Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБЛЕННЯ ЗАСТОСУНКУ ДЛЯ ВІДОБРАЖЕННЯ АЛГОРИТМІВ НА ГРАФАХ З ВЕЛИКОЮ КІЛЬКІСТЮ ВЕРШИН
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-19-1
Зарниця Д.Д.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Любченко В.А.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Зарниці Діані Денисівні
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення застосунку для відображення алгоритмів на графах з великою кількістю вершин

затверджена наказом університету від 15 травня 2023 року № 474 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 29 травня 2023 р.
3. Вихідні дані до роботи літературні джерела, матеріали роботи.

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд предметної області «Розроблення застосунку для відображення алгоритмів на графах з великою кількістю вершин».
2. Проєктування системи та матиматичне обґрунтування обраних методів візуалізації.
3. Комп'ютерна модель застосунку для візуалізації алгоритмів на графах.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми розроблення застосунку для відображення алгоритмів на графах, постановка задачі, тестові зображення.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Консультант з дотримання діючих стандартів та норм	Доцент Творошенко І.С.		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	10.04.2023	
2	Аналіз завдання, підбір літератури	12.04.23-15.04.23	
3	Аналіз літератури з досліджуваної проблеми	24.04.23-26.04.23	
4	Аналіз технічних засобів візуалізації графів	27.04.23-05.05.23	
5	Розробка методу візуалізації графів	07.05.23-15.05.23	
6	Програмна реалізація	16.05.23-23.05.23	
7	Оформлення пояснювальної записки	24.05.23-26.05.23	
8	Перевірка на плагіат	27.05.23	
9	Рецензування	28.05.23	
10	Підготовка презентації та доповіді	29.05.23-30.05.23	
11	Занесення роботи в електронний архів	31.05.23	
12	Попередній захист кваліфікаційної роботи	07.06.23	

Дата видачі завдання 10 квітня 2023 р.

Студент _____
(підпис)

Керівник роботи _____ ____доц. Любченко В.А._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 71 с., 13 табл., 29 рис., 1 дод., 30 джерел.

ГРАФ, АЛГОРИТМИ НА ГРАФАХ, ВІДОБРАЖЕННЯ АЛГОРИТМІВ, C++, SFML , VISUAL STUDIO, МЕТОДИ ВІЗУАЛІЗАЦІЇ ГРАФІВ.

Об’єктом роботи є дослідження та реалізація алгоритму графічного відображення графів з подальшою візуалізацією алгоритмів на графах.

Метою роботи є реалізація методів, що базуються на використанні методів графічного відображення об’єктів, які дозволяють коректно відображати графи з великою кількістю вершин з подальшою можливістю аналізу роботи алгоритмів з максимальною точністю відображення їх роботи.

У процесі розробки застосунку були використані сучасні технології, такі як C++, SFML та Visual Studio.

У результаті роботи здійснена програмна реалізація системи для візуалізації алгоритмів на графах з великою кількістю вершин .

GRAPH, GRAPH ALGORITHMS, ALGORITHMS VISUALIZATION, C++, SFML, VISUAL STUDIO, GRAPH VISUALIZATION METHODS.

The object of the work is to study and implement an algorithm for graphical visualization of graphs with subsequent visualization of algorithms on graphs.

The aim of the work is to implement methods based on the use of graphical object display methods that allow to correctly display graphs with many vertices with the subsequent possibility of analyzing the work of algorithms with maximum accuracy of displaying their work.

In the process of developing the application, modern technologies such as C++, SFML and Visual Studio were used. As a result of the work, a software implementation of the system for visualizing algorithms on graphs with a large number of vertices was carried out.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface (прикладний програмний інтерфейс)

2D – двовимірний простір

PCA – Principal Component Analysis (метод головних компонент)

UMAP – Uniform Manifold Approximation and Projection (алгоритм машинного навчання, що виконує нелінійне зниження розмірності)

T-SNE – T-distributed Stochastic Neighbor Embedding (T-розподілене вкладення стохастичної близькості)

BFS – Breadth-first search (пошук у ширину)

DFS – Depth-first search (пошук в глибину)

SFML – Simple and Fast Multimedia Library (проста і швидка мультимедійна бібліотека)

SDL – Simple DirectMedia Layer (кросплатформна мультимедійна бібліотека)

KDE – K Desktop Environment (спільнота, розробляюча вільне програмне забезпечення)

XML – Extensible Markup Language (розширювана мова розмітки)

Web – інтернет-простір

GUI – Graphical User Interface (графічний інтерфейс користувача)

TGUI – Texus Graphical User Interface (графічний інтерфейс користувача Тексус)

JSON – JavaScript Object Notation (запис об’єктів JavaScript)

UML – Unified Modeling Language (уніфікована мова моделювання)

ВСТУП

Теорія графів вивчає математичні функції, які відображають відношення між різними об’єктами. Вона має величезну кількість застосувань, починаючи від комп’ютерних і дорожніх мереж і навіть охоплюючи такі теми, як людські стосунки і лінгвістика. Іншими словами, будь-які дані, які мають зв’язки або відносини, можуть бути представлені у форматі графа.

Актуальність роботи полягає у тому, що безліч проведений досліджень і робіт використовують лише загальне представлення графів, і коли справа доходить до конкретної проблеми, це не завжди так просто, як взяти готове рішення і застосувати його у вашій задачі. У деяких випадках дані графа можуть залежати від інших зовнішніх факторів, і базовий алгоритм повинен бути модифікований, щоб врахувати ці фактори належним чином. Коли справа доходить до складних змін базового алгоритму, добре, якщо є очікування або істина, яка вже встановлена і на яку можна покластися. Але це не завжди так, і модифікації коду можуть призвести до появи нових помилок.

Найпростіший спосіб знайти помилку – це представити дані візуально. Розробка таких застосунків вимагає часу, а витрачати час на реалізацію застосунку замість роботи над реальними проблемами – марна трата часу.

Тому мета цієї роботи – представити за стосунок для візуалізації будь-якого алгоритму на графах, щоб заощадити час на роботу над окремим застосунком і більше зосередитися на вирішенні проблем.

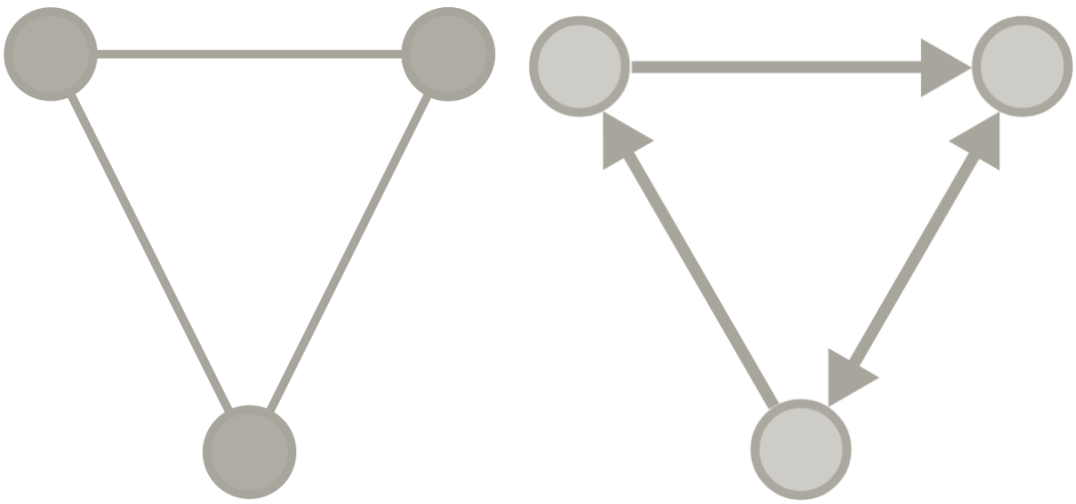
1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ «РОЗРОБЛЕННЯ ЗАСТОСУНКУ ДЛЯ ВІДОБРАЖЕННЯ АЛГОРИТМІВ НА ГРАФАХ З ВЕЛИКОЮ КІЛЬКІСТЮ ВЕРШИН»

1.1 Огляд загальної теорії про графи

1.1.1 Визначення графу

Графом [1, 2] називається пара (V, E), де V – множина об’єктів, які називаються вершинами, а E – множина двоелементних підмножин V, які називаються ребрами. Вершини й ребра графа називаються також елементами графа, число вершин у графі V – порядком, число ребер E – розміром графа. Для кожного ребра можна вказати напрямок від першої інцидентної вершини до другої інцидентної вершини.

Напрямлені ребра зазвичай називають дугами, а граф, що їх містить – орієнтованим (рис. 1.1 а)). У протилежному випадку (вершини з рівними інцидентними ребрами) ми називаємо граф неорієнтованим (рис. 1.1 б)).

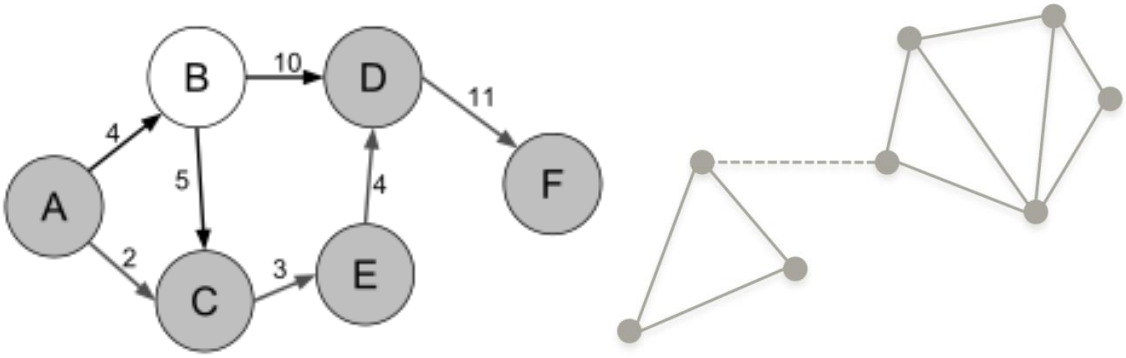


а) б)

Рисунок 1.1 – Види графів за орієнтованістю:
а) неорієнтований граф; б) орієнтований граф

Зважений граф – граф, кожному ребру якого поставлено у відповідність деяке значення (вага ребра) (рис. 1.2 а)).

Зв’язний граф – граф, в якому всі вершини зв’язані між собою (рис. 1.2 б)).



а) б)

Рисунок 1.2 – Види графів:
а) зважений граф; б) зв’язний граф

1.1.2 Способи представлення графу

Одним з основних кроків для розглядаємого застосунку є надання можливості користувачеві побудувати граф, який згодом буде використаний для візуалізації алгоритму. Три найпоширеніші способи представлення графа за допомогою структур даних – це список суміжності, матриця суміжності та матриця інцидентності.

У таблиці 1.1 [2] описані основні відмінності між існуючими підходами. Оскільки очікується, що графи, які будуть оброблятися застосунком, матимуть великий розмір (більше ніж 10000 вершин), дуже важливо сакцентувати увагу на оптимізації збереження даних. З цієї причини список суміжності є найбільш вигідним способом зберігання графів і буде використовуватися для подальшої розробки застосунку.

Таблиця 1.1 – Порівняння структур даних

	Список суміжності	**Матриця суміжності**	**Матриця інцидентності**
Зберігання графу	$O(\|V\| + \|E\|)$	$O(\|V\|^2)$	$O(\|V\| \cdot \|E\|)$
Додавання вершин	$O(1)$	$O(\|V\|^2)$	$O(\|V\| \cdot \|E\|)$
Видалення вершин	$O(\|E\|)$	$O(\|V\|^2)$	$O(\|V\| \cdot \|E\|)$
Перевірка суміжності	$O(\|V\|)$	$O(1)$	$O(\|E\|)$
Примітки	Повільно видаляє вершини та ребра, оскільки йому потрібно знайти абсолютно всі вершини або ребра	Повільно додає або видаляє вершини через необхідність копіювання матриці або зміни розміру	Повільно додає або видаляє вершини та ребра через необхідність копіювання матриці або зміни розміру

1.2 Дослідження доцільності створення візуалізатора алгоритмів на графах

Кожен data scientist та інженер програмного забезпечення, якщо він працює з алгоритмами на графах, рано чи пізно стикається з необхідністю відображати свій алгоритм. Причин для цього може бути багато. Найпоширеніші з них – це пошук несправностей створеного алгоритму та краще представлення даних. У першому випадку все досить просто: відбулося оновлення алгоритму, яке спричинило нову поведінку. Це можуть бути покращення, які просто змінюють спосіб обробки даних, або, з іншого боку,

зміни могли призвести до появи нових помилок. Замість того, щоб використовувати відладчик, який є чудовим інструментом, але у випадку величезних графів неможливо відстежувати великі фрагменти даних. Сприйняття процесів через чисельне представлення набагато складніше графічної візуалізації. Набагато простіший підхід, особливо якщо поведінка значно змінилася – це візуалізувати процес. Це зробить проблему очевидною і заощадить час використання відладчика.

Другий випадок, який є кращим представленням даних, частково перетинається з першим сценарієм. Він допомагає відстежувати, як різні зміни впливають на виконання алгоритму. Але головне – з'ясувати, які саме дані обробляються. По суті, граф – це сирі числа. Їх можна використовувати для обчислення певної статистики та метрик графа, але цього недостатньо, щоб отримати уявлення про структуру даних. У графі є багато структур, таких як кластери та мости, які можна використати для покращення продуктивності програми, скориставшись їхніми перевагами.

Зазвичай для більшості проєктів, пов'язаних з графікою, для вирішення вищезгаданих проблем самостійно розробляється інструмент візуалізатора. Але на їх впровадження та налагодження роботи потрібен час. Напрямом цієї роботи є створення єдиного застосунку, який допоможе дослідникам, розробникам та бізнесу в цілому заощадити час на розробку таких інструментів.

