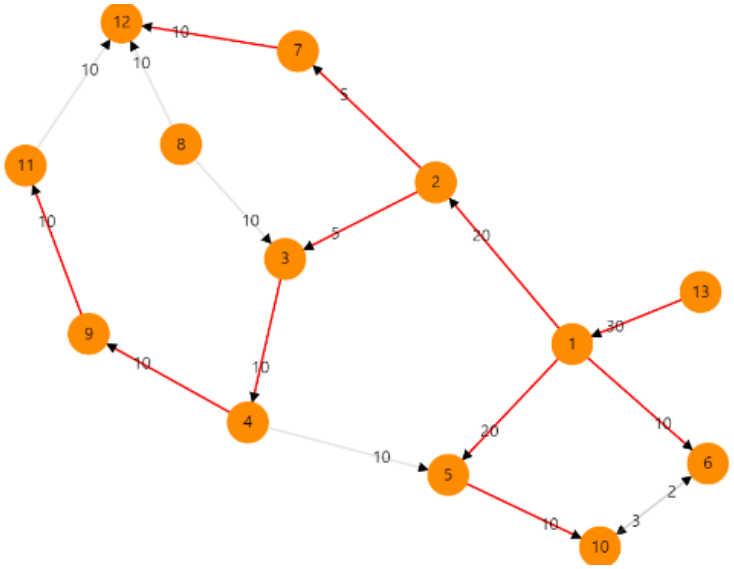

Рисунок 6.8 Обхід графа з рис.4.4 за допомогою BFS-методу з початком у вершині 13.

BFS:

Vertex	BFS-Number	Queue
13	1	13
1	2	13, 1
-	-	1
2	3	1, 2
5	4	1, 2, 5
6	5	1, 2, 5, 6
-	-	2, 5, 6
3	6	2, 5, 6, 3
7	7	2, 5, 6, 3, 7
-	-	5, 6, 3, 7
10	8	5, 6, 3, 7, 10
-	-	6, 3, 7, 10
-	-	3, 7, 10
4	9	3, 7, 10, 4
-	-	7, 10, 4
12	10	7, 10, 4, 12
-	-	10, 4, 12
-	-	4, 12
9	11	4, 12, 9
-	-	12, 9
-	-	9
11	12	9, 11
-	-	11
-	-	Ø
8	13	8
-	-	Ø

Рисунок 6.9 Протокол обходу графа BFS-методом.

7. ПУБЛІКАЦІЯ ВЕБ-САЙТУ В ЗАГАЛЬНИЙ ДОСТУП

Код веб-сайту написаний і готовий до публікації в загальний доступ в інтернет, що є однією з головних цілей роботи – це загальнодоступність. Будь-хто маючи посилання зможе отримати доступ до веб-сайту.

Першим ділом потрібно завантажити вже наявний код в інтернет, для того щоб цей код в майбутньому можна було публікувати. Для цього скористуємось одним з найбільших веб-сервісів для спільної розробки програмного забезпечення – *GitHub*. GitHub базується на системі керування версій Git, яка забезпечує просте оновлення коду як локально, так і віддалено, зберігаючи всю історію змін до програми. Таким чином будь-хто маючи доступ до віддаленого репозиторію (який містить у собі всі файли програми) зможе їх отримати локально на свій комп'ютер.

