МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

**«УЖГОРОДСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ»**

**ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

**Кафедра інформаційних управляючих систем та технологій**

Реєстраційний номер №\_\_\_\_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Курсова робота**

На тему:

«Метод Дейкстри. Знаходження найкоротшого шляху»

Рекомендовано до захисту:

„\_\_\_”\_\_\_\_\_\_\_\_\_\_\_2023 року

Робота захищена

„\_\_\_\_”\_\_\_\_\_\_\_\_\_\_2023 року

з оцінкою

„\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_”

Підписи членів комісії

***Виконав:***

студент 2-го курсу

денної форми навчання

Спеціальності «Комп’ютерні науки»

**Лобанов Микола Миколайович**

***Науковий керівник:***

**проф. Міца О**. **В.**

# **Зміст**

[**ВСТУП………………………………………………………………………3**](#_ВСТУП)

[**РОЗДІЛ 1. БАЗОВІ ПОНЯТТЯ МЕТОДУ ДЕЙКСТРИ……4**](#_Розділ_1._Базові)

1. [**Визначення графу……..…………………………………….4**](#_Визначення_графу)
2. [**Базові поняття ребра та його ваги……..……………5**](#_Базові_поняття_ребра)
3. [**Графи на реальних прикладах………………………..6**](#_Графи_на_реальних)

[**РОЗДІЛ 2. ДЕТАЛІЗАЦІЯ, ПРИНЦИПИ ТА МЕХАНІЗМ РОБОТИ АЛГОРИТМУ………………………………………………8**](#_Розділ_2._Деталізація,)

[**РОЗДІЛ 3. РЕАЛЬНІ ПРИКЛАДИ ВИКОРИСТАННЯ АЛГОРИТМУ ДЕЙКСТРИ………………………………………..11**](#_Розділ_3._Реальні)

1. [**Компанії сотового зв’язку……………………………..12**](#_Компанії_сотового_зв’язку)
2. [**GPS навігація………………………………………………….13**](#_GPS_навігація)
3. [**Соціальні мережі…………………………………………..13**](#_Соціальні_мережі)

[**РОЗДІЛ 4. РОЗРОБКА ПРОГРАМИ ДЛЯ ПОШУКУ НАЙКОРОТШОГО ШЛЯХУ………………………………………13**](#_Розділ_4._Розробка)

[**ВИСНОВКИ…………………………………………………………….20**](#_Висновки)

# **ВСТУП**

Алгоритм Дейкстри - один з фундаментальних алгоритмів в галузі теорії графів і оптимізації шляхів. Цей алгоритм був розроблений голландським математиком Едсгером Дейкстрою в 1956 році і здобув широку визнаність і застосування в різних галузях інформатики.

Головна мета алгоритму Дейкстри - знайти найкоротший шлях між двома вершинами у направленому або ненаправленому зваженому графі. Цей алгоритм вирішує таку задачу ефективно, знаходячи найкоротші шляхи від однієї вершини до всіх інших вершин у графі, якщо відомі ваги ребер.

У даній курсовій роботі ми розглянемо принципи та роботу алгоритму Дейкстри, його математичне обґрунтування та реалізацію на мові програмування Python.

Ця робота дозволить вам поглибити розуміння теорії графів, а також набути практичних навичок у реалізації алгоритму Дейкстри. Вона стане важливим кроком у вашому професійному розвитку в галузі інформатики та допоможе вам зрозуміти основи оптимізації шляхів в різних задачах.

Мета курсової роботи – створити інтерактивну програму пошуку найкоротшого шляху простими методами, щоб будь хто, незалежно від знань в області математики та алгоритмів зміг зрозуміти

# Розділ 1. Базові поняття методу Дейкстри

Хоча алгоритм Дейкстри був розроблений у 1959 році, він використовується як невід'ємна складова багатьох додатків та програм. Один із найвідоміших прикладів його застосування - це Google Maps, де він використовується для знаходження найкоротшого шляху між двома точками на мапі. Для кращого розуміння важливості пошуку найкоротшого шляху в графі, необхідно мати чітке уявлення про те, що ми маємо на увазі, коли говоримо про граф.

## Визначення графу

Граф – це абстрактна структура даних, яка складається з двох компонентів – вершин (Вузлів) та ребер (зв’язків між вузлами). Зобразимо такий граф за допомогою безкоштовної програми “Figma”:

Изображение выглядит как круг, диаграмма, часы, дизайн

Автоматически созданное описание

Це один з найпростіших графів, який складається з 6ти вузлів, які позначені колами, з цифрою в центрі. Цей номер ми будемо використовувати для того, щоб відрізняти ці вузли.

## Базові поняття ребра та його ваги

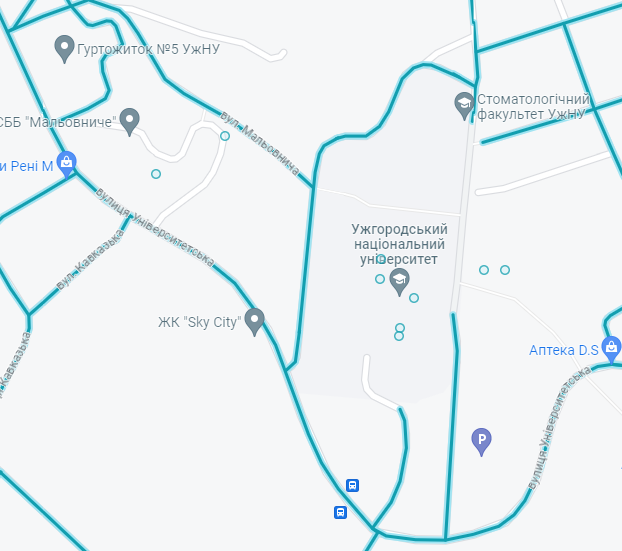
У графі вузли пов'язані між собою лініями, які мають певне числове значення, відоме як вага ребра. Ця вага може вимірюватись у відстані або часі, потрібному для подолання ребра. Надалі ми будемо використовувати термін "довжина ребра" для спрощення. У графі вершина "6" з'єднана з усіма іншими вершинами, за винятком вершини "1", тому не можна безпосередньо перейти від вершини "6" до вершини "1". Отже, треба пройти через одну або кілька проміжних вершин, які їх з'єднують. Це створює проблему: "Який шлях потрібно обрати, щоб якнайшвидше досягти кінцевої точки?". Щоб з'ясувати найкоротший шлях, необхідно зрозуміти, як саме вимірюється довжина цього шляху.

На графі, біля кожного зв’язку є число. Воно відповідає за довжину цього зв’язку, тобто відстань від одної вершини до іншої.

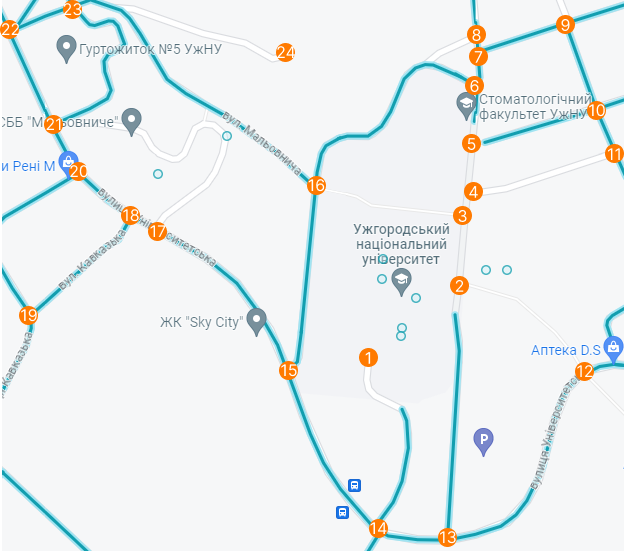
## Графи на реальних прикладах

Щоб краще розібратись у роботі графів, можна провести аналогію з реальними об'єктами, з якими ми знайомі. Уявімо, що кожна точка в графі відповідає перехрестю доріг, а ребра - самим дорогам. Така аналогія допоможе нам зрозуміти основні поняття та принципи графів.