1.3 Аналіз основних сфер використання візуалізації графів

Візуалізація графових алгоритмів використовується в різних галузях і сферах, де розуміння та аналіз складних мереж має вирішальне значення. Області, де візуалізація графових алгоритмів знаходить доволі активне застосування, включають в себе наступні:

– мережевий аналіз. Графові алгоритми та їх візуалізація широко використовуються в мережевому аналізі. Вони допомагають зрозуміти структуру, зв’язки та властивості мереж, таких як соціальні мережі, комп’ютерні мережі (рис. 1.3 [3]), біологічні мережі, транспортні мережі тощо. Візуалізація допомагає визначити ключові вузли, спільноти, центри та закономірності в мережі;

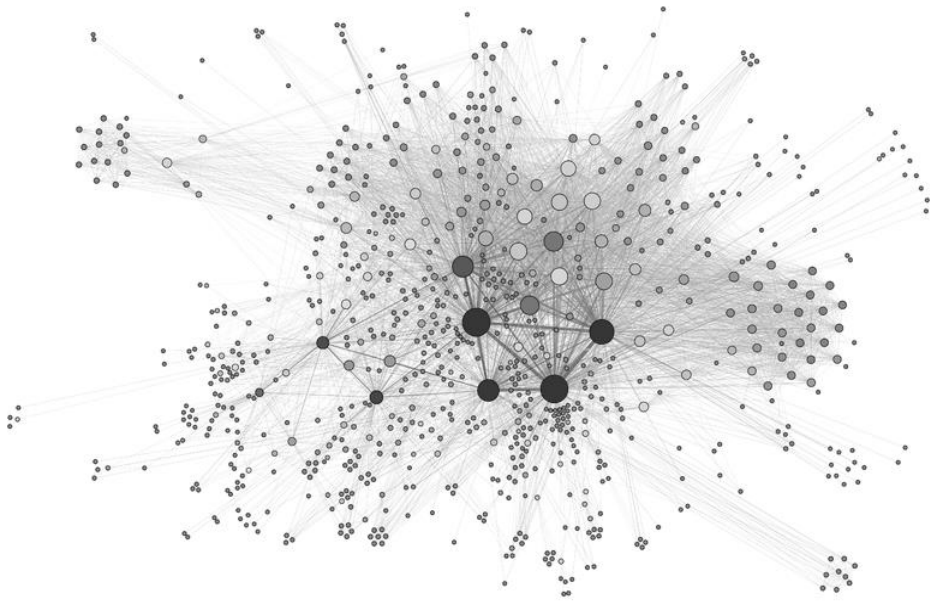


Рисунок 1.3 – Візуалізація графу комп’ютерної мережі

– наука про дані та машинне навчання. Графові алгоритми відіграють важливу роль у задачах науки про дані та машинного навчання. Візуалізація графових алгоритмів допомагає досліджувати та інтерпретувати взаємозв’язки, кластеризацію, класифікацію та завдання прогнозування. Це допомагає визначити відповідні особливості, оцінити продуктивність моделі та пояснити результати моделей машинного навчання на основі графів;

– інженерія програмного забезпечення та аналіз коду. Графові алгоритми та візуалізації застосовуються в інженерії програмного забезпечення для аналізу коду, розуміння програм та візуалізації програмного забезпечення. Вони можуть представляти залежності між модулями коду,

графіки викликів, графіки потоків управління та ієрархії класів, допомагаючи розробникам розуміти та орієнтуватися в складних програмних системах;

– обчислювальна біологія та біоінформатика. Графові алгоритми та візуалізації широко використовуються в обчислювальній біології та біоінформатиці. Вони допомагають аналізувати біологічні мережі, такі як мережі білкових взаємодій (рис. 1.4 [4]), генні регуляторні мережі, метаболічні шляхи та філогенетичні дерева. Візуалізація допомагає зрозуміти біологічні процеси, ідентифікувати функціональні модулі та прогнозувати функції білків;

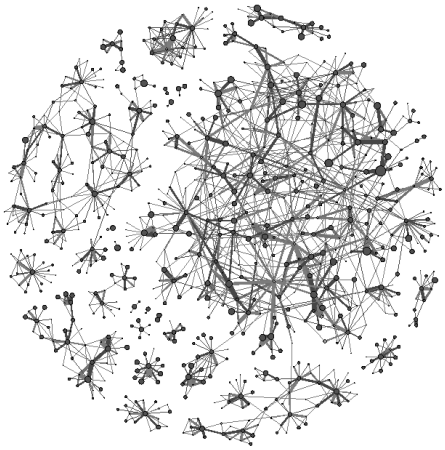


Рисунок 1.4 – Візуалізація графу мережі білкових взаємодій

– аналіз вебсторінок та соціальних мереж. Графові алгоритми використовуються для аналізу вебграфіки та соціальних мереж. Візуалізації можуть допомогти зрозуміти структуру Інтернету, проаналізувати алгоритми аналізу посилань, такі як PageRank, визначити впливові вебсторінки або користувачів, а також виявити спільноти або патерни в соціальних мережах;

– візуалізація даних та візуалізація інформації. Візуалізація графових алгоритмів використовується як окрема техніка у сферах візуалізації даних та візуалізації інформації. Вона дозволяє представляти складні взаємозв’язки даних, шаблони та структури в інтуїтивно зрозумілій та інтерактивній формі. Вона допомагає в дослідницькому аналізі даних та прийнятті рішень.

1.4 Аналіз графічних бібліотек для створення графічного інтерфейсу

Основна мета розробки застосунку – надати користувачам можливість візуалізувати графи та алгоритми на графах. У цьому розділі приведен аналіз існуючих графічних бібліотек для створення візуалізатора. Особлива увага була приділена бібліотекам, які мають інтеграцію з мовою програмування C++, тому що C++ активно використовується для роботи з графами та має доволі високу швидкодію.

1.4.1 SFML

SFML [5] – це об’єктно-орієнтований фреймворк для мови програмування C++. Його філософія полягає в тому, щоб мати простий та зручний інтерфейс прикладного програмування (API), що дозволяє мати як високу продуктивність програмного забеспечення, так і швидку розробку. SFML має відкритий код, що означає, що ви маєте доступ до його повної реалізації.

SFML – це мультимедійна бібліотека, тобто вона забезпечує прошарок між вами та обладнанням. Вона розділена на чотири модулі:

– система. Це основний модуль, на якому будуються всі інші модулі. Він надає класи двовимірних та тривимірних векторів, годинники, потоки, рядки Unicode;

– window. Модуль Window дозволяє створювати вікна застосунків та збирати дані, що вводяться користувачем, такі як рух миші або натискання клавіш;

– графіка. Цей модуль надає всі функціональні можливості, пов’язані з двовимірним рендерингом, тобто можливість використорувати зображення, текст, фігури та кольори;

– аудіо. SFML також пропонує модуль для роботи зі звуком. Якщо є необхідність завантаження музичної теми і відтворення її на динаміках комп’ютера, це саме той модуль, який потрібен.

1.4.2 Qt

Qt [6] – це фреймворк для розроблення кросплатформного програмного забезпечення мовою програмування C++. З часу своєї появи бібліотека лягла в основу багатьох програмних проєктів. Крім того, Qt є фундаментом популярного робочого середовища KDE, що входить до складу багатьох дистрибутивів Linux.

Qt дає змогу запускати написане з його допомогою програмне забезпечення в більшості сучасних операційних систем шляхом простої компіляції програми для кожної системи без зміни вихідного коду. Містить усі основні класи, які можуть знадобитися під час розроблення прикладного програмного забезпечення, починаючи від елементів графічного інтерфейсу і закінчуючи класами для роботи з мережею, базами даних і XML. Є повністю об’єктно-орієнтованим, розширюваним і підтримує техніку компонентного програмування.

1.4.3 SDL

Simple DirectMedia Layer [7] (SDL) – це крос-платформна мультимедійна бібліотека, створена Семом Оскаром Латінга. Вона забезпечує низькорівневий доступ до вводу (через мишу, клавіатуру та геймпадів/джойстиків), 3D-апаратури та буферу 2D-відеокадрів. SDL написана мовою програмування C, але має вбудовану підтримку C++.

SDL має велику базу користувачів і активно оновлюється та підтримується. Існує також також чуйна спільнота та корисний список розсилки. Документація для SDL 2.0 є актуальною і постійно підтримується.

Під час аналізу існуючих графічних бібліотек було закцентовано увагу на таких особливостях, як можливість підтримувати будь-які операційні системи, швидкодія коду та легкість освоювання певної бібліотеки. Серед представлених бібліотек була виділена SFML через відповідність усім вищезазначеним критеріям та наявному досвіду роботи з даною бібліотекою.

1.5 Зберігання та зчитування графів

Найпопулярніший і найзручніший підхід до запису і читання з жорсткого диска використовувався в Chaco. Chaco – це бібліотека інтерактивного 2D візуалізації з відкритим вихідним кодом, яка є частиною набору інструментів Enthought, який, в свою чергу, побудований на основі бібліотеки інтерактивного 2D рисования, сумісної з PyQt, WxPython, Pyglet і VTK. Концепція полягає в зберіганні графів у форматі списку суміжності у файлі. В застосунку буде використано той самий формат, оскільки він легко конвертується у список суміжності.

Задля оптимізації використання пам’яті та швидкодії не був використаний підхід зберігання даних у реляційній базі даних через необхідність створення великої кількості запитів, що у значній мірі б навантажувала систему.

1.6 Аналіз існуючих аналогів, що реалізують функції обраної предметної області

Витративши деякий час на пошук подібних застосунків, не було знайдено аналогів, які б реалізовували ті самі функції. Хоча існує багато наукових досліджень, пов’язаних з проблемами візуалізації графів, і багато хороших програмних застосунків (табл. 1.2 [8, 9]), які реалізують ці роботи, жоден з них не має можливості візуалізації розширених або користувацьких алгоритмів. Найкращі аналоги, які були знайдені, могли відображати лише попередньо визначені алгоритми і, як правило, лише базові (DFS, Дейкстри [2] тощо). Це досить маленький функціонал, який можна використовувати лише в навчальних цілях.

Таблиця 1.2 – Порівняння з аналогами

Назва аналога	**Visualgo**	**Gephi**
1	**2**	**3**
Платформа застосунку	Вебзастосунок	Desktop (Windows/Mac OS/Linux)
Використання власного графу	Додавання у застосунку лише вручну	Так
Візуалізація базових алгоритмів	Так	Ні
Підтримка різних типів візуалізації графів	Ні	Так

Продовження таблиці 1.2

1	2	3
Наскільки інтерфейс є user-friendly	5/5	2/5
Додатки	Не застосовується для великих графіків через підтримку лише ручного введення. Але є доволі гарним навчальним інструментом	Професіональний інструмент візуалізації. Надзвичайно добре підходить для дослідників даних. Але не може забезпечити візуалізацію алгоритмів.

1.6.1 Gephi

Gephi [9] – це програмне забезпечення з відкритим вихідним кодом для візуалізації та аналізу мереж. Воно допомагає аналітикам даних інтуїтивно виявляти закономірності та тенденції, виокремлювати викиди та розповідати історії за допомогою своїх даних. Він використовує механізм 3D-візуалізації для відображення великих графіків у реальному часі та пришвидшення дослідження. Був розроблений студентами французького Технологічного університету Комп'єня у 2009 році. Це інструмент для людей, яким потрібно досліджувати і розуміти графи. Подібно до Photoshop, але для графів, користувач взаємодіє з представленням, маніпулює структурами, формами та кольорами, щоб виявити приховані властивості.

Мета – допомогти аналітикам даних висувати гіпотези, інтуїтивно виявляти закономірності, ізолювати сингулярності структури або помилки під

час пошуку даних. Це додатковий інструмент до традиційної статистики, оскільки візуальне мислення з інтерактивними інтерфейсами тепер визнано як таке, що полегшує міркування. Це програмне забезпечення для дослідницького аналізу даних – парадигми, що з'явилася в галузі візуальної аналітики. Gephi поєднує в собі вбудовані функції та гнучку архітектуру для:

- дослідження;
- аналізу;
- просторової візуалізації;
- фільтрації;
- кластерізації;
- експорту.

Gephi базується на парадигмі візуалізації та маніпулювання, яка дозволяє будь-якому користувачеві відкривати для себе мережі та властивості даних. Більше того, він розроблений для того, щоб прослідкувати ланцюжок тематичного дослідження, від файлу даних до красивих карт, які можна роздрукувати.

1.6.2 Visualgo

Спроєктований у 2011 році доктором Стівеном Халімом, VisuAlgo [8] мав на меті сприяти глибшому розумінню структур даних та алгоритмів його студентами, надаючи інтерактивну навчальну платформу для самостійного опрацювання. Завдяки численним передовим алгоритмам, описаним у книзі доктора Стівена Халіма «Конкурентне програмування», написаній у співавторстві з доктором Феліксом Халімом, VisuAlgo залишається ексклюзивною платформою для візуалізації та анімації деяких з цих складних алгоритмів навіть через десятиліття.

Будучи в першу чергу розробленою для студентів Національного університету Сінгапуру, які навчаються на різних курсах зі структури даних та алгоритмів, VisuAlgo також слугує цінним ресурсом для допитливих умів у всьому світі, сприяючи онлайн-навчанню.

Оснащена вбудованим генератором запитань і верифікатором відповідей, «система онлайн-тестів» VisuAlgo дозволяє учням перевірити свої знання основних структур даних і алгоритмів. Запитання генеруються випадковим чином на основі певних правил, а відповіді студентів автоматично оцінюються після надсилання на наш сервер оцінювання.

Оскільки все більше викладачів комп'ютерних наук у всьому світі використовують цю систему онлайн-тестів, вона може ефективно усунути ручне вивчення базових структур даних та алгоритмів зі стандартних іспитів з інформатики в багатьох університетах.

1.7 Постановка задачі

Таким чином, створення відображення роботи певних алгоритмів на графах є доволі актуальним через відсутність аналогів з таким функціоналом та практичну цінність у такий галузі як освіта. Тому ставиться мета розробки програмного застосунку для відображення роботи алгоритмів на великій кількості вершин.

Об'єктом роботи є дослідження та реалізація алгоритму графічного відображення графів з подальшою візуалізацією алгоритмів на графах.

Метою роботи є реалізація методів, що базуються на використанні методів графічного відображення об'єктів, які дозволяють коректно відображати графи з великою кількістю вершин з подальшою можливістю аналізу роботи алгоритмів з максимальною точністю відображення їх роботи.

Для досягнення мети необхідно вирішити такі завдання:

– провести аналіз існуючих методів графічного відображення графів;

– розробити базові алгоритми на графах для подальшої їх демонстрації;

– розробити користувацький інтерфейс для взаємодії з візуалізатором;

– створити алгоритм завантаження власного алгоритму та графів у застосунок з подальшою візуалізацією;

– реалізувати візуалізацію графів з великою кількістю вершин;

– розробити покрокову візуалізацію алгоритмів на графах;

– створити систему зберігання створеного користувачем графу.

2 ПРОЄКТУВАННЯ СИСТЕМИ ТА МАТИМАТИЧНЕ ОБҐРУНТУВАННЯ ОБРАНИХ МЕТОДІВ ВІЗУАЛІЗАЦІЇ

2.1 Аналіз вимог

Аналіз функціональних та нефункціональних вимог є важливим етапом проєктування програмного застосунку. Функціональні вимоги [10] – це набір функцій, які визначають, як працює система. Кожна функція містить у собі технічні деталі, які визначають, що повинна робити система. Нефункціональними вимогами називаються обмеження, які накладані на застосунок. Вони стосуються таких проблем, як можливість масштабування, продуктивність, портативність, безпека, надійність і багато інших.

Функціональні вимоги:

– можливість зчитування графу з файла;

– покрокова візуалізвція алгоритму на графі;

– будування графу з нуля;

– наявність інструменту для збереження побудованого графу у файл;

– надання алгоритмів BFS та DFS як базових вбудованих алгоритмів для візуалізації;

– інструмент генерації випадкового графу;

– можливість реалізації додаткових алгоритмів на графах;

– впровадження більш просунутих методи візуалізації графів (Force Atlas, Fruchterman-Reingold тощо).

Нефункціональні вимоги:

– час навчання користувача щодо використання візуалізатора не повинен перевищувати 30 хвилин;

– час навчання користувача щодо будування графу у конструкторі графів не повинен перевищувати 5 хвилин;

– застосунок може працювати протягом 12 годин без будь-яких дефектів;

– зчитування даних графу з файлу та відображення його на екрані повинно займати до 5 секунд;

– збереження графу у файл займає менше 5 секунд;

– використання RAM не перевищує 8 гігабайт;

– програма повинна сповіщати користувача у випадку неправильного введення даних;

– користувач повинен мати можливість масштабування та переміщення вікна візуалізації в межах площини.

2.2 Проєктування користувацького інтерфейсу

Інтерфейс користувача [11, 12] складається з 5 основних екранів. Коли користувач входить в систему, відображається головний екран (рис. 2.1).

Рисунок 2.1 – Макет головного меню

В залежності від потреб користувача, за допомогою відповідної кнопки можна відкрити вікно конструктора графіків (рис. 2.2) або вікно візуалізатора (рис. 2.3).

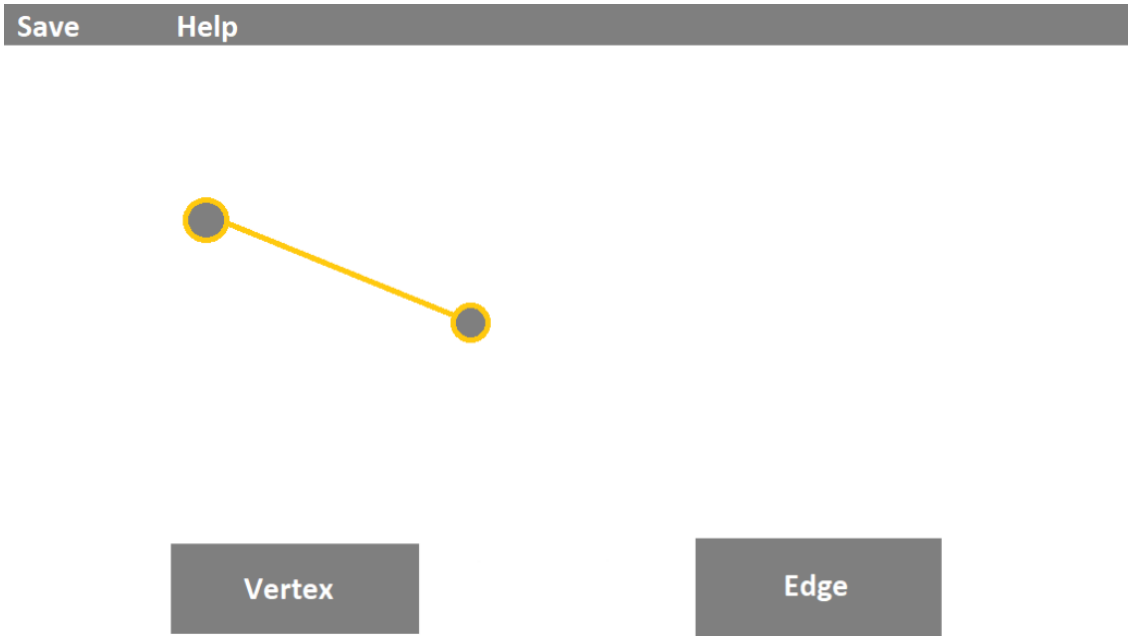


Рисунок 2.2 – Макет вікна конструктора графів

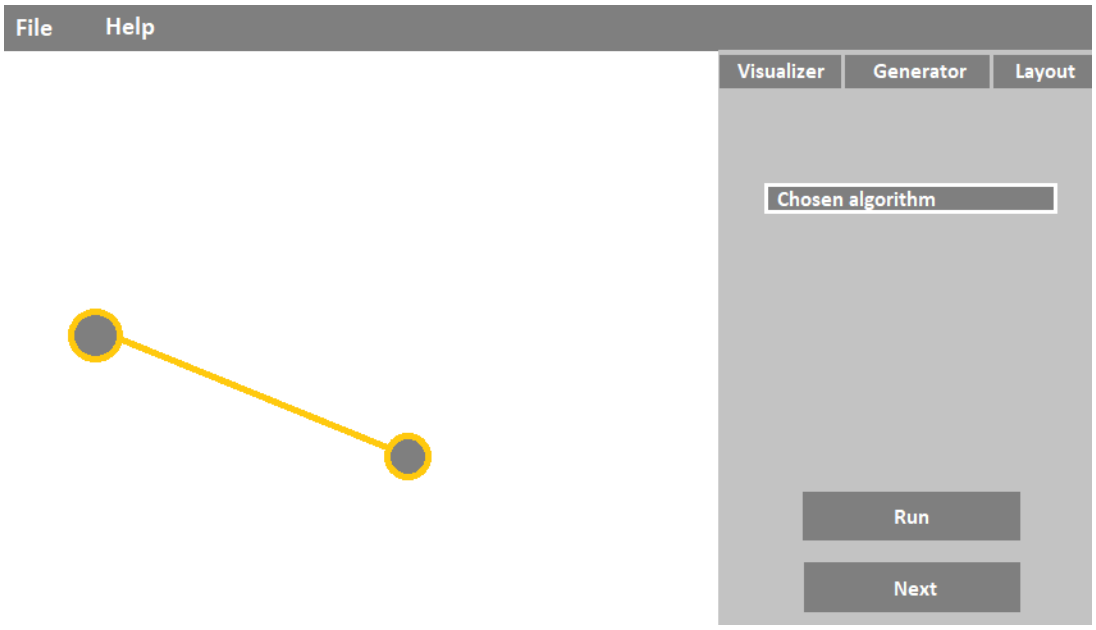


Рисунок 2.3 – Макет вікна візуалізатора алгоритмів

На кожному екрані є кнопка «Довідка». На головному екрані вона відображається кнопкою зі знаком «?». Вона потрібна для того, щоб показати екран з інструкцією користувача (рис. 2.4) для кращої взаємодії з програмою.

Рисунок 2.4 – Макет вікна інструкції користувача

На екрані конструктора графа знаходиться вікно для створення користувачем елементів графу (вершини та ребра). У вікні візуалізатора знаходиться вікно відображення графу, з яким користувач може взаємодіяти у разі масштабування та переміщення графу. За допомогою натискання лівої нижньої кнопки відбувається перехід до наступного кроку роботи алгоритму.

2.3 Аналіз алгоритмів візуалізації графів

Графічне відображення графів [13] саме по собі є складним завданням. Існує безліч алгоритмів і навіть підходів для вирішення цієї проблеми.

Можна виділити три основні способи візуалізації графів:

- силовий підхід;
- підхід зменшення розмірності;
- компонування на основі функцій.

2.3.1 Силовий підхід

Це сімейство методів базується на моделюванні фізичної системи. Вершини представляються як заряджені частинки, що відштовхуються одна

від одної, а ребра розглядаються як пружні струни [14, 15]. Ці методи намагаються змоделювати динаміку цієї системи. Такі методи зазвичай дають дуже хороший результат. Отримані графіки дуже добре відображають топологію графа. Але вони також складні в обчисленнях і мають багато параметрів для налаштування. Найбільш важливими представниками цього сімейства є Force Atlas, алгоритм Фрухтермана-Рейнгольда, алгоритм Камада-Каваї та OpenOrd.

2.3.1.1 Force Atlas

ForceAtlas [16] є представником силового підходу візуалізації, який зазвичай використовується для візуалізації мереж або графів. Головна мета даного алгоритму – розташувати вершини графа таким чином, щоб виявити його базову структуру та зв’язки. Алгоритм використовує сили притягання та відштовхування для досягнення задовільного розташування.

Основні кроки роботи алгоритму ForceAtlas:

Крок 1. Ініціалізація позицій. Алгоритм розпочинає свою роботу з присвоєння випадкових початкових позицій усім вершинам графа.

Крок 2. Обчислення сили відштовхування. Для кожної пари вершин обчислюється сила відштовхування на основі їхніх позицій. Ця сила обернено пропорційна відстані між вузлами, тобто ближчі вузли відштовхують один одного сильніше. Це допомагає розосередити вузли і запобігти їхньому скупченню.

Крок 3. Обчислення сил притягання. Для кожної пари вершин, з’єднаних ребром, обчислюється сила притягання, виходячи з їх положення. Сила прямо пропорційна відстані між вузлами, тобто з’єднані вузли притягуються один до одного. Це допомагає створити згуртованість і зблизити пов’язані вузли.

Крок 4. Оновлення позицій. Алгоритм додає сили притягання і відштовхування, щоб обчислити сумарну силу, що діє на кожен вузол.

Оновлюються позиції вузлів на основі цих сил. В процесі вузли рухаються у напрямку дії сили, але відстань, на яку вони переміщуються, контролюється такими параметрами, як швидкість зближення.

Крок 5. Ітеративний процес. Йде повторення Кроків 2-4 протягом певної кількості ітерацій або до тих пір, поки не буде виконана умова збіжності. На кожній ітерації позиції вузлів оновлюються на основі сил, доки система не досягне відносно стабільного стану.

Крок 6. Додаткове налаштування. Після основних ітерацій деякі реалізації алгоритму застосовують додаткові налаштування або оптимізацію. Вони можуть включати зменшення перекриття вершин або подальше вдосконалення макета на основі певних критеріїв.

Алгоритм ForceAtlas врівноважує сили притягання і відштовхування для досягнення рівноваги, коли вузли рівномірно розподілені і добре розташовані відповідно до їх ребер. Він часто створює естетичні та інформативні макети графів, які можуть допомогти виявити закономірності та структури в мережевих даних. Варто зазначити, що існують різні варіації та реалізації алгоритму ForceAtlas, і деякі з них можуть мати додаткові функції або параметри для налаштування процесу візуалізації.

2.3.1.2 Алгоритм Фрухтермана-Рейнгольда

Алгоритм Фрухтермана-Рейнгольда [17] – доволі популярний алгоритм побудови графів, мета якого полягає у розташуванні вершин графа візуально комфортним виглядом. Основна відмінність між алгоритмом Фрухтермана-Рейнгольда і ForceAtlas полягає у специфічному підході до обчислення сил притягання і відштовхування.

В алгоритмі Фрухтермана-Рейнгольда сили притягання і відштовхування обчислюються наступним чином:

– обчислення сили відштовхування. Для кожної пари вузлів проводиться обчислення сили відштовхування, виходячи з її положення. Сила відштовхування розраховується за наступною формулою:

$$f_r(d) = k^2 / d , \quad (2.1)$$

де $f_r(d)$ – сила відштовхування;

k – константа, що представляє бажану середню довжину ребра;

d – Евклідова відстань між вершинами.

Вершини, які знаходяться ближче один до одного, відчувають сильнішу силу відштовхування, що змушує їх відштовхуватися один від одного;

– обчислення сили притягання. Для кожної пари вершин, з'єднаних ребром, йде обчислення сили притягання, виходячи з її положення. Сила притягання визначається за наступною формулою:

$$f_a(d) = d^2 / k , \quad (2.2)$$

де $f_r(d)$ – сила притягання;

k – константа, що представляє бажану середню довжину ребра;

d – Евклідова відстань між вершинами.

Це означає, що з'єднані вершини, які знаходяться далі один від одного, відчувають сильнішу силу притягання, що змушує їх тягнутися один до одного.

Після розрахунку основних сил виконується обчислення сумарної сили, що діє на кожну вершину. Вершини будуть рухатися у напрямку дії сили, але відстань, на яку вони переміщуються, контролюється коефіцієнтом, який поступово зменшується протягом ітерацій, дозволяючи системі прийти до стабільного стану.

Основна відмінність ForceAtlas від Fruchterman-Reingold полягає в специфічних формулах, що використовуються для розрахунку сил. ForceAtlas використовує більш складний підхід, який включає додаткові параметри, такий як швидкість зближення. Алгоритм Фрухтермана-Рейнгольда використовує простішу формулу, яка фокусується на балансуванні сил притягання і відштовхування на основі бажаної середньої довжини краю.

Обидва алгоритми мають свої сильні та слабкі сторони і можуть створювати різні макети на основі своїх формул.

2.3.1.3 Алгоритм Камада-Каваї

Камада-Каваї [18] – ще один алгоритм силового підходу візуалізації, який розроблен для розташувати вершини графа у візуально привабливий спосіб, враховуючи довжини ребер. Він відрізняється від алгоритмів ForceAtlas та Фрухтермана-Рейнгольда своїм підходом до обчислення розміщення.

Основні особливості та відмінності алгоритму Камада-Каваї:

– оптимізація найкоротшого шляху. Камада-Каваї фокусується на мінімізації загальної довжини найкоротших шляхів між усіма парами вершин у графі. Він враховує не тільки сили притягання та відштовхування, але й фактичну довжину ребер, що з’єднують вершини;

– модель пружинного вбудовування. Алгоритм моделює ребра між вершинами як пружини, з оптимальною довжиною, що визначається початковою довжиною ребра. Таке моделювання дозволяє Камада-Каваї знайти відображення, який точно відповідає заданим довжинам ребер;

– нелінійна оптимізація. Камада-Каваї використовує метод нелінійної оптимізації, точніше алгоритм Ньютона-Рафсона, для ітеративного пошуку положень вершин, які мінімізують функцію напружень. Функція вимірює різницю між фактичною та бажаною відстанню між парами вершин;

– функція напруження. Функція напруження, що використовується в методі Камада-Каваї, кількісно оцінює загальне спотворення макета графа. Це сума різниць між фактичними відстанями та бажаними відстанями для кожної пари з’єднаних вершин. Алгоритм має на меті мінімізувати цю функцію напруження для досягнення оптимальної структури;

– змінний коефіцієнт масштабування. Камада-Каваї вводить масштабний коефіцієнт для контролю сили притягання і відштовхування на основі бажаної довжини ребра. Це дозволяє гнучко налаштовувати макет відповідно до конкретних вимог;

– ітеративне вдосконалення. Алгоритм проходить кілька ітерацій, на кожній з яких уточнюються положення вершин на основі розрахованих сил. Ітерації продовжуються до тих пір, поки не буде виконана умова збіжності.

Підсумовуючи усі переваги та особливості можна зробити висновок, що Камада-Каваї відрізняється від алгоритмів ForceAtlas і Фрухтермана-Рейнгольда своїм акцентом на оптимізації найкоротших відстаней і використанням нелінійного оптимізаційного підходу. Враховуючи фактичні довжини ребер і прагнучи мінімізувати функцію напруження, Камада-Каваї може створювати відображення, які зберігають початкові довжини ребер і показують основну структуру мережі. Однак, він може бути більш трудомістким у порівнянні з іншими алгоритмами через нелінійний процес оптимізації.

2.3.1.4 OpenOrd

OpenOrd [19] – це один з небагатьох алгоритмів візуалізації, який може масштабуватися до понад 1 мільйона вузлів, що робить його ідеальним варіантом для роботи з великими графами. Однак невеликі графи (сотня або менше вершин) не завжди виглядають прийнятно. Цей алгоритм працює з

неорієнтованими зваженими графами і розрахован на покращення процесу розрізняння кластерів.

Алгоритм заснований на методі Фрухтермана-Рейнгольда і працює з фіксованою кількістю ітерацій. Він має п’ять різних фаз. Кожна стадія становить частину від загальної кількості ітерацій, при цьому змінюються декілька параметрів, таких як температура, притягання та демпфування.

Основні характеристики та особливості алгоритму OpenOrd:

– ієрархічна кластеризація. OpenOrd спочатку виконує ієрархічну кластеризацію вершин на основі їх зв’язності. Він використовує варіацію метода Лувена або інші алгоритми виявлення спільнот для визначення груп або кластерів в мережі;

– багаторівневе огрублення. OpenOrd застосовує багаторівневе огрублення до графа. Це передбачає поступове агрегування вершин і ребер для створення грубішого представлення мережі. Цей крок зменшує складність задачі компонування і прискорює обчислення;

– оптимізація енергоспоживання. OpenOrd використовує техніку енергетичної оптимізації для пошуку такого розташування вершин, що мінімізує енергетичну функцію. Енергетична функція визначається на основі відстаней поміж вершинами з метою створення макета, який врівноважує сили притягання та відштовхування;

– багатопотокова реалізація. OpenOrd використовує переваги паралельних обчислень, використовуючи декілька потоків або процесорів для одночасного виконання розрахунків макета. Це дозволяє ефективно обробляти великомасштабні графи і швидше генерувати макети;

– адаптивний графік охолодження. OpenOrd використовує адаптивний графік охолодження під час процесу компонування. Графік охолодження контролює швидкість, з якою система переходить від початкового високотемпературного стану (що дозволяє дотримуватись більшої швидкості руху вершин) до більш низькотемпературного стану (зближення до стабільної

компоновки вершин). Адаптивний графік допомагає збалансувати експлуатацію під час оптимізації компонування;

– високоякісні компонування. OpenOrd здатен створювати високоякісні віщуалізації, які зберігають як глобальну, так і локальну структуру мережі. Алгоритм намагається уникати перекриттів між вузлами та ребрами та виявляти основні патерни зв’язків вершин.

Алгоритм OpenOrd, що поєднує ієрархічну кластеризацію, багаторівневе огрублення, оптимізацію енергоспоживання та багатопотокову реалізацію, робить його особливо ефективним для візуалізації великих, складних мереж. Використовуючи паралельні обчислення та адаптивні стратегії, OpenOrd може ефективно генерувати макети, зберігаючи при цьому важливі структурні властивості мережі.

2.3.2 Підхід зменшення розмірності

Граф можна визначити як матрицю суміжності $N \times N$, де N – кількість вершин. Цю матрицю можна також розглядати як таблицю з N об’єктів у N-вимірному просторі. Таке представлення дозволяє використовувати універсальні методи зменшення розмірності [20, 21], як PCA. Інший спосіб – обчислити теоретичні відстані між вузлами, а потім спробувати зберегти пропорції при переході до простору меншої розмірності.

2.3.3 Компонування на основі функцій

Зазвичай інформація, яка зберігається у графі пов’язана з деякими об’єктами реального світу. Отже, вершини та ребра можуть мати свої особливості. Тому можна використовувати ці особливості для представлення їх на площині. Є можливість працювати з характеристиками вершин, як зі

звичайними табличними даними, використовуючи вищезгадані методи зменшення розмірності або безпосередньо будуючи діаграму розсіювання для пар характеристик. Варто згадати про вуликову діаграму, оскільки вона дуже відрізняється від усіх інших методів. У вуликовій діаграмі вузли вирівнюються за кількома радіальними осями, а ребра вигинаються між ними.

Існують також різні методи візуалізації графів критично великих розмірів (>10 мільйонів вершин), але вони виходять за рамки цієї роботи. Розроблений інструмент повинен принаймні реалізовувати макети на основі функцій, але для просунутих версій інструменту рекомендується підтримка інших методів візуалізацій.

Під час аналізу наявних методів графічного відображення графів було зроблено акцент на необхідність балансу поміж швидкістю роботи алгоритму та якістю отриманого результату. В якості вирогідного алгоритму не будуть розглядатися представники компонування на основі функцій через нестачу необхідних даних. Під час аналізу представників силового підходу було виявлено, що на відміну від алгоритмів Force Atlas та Фрухтермана-Рейнгольда алгоритм Камада-Каваї має нелінійний процес оптимізації, тобто доволі низьку швидкість роботи. Через це даний варіант не буде розглядатися як потенційний алгоритм візуалізації. Алгоритм OpenOrd має доволі складну релізацію та має особливість утворювати доволі великі проміжки між вершинами. Force Atlas має доволі велику чутливість до початкових параметрів, що також не робить його найкращим претендентом. Алгоритм Фрухтермана-Рейнгольда відповідає усім поставленим вимогам. Він доволі простий в реалізації, швидкий та точний на великих та маленьких графах та має доволі високу швидкість роботи.

2.4 Аналіз базових алгоритмів для візуалізації на графах

Додатковий спосіб використання інструменту – швидка перевірка базових властивостей графів, таких як зв’язність. Для цього мають бути реалізовані деякі базові алгоритми роботи з графами. Задля найбільш зрозумілих прикладів були обрані алгоритми пошуку в ширину та пошуку в глибину.

2.4.1 Пошук в ширину

Пошук в ширину (BFS) [22] – це алгоритм обходу графа, який використовується для систематичного дослідження та обходу вершин графа. Він працює вшир, тобто відвідує всі сусідні вершини на поточному рівні глибини, перш ніж перейти до вершин на наступному рівні глибини. BFS є фундаментальним алгоритмом в теорії графів і має різні застосування в інформатиці, мережевому аналізі та інших галузях.

BFS починає із заданої початкової вершини і досліджує її сусідів. Потім він переходить до сусідів сусідів, переконуючись, що всі вершини на заданому рівні глибини відвідані, перш ніж перейти до вершин на наступному рівні. Таке дослідження в ширину гарантує, що BFS знайде найкоротший шлях від початкової вершини до всіх доступних вершин. Він використовує структуру даних First-In-First-Out (FIFO) для відстеження вершин, які потрібно дослідити, що робить його ефективним алгоритмом для обходу графа.

Кроки роботи алгоритму BFS:

Крок 1. Ініціалізація черги та позначення початкової вершини як вже відвіданої.

Крок 2. Постановка початкової вершини у чергу.

Крок 3. Видалення вершини з початку черги.

Крок 4. Опрацювання вершини.

Крок 5. Постановка у чергу всіх невідвіданих сусідів поточної вершини і позначеня їх як відвідані.

Крок 6. Якщо у графі все ще залишаються невідвідані вершини, йде обирання одної з них як нову вихідну вершину. Після цього йде повторення Кроків 2 та 3.

Крок 7. Повторення Кроків 4-6, поки черга не стане порожньою.

Крок 8. Якщо всі вершини було відвідано, алгоритм BFS завершає роботу.

BFS має декілька характеристик і властивостей, які роблять його потужним алгоритмом:

– складність. Часова складність BFS становить $O(V + E)$, де V – кількість вершин (вузлів), а E – кількість ребер у графі;

– обхід в порядку рівня. BFS досліджує вершини в порядку рівня, забезпечуючи систематичне дослідження вершин на кожному рівні глибини;

– найкоротший шлях. BFS гарантує знаходження найкоротшого шляху в незважених графах, що робить його цінним для задач пошуку шляхів;

– використання пам’яті. BFS потребує додаткової пам’яті для відстеження відвіданих вершин та черги. Звичайна складність складає $O(V)$, де V – кількість вершин;

– реалізація. BFS може бути реалізовано як ітеративним підходом з використанням черги, так і рекурсивним підходом з використанням викликів функцій.

Застосування пошуку в ширину варіюється від пошуку найкоротших шляхів до тестування двозв’язності графів та аналізу мереж. Завдяки своїй ефективності, здатності знаходити найкоротші шляхи та універсальності, BFS слугує важливим інструментом у різних галузях.

2.4.2 Пошук в глибину

Пошук у глибину (DFS) [22] – це алгоритм обходу графа, який використовується для систематичного дослідження та обходу вершин графа. Він працює в глибину, тобто досліджує якомога далі вздовж кожної гілки перед тим, як повернутися назад. DFS є фундаментальним алгоритмом в теорії графів і має різні застосування в інформатиці, штучному інтелекті та інших галузях.

DFS починає роботу із заданої початкової вершини і досліджує кожну гілку якомога глибше, перш ніж повернутися назад. Він проходить граф так далеко, як тільки може, досліджуючи вершини вглиб, поки не досягне вершини, яка не має невідвіданих сусідів. Потім він повертається до останньої відвіданої вершини з недослідженими сусідами і продовжує дослідження, доки не буде відвідано всі вершини або не буде виконано потрібну умову. DFS може бути реалізований як ітераційним підходом зі стеком, так і рекурсивним підходом з використанням викликів функцій.

Кроки роботи алгоритму DFS:

Крок 1. Ініціалізація стека та позначення початкової вершини як вже відвіданої.

Крок 2. Переміщення початкової вершини у стек.

Крок 3. Витягніть вузол з вершини стека.

Крок 4. Опрацювання вершини.

Крок 5. Переміщення усіх невідвіданих сусідів поточної вершини до стеку і позначення їх як відвіданих.

Крок 6. Якщо у графі ще залишилися невідвідані вершини, виконується обрання однієї з них як нову вихідну вершину і йде повторення Кроків 2 і 3.

Крок 7. Повторення Кроків 3–6, поки стек не стане порожнім.

Крок 8. Якщо всі вершини було відвідано, DFS завершає роботу.

DFS має широкий спектр застосувань:

– обхід графа. DFS дозволяє обходити граф і відвідувати всі його вершини, досліджуючи їх вглиб. Це корисно для дослідження або пошуку в графах, визначення пов’язаних компонентів і виявлення циклів;

– топологічне сортування. DFS можна використовувати для топологічного сортування орієнтованих ациклічних графів (DAG), яке впорядковує вершини на основі їх залежностей;

– пошук сильно зв’язаних компонентів. DFS допомагає знаходити сильно зв’язані компоненти в орієнтованих графах, які є групами вершин, де кожна вершина досяжна з будь-якої іншої вершини в групі;

– розв’язання лабіринтів. DFS можна використовувати для розв’язання лабіринтів або навігації складними шляхами, досліджуючи можливі шляхи та повертаючись назад, коли ви потрапляєте в глухий кут;

– виявлення циклів. DFS може виявляти цикли на графіку, вказуючи на наявність циклів або повторюваних шаблонів у даних.

3 КОМП'ЮТЕРНА МОДЕЛЬ ЗАСТОСУНКУ ДЛЯ ВІЗУАЛІЗАЦІЇ АЛГОРИТМІВ НА ГРАФАХ

3.1 Обґрунтування вибору середовища програмної реалізації

Однією з причин розробки програми було спрощення налагодження різних реалізацій алгоритмів. Оскільки розробники можуть використовувати різні операційні системи, буде правильним рішенням зробити програмне забезпечення крос-платформним.

Проєкт буде мати справу з величезними обсягами даних і має багато можливостей для розширення та вдосконалення в майбутньому. Тому вимоги до мови включають високу продуктивність і бажано, щоб вона була об'єктно-орієнтованою, щоб спростити впровадження нових функцій. C++ ідеально відповідає цим вимогам.

Оскільки стандартна бібліотека C++ не надає жодного інструменту для роботи з графікою, для цього було обрано SFML [5]. Ця бібліотека використовується для створення вікна відображення та для більшості графіки. Також бібліотека підтримує декілька платформ, що є важливим для такого інструменту. Список підтримуваних платформ включає Windows, Linux, macOS, і навіть частково підтримується Android.

Тим не менш, продукт буде з відкритим вихідним кодом. Такий підхід має багато переваг:

– користувачі з усього світу можуть зробити свій внесок у проєкт;

– нескінченний механізм зворотного зв'язку як щодо деталей реалізації, так і щодо якості коду;

– дозволяє кінцевим користувачам модифікувати програму, якщо чогось не вистачає, що вони хотіли б використовувати.

3.2 Дизайн сховища даних

Розроблене програмне забезпечення не містить жодних баз даних, оскільки вони не застосовуються в цих сценаріях. Замість них у цьому підрозділі будуть описані формати обміну даними та рішення, що використовуються для цього.

Як було зазначено раніше, існує два основних джерела даних, які використовує програма. Це граф, представлений у текстовому форматі, та кроки візуалізації.

3.2.1 Представлення графу в текстовому форматі

Граф зберігатиметься у форматі представлення графів на основі METIS, який буде розширено, оскільки очікується, що програмне забезпечення, розроблене в цій роботі, охоплюватиме всі можливі типи графіків. Можливі варіанти типів графів, які будуть підтримуватися:

- спрямований/не спрямований;
- зважений;
- зв’язний.

І хоча більшість типів графів було покрито в процесі створення моделі застосунку, все ще існують типи графів (наприклад, мультиграфи), які не будуть розглянуті в цій роботі, але стануть чудовим доповненням функціоналу для подальшого розвитку інструменту.

Граф $G = (V, E)$ з N вершинами зберігається у звичайному текстовому форматі у файлі з $N + 1$ рядка. Перший рядок містить два цілих числа N та F, які задають номер вершини та формат графа.

Оскільки існує декілька можливих комбінацій типів графів, було прийнято рішення використовувати стандартний спросіб для зберігання цієї інформації в одному цілому числі, де кожен біт вказує чи використовується

певна ознака, чи ні. Наприклад, якщо один атрибут має значення 0x08, а інший 0x02, щоб вказати, що граф буде використовувати обидва атрибута, до цих значень застосовується побітове АБО, і отримане число (10 в даному випадку) буде збережено як тип графа. Всі проводимі обчислення не будуть продемонстровані користувачеві задля збереження простого інтерфейса.

Якщо ребра графа мають певну вагу, то значення повинно містити 0x01, якщо вершини мають вагу – 0x02, якщо положення вершин можна задати через координати (може використовуватися у випадку макетів на основі функцій, а також для відновлення положень вершин після етапу побудови графа) – 0x04, для орієнтованих графів – 0x08.

Решта *N* рядків файлу зберігають інформацію про фактичну структуру графа. Зокрема, *i*-й рядок містить інформацію про *i*-ту вершину. Залежно від значення *F*, інформація, що зберігається у кожному рядку, дещо відрізняється. У найбільш загальному вигляді (при $F = 15$) кожен рядок має наступну структуру:

$$c\ x\ y\ v1\ w1\ v2\ w2\ \ldots, \tag{3.1}$$

де c – вага вершини;

x, y – відповідні координати на площині;

v_i – вершина, суміжна з поточною;

w_i – вага ребра, що веде до суміжної вершини.

3.2.2 Збереження кроків візуалізації алгоритму та графів

Візуалізація містить інформацію про кожен крок процесу роботи алгоритма. Наприклад, підсвічування вершин. Звичайно, деякі алгоритми вимагають більш складних моментів для візуалізації, таких як рівні розбиття,

але більшість алгоритмів не потребують цього, і наданої функціональності буде більш ніж достатньо.

Збереження власного алгоритму чи графу відбувається за допомогою окремої бібліотеки, що підключається до проєкту користувача. За допомогою методів висвітлення прохідних вершин та ребер впродовж усіх ітерацій алгоритму та генерації окремого файлу з даними відбувається збереження персонального алгоритму на графі для подальшого використання у створеному візуалізаторі. Одночасно йде збереження графу з використанням методу збереження вершин та ребер за аналогічним алгоритмом .

Для покращення користувацьких можливостей застосунок надає можливість об’єднувати декілька кроків візуалізації в групи. Причина цього досить проста: якщо для спрощення використовувати BFS як алгоритм для візуалізації, іноді немає необхідності розглядати кожну перевірку, а достатньо буде спостерігати за станом графа лише після кожної ітерації алгоритму. Однією з причин, чому JSON було обрано замість інших існуючих підходів до зберігання даних, є те, що він дозволяє серіалізацію користувацьких об’єктів, а також підтримує масиви, що буде мати критичне значення в даному випадку. Приклад збереження базового алгоритму та збереження його кроків роботи представлені у додатку А.

3.3 Програмна реалізація застосунку

UML-діаграми [23, 24] є важливими для проєктування програмного забезпечення, оскільки вони допомагають візуалізувати, як працює система, навіть не заглядаючи в код. Загальна діаграма представлена на рисунку 3.1. Оскільки проєкт є досить складним і на представленій діаграмі важко побачити всі деталі, кожна частина програмного забезпечення буде розглянута окремо [25].

Початковою точкою програми є клас Application (рис. 3.2). Він містить головний цикл, обробляє частоту кадрів та делегує події користувача іншим класам [26]. Клас Window збирає події користувача, управляє рухом камери і всім, що пов’язано з оновленням і відображенням головного вікна програми. Щоб зберегти екран користувача чистим, програма працює в одновіконному режимі і створює додаткові вікна лише для того, щоб повідомити користувача про неправильне використання програми. StateManager дозволяє переводити програму з одного стану в інший.

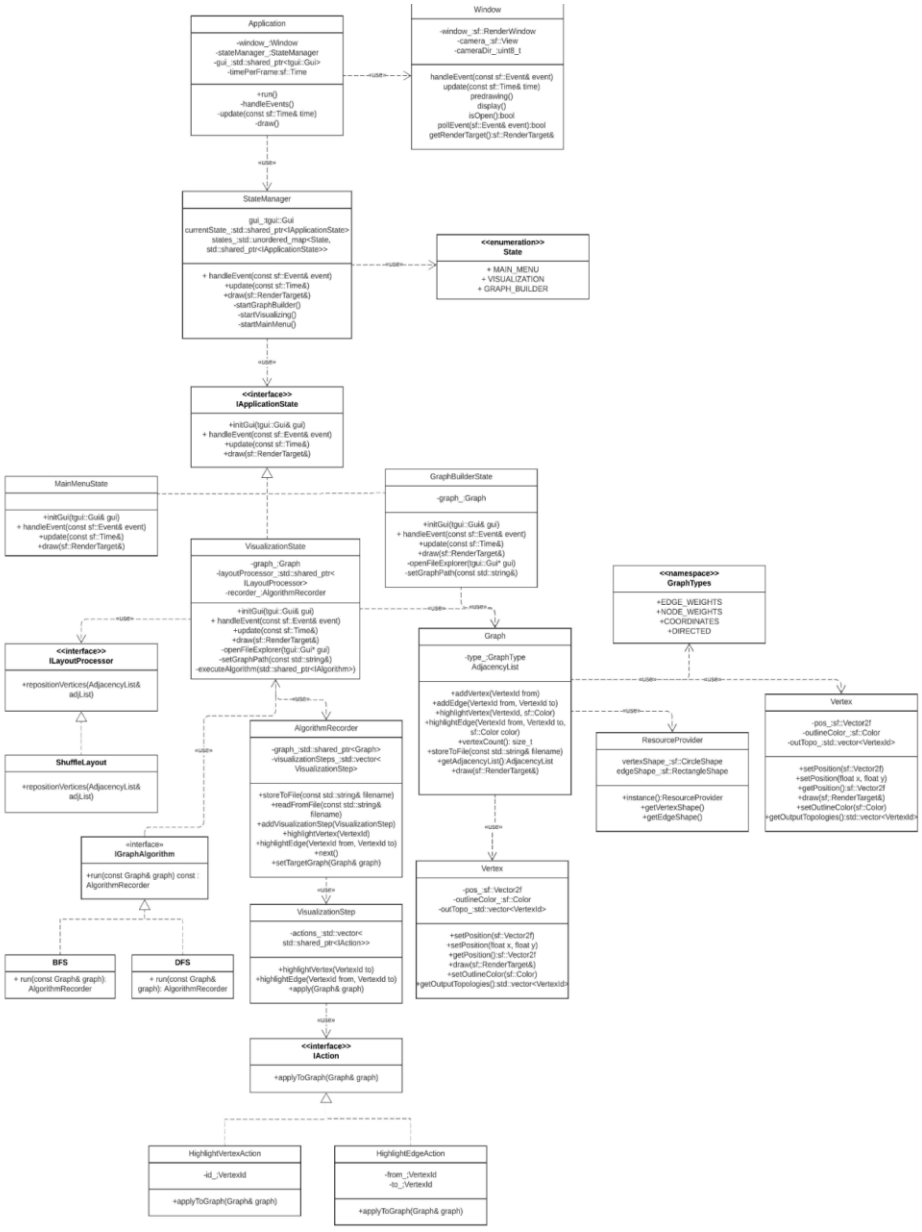


Рисунок 3.1 – UML діаграма застосунку

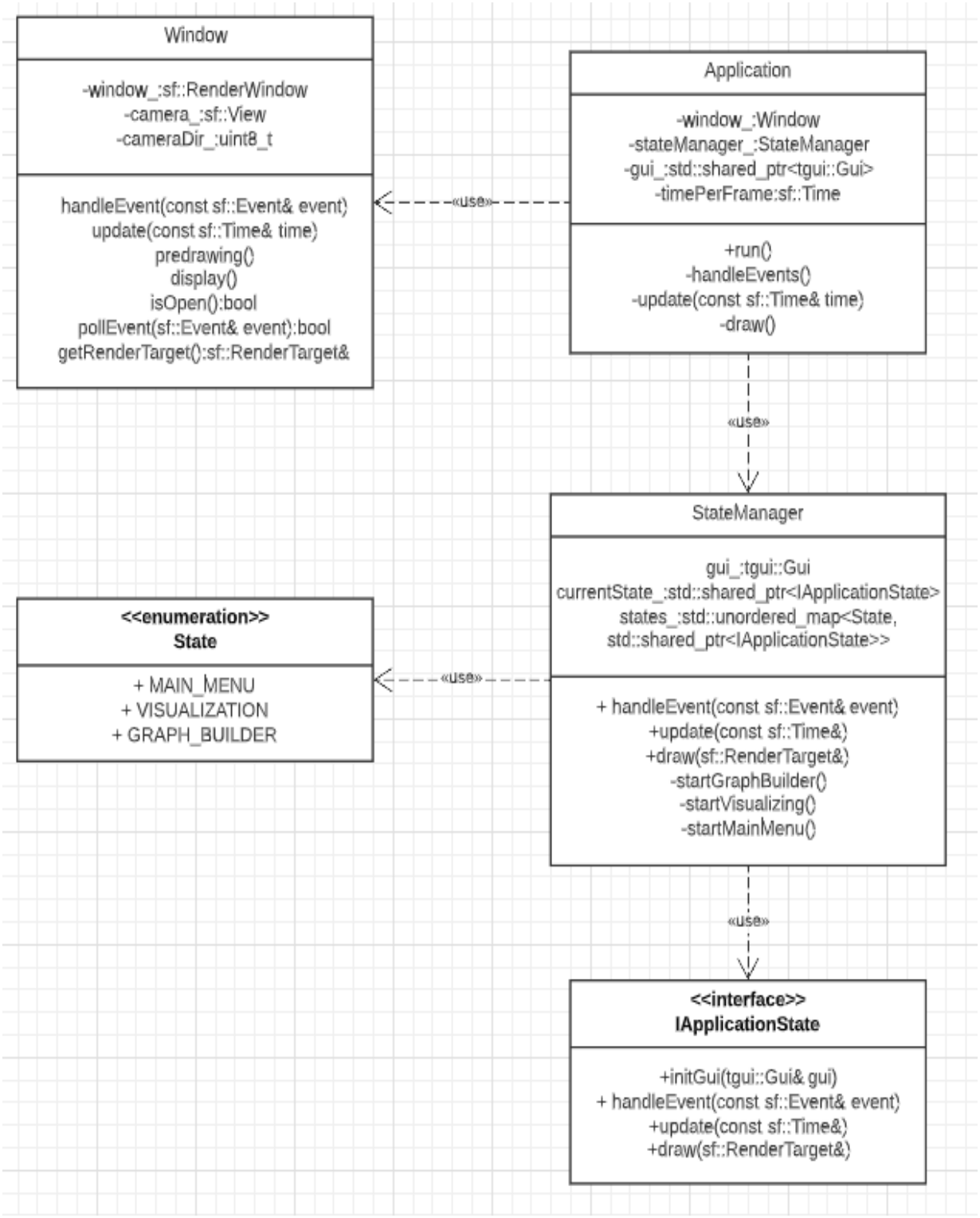


Рисунок 3.2 – UML діаграма базових класів

Існує три основні стани (рис. 3.3), які відповідають трьом основним екранам застосунку:

– MainMenuState – це початковий екран, який дозволяє користувачеві вибрати, чи хоче він запустити візуалізатор або конструктор графіків;

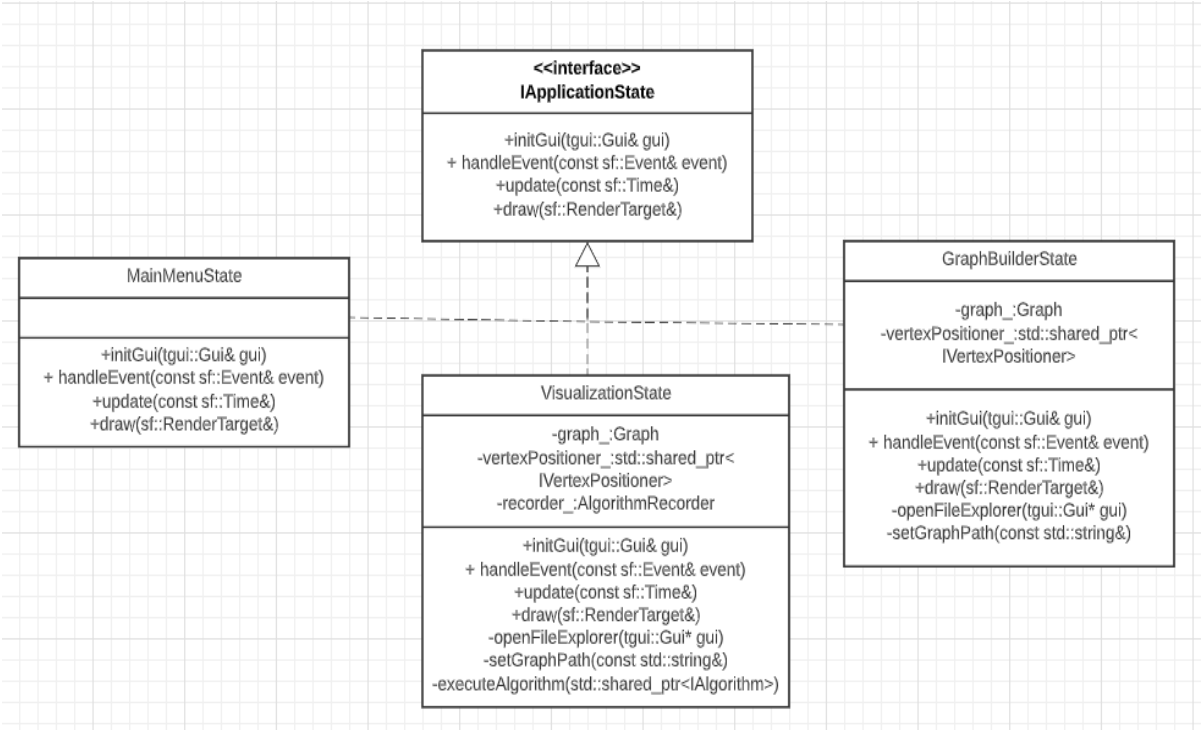


Рисунок 3.3 – UML діаграма станів застосунку

– GraphBuilderState (рис. 3.4) – екран, який дозволяє користувачеві вручну побудувати графік. Він має декілька режимів, таких як «Рука», «Додати вершину», «Додати ребро». Набагато простіше побудувати графік за допомогою інструменту користувацького інтерфейсу, ніж записувати необроблені значення у звичайний текстовий файл;

– VisualizationState (рис. 3.5), який є найскладнішою частиною програми. Він включає в себе алгоритми компонування графів, а також приклади алгоритмів для демонстрації роботи програми і дозволяє візуалізувати алгоритм, наданий користувачем.

Інтерфейс IGraphAlgorithm використовується для попередньо визначених алгоритмів у програмі. Наприклад, як показано на рисунку 3.6, наведено два базових алгоритми, які було реалізовано з метою тестування програмного забезпечення. Обидва вони реалізують метод run(), який виконує алгоритм, використовуючи наданий граф, і повертає дані для процесу візуалізації.

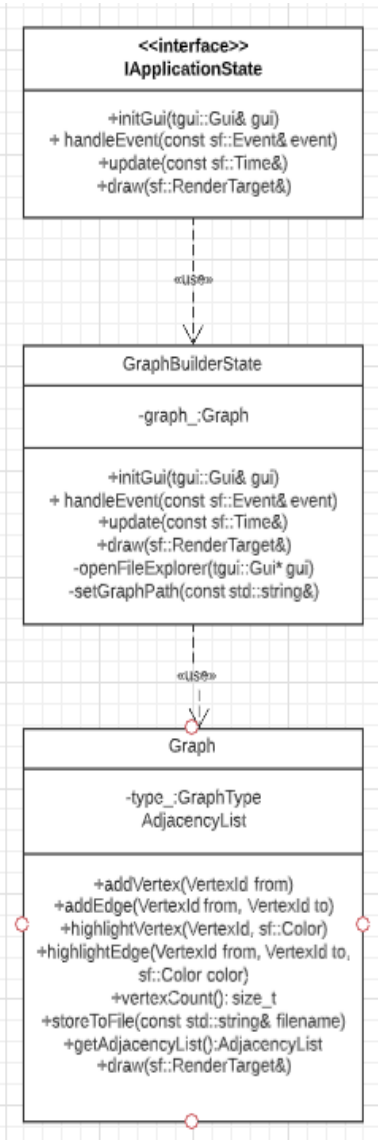


Рисунок 3.4 – Стан побудови графу

Клас Graph містить список суміжності, який по суті є вектором з вершин (рис. 3.7). Кожна вершина містить список вихідних топологій або, іншими словами, всіх суміжних вершин, а також містить колір, що відповідає цій вершині.

ResourceProvider – утиліта, яка надає форми для вершин і ребер. Такі класи необхідно мати для великого проєкту згідно з принципом єдиної відповідальності [27].

AlgorithmRecorder зберігає та допомагає відтворити процес візуалізації на графах (рис. 3.8). Цей клас також передбачений в застосунку для того, щоб користувачі могли створювати дані про кроки алгоритму.

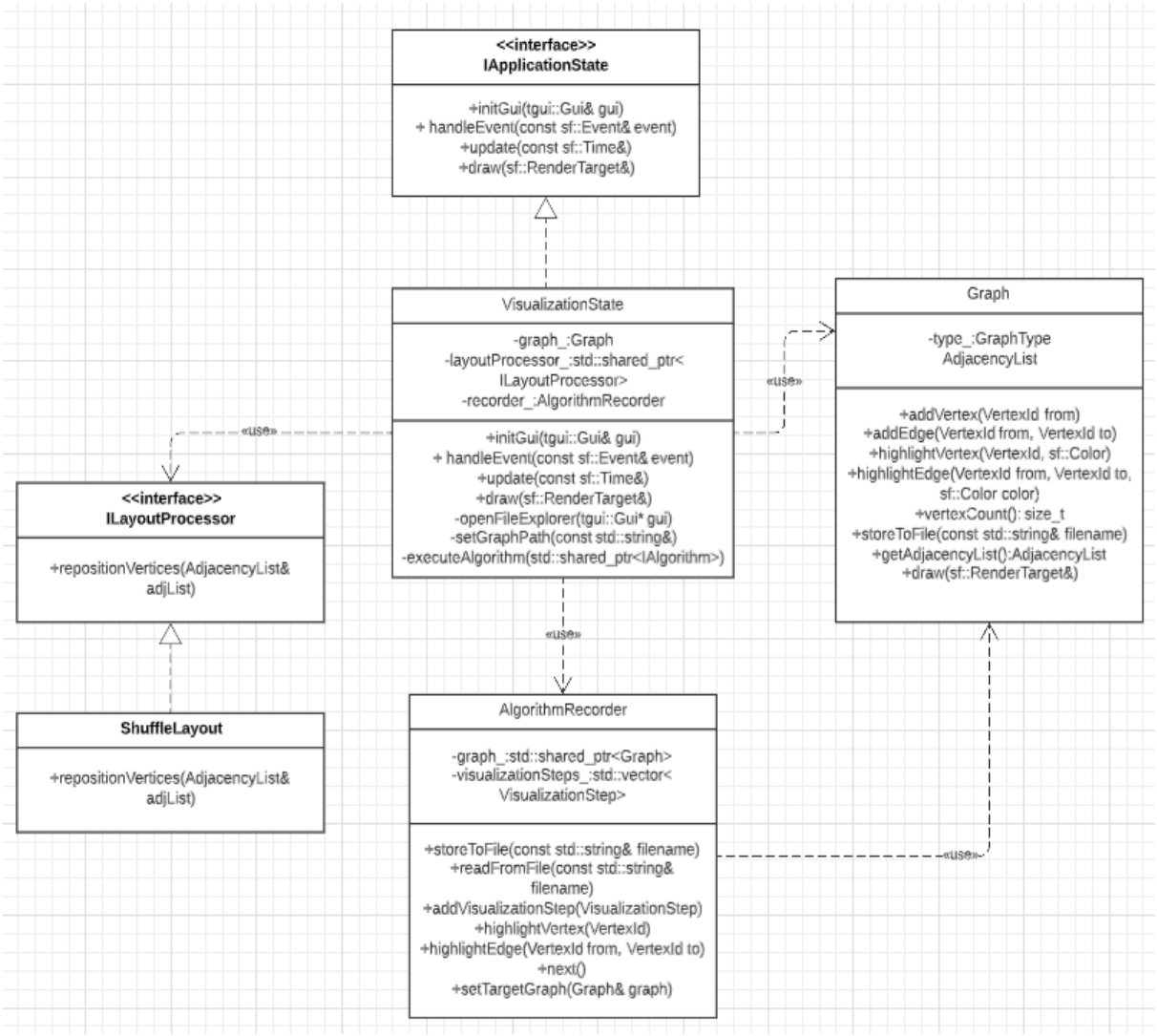


Рисунок 3.5 – Стан візуалізації графу

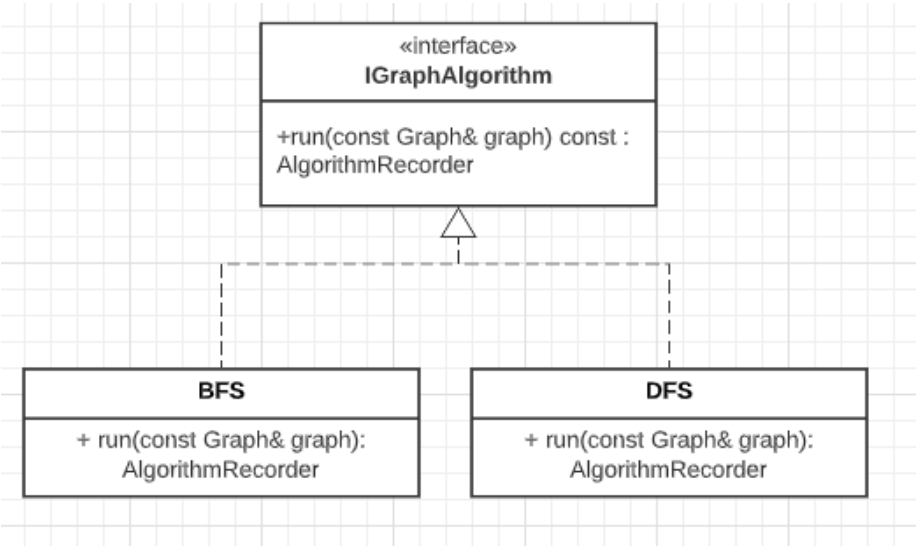


Рисунок 3.6 – Будова базових алгоритмів на графах

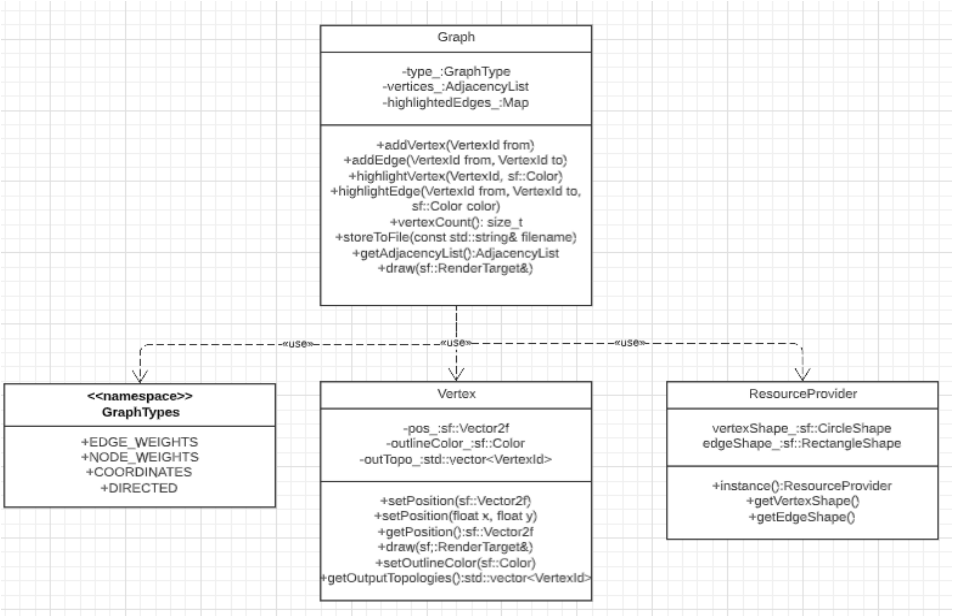


Рисунок 3.7 – UML діаграма класу Graph

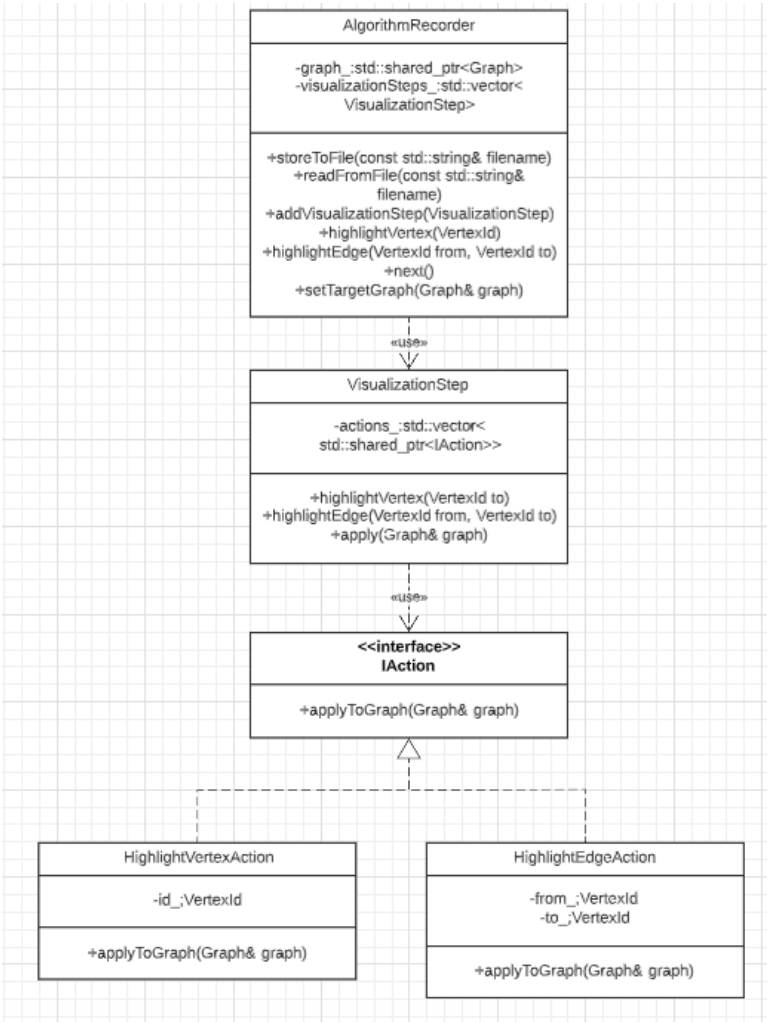


Рисунок 3.8 – UML діаграма класу записувача алгоритму

3.4 Тестування програмного забезпечення

Згідно з [28, 29] існує декілька стратегій тестування:

– інтеграційні тести. Дозволяють об’єднати декілька модулів для перевірки того, що інтерфейсна взаємодія працює так, як очікується. Існує кілька підходів до реалізації інтеграційних тестів, але всі вони об’єднані спільною ідеєю заміни макетів і заглушок даних, що використовуються на етапі модульного тестування, на реальні класи, які будуть використовуватися в застосунку;

– модульні тести. Цей тип тестів перевіряє, що кожен блок програми працює належним чином, відокремлено від оточення. Будь-які залежності в коді повинні бути імітовані, щоб зробити код повністю ізольованим. Правильно написані модульні тести зменшують простір пошуку проблеми до одного блоку (або навіть методу), що робить ціну помилки мінімально можливою;

– системне тестування. На цьому етапі вмикаються всі компоненти системи і система тестується за допомогою реальних запитів, які проходять від початку до кінця програмного забезпечення. На цьому етапі повинні бути виявлені всі проблеми безпеки та сумісності.

3.4.1 Створення функціональних вимог до тестування програмного застосунку

В процесі розробки програмного застосунку тестування [30] проводилося вручну відповідно до наступних функціональних вимог:

а) зчитування графа з файлу:

1) переконатися, що записані дані графіка будуть успішно зчитуватися та відображатися;

2) у випадку неправильно записаного або пошкодженого файлу, користувач повинен бути повідомлений про те, що файл не відповідає вимогам до формата;

б) можливість візуалізації алгоритмів, наданих користувачем:

1) переконатися, що візуалізація виконується належним чином;

2) повідомлення повинно відображатися у випадку, якщо було надано файл з невірними даними;

в) можливість будування графів за допомогою конструктора графів;

г) перевірити відсутність проблем зі зберіганням створеного користувачем графу;

д) надані базові алгоритми для інструменту візуалізації:

1) перевірити наявність алгоритмів BFS та DFS;

2) обидва алгоритми повинні перевіряти вхідні дані та попереджати користувача у випадку невірного введення даних;

е) перевірити, що граф може бути згенеровано із заздалегідь визначеною кількістю вершин та ребер;

ж) переконатися, що алгоритм Фрухтермана-Рейнгольда працює належним чином;

з) перевірити роботу вікна допомоги користувачеві.

3.4.2 Тестування створеного застосунку

Тестовий приклад №1 (табл. 3.1).

Мета: перевірити, що дані графіка, які коректно зберігаються у файлі, можуть бути прочитані та відображені належним чином.

Вимоги до тесту: а.1.

Передумови: система перебуває у стані візуалізації.

Критерій прийнятності: очікування відповідають фактичним результатам.

Таблиця 3.1 – Тестовий план: «Зчитування коректних даних з файлу»

№	Опис етапу	Очікування	Фактичний результат	Критерій прийнятності
1	Відкрити провідник, натиснувши Файл=> Завантажити	Відкрито файловий провідник	Відкрито файловий провідник	Прийнято
2	Вибрати потрібний файл у відкритому файловому провіднику	Провідник файлів закрито, а граф відображено на екрані	Провідник файлів закрито, а на екрані відображено граф (рис. 3.9)	Прийнято

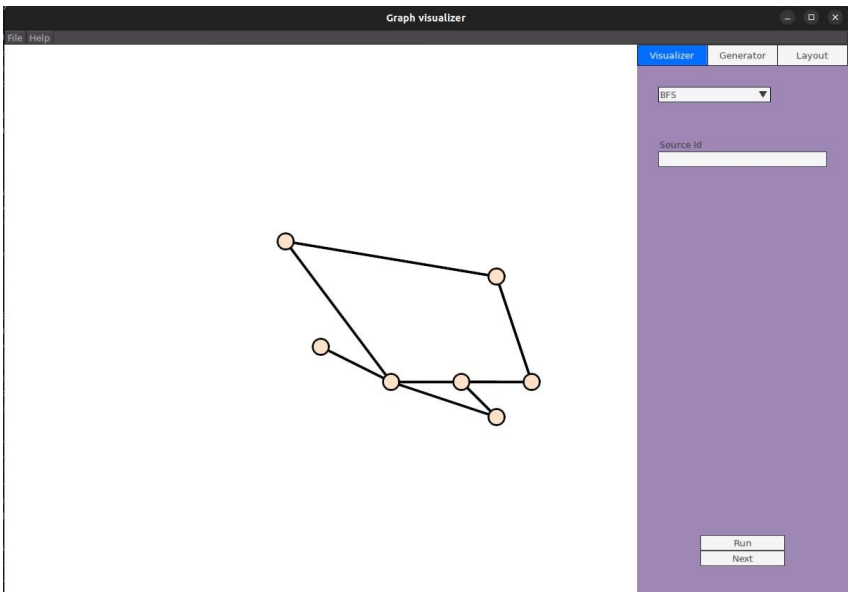


Рисунок 3.9 – Граф зчитано успішно

Тестовий приклад №2 (табл. 3.2).

Мета: переконатися, що користувач отримає сповіщення у випадку пошкодження графічного файлу.

Вимоги до тесту: a.2.

Передумови: система знаходиться у стані візуалізції.

Критерій прийнятності: результат відповідають очікуванням.

Таблиця 3.2 – Тестовий план: «Зчитування некоректних даних з файлу»

№	**Опис етапу**	**Очікування**	**Фактичний результат**	**Критерій прийнятності**
1	Відкрити провідник, натиснувши Файл=> Завантажити	Відкрито файловий провідник	Відкрито файловий провідник	Прийнято
2	Вибрати потрібний файл у відкритому файловому провіднику	Провідник файлів закрито і відображено повідомлення про те, що файл містить некоректні дані	Провідник файлів закрито і відображено повідомлення про те, що файл містить некоректні дані (рис. 3.10)	Прийнято

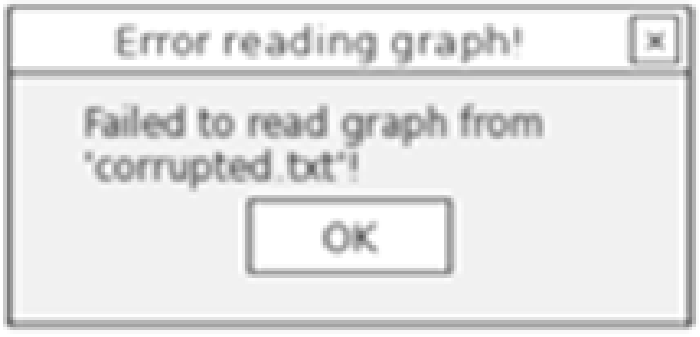


Рисунок 3.10 – Помилка зчитування графу

Тестовий приклад №3 (табл. 3.3).

Мета: переконатися, що наданий користувачем алгоритм відображається.

Вимоги до тесту: б.1.

Передумови: застосунок знаходиться в режимі відображення.

Критерій прийнятності: результат відповідають очікуванням.

Таблиця 3.3 – Тестовий план: «Етапи візуалізації алгоритму»

№	**Опис етапу**	**Очікування**	**Фактичний результат**	**Критерій прийнятності**
1	**2**	**3**	**4**	**5**
1	Відкрити провідник, натиснувши Файл => Завантажити	Відкрито файловий провідник	Відкрито файловий провідник	Прийнято
2	Вибрати значення «Custom» у полі «Algorithms»	Значення поля «Algorithms» змінено на «Custom»	Значення поля «Algorithms» змінено на «Custom»	Прийнято
3	Відкрити файловий провідник, натиснувши «Load»	Відкритий файловий провідник	Відкритий файловий провідник	Прийнято
4	Відкрити файл користувача з кроками алгоритму	Відображено повідомлення про успішне завантаження	Відображено повідомлення про успішне завантаження	Прийнято

Продовження таблиці 3.3

1	2	3	4	5
5	Виконати візуалізацію	Алгоритм відображається відповідно до даних у завантаженому файлі	Алгоритм відображається відповідно до даних у завантаженому файлі (рис. 3.11)	Прийнято

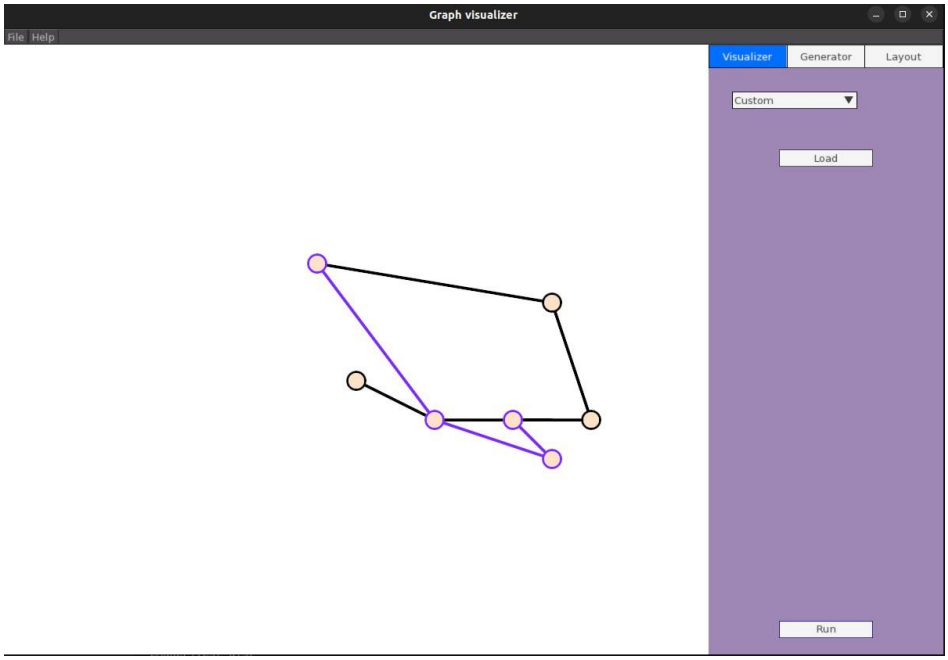


Рисунок 3.11 – Відображення дії алгоритму

Тестовий приклад №4 (табл. 3.4).

Мета: переконатися, що у випадку надання невірних даних для візуалізації алгоритму, користувач отримає відповідне повідомлення.

Вимога до тесту: б.2.

Початкові умови: застосунок знаходиться у стані відображення.

Критерії прийнятності: результати відповідають очікуванням.

Таблиця 3.4 – Тестовий план: «Етапи візуалізації алгоритму»

№	Опис етапу	Очікування	Фактичний результат	Критерій прийнятності
1	Вибрати значення «Custom» у полі «Algorithms»	Значення поля «Algorithms» змінено на «Custom»	Значення поля «Algorithms» змінено на «Custom»	Прийнято
2	Відкрити файловий провідник, натиснувши «Load»	Відкритий файловий провідник	Відкритий файловий провідник	Прийнято
3	Відкрити некоректний файл користувача з кроками алгоритму	Відображене повідомлення про некоректність даних	Відображене повідомлення про некоректність даних (рис. 3.12)	Прийнято

Рисунок 3.12 – Помилка зчитування кроків роботи алгоритму

Тест №5 (табл. 3.5).

Мета: переконатися, що граф можна побудувати вручну за допомогою конструктора графів.

Вимоги до тесту: в.

Початкові умови: застосунок знаходиться у стані конструктора графів.

Критерій прийнятності: результати відповідають очікуванням.

Таблиця 3.5 – Тестовий план: «Тестування конструктора графів»

№	**Опис етапу**	**Очікування**	**Фактичний результат**	**Критерій прийнятності**
1	За допомогою інструменту «Add vertex» додати пару вершин на площину	Дві вершини з’явилися у вибраному користувачем місці	Дві вершини з’явилися у вибраному користувачем місці	Прийнято
2	З’єднати вершини за допомогою інструменту «Add vertex»	З’явилося ребро, що з’єднує дві вершини	З’явилося ребро, що з’єднує дві вершини	Прийнято
3	Інструментом «Hand» перемістити одну з вершин	Вершина переміщується на нове місце, а ребро все ще з’єднує дві вершини	Вершина переміщується на нове місце, а ребро все ще з’єднує дві вершини (рис. 3.13)	Прийнято

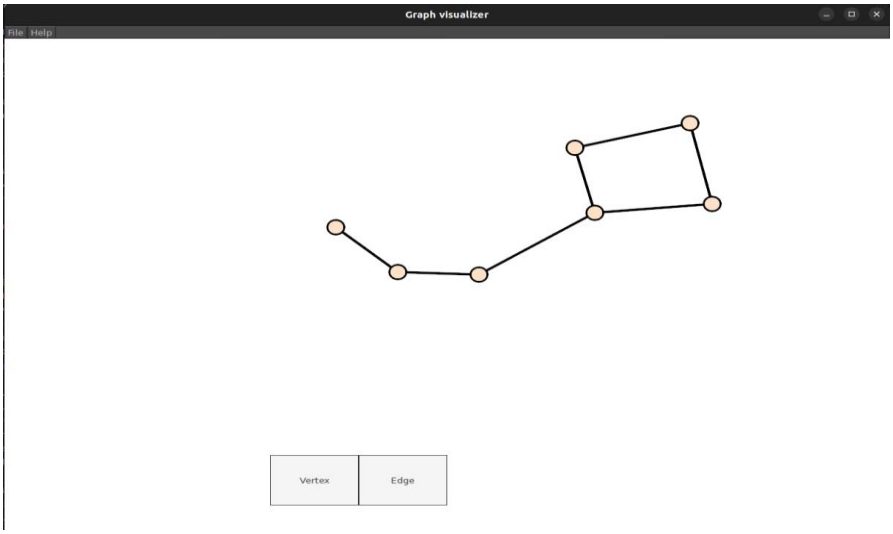


Рисунок 3.13 – Створення користувачем декількох вершин з ребрами

Тестовий приклад №6 (табл. 3.6).

Мета: перевірити, чи правильно реалізовано збереження графу на диску.

Вимоги до тесту: г.

Передумови: застосунок знаходиться у стані конструктора графів, на екрані відображається граф, який було створено на попередньому кроці.

Критерій прийнятності: результати відповідають очікуванням.

Таблиця 3.6 – Тестовий план: «Збереження графу у файлі»

№	Опис етапу	Очікування	Фактичний результат	Критерій прийнятності
1	**2**	**3**	**4**	**5**
1	Відкрити провідник, натиснувши Файл => Зберегти	Відкрито файловий провідник	Відкрито файловий провідник	Прийнято

Продовження таблиці 3.6

1	2	3	4	5
5	Обрати бажане місце для зберігання файлу з графом	Файловий провідник закривається і відображається повідомлення про успішне виконання операції	Файловий провідник закривається і відображається повідомлення про успішне виконання операції	Прийнято

Тестовий приклад №7 (табл. 3.7).

Мета: перевірити коректність роботи базових алгоритмів.

Вимоги до тесту: д.1.

Початкові умови: застосунок знаходиться у стані візуалізації та завантажено графік.

Критерій прийнятності: результати відповідають очікуванням.

Таблиця 3.7 – Тестовий план: «Тестування базових алгоритмів»

№	Опис етапу	Очікування	Фактичний результат	Критерій прийнятності
1	2	3	4	5
1	Відкрити поле «Algorithms» і обрати «BFS»	Бічне меню змінено відповідно до конфігурації BFS	Бічне меню змінено відповідно до конфігурації BFS	Прийнято

Продовження таблиці 3.7

1	2	3	4	5
2	Ввести коректний номер вершини і натиснути «Run»	Алгоритм BFS відображається належним чином	Алгоритм BFS відображається належним чином (рис. 3.14)	Прийнято
3	Відкрити поле «Algorithms» і обрати «DFS»	Бічне меню змінено відповідно до конфігурації DFS	Бічне меню змінено відповідно до конфігурації DFS	Прийнято
4	Ввести коректний номер вершини і натиснути «Run»	Алгоритм DFS відображається належним чином	Алгоритм DFS відображається належним чином (рис. 3.15)	Прийнято

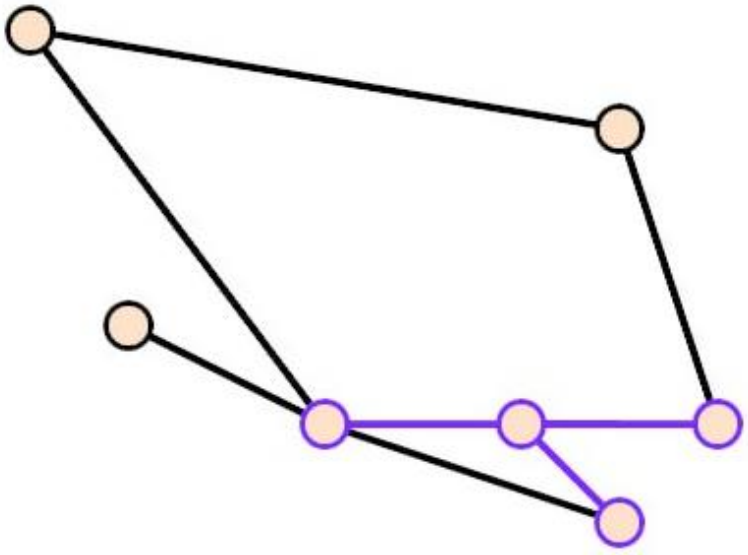


Рисунок 3.14 – Процес відображення роботи базового алгоритму BFS

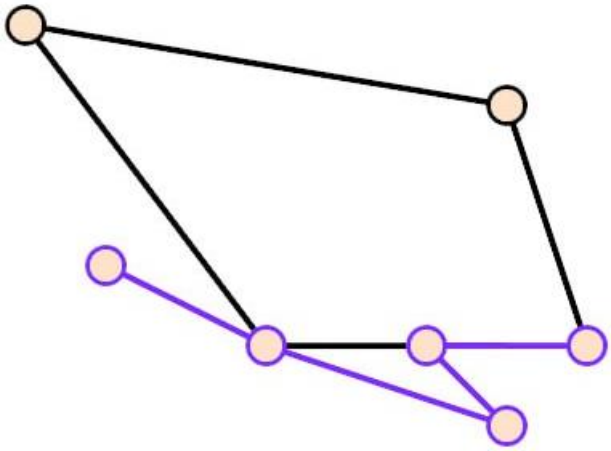


Рисунок 3.15 – Процес відображення роботи базового алгоритму DFS

Тестовий приклад №8 (табл. 3.8).

Мета: переконатися, що користувач отримує сповіщення у разі неправильного введення даних для базових алгоритмів.

Вимоги до тесту: д.2.

Початкові умови: застосунок знаходиться у стані візуалізації.

Критерій прийнятності: результати відповідають очікуванням.

Таблиця 3.8 – Тестовий план: «Тестування неправильно введених даних для базових алгоритмів»

№	Опис етапу	Очікування	Фактичний результат	Критерій прийнятності
1	**2**	**3**	**4**	**5**
1	Відкрити поле «Algorithms» і обрати «BFS»	Бічне меню змінено відповідно до конфігурації BFS	Бічне меню змінено відповідно до конфігурації BFS	Прийнято

Продовження таблиці 3.8

1	2	3	4	5
2	Ввести некоректний номер вершини і натиснути «Run»	Відображається повідомлення з помилкою щодо неправильного ідентифікатора	Відображається повідомлення з помилкою щодо неправильного ідентифікатора (рис. 3.16)	Прийнято
3	Відкрити поле «Algorithms» і обрати «DFS»	Бічне меню змінено відповідно до конфігурації DFS	Бічне меню змінено відповідно до конфігурації DFS	Прийнято
4	Ввести некоректний номер вершини і натиснути «Run»	Відображається повідомлення з помилкою щодо неправильного ідентифікатора	Відображається повідомлення з помилкою щодо неправильного ідентифікатора (рис. 3.17)	Прийнято

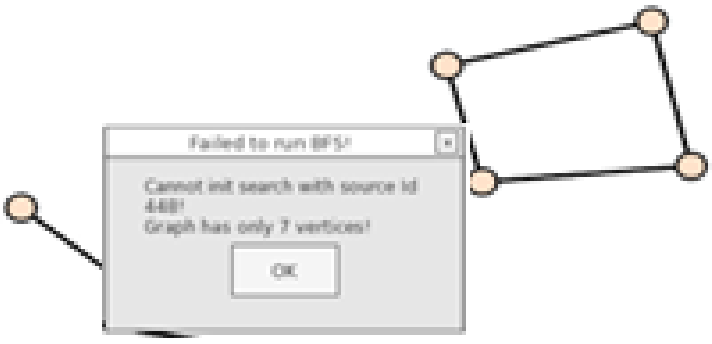


Рисунок 3.16 – Помилка при введенні неправильних даних для роботи алгоритму BFS

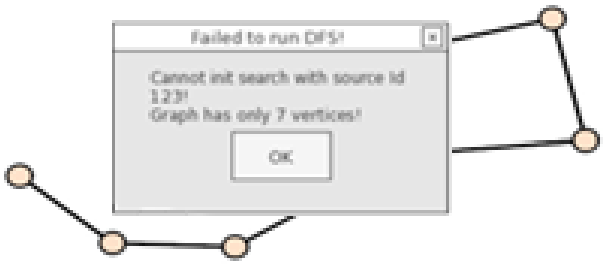


Рисунок 3.17 – Помилка при введенні неправильних даних для роботи алгоритму DFS

Тестовий приклад №9 (табл. 3.9).

Мета: переконатися, що генерація випадкових графів працює коректно.

Вимоги до тесту: 6.е.

Початкові умови: застосунок знаходиться на стадії візуалізації та відкрита вкладка «Generator».

Критерії прийнятності: згенеровано граф із заданою кількістю ребер та вершин.

Таблиця 3.9 – Тестовий план: «Тестування випадкової генерації графу»

№	Опис етапу	Очікування	Фактичний результат	Критерій прийнятності
1	**2**	**3**	**4**	**5**
1	Ввести потрібну кількість ребер та вершин і натиснути кнопку «Generate graph»	На екран виводиться випадково згенерований графік	На екран виводиться випадково згенерований графік	Прийнято

Продовження таблиці 3.9

1	2	3	4	5
2	Перевірити правильність підрахунку вершин та ребер	Кількість вершин та ребер дорівнює введеній	Кількість вершин та ребер дорівнює введеній (рис. 3.18)	Прийнято

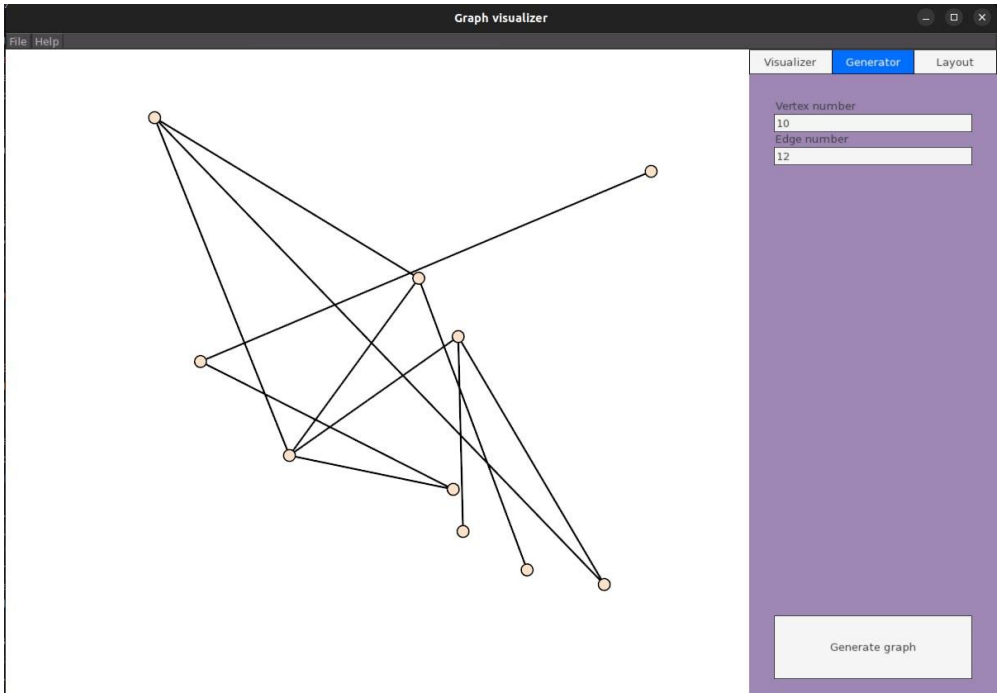


Рисунок 3.18 – Випадково згенерований граф

Тестовий приклад №10 (табл. 3.10).

Мета: перевірити роботу алгоритму візуалізації (Фрухтермана-Рейнгольда).

Вимоги до тесту: ж.

Передумови: застосунок знаходиться у стані візуалізації. Використовувати граф, згенерований на попередньому кроці.

Критерії прийнятності: результати відповідають очікуванням.

Таблиця 3.10 – Тестовий план: «Перевірка роботи алгоритму візуалізації»

№	Опис етапу	Очікування	Фактичний результат	Критерій прийнятності
1	Відкрити вкладку «Layout»	Вкладка «Layout» відкрита	Вкладка «Layout» відкрита	Прийнято
2	Обрати «Fruchterman-Reingold» та виконати алгоритм	Вершини пересуваються відповідно до роботи обраного алгоритму	Вершини пересуваються відповідно до роботи обраного алгоритму (рис. 3.19)	Прийнято

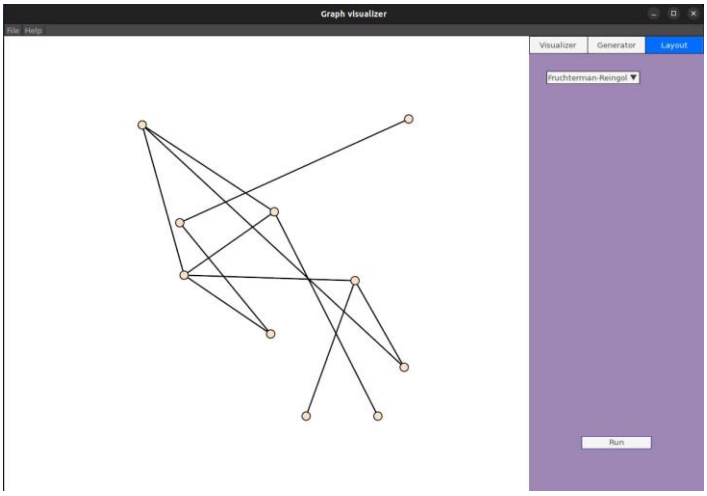


Рисунок 3.19 – Відображення роботи алгоритму Фрухтермана-Рейнгольда

Тестовий приклад №11 (табл. 3.11).

Мета: перевірити роботу вікна допомоги користувачеві.

Вимоги до тесту: з.

Передумови: система перебуває у режимі меню.

Критерій прийнятності: очікування відповідають фактичним результатам.

Таблиця 3.11 – Тестовий план: «Перевірка роботи алгоритму візуалізації»

№	Опис етапу	Очікування	Фактичний результат	Критерій прийнятності
1	Відкрити вкладку «?»	Вкладка «?» відкрита та відображен текст допомоги для користувача	Вкладка «?» відкрита та відображен текст допомоги для користувача (рис. 3.20)	Прийнято

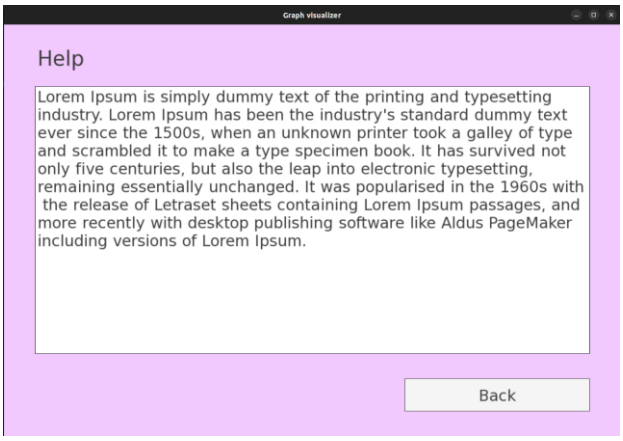


Рисунок 3.20 – Відображення інтерфейсу допомоги для користувача

Під час тестування розробленого застосунку дефектів виявлено не було. Застосунок працює відповідно до очікувань та відповідає усім створеним функціональним вимогам.

ВИСНОВКИ

У рамках кваліфікаційної роботи було розроблено і реалізовано застосунок для візуалізації алгоритмів на графах з великою кількістю вершин. В процесі розробки було вивчено багато літератури та значно покращено загальні знання з теорії графів.

Дана робота є цінним інструментом для всіх дослідників, фахівців з обробки даних та інших людей, які спеціалізуються на роботі з графовими алгоритмами, оскільки візуалізація даних є потужним інструментом для налагодження та аналізу великих структур даних. Роботу застосунку було успішно проаналізовано, сервіс демонструє високу швидкість та продуктивність.

Розроблене програмне забезпечення має багато можливостей для розширення завдяки добре продуманому інтерфейсу та глибині тематики. Можна застосувати нові алгоритми компонування, розширити підтримку інших властивостей графів, таких як багаторівневе розбиття, мультиграфи тощо. Тим не менш, покращення продуктивності буде мати вирішальне значення для підтримки графів з ще більшою кількістю вершин і ребер. Також можна оптимізувати інструменти (наприклад, використовувати OpenGL замість SFML) для роботи з графікою на нижчому рівні.

.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. В. О. Гороховатський. (2017). Алгоритми та структури даних: навч. посібник. *Харків: ХНУРЕ*.

2. С. В. Машталір, Н. В. Васильцова, Л. Е. Чала. (2017). Дискретна математика: навч. посібник. *Харків: ХНУРЕ*.

3. Grandjean, M. (2015). Introduction à la visualisation de données: l'analyse de réseau en histoire. Geschichte und Informatik, (18/19), 109-128. с.

4. Wang, F., Srinivasan, U., Uddin, S., & Chawla, S. (2014, August). Application of network analysis on healthcare. In 2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014) (pp. 596-603). IEEE.

5. Haller, J., & Hansson, H. V. (2013). SFML Game Development. Packt Publishing Ltd.

6. A beginner's guide to graph data visualization. URL: https://cambridge-intelligence.com/a-beginners-guide-to-graph-visualization/ (дата звернення 04.05.2023).

7. Mitchell, S. (2013). SDL Game Development. Packt Publishing Ltd.

8. S. (2015). Visualgo–visualising data structures and algorithms through animation. Olympiads in informatics, 9, 243-245.

9. Bastian, M., Heymann, S., & Jacomy, M. (2009, March). Gephi: an open source software for exploring and manipulating networks. In Proceedings of the international AAAI conference on web and social media (Vol. 3, No. 1, pp. 361-362).

10. А. С. Чуприна (2019). Аналіз вимог до програмного забезпечення: навч. посібник. *Харків: ХНУРЕ*.

11. О. В. Бєляєв (2018). Дизайн інтерфейса: навч. посібник. *Харків: ХНУРЕ*.

12. Ю. С. Бокарєва (2017). Основи графічного дизайну: навч. посібник. *Харків: ХНУРЕ*.

13. О. В. Золотухін (2019). Візуалізація даних та аналітичні сховища: навч. посібник. *Харків: ХНУРЕ*.

14. Noack, A. (2007). Energy models for graph clustering. J. Graph Algorithms Appl., 11(2), 453-480.

15. Walshaw, C. (2001). A multilevel algorithm for force-directed graph drawing. In Graph Drawing: 8th International Symposium, GD 2000 Colonial Williamsburg, VA, USA, September 20–23, 2000 Proceedings 8 (pp. 171-182). Springer Berlin Heidelberg.

16. Noack, A. (2009). Modularity clustering is force-directed layout. Physical review E, 79(2), 026102.

17. Fruchterman, T. M., & Reingold, E. M. (1991). Graph drawing by force-directed placement. Software: Practice and experience, 21(11), 1129-1164.

18. Kamada, T., & Kawai, S. (1989). An algorithm for drawing general undirected graphs. Information processing letters, 31(1), 7-15.

19. Martin, S., Brown, W. M., Klavans, R., & Boyack, K. W. (2011, January). OpenOrd: an open-source toolbox for large graph layout. In Visualization and Data Analysis 2011 (Vol. 7868, pp. 45-55). SPIE.

20. Gajdoš, P., Ježowicz, T., Uher, V., & Dohnálek, P. (2016). A parallel Fruchterman–Reingold algorithm optimized for fast visualization of large graphs and swarms of data. Swarm and evolutionary computation, 26, 56-63.

21. Large Graph Visualization Tools and Approaches. URL: Access mode : https://towardsdatascience.com/large-graph-visualization-tools-and-approaches-2b8758a1cd59 (дата звернення 05.05.2023).

22. В. Білоус (2017). Комп’ютерна дискретна математика: навч. посібник. *Харків: ХНУРЕ*.

23. І. А. Штих (2020). Інформаційні UML технології: навч. посібник. *Харків: ХНУРЕ*.

24. І. С. Творошенко (2018). Проектування інформаційних систем: навч. посібник. *Харків: ХНУРЕ*.

25. С. Ф. Чалий (2021). Патерни програмування і проектування: навч. посібник. *Харків: ХНУРЕ*.

26. Є. П. Путятін, В. П. Степанов, В. П. Пчелінов (2005). Основи програмування мовою С++: навч. посібник. *Харків: ХНУРЕ*.

27. В. А. Любченко (2019). Об’єктно-орієнтоване програмування: навч. посібник. *Харків: ХНУРЕ*.

28. Є. П. Павленко (2016). Технології управління якістю та тестування програмного забезпечення: навч. посібник. *Харків: ХНУРЕ*.

29. О. В. Тітова (2021). Технології тестування програмного забезпечення комп’ютерних систем: навч. посібник. *Харків: ХНУРЕ*.

30. Н. В. Голян (2017). Управління тестуванням: навч. посібник. *Харків: ХНУРЕ*.