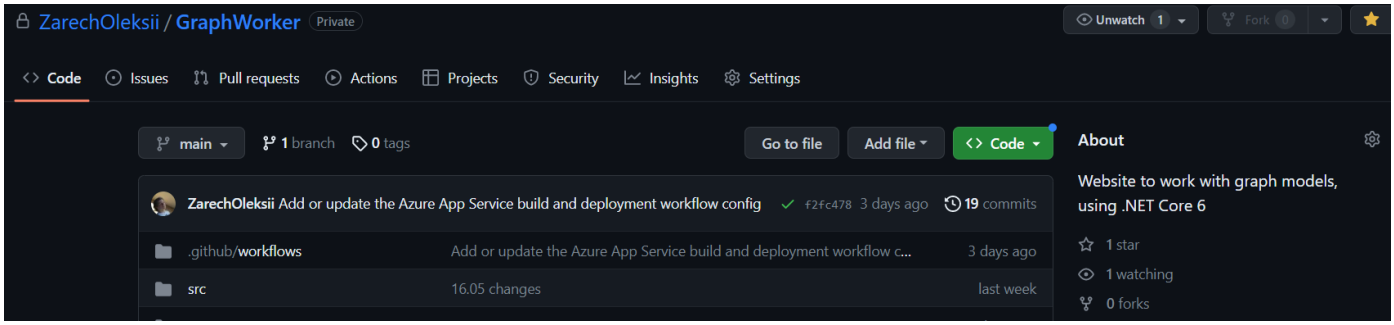

Рисунок 7.1 Скріншот готового репозиторію з завантаженим кодом веб-сайту.

Проте Git корисний не лише для спільної розробки ПЗ, але і для публікації веб-сайту в публічний доступ. Відмінність в тому, що отримувати код будуть на люди локально, а віддалений сервер, на якому програма буде побудована і запущена.

Наступним кроком буде вибір сервісу для хостингу програми. Потрібно щось що буде сумісне з нашим технологічним стеком, надійне, достатньо просте в використанні та як можна дешевше. Під усі ці параметри ідеально підходить *Microsoft Azure* або просто *Azure* – хмарна платформа для обчислень.

Тепер про кожен пункт детальніше:

- Функціонал Azure неймовірний – від веб-сайтів та віртуальних машин до баз даних та контейнерів Docker. Тут точно можна буде отримати те що потрібно для того щоб опублікувати наш веб-сайт, при тому різними способами якщо потрібно. Також Azure розроблявся компанією Microsoft, так само як і .NET Core на якому написаний бекенд частина веб-сайту, так само як і IDE Visual Studio, відповідно можна припустити що на Azure буде найкраща сумісність з написаним сайтом.
- Враховуючи що компанія Microsoft є одним з провідних розробників програмного забезпечення у світі, можна не сумніватись у якості послуг які надаються на Azure. Проте є наявні навіть додаткові сервіси для підвищення надійності вже наявних, якщо вони нам знадобляться, такі як бекапи, репліки, копії даних, тощо.
- Простота у використанні теж забезпечена, адже усі ресурси Azure можна створювати, редагувати, видаляти через веб-сайт Azure. Не потрібно писати скриптів чи команди в консолі, усе дуже просто і зрозуміло можна зробити через сайт. Для складніших операцій це можливо би було потрібно, але так як наш веб-сайт доволі простий та не має багатьох компонентів, то нам це не потрібно.
- Остання категорія це ціна, де теж усе добре. Згідно документації Azure можна мати до 10 веб програм, веб-апі, або мобільних програм, абсолютно безкоштовно необмежений період часу. Звісно у всього безкоштовного є обмеження – 1 гігабайт сховища та одна година процесорних обрахунків на день. Проте враховуючи що всі операції з графом виконуються в скрипт JavaScript файлах, то це не є великим мінусом, тому що ці файли виконуються уже в клієнта після завантаження сторінки і не створюють додаткове навантаження на процесор. Таким чином навантаження на процесор створюється лише при завантажені сторінки, тому однієї години доступу на день буде

достатньо для початку. Якщо цього ліміту буде не вистачати, це буде означати що сайт став достатньо великим щоб його монетизувати, і з цього прибутку оплатити дорожчий хостинг план.

Таким чином ми обрали платформу де буде відбуватись публікація веб-сайту. Перейдемо безпосередньо до цього процесу, спочатку потрібно зареєструватись на Azure. При початковій реєстрації Azure надає безкоштовне користування найпопулярнішими сервісами на 1 рік і також нараховуються 200 бонусних доларів на використання сервісів Azure. Враховуючи що на даному етапі веб-сайт не вимагає нічого окрім хостингу самого сайту, це на нас не впливає, проте згодом може використано для можливого розширення функціоналу сайту (наприклад для додавання бази даних).

Після реєстрації перейдемо до безпосереднього створення веб-сайту. Сервіс який буде використовуватись – *Azure App Service*. Спочатку потрібно обрати групу ресурсів. В одній групі будуть усі ресурси пов’язані з одним окремим продуктом. Відповідно створимо нову групу GraphCreator. Створений сервіс Azure App Service буде доданий до цієї групи і усі наступні сервіси які будуть стосуватись цього продукту у майбутньому теж будемо додавати в цю групу.

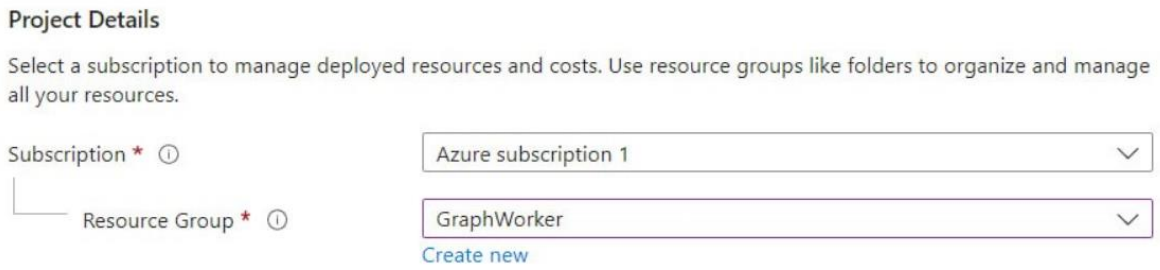

Рисунок 7.2 Вибір групи ресурсів, в якій будуть зберігатись усі сервіси поєднані з веб-сайтом.

Далі потрібно вказати назву, яка також буде слугувати посиланням на продукт. Так як Azure надає цей домен безкоштовно, то в посиланні на сайт буде приписка Azure, проте в майбутньому можна буде купити кращий для користувачів

домен і додати його до цього сервісу, тоді програма буде працювати на кількох доменах.

Також потрібно обрати що саме буде публікуватись, код, Docker контейнер чи статичний веб-сайт. Так як у нас є бекенд та немає Docker контейнеру обираємо варіант Code. Також потрібно обрати програмний стек який буде використаний для побудови програми, обираємо .NET 6, на якому написаний бекенд веб-сайту. Наступним параметром налаштувань є операційна система на якій буде хоститсь веб-сайт. Між Linux та Windows є різниця у технологіях які підтримуються та у ціні, проте враховуючи що наш веб-сайт буде використовувати безкоштовний план, а .NET Core 6 є крос-платформним, ми можемо обрати будь який варіант.

Далі необхідно вказати регіон де буде хоститись програма. Від цього залежать технічні плани які будуть нам доступні, а також затримка при доступі та користуванні з нашого місця знаходження. Для нашої програми це не є критичним, проте краще вказати щось поближче до України, наприклад Західна Європа.

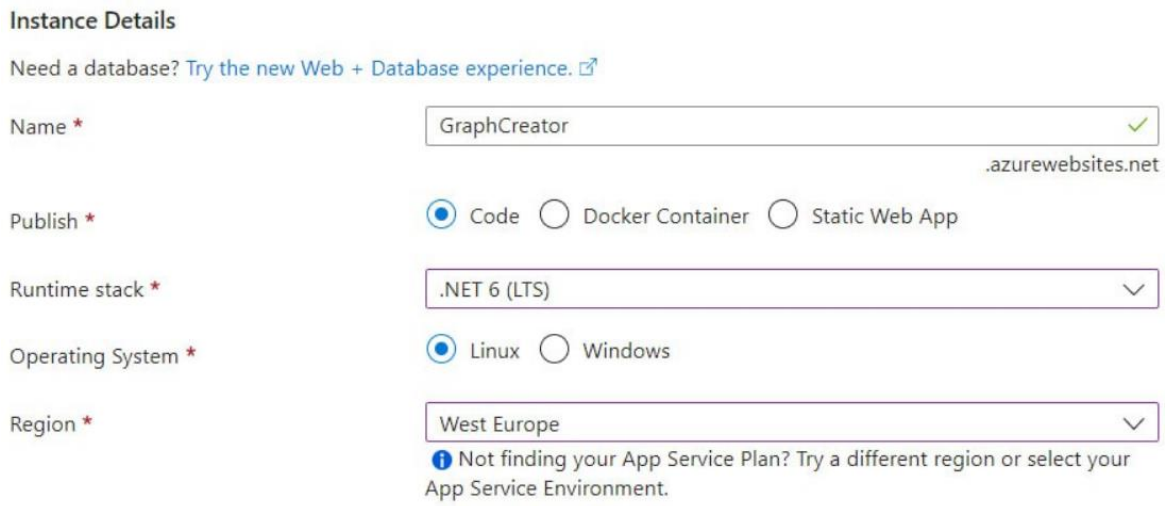

Рисунок 7.3 Задання імені (домену), способу публікації, технологічного стеку, операційної системи та регіону хостингу.

Після чого вказуємо технічний план, як вже було обговорено раніше безкоштовний план *F1* абсолютно покриває наші теперішні потреби тому обираємо його, в будь який момент його можна буде покращити.

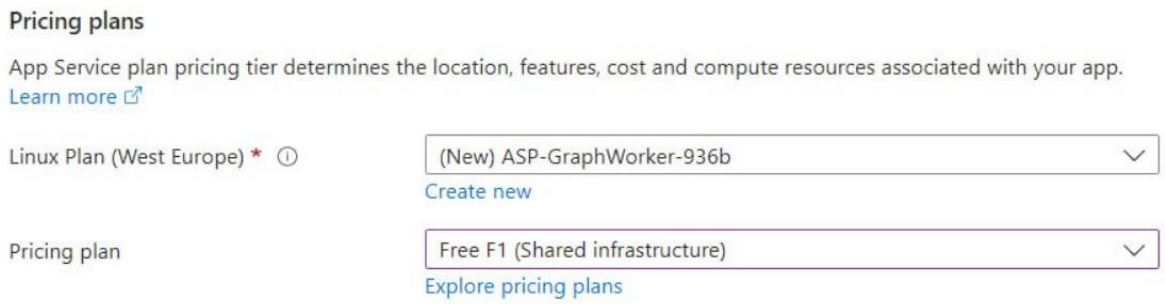

Рисунок 7.4 Вибір тарифного плану для публікації.

Після цього в вкладці *Networking* вмикаємо публічний доступ до веб-сайту, а у вкладці *Monitoring* вимикаємо функцію *Application Insights* для уникнення додаткових грошових стягнень.

Після завершення усіх налаштувань натискаємо кнопку Create і бачимо в результаті ресурси які були створенні в ресурсній групі. Серед них власне сам веб-сайт, сервіс план, який відслідковує щоб ліміти не були перевищені, та віртуальна мережа, яка буде дозволяти декільком сервісам в одній групі мати доступ одне до одного наче через локальну мережу.

Рисунок 7.5 Створені ресурси, які належать до однієї ресурсної групи.

Останнім кроком лишилось опублікувати веб-сайт з репозиторію на GitHub на створеному ресурсі Azure. Переходимо до створеного App Service, і переходимо до пункту *Deployment Center*. Тут у вкладці *Settings* можна вказати джерело з якого власне буде взятий код програми. Надаємо доступ Azure до нашого аккаунту GitHub, вибираємо організацію, репозиторій та гілку з якої буде відбуватись публікація програми та зберігаємо.

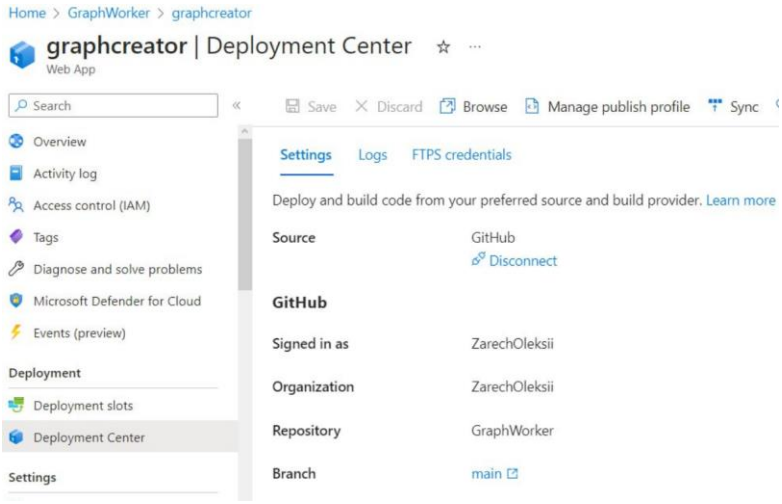

Рисунок 7.6 Поєднання репозиторію та Azure сервісу, яке дозволяє опублікувати веб-сайт.

Після пов’язання репозиторію та Azure ресурсу, будуть додані нові файли до репозиторію. Це файли конфігурації, які Azure створює автоматично. У цьому файлі написана інструкція для того щоб GitHub міг опублікувати веб-сайт. Після додавання цього файлу, за цією інструкцією почнеться опублікування веб-сайту.

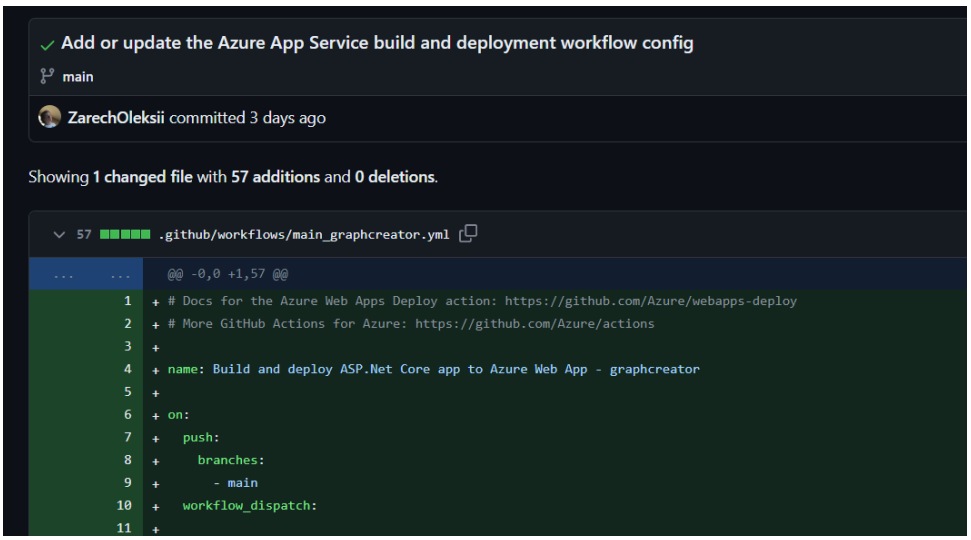

Рисунок 7.7 Файли додані Azure до репозиторію.

Прогрес публікації можна переглянути на самому репозиторії у вкладці *Actions*. Тут можна побачити крок-за-кроком процес побудови програми та публікації побудованої програми. Усе відбувається згідно файлу інструкції вказаного вище.

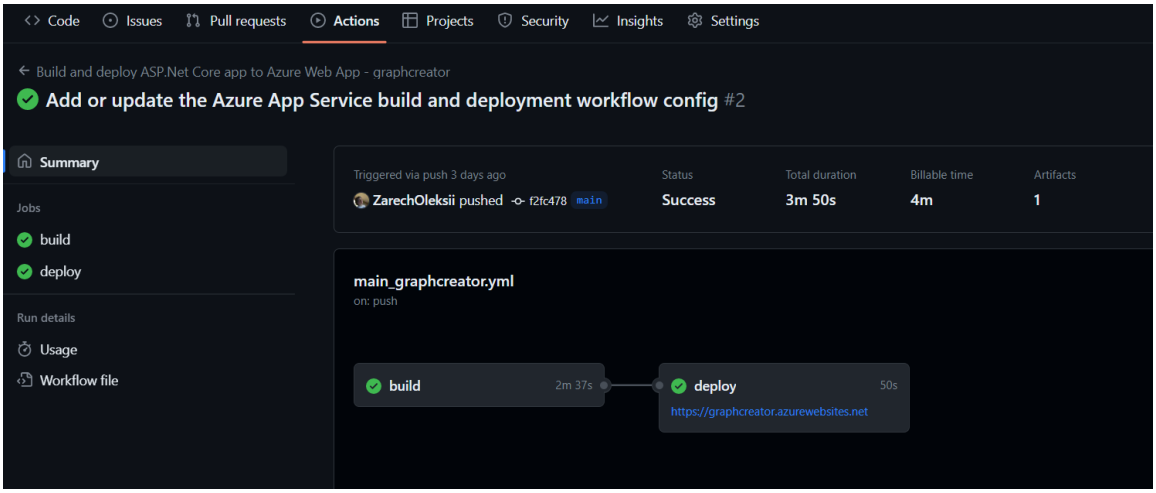

Рисунок 7.8 Прогрес і результат побудови та публікації програми.

За кілька хвилин переходимо по посиланню, яке ми задавали під час створення як ім’я App Service, або його можна знайти натиснувши на пункт *Overview*. Бачимо граф, який з’явився за цим посиланням, яке можна відкрити з будь якого пристрою і він буде відображатись. Публікація веб-сайту у публічний доступ завершена.

ВИСНОВКИ

Метою цієї роботи було створення максимально доступного та зручного інструменту для роботи з графовими моделями, що і було досягнуто. Були виконані також усі поставлені задачі:

- є можливість створювати граф та модифікувати його;
- є можливість переглянути створений граф в інших видах представлення графів, таких як списки суміжності чи різноманітні матриці;
- користувач може користуватись такими найпотрібнішими алгоритмами для роботи з графами як пошук найкоротшого шляху (використовуючи алгоритми Дейкстри та Флойда) і обхід графа (пошук вглиб та вшир);
- веб-сайт опубліковано в публічний доступ за посиланням: https://graphcreator.azurewebsites.net.

В ході роботи було в деталях розібрано кожен з кроків створення веб-сайту, починаючи від вибору технологій до відображення результатів роботи алгоритмів на побудованому графі. Варто зазначити що роботу над веб-сайтом можна продовжити і далі, наприклад додавши можливість збереження зареєстрованим користувачам їх графів або додаючи більшу кількість існуючих алгоритмів, особливо корисно було би для цього використати відгуки реальних користувачів, які використають веб-сайт, проте це виходить за межі поставлених в цій роботі завдань і тут описано не буде.

ПОСИЛАННЯ

1. Нікольський Ю.В. Дискретна математика Підручник / Нікольський, Пасічник, Щербина - Видавнича група BHV, 2007.
2. Chris Caldwell Graph Theory Glossary Онлайн глосарій 1995 – Режим доступу: https://primes.utm.edu/graph/glossary.html.
3. S. Gill Williamson Lists, Decisions and Graphs With an Introduction to Probability Підручник / Williamson, Bender – Режим доступу: https://books.google.com.ua/books?id=vaXv_yhefG8C
4. Документація по .NET Core 6.0 – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-6.0
5. Веб-сайт бібліотеки D3 – Режим доступу: https://d3js.org.
6. Безкоштовні сервіси Azure – Режим доступу: https://azure.microsoft.com/en-us/pricing/free-services
7. Портал Microsoft Azure – Режим доступу: https://portal.azure.com
8. Консоль Google для індексування сайту – Режим доступу: https://search.google.com/u/2/search-console/welcome
9. Веб-сервіс для спільної розробки GitHub – Режим доступу: https://github.com
10. Веб-сервіс для генерації мета тегів Open Graph – Режим доступу: https://www.opengraph.xyz