МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРИВОРІЗЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ
Фізико-математичний факультет
Кафедра інформатики та прикладної математики

«Допущено до захисту»
В.о. завідувача кафедри
_____ Моісеєнко Н.В.
«___» _____ 2023 р.

Реєстраційний № _____
«___» _____ 2023 р.

СИСТЕМА ПОШУКУ ЗАЛІЗНИЧНОГО МАРШРУТУ ПАСАЖИРСЬКИХ ПЕРЕВЕЗЕНЬ

Кваліфікаційна робота
студента групи І-19
ступінь вищої освіти «бакалавр»
спеціальності
014.09 Середня освіта (Інформатика)
Дзюбіна Олександра Артуровича

Керівник: асистент
Степанюк Олександр Миколайович

Оцінка:
Національна шкала _____
Шкала ECTS ___ Кількість балів ___
Голова ЕК _____ _____
Члени ЕК _____ _____
_____ _____
_____ _____
_____ _____

Кривий Ріг – 2023

ЗАПЕВНЕННЯ

Я, Дзюбін Олександр Артурович, розумію і підтримую політику Криворізького державного педагогічного університету з академічної доброчесності. Запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Я не надавав і не одержував недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають покликання на відповідне джерело. Із чинним Положенням про запобігання та виявлення академічного плагіату в роботах здобувачів вищої освіти Криворізького державного педагогічного університету ознайомлений. Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

ЗМІСТ

ВСТУП

Актуальність роботи полягає в тому, що залізниці є важливим складовим елементом транспортної інфраструктури багатьох країн і мають велике значення для пасажирських перевезень. Вирішення задачі пошуку оптимального залізничного маршруту з використанням теорії графів стає дедалі важливішим з прискореним розвитком транспортних систем і зростанням потреб пасажирських перевезень.

Теорія графів надає математичну модель для аналізу та оптимізації залізничних мереж і маршрутів. Вона дозволяє визначити оптимальний маршрут для пасажирів з урахуванням таких факторів, як відстань, час подорожі, наявність пересадок, економічна ефективність та інші обмеження.

Із зростанням кількості пасажирів та складності транспортних систем, ефективне планування та управління залізничними маршрутами стає все більш складною задачею. Використання теорії графів дозволяє зменшити складність обчислень і знайти оптимальні рішення для розподілу ресурсів, планування графіків руху поїздів, підвищення ефективності перевезень та зменшення затрат.

Пошук оптимальних залізничних маршрутів також має практичне застосування в реальному часі, де важливо швидко та ефективно обробляти великі обсяги даних про розклади руху поїздів, наявність затримок, популярні напрямки та інші фактори, що впливають на вибір маршруту.

Застосування теорії графів у пошуку залізничних маршрутів допомагає покращити якість пасажирських перевезень, зменшити час подорожі та затрати, забезпечити більш ефективне використання ресурсів та зробити транспортну систему більш стійкою до змінних умов.

Отже, актуальність роботи полягає у важливості використання теорії графів для пошуку оптимального залізничного маршруту пасажирських перевезень, що допоможе вирішити складні завдання управління та покращити ефективність транспортної системи.

Мета даної роботи полягає у розробці алгоритму для пошуку залізничного маршруту пасажирських перевезень. При цьому *об'єктом дослідження* є теорія графів, яка є основою для вирішення задачі пошуку маршрутів на залізничній мережі. *Предметом дослідження* є алгоритми, які використовують для пошуку залізничних маршрутів з використанням теорії графів.

Для досягнення поставленої мети необхідно виконати кілька завдань. По-перше, потрібно вивчити літературу та теоретичні матеріали, що стосуються теорії пошуку маршрутів. Це дозволить ознайомитись зі знаряддями та методами, які використовуються в цій галузі.

По-друге, необхідно визначити основні методики та підходи до пошуку маршрутів. Це можуть бути, наприклад, алгоритми Дейкстри, алгоритми пошуку в ширину або в глибину, алгоритми з використанням евристичних підходів тощо. Вибір методики залежить від конкретних вимог і обмежень, які поставлені перед системою пошуку маршрутів.