Для покращення нашого розуміння, давайте скористаємося Google Maps для перегляду зображення доріг одного з районів Ужгорода. Цей онлайн-сервіс дозволяє нам візуалізувати реальний світ через карти та вуличні зображення. Для отримання більш детальної картини, ввімкнемо режим "Перегляд вулиць" у налаштуваннях шарів відображення. Тоді на екрані з'явиться детальне зображення доріжок та інфраструктури міста.

Це допоможе нам наочно побачити, як виглядають дороги та перехрестя, як вони пов'язані між собою. Так само, як і в графах, вузли та ребра у графі доріг мають свої взаємозв'язки та шляхи з'єднання. За допомогою такої аналогії ми можемо краще уявити собі принципи навігації та пошуку найкоротшого шляху в графі.  


Поки що, це виглядає як звичайна фотографія доріг, але достатньо додати до них вершини, і звичайна карта перетворюється в граф. Скористаємося знову програмою Figma, щоб зобразити вершини:



Зверніть увагу, що ми встановлюємо вершини лише на розвилках доріг, а також на кінці доріг, якщо вони припиняються. Наприклад, вершина "1" веде до УжНУ. Варто також відзначити, що вершини можна розміщувати де завгодно, навіть посеред дороги між двома іншими вершинами, навіть якщо немає розвилок чи перехресть. Це дає нам можливість встановити вершину в будь-якій значущій точці на мапі, до якої також можна запланувати маршрут. На цьому фото ми можемо уявити, що кожна точка з надписом є окремою вершиною, які відповідають УжНУ, гуртожитку №5, аптекам та навіть автобусним зупинкам. Таке уявлення допомагає нам більш детально представити граф доріг і розглянути різні місця як окремі вершини з їх власними взаємозв'язками.

Для того, щоб виміряти найкоротший шлях нам знадобиться вага кожного ребра. Якщо вірити Google Maps, відстань між вершинами «1» та «14» буде приблизно 200м. Аналогічно з цим, можна розрахувати відстань між усіма іншими вершинами.

# Розділ 2. Деталізація, принципи та механізм роботи алгоритму

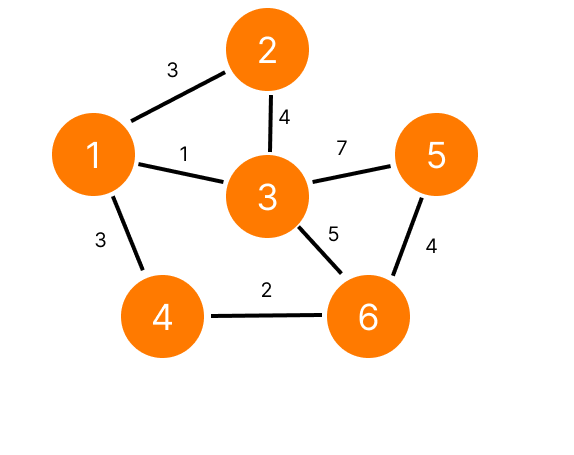
Мета алгоритму Дейкстри - знайти найкоротший шлях між двома вершинами. Ми уже ознайомились зі способами вимірювання відстані між вершинами та структурою графа. Тепер давайте розглянемо, як саме працює алгоритм Дейкстри.

Основна ідея алгоритму дуже проста. Ми розглядаємо всі можливі шляхи від стартової вершини до всіх інших вершин. Для початку необхідно обрати початкову позицію. Візьмемо простий граф з першого прикладу. Щоб спростити розрахунки, зменшимо відстань між вершинами, але вилучимо два ребра: одне між вершинами "2" і "5", інше - між вершинами "3" і "4".

Алгоритм Дейкстри працює наступним чином:

1. Встановлюємо вагу стартової вершини на 0, а вагу всіх інших вершин - нескінченність.
2. Обираємо поточну вершину і розглядаємо всі сусідні вершини, які з нею пов'язані ребром.
3. Розраховуємо відстань до кожної сусідньої вершини, яка проходить через поточну вершину. Якщо отримана відстань менша за поточну вагу сусідньої вершини, оновлюємо вагу.
4. Позначаємо поточну вершину як відвідану.
5. Повторюємо кроки 2-4 для усіх невідвіданих вершин, поки не відвідаємо всі вершини або не досягнемо кінцевої вершини.
6. Коли всі вершини відвідані, маємо найкоротший шлях до кожної вершини від стартової.

Це основний принцип його роботи, який можна застосувати в різних сферах, включаючи навігаційні системи, маршрутизацію мереж та багато інших.

Граф зараз має ось такий вигляд:

Оберемо стартову позицію точку «1» та розрахуємо мінімальну відстань від вершини «1» до усіх інших. Щоб це зробити нам допоможе таблиця найкоротших шляхів. Позначимо усі точки нашого графу, а в першому рядку таблиці позначимо вершину «1» цифрою 0, так як відстань від вершини «1» до вершини «1» дорівнює нулю. Для усіх інших вершин нам потрібно вказати нескінченність, тому що ми ще не знаємо, яка відстань від точки 1 до інших.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № ітерації | Вершина 1 | Вершина 2 | Вершина 3 | Вершина 4 | Вершина 5 | Вершина 6 |
| 1 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ |

На першій ітерації нашого алгоритму ми виявили, що у нас всього 6 вершин, а ми знаходимося в вершині «1».  
Тепер, потрібно знайти усі вершини, до яких ми можемо потрапити з вершини «1» напряму.  
Ми можемо потрапити до вершин «2», «3» та «4». Запишемо цю відстань у нашу таблицю.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № ітерації | Вершина 1 | Вершина 2 | Вершина 3 | Вершина 4 | Вершина 5 | Вершина 6 |
| 1 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 |  | 3 | 1 | 3 | ∞ | ∞ |

Поле, з першою вершиною ми пропускаємо, так як ми вже знайшли найкоротшу відстань до першої вершини. Ми будемо позначати найкоротшу відстань зеленим кольором.

Правило ітерацій просте – ми беремо відстань, яка була потрібна на попередній ітерації, та додаємо її до відстані, яку нам потрібно пройти до вершини. Дістатись до вершин «5» та «6» напряму з вершини «1» ми не можемо, тому поки що залишаємо нескінченність

Згідно алгоритму Дейкстри, ми маємо обрати іншу вершину, з якою будемо працювати в наступній ітерації. Вершину потрібно обирати згідно за відстанню до неї. Найближча до поточної вершини буде вершина «3» з відстанню 1. Повторюємо операцію, яку ми робили на другій ітерації, але тепер за основу беремо вершину «3». Пройти з вершини «3» ми можемо до вершин «2», «5» та «6». Вершину «1» ми не розглядаємо, тому що ми з неї почали, тобто ми її вже розглянули. Повертатись назад у вершини, які вже були розглянуті не можна.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № ітерації | Вершина 1 | Вершина 2 | Вершина 3 | Вершина 4 | Вершина 5 | Вершина 6 |
| 1 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 |  | 3 | 1 | 3 | ∞ | ∞ |
| 3 |  | 3 |  | 3 | 8 | 6 |

Розглянувши відстань від вершини «3» до «2» ми маємо відстань 4, але, так як ми знайшли, що відстань до вершини «2» з вершини «1» рівна 3, ми залишаємо найменше з них, тобто 3. Потрібно пам’ятати, що ми знаходимо найкоротшу відстань від вершини «1» до іншої вершини. В нашому випадку – до усіх інших вершин.

Відстань від вершини «3» до вершини «5» рівна 7, але ми пам’ятаємо, що нам потрібно знаходити суму попередньої відстані, тобто відстані до вершини «3», та відстань до вершини, в яку ми прямуємо. Відстань до вершини «3» складає 1, тому відстань до вершини «5» буде рівна 7+1. Повторюємо ту саму операцію для вершини «6».

Потрапити з вершини «3» у вершину «4» напряму ми не можемо, але так, як ми знаємо як дістатись до вершини «4», ми залишаємо значення 3 з попередньої ітерації.

Продовжуємо ітерації. Нам знову потрібно обрати мінімальне значення. У нас вони рівні у вершинах «2» та «4». Ми можемо обрати будь яку. Наприклад, оберемо вершину «2».

З вершини «2» ми можемо потрапити лише у вершину «1», яка вже була розглянута, та у вершину «3», яку ми також розглянули. Отже, ми можемо сказати, що з вершини «1» ми можемо дістатись до вершини «2» з мінімальною відстанню 3.

Якщо ми шукали відстань від вершини «1» саме до вершини «2», то на цьому наш алгоритм завершив би роботу, але ми продовжимо та знайдемо відстань до усіх вершин.

На наступній ітерації візьмемо вершину «4», так як наразі це не розглянута вершина з мінімальною відстанню до неї.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № ітерації | Вершина 1 | Вершина 2 | Вершина 3 | Вершина 4 | Вершина 5 | Вершина 6 |
| 1 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 |  | 3 | 1 | 3 | ∞ | ∞ |
| 3 |  | 3 |  | 3 | 8 | 6 |
| 4 |  |  |  | 3 | 8 | 6 |

З вершини «4», у не розглянуті вершини ми можемо потрапити лише у вершину «6». Відстань між вершинами «4» та «6» рівна 2. Додаємо 2 до відстані до поточної вершини (Вершини «4») та отримуємо 5.

Зараз вершина «6» має значення 6 як мінімальне, але ми знайшли, що відстань до вершини «6» з вершини «1» через вершину «4» буде меншою, отже записуємо 5 замість поточного.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № ітерації | Вершина 1 | Вершина 2 | Вершина 3 | Вершина 4 | Вершина 5 | Вершина 6 |
| 1 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 |  | 3 | 1 | 3 | ∞ | ∞ |
| 3 |  | 3 |  | 3 | 8 | 6 |
| 4 |  |  |  | 3 | 8 | 6 |
| 5 |  |  |  |  | 8 | 5 |

Залишається лише одна нерозглянута вершина - вершина «5». Знайдемо відстань до неї. До поточної відстані 5 ми додаємо відстань до вершини «5» з вершини «6» 5+4=9. Поточна відстань до вершини «5» - 8, що менше за 9, а отже, відстань до вершини «5» через вершину «6» буде довшою, ніж через вершину «3». Залишаємо 8.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № ітерації | Вершина 1 | Вершина 2 | Вершина 3 | Вершина 4 | Вершина 5 | Вершина 6 |
| 1 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 |  | 3 | 1 | 3 | ∞ | ∞ |
| 3 |  | 3 |  | 3 | 8 | 6 |
| 4 |  |  |  | 3 | 8 | 6 |
| 5 |  |  |  |  | 8 | 5 |
| 6 |  |  |  |  | 8 |  |

На цьому робота алгоритму закінчується, адже ми розглянули усі вершини.

Ця таблиця дає нам інформацію, яку мінімальну відстань потрібно здолати, щоб потрапити з вершини «1» до будь якої іншої. Наприклад, якщо ми хочемо потрапити до вершини «4», то мінімальна відстань до неї буде 3 (Напряму з вершини «1» до вершини «4»). А якщо оберемо вершину «5», то відстань до неї буде 8 (Від вершини «1» через вершину «3» до вершини «5»).

# Розділ 3. Реальні приклади використання алгоритму Дейкстри

Ми розглянули як працює алгоритм Дейкстри, але слід розуміти, що сам алгоритм універсальний. Розглянемо приклади де використовується алгоритм Дейкстри та яку роль він там виконує.

На думку в першу чергу приходить Google Maps, який ми розглянули у першому розділі. Google Maps використовує алгоритм для того, щоб знайти найкоротший шлях від однієї точки до іншої. Точками можуть бути будинки або ресторани, автобусні зупинки або цілі міста. Але як і будь який складний алгоритм, він іноді дає збій. Згадаємо граф з першого розділу. Якщо прокласти маршрут з вершини «8» до вершини «13» найкоротшим шляхом буде пройти напряму через одразу декілька вершин, які були зображені:

Изображение выглядит как карта, диаграмма, текст, атлас

Автоматически созданное описание

Але, якщо спитати у Google Maps, то алгоритм підкаже інший шлях.

Изображение выглядит как карта, диаграмма, атлас, План

Автоматически созданное описание

Так, або інакше, він підказує один з найближчих шляхів.

Розглянемо приклади, де може використовуватись алгоритм Дейкстри, окрім Google Maps.

## Компанії сотового зв’язку

Мобільні телефони, планшети з SIM картою використовують підключення до вишок сотового зв’язку. Але звідки береться інформація на ці вишки?  
Компанії, такі як Kyivstar або Vodafone використовують алгоритм Дейкстри для того, шоб краще передати сигнал на вишки. Але використовують вони не лише довжину шляху, але і пріорітет. Пріорітет дуже важливий для подібних алгоритмів, адже він дасть змогу розділити навантаження на окремі вершини. Якщо навести приклад зі звичайними дорогами, то ми можемо уявити, що на одній з доріг зараз намагаються проїхати велика кількість машин. Сам шлях через цю дорогу буде найкоротшим, але на прикладі через велику кількість автомобілів рух буде ускладнений, що може привести до сповільнення руху, або навіть повного припинення. Так само працює пріорітет пошуку у провайдерів сотового зв’язку. Але замість машин затори створюють клієнти сервісу. Пріорітет працює наступним чином, якщо одна з вишок зв’язку занадто навантажена кількістю клієнтів, то пріорітет на неї буде зменшений. Під час пошуку вишки сотового зв’язку (Наприклад під час дзвінку по телефону) буде обрана інша вишка через те, що найближча має менший пріорітет. Це розвантажить вишки сотового зв’язку від великої кількості клієнтів одночасно.

## GPS навігація

Google Maps, Apple Maps та інші подібні програми використовують алгоритм Дейкстри для того, щоб знайти найкоротший шлях від точки А до точки Б. В цих програмах алгоритм прораховує довжину шляху, але також прораховує затори на дорогах, що зменшує пріорітет для доріг, якщо на ній зараз багато автомобілів.

## Соціальні мережі

Від найочевидніших випадків до більш цікавих алгоритмів. Соціальні мережі, такі, як Facebook, Twitter та Instagram використовують алгоритм Дейкстри для пошуку друзів, знайомих та просто людей, які можуть зацікавити користувача. За схожим алгоритмом працює YouTube, який підбирає рекомендації залежно від минулих запитів користувача.

# Розділ 4. Розробка програми для пошуку найкоротшого шляху

Для розробки програми ми використаємо Python. Встановимо бібліотеку pygame, щоб створити графічне відображення алгоритму.

Я використовую Visual Studio Code, тому що звик працювати саме в цьому середовищі. Для зручності можна було б використати PyCharm, який підтримує встановлення усіх бібліотек через інтерфейс. В моєму випадку доведеться скористатись терміналом.  
Встановлюємо пакет pygame за допомогою “pip install pygame”  
Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

В ході роботи, ми розробимо програму, де користувач буде мати можливість «малювати» лабіринт за допомогою курсору. Програма має за допомогою алгоритму Дейкстри знайти найкоротший шлях від однієї точки до іншої.

Додаємо tkinter для створення message боксів, sys для закриття програми та додамо deque з бібліотеки collections, щоб створювати черги переглянутих вузлів.  
Напишемо код класу Cell та просту функцію main з зчитуванням кліку мишки:

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Додамо функцію створення стінки при кліку мишки

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Ми підготували середовище для початку написання коду алгоритму. Для того, щоб алгоритм спрацював нам потрібно додати початкову та кінцеві точки. Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Як і в нашому прикладі вище, ми позначимо початкову точку вже переглянутою. Алгоритм буде проходити усі сусідні точки починаючи з лівого верхнього краю програми

Ми додали початкову і кінцеві точки, але нам потрібно виділити їх на фоні інших, тому що зараз вони мають такий самий колір. Змінюємо код циклу всередині функції main, щоб змінити колір клітинок

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, программное обеспечение

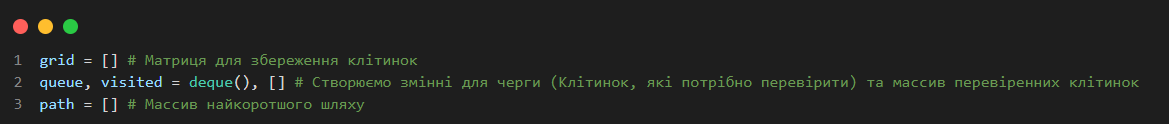
Автоматически созданное описание

Після запуску програми бачимо 2 клітинки іншого кольоруИзображение выглядит как снимок экрана, текст, дисплей, программное обеспечение

Автоматически созданное описание

Перейдемо до написання алгоритму.

Створюємо масиви для того, щоб зберегти дані про клітинки, які вже були переглянуті, ті, які ще потрібно перевірити, та масив path для збереження послідовності клітинок найкоротшого шляху від початкової точки до кінцевої.



Тепер, додамо подію при натисканні клавіші ENTER, щоб скрипт почав роботу:

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Змінна startflag буде переходити в True, як тільки була натиснута клавіша Enter. З цього моменту, програма має знайти найкоротший шлях. Напишемо алгоритм пошуку.

Алгоритм буде працювати наступним чином:  
Кожну ітерацію програми скрипт буде перевіряти масив queue на наявність клітинок для перевірки. Якщо в масиві не буде клітинок, то це означає, що алгоритм не знайшов шляху до точки.

Якщо ще є не перевірені клітинки, скрипт перевіряє, чи є ця клітинка кінцевою, яку ми шукаємо.

Точки будуть перевірятись послідовно. Під час перевірки кожної клітинки перевіряються усі сусідні клітинки. Алгоритм почне пошук в усіх напрямках одночасно і як тільки він дійде до кінцевої – пошук зупиняється, після чого скрипт проходить весь шлях назад, поки не дійде до першої точки. Це і буде найкоротший шлях.

Весь алгоритм можна записати в короткий код:

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Програма завершена, перевіримо роботу алгоритму.  
Малюємо простий лабіринт для прикладу:

Изображение выглядит как снимок экрана, шаблон, прямоугольный, Прямоугольник

Автоматически созданное описание

Натискаємо Enter, та через декілька секунд бачимо результатИзображение выглядит как шаблон, прямоугольный, Красочность, Прямоугольник

Автоматически созданное описание

# Висновки

У даній курсовій роботі був розглянутий алгоритм Дейкстри, який використовується для пошуку найкоротшого шляху в графі. Графи є потужним інструментом моделювання та розв'язування різних завдань, які включають мережі, транспортні системи, соціальні мережі та багато інших.

Основна ідея алгоритму Дейкстри полягає у повноцінному пошуку всіх можливих шляхів від однієї вершини до іншої, з метою знайти найкоротший шлях. Він працює шляхом оновлення ваги вершин та розрахунку відстаней через поточну вершину. Після завершення алгоритму, ми отримуємо найкоротший шлях від стартової вершини до кожної іншої вершини в графі.

Алгоритм Дейкстри має широке застосування в різних областях, включаючи транспортну логістику, маршрутизацію мереж, планування маршрутів у навігаційних системах та багато інших. Він дозволяє ефективно вирішувати задачі знаходження найкоротшого шляху і відомий своєю швидкістю та точністю.

У процесі вивчення алгоритму Дейкстри були розглянуті основні принципи та механізми його роботи. Було розібрано важливі поняття, такі як графи, вершини, ребра та їх ваги. Також була проведена аналогія з реальними об'єктами, що сприяє кращому розумінню та візуалізації роботи алгоритму.

Використання алгоритму Дейкстри дозволяє ефективно розв'язувати задачі знаходження найкоротшого шляху, що має велике значення в різних сферах. Розуміння його принципів та особливостей допомагає розробникам і інженерам створювати більш ефективні та оптимальні системи та програми.

Однак, варто враховувати, що алгоритм Дейкстри має обмеження в ситуаціях, коли в графі присутні цикли або ваги ребер неоднакові, або ваги від’ємні. У таких випадках, існують інші алгоритми, які можуть бути більш підходящими для вирішення задач.

Отже, алгоритм Дейкстри є потужним інструментом для знаходження найкоротшого шляху в графі. Вивчення його принципів та застосування дозволяють розширити знання про графи та розв'язування пов'язаних задач, сприяючи розвитку ефективних алгоритмів і програм. Не варто забувати, що існує велика кількість алгоритмів, які використовують алгебраїчні операції. Отже, знання математики все ж таки необхідні для програмістів.