По-третє, потрібно описати теорію та побудувати алгоритм пошуку залізничного маршруту з використанням теорії графів. Це означає, що потрібно розглянути залізничну мережу як граф, де вузлами є станції, а ребрами - залізничні лінії, і розробити алгоритм, який знайде найкоротший шлях між вказаними станціями з урахуванням розкладу руху поїздів та інших обмежень.

На завершення, необхідно розробити алгоритм та його реалізувати в додатку. Це означає, що після теоретичного розроблення алгоритму його

потрібно перетворити на програмний код, який буде виконувати пошук маршрутів на основі введених даних про залізничну мережу.

В результаті виконання даної роботи буде отримано алгоритм та програмний додаток, які дозволять здійснювати пошук залізничних маршрутів. Це може бути корисно для пасажирів, які планують подорожі залізницею і хочуть знати оптимальний маршрут та час прибуття.

Структура та обсяг роботи: кваліфікаційна робота складається із вступу, трьох розділів, висновків до кожного розділу, загальних висновків, 9 зображень та списку використаних джерел, що нараховує 17 джерел.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Алгоритми пошуку оптимального шляху в графах

Пошук шляху є методом для обчислення найкоротшого маршруту між двома точками з використанням спеціальних алгоритмів для персонального комп'ютера [1]. Ці дослідження базуються на використанні алгоритму Дейкстри [4] для знаходження найкоротшого шляху з використанням зваженого графа.

За суттю, алгоритм знаходження шляху досліджує граф, починаючи з одного вузла і переходячи до сусідніх вузлів, поки не знайде цільовий вузол, з метою пошуку маршруту, що є найкоротшим [1].

Основні задачі пошуку шляху [1] включають:

1) Задача найкоротших шляхів з одним вхідним вузлом, де потрібно знайти найкоротші шляхи від початкового вузла до всіх інших вузлів у графі.
2) Задача найкоротших шляхів з одним вихідним вузлом, де потрібно знайти найкоротші шляхи від усіх вузлів у графі до заданого вихідного вузла. Ця задача може бути скорочена до задачі з одним вхідним вузлом шляхом зміни напрямку ребер у графі.
3) Задача найкоротших шляхів для всіх пар вузлів, де потрібно знайти найкоротші шляхи між кожною парою вузлів у графі.

Алгоритми пошуку у ширину або у глибину, розглядають всі можливості, розпочинаючи зі стартового вузла, і ітеративно досліджують шляхи, до моменту досягнення цільового вузла. Дані алгоритми мають часову складність O(|V| + |E|), де |V| - кількість вузлів, а |E| - кількість ребер між вузлами.

Основні алгоритми [1, 2, 5] для вирішення цих задач включають:

1) Алгоритм Дейкстри використовується для знаходження найкоротшого шляху в графі з однією парою вхідного та вихідного вузлів. Він використовує жадібний підхід, обираючи найближчий вузол та оновлюючи відстані до сусідніх вузлів, якщо знайдений шлях коротший.

2) Алгоритм Беллмана-Форда вирішує задачу з одним вхідним вузлом, коли ваги ребер можуть бути від'ємними. Він використовує ітераційний підхід, обходячи всі ребра графу кілька разів та оновлюючи відстані до вузлів до тих пір, поки немає можливості скоротити шляхи.

3) Алгоритм пошуку A* є евристичним алгоритмом для знаходження найкоротшого шляху в графі між двома вузлами. Він поєднує в собі інформацію про відстані від початкового вузла до поточного вузла та евристичну оцінку від поточного вузла до цільового вузла. Це дозволяє алгоритму зосередитися на найбільш оптимальних шляхах та прискорити пошук.

4) Алгоритм Флойда-Воршелла вирішує задачу знаходження найкоротших шляхів між всіма парами вузлів у графі. Він використовується для орієнтованих та незважених графів. Алгоритм працює на принципі динамічного програмування, оновлюючи відстані за допомогою проміжних вузлів.

5) Алгоритм Джонсона вирішує задачу знаходження найкоротших шляхів між всіма парами вузлів у графі. Він може бути ефективнішим за алгоритм Флойда-Воршелла на розріджених графах. Алгоритм Джонсона використовує модифіковану версію алгоритму Беллмана-Форда для знаходження потенційних ваг ребер, що дозволяє йому ефективно обробляти ребра з від'ємними вагами.

6) Алгоритм знаходження локально найкоротшого шляху з використанням теорії збурень використовує теорію збурень для знаходження найкоротшого шляху в графі. Він розглядає випадки, коли невеликі збурення

відбуваються в графі, та оновлює шляхи залежно від цих збурень. Цей підхід дозволяє знаходити локально найкоротші шляхи та враховувати зміни в графі з малими впливами на інші шляхи.

1.2. Аналіз алгоритмів

Алгоритм Дейкстри [4] є ефективним інструментом для пошуку найкоротшого шляху в графі з невід'ємними вагами ребер. В контексті залізничних перевезень, коли ми шукаємо оптимальний маршрут, цей алгоритм має деякі переваги:

1. Ефективність: Алгоритм Дейкстри має складність O((V + E) log V), де V, E – кількість вершин та ребер відповідно. Це означає, що він швидко обробляє графи з великою кількістю вершин і ребер, що є важливим фактором для залізничних систем з багатьма станціями і маршрутами.

2. Невід'ємні ваги ребер: Алгоритм Дейкстри працює ефективно тільки з графами, де ваги ребер є невід'ємними. У випадку залізничних перевезень, це часто відповідає реальності, оскільки час або вартість подорожі не може бути від'ємним значенням.

3. Знаходження найкоротшого шляху: Алгоритм Дейкстри використовується для пошуку найкоротшого шляху між вершинами у графі. Це дозволяє визначити оптимальний маршрут залізничних перевезень, який мінімізує витрати часу або вартості, в залежності від ваг ребер.

4. Підтримка попередників: Алгоритм Дейкстри зберігає інформацію про попередників на шляху до кожної вершини. Це дозволяє відтворити сам маршрут, який включає оптимальний шлях, що знайшов алгоритм.

5. Гнучкість: Алгоритм Дейкстри може бути модифікований для врахування різних факторів, таких як обмеження швидкості, пересадки або

пріоритети. Це дозволяє використовувати його для визначення оптимального маршруту залежно від конкретних вимог і умов залізничних перевезень.

Загалом, алгоритм Дейкстри є потужним і ефективним інструментом для пошуку оптимального маршруту в залізничних перевезеннях, забезпечуючи швидкість, точність і можливість урахування різних факторів і обмежень.

1.3. Постановка задачі пошуку залізничних маршрутів пасажирських перевезень

Постановка задачі полягає в розробці системи пошуку залізничних маршрутів пасажирських перевезень з використанням теорії графів та створенні веб-додатку, що базується на цій системі. Завдання можна розбити на кілька етапів:

1. Розробка структур даних для організації зберігання даних графа:

1) Потрібно визначитися, як будуть представлені вузли і ребра графа в системі. Вузли можуть бути представлені станціями залізниці, а ребра - залізничними маршрутами, які з'єднують станції.

2) Для зберігання графа можна використовувати певну структуру даних, наприклад, список суміжності або матрицю суміжності. Вибір структури даних залежить від вимог ефективності операцій пошуку та масштабів задачі.

2. Розробка алгоритму пошуку оптимального маршруту:

1) Потрібно визначити параметр, за яким буде оцінюватись оптимальність маршруту, наприклад, найкоротший час подорожі, найменша кількість пересадок або найнижча ціна квитка.

2) Застосування певного алгоритму пошуку шляху в графі, такого як алгоритм Дейкстри, для знаходження оптимального маршруту між вказаними початковою та кінцевою станціями.

3) Врахування специфічних обмежень та умов, які пов'язані з пасажирськими перевезеннями, наприклад, розкладом руху поїздів, наявністю місць, можливими пересадками тощо.

3. Реалізація розробленого алгоритму у веб-додатку:

1) Розробка веб-інтерфейсу, що дозволяє користувачам вводити початкову та кінцеву станції та отримувати оптимальний маршрут.

2) Інтеграція алгоритму пошуку у веб-додаток і забезпечення зручної взаємодії користувача з системою.

3) Реалізація функціональності, яка дозволяє відображати додаткову інформацію про маршрути, таку як тривалість подорожі, наявність пересадок, розклад руху тощо.

4. Аналіз складності розробленого алгоритму:

1) Визначення часової та просторової складності алгоритму, тобто оцінка часу виконання та обсягу пам'яті, які необхідні для його реалізації.

2) Оцінка ефективності розробленого алгоритму порівняно з іншими алгоритмами пошуку шляхів в графах.

3) Виявлення можливих шляхів оптимізації алгоритму та покращення його продуктивності.

В результаті виконання цих завдань буде розроблена система пошуку залізничних маршрутів пасажирських перевезень на основі теорії графів, яка буде доступна через веб-додаток. Користувачі зможуть швидко знаходити оптимальні маршрути для своїх поїздок та отримувати додаткову інформацію про них.

Висновки до розділу 1

У розділі 1 розглянуто теорію графів та алгоритми пошуку оптимального шляху в графах. Пошук шляху є методом для обчислення найкоротшого маршруту з використанням алгоритмів для комп’ютера.

Основні задачі пошуку шляху в графі включають задачу найкоротших шляхів з одним вхідним вузлом та задачу найкоротших шляхів з одним вихідним вузлом.

Для розробки системи пошуку залізничних маршрутів пасажирських перевезень на основі теорії графів, необхідно визначитися, як будуть представлені вузли і ребра графа в системі, а також використовувати певну структуру даних для зберігання графа.

Розробка алгоритму пошуку оптимального маршруту включає визначення параметра, за яким буде оцінюватись оптимальність маршруту, застосування певного алгоритму пошуку шляху в графі та врахування специфічних обмежень та умов, які пов'язані з пасажирськими перевезеннями.

Реалізація розробленого алгоритму у веб-додатку включає розробку веб-інтерфейсу, що дозволяє користувачам вводити початкову та кінцеву станції та отримувати оптимальний маршрут, інтеграцію алгоритму пошуку у веб-додаток і реалізацію функціональності, яка дозволяє відображати додаткову інформацію про маршрути.

Аналіз складності розробленого алгоритму включає визначення часової та просторової складності алгоритму, тобто оцінка часу виконання та обсягу пам'яті, які необхідні для його реалізації.

РОЗДІЛ 2. РОЗРОБКА АЛГОРИТМУ ПОШУКУ МАРШРУТУ

2.1. Опис структур даних

Щоб зберегти інформацію в пам'яті комп'ютера, потрібно, щоб вона мала підходящий формат для комп'ютера. Якщо ця умова виконується, то потрібно обрати структуру, яка найкраще підходить для даної інформації і надає необхідний набір можливостей для роботи з нею [17].

Структура означає спосіб представлення інформації, де окремі елементи взаємно пов'язані між собою, утворюючи ціле. Якщо дані правильно скомпоновані і логічно пов'язані, вони можуть бути ефективно оброблені, оскільки загальна структура надає можливість управління ними. Це допомагає досягти високих результатів при розв'язанні різних завдань [17].

На відміну від класичних задач на пошук оптимального шляху, розробка алгоритму для пасажирських перевезень залізницею має свої особливості. Кожен потяг має початкову станцію, проміжні та кінцеву. Його маршрут не є оптимальним з точки зору відстані, часу та вартості. Але він є наперед визначеним. Крім того, час проходження відстані між населеними пунктами у кожного потяга різний і вартість також.

Враховуючи ці особливості треба визначити ті структури даних, які в найпростішій формі дозволять представити граф, який буде відображенням карти потягів.

Для розробки системи пошуку залізничних маршрутів пасажирських перевезень з використанням теорії графів в мові програмування Python [12] необхідно використовувати наступні структури даних:

1) Структура даних для збереження населених пунктів:

class Settlement:

```
def __init__(self, id, name):
    self.id = id
    self.name = name
```

Дана структура містить два поля: id (номер) та name (назва) населеного пункту.

2) Структура даних для збереження інформації про потяги:

```
class Train:
    def __init__(self, id, settlements):
        self.id = id
        self.settlements = settlements
```

Дана структура містить два поля: id (номер) потяга та settlements (послідовність населених пунктів), які представлені списком або масивом їх номерів.

3) Квадратна матриця для збереження інформації про час та вартість руху між населеними пунктами:

```
class Schedule:
    def __init__(self, size):
        self.matrix = [[[] for _ in range(size)] for _ in range(size)]

    def add_connection(self, source, destination, train_id, time, cost):
        self.matrix[source][destination].append({'train_id': train_id, 'time': time, 'cost': cost})
```

Дана структура містить квадратну матрицю розмірністю, що відповідає кількості населених пунктів. Кожна комірка матриці містить список словників,

які є зв'язками між населеними пунктами. Кожен словник містить поля: train_id (номер потяга), time (час проходження) та cost (вартість руху).

Ці структури даних дозволять представити граф, де вершинами є населені пункти, а ребрами відображаються зв'язки між ними у вигляді потягів з відповідними часом та вартістю руху. За допомогою цих структур можна здійснювати пошук оптимальних залізничних маршрутів.

2.2. Розробка алгоритму

Графові моделі, як математичні моделі, широко використовуються для моделювання різноманітних явищ, процесів і систем. Це дозволяє вирішувати багато теоретичних і прикладних завдань, використовуючи аналіз графових моделей. Існує набір типових алгоритмів обробки графів, які можна застосовувати для цих задач.

Існують два стандартних способи представлення графа G = (V, E): списки суміжних вершин або матриця суміжності. Ці способи використовуються для орієнтованих і неорієнтованих графів.

Алгоритм Дейкстри [4] використовується для знаходження найкоротшого шляху з однієї вершини у вагованому орієнтованому графі G = (V, E), коли ваги ребер є невід'ємними. Для всіх ребер (u, v) ∈ E виконується умова $w(u, v) \geq 0$.

У алгоритмі Дейкстри використовується множина вершин X, для яких вже враховані остаточні ваги найкоротших шляхів зі стартової вершини s. На кожному кроці алгоритму вибирається вершина u ∈ V[G] - X з найменшою оцінкою найкоротшого шляху.

На рисунку 2.1 зображено ілюстрацію алгоритму Дейкстри. Стартова вершина s розташована зліва від інших вершин. У кожній вершині показана

оцінка найкоротшого шляху до неї, а виділені ребра вказують попередників. Чорним кольором позначені вершини, які були додані до множини X, а білим - вершини, що ще не належать до V - X.

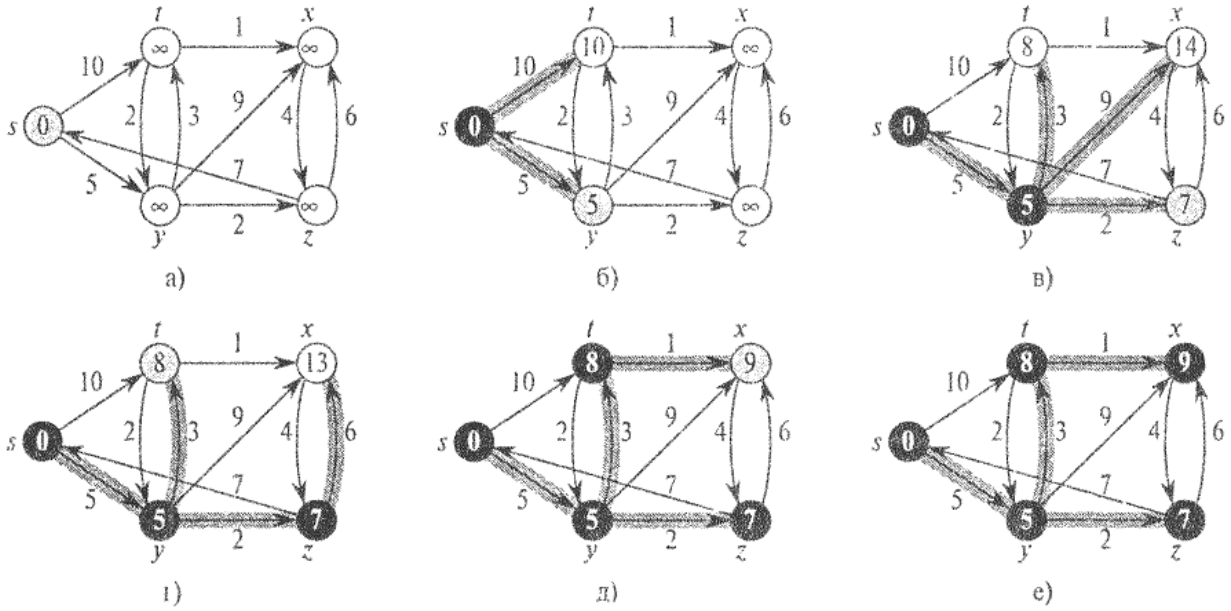

Рис. 2.1. Виконання алгоритму Дейкстри

Реалізація даного алгоритму псевдокодом представлена далі [4].

```
Dijkstra(G, s)
for (для) кожної вершини v є V[G] do
 A[v] ← ∞; B[v] ← NIL
A[s] ← 0
X ← {s}; v ← s
while X ≠ V[G]:
  for (для) каждого ребра (v, u) є E[G] та u є V[G] – X:
    if A[u]>A[v]+w(v, u) then
      A[u] ← A[v]+w(v, u)
      B[u] ← v
  for серед усіх значень вершин з V[G]– X знайти v* з min значенням A[v]
    X ← X + {v*}; v = v*
  return A, B
```

На рисунку, частина а) демонструє початкову ситуацію перед першою ітерацією циклу while у рядках 5-11. Виділена сірим кольором вершина обрана як вершина "u" для наступної ітерації. У частинах б) - е) показані ситуації після виконання наступної ітерації циклу while. У кожній з цих частин виділена

сірим кольором вершина обрана як вершина "u". У частині е наведені кінцеві значення найкоротших шляхів.

Алгоритм працює наступним чином: на початку, у рядках 1-4, ініціалізуються значення "A[v]" і "B[v]". Початкове значення найкоротшої відстані від початкової вершини "s" до всіх інших вершин "v" у множині "V" встановлюються як нескінченні (A[v] = ∞). Масив "B" використовується для збереження попередників у найкоротшому шляху. У рядку 3 вказується, що найкоротша відстань від "s" до себе ж дорівнює 0. У рядку 4 ініціалізується множина "X", яка спочатку містить лише початкову вершину "s" і вказується, що її попередником є сама "s".

В алгоритмі дотримується інваріант, що цикл while виконується до тих пір, поки множина "X" не містить всі вершини графу "G". У кожній ітерації циклу while розглядаються всі ребра, що виходять з вершини "v" до всіх вершин, які належать до множини "V - X". Якщо поточний найкоротший шлях до вершини "u" може бути поліпшений, проходячи через вершину "v", оновлюється значення "A[u]" і попередник "B[u]".

Після цього обирається нова поточна вершина "v*", яка має мінімальне значення "A[v*]", і додається до множини "X". Оскільки в алгоритмі Дейкстри з множини "V - S" завжди вибирається найлегша або найближча вершина для додавання до множини "X", алгоритм вважається жадібним.

Таким чином, робота розглянутого алгоритму Дейкстри полягає у послідовному оновленні найкоротших шляхів до вершин, використовуючи жадібний підхід, поки всі вершини не будуть включені до множини "X".

Оскільки в алгоритмі Дейкстри з множини V - S для приміщення в множину X завжди вибирається сама "легка" або "близька" вершина, кажуть, що цей алгоритм дотримується жадібної стратегії.

Для побудови графа, що моделює залізничні пасажирські перевезення, необхідно визначитися зі структурою вузлів та ребер.

У звичайних обставинах, час та вартість поїздки не є прямо пропорційними. Скоріше навпаки, вони обернено пропорційні. Чим швидше пасажир пройде відстань між населеними пунктами, тим дорожче обійдеться поїздка. Тому говорити про абсолютно оптимальний маршрут не має сенсу.

Крім того, часто виникають пересадки, що є найбільш незручною частиною пасажирських перевезень.

Тому був розроблений наступний алгоритм пошуку оптимального маршруту за окремими критеріями - вартістю та часом (рис. 2.2). Замість пошуку єдиного найкоротшого маршруту, цей алгоритм визначає оптимальний маршрут залежно від обраного критерію, будь то вартість або час подорожі.

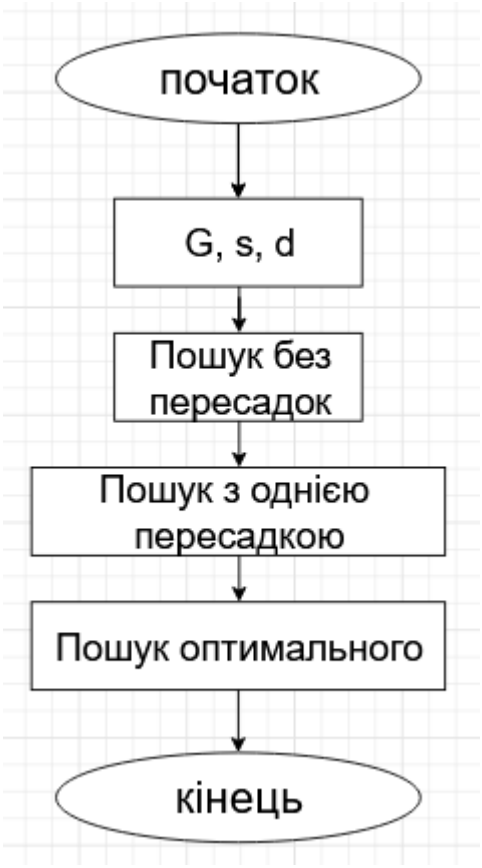

Рис. 2.2. Алгоритм пошуку маршрутів

Граф G представляє собою мережу населених пунктів, де вузлами є самі пункти, а ребра відповідають мінімальним значенням певного критерію

(наприклад, оптимальність, час або ціна), s визначає початковий вузол маршруту, а d - кінцевий вузол.

Пошук маршруту без пересадок полягає в виборі найменшого значення критерію з прямих поїздів між вузлами.

Пошук маршруту з однією пересадкою включає формування масиву маршрутів між вузлами, шляхом використання пошуку в глибину на два вузли. Потім вибирається маршрут з найменшим значенням критерію.

Пошук оптимального маршруту за певним критерієм виконується за допомогою алгоритму Дейкстри, який був описаний вище.

Отриманий оптимальний шлях може збігатися з одним зі шляхів, знайдених на попередніх етапах.

2.3. Складність розробленого алгоритму

Алгоритм, який був розроблений, складається з трьох послідовних кроків:

1) На першому кроці ми шукаємо найменше значення параметру зі списку. Оскільки список не впорядкований, складність цього кроку алгоритму становить O(n), де n - кількість прямих поїздів між населеними пунктами.

2) На другому кроці ми визначаємо кількість маршрутів з однією пересадкою. У найгіршому випадку, глибина підграфа з двома вузлами може бути такою ж, як і початковий граф. Складність алгоритму пошуку в ширину в цьому кроці становить O(|V| + |E|), де |V|, |E| - кількість вузлів та ребер графа відповідно.

3) На третьому кроці застосовується алгоритм Дейкстри. Його складність складає O((|V| + |E|) log(|V|)), де |V| - кількість вузлів, а |E| - кількість ребер графа.

Таким чином, загальна складність розробленого алгоритму пошуку шляху становить O((|V| + |E|) log(|V|)), де |V| - кількість вузлів, а |E| - кількість ребер графа.

Висновки до розділу 2

У розділі 2 дано визначення структур даних для ефективної обробки інформації в алгоритмах пошуку маршруту. Визначення правильних структур даних є ключовим етапом у розробці алгоритмів, оскільки це надає можливості для ефективної обробки даних та досягнення оптимальних результатів.

У розділі описано важливість визначення структур даних для зберігання інформації про населені пункти та потяги, а також про час та вартість руху між ними. Для цього було запропоновано використання структур даних, таких як Settlement та Train, а також квадратної матриці для зберігання часу та вартості руху між населеними пунктами.

Отже, правильне визначення структур даних та урахування особливостей задачі є важливими етапами у розробці алгоритмів пошуку маршруту для пасажирських перевезень.

Для пошуку найкоротшого шляху з однієї вершини у вагованому орієнтованому графі було запропоновано використання алгоритму Дейкстри. У псевдокоді алгоритму було детально описано кожен крок алгоритму.

Отже, розділ 2 надає інформацію про процес розробки алгоритму для пошуку маршруту залізничного транспорту з використанням теорії графів. Це необхідні знання про структури даних, моделі на графах та алгоритми, які можуть бути використані для розробки подібних систем.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ АЛГОРИТМУ

3.1. Вибір платформи роботи додатку

Було вирішено розробити веб-додаток. Для розробки сайту було вибрано мову програмування Python [12] та фреймворк Django [9-16].

Django був розроблений для швидкого та простого виконання загальних завдань веб-розробки із врахуванням швидкого розвитку середовища редакції новин. Ось огляд того, як створити веб-додаток, що працює з базою даних, за допомогою Django.

Проектування моделі

Хоча Django можна використовувати і без бази даних, він має вбудоване об'єктно-реляційне відображення, за допомогою якого описується структура бази даних у вигляді Python-коду. Синтаксис моделі даних надає багато способів представлення даних, що дозволяє використовувати різні системи управління базами даних. Модель є єдиним джерелом інформації про дані. Вона включає основні поля та поведінку, яку потрібно зберігати. Зазвичай кожна модель відображається в таблиці бази даних.

Основні принципи

1) Кожна модель є класом Python, що є підкласом django.db.models.Model.

2) Кожне поле моделі - це поле бази даних.

3) Django автоматично надає доступ до API функцій бази даних, який генерується автоматично.

Використання моделей

Після того, як ви визначили свої моделі, вам потрібно повідомити Django, що ви плануєте використовувати ці моделі. Для цього ви відредагуєте

файл налаштувань і змініте параметр INSTALLED_APPS, додавши ім'я модуля, який містить ваш файл models.py.

Наприклад, якщо ваші моделі розташовані у модулі myapp.models (це структура пакета, яка створюється для програми за допомогою команди manage.py startapp), параметр INSTALLED_APPS буде виглядати наступним чином:

```
INSTALLED_APPS = [
    ...
    'myapp',
    ...
]
```

Цим ви включаєте вашу програму (модуль) до списку встановлених додатків Django, що дає Django знати, що ви плануєте використовувати моделі з цього модуля у вашому проекті.

При додаванні нових програм до INSTALLED_APPS обов'язково запустіть команду manage.py migrate. У разі потреби, спочатку зробіть міграцію для цих програм за допомогою команди manage.py makemigrations.

Поля

Найважливіша і обов'язкова частина моделі - це список полів бази даних, які вона визначає. Поля визначаються як атрибути класу.

Типи полів

Кожне поле у вашій моделі повинно бути екземпляром відповідного класу Field. Django використовує класи полів для визначення кількох речей:

- Тип стовпця, який повідомляє базі даних, який тип даних буде зберігатися (наприклад, INTEGER, VARCHAR, TEXT).

- HTML-віджет за замовчуванням, який використовується при візуалізації поля форми (наприклад, `<input type="text">`, `<select>`).

- Мінімальні перевірки, які використовуються в адміністративному інтерфейсі Django та автоматично генерованих формах.

Django має вбудовані десятки типів полів; повний список можна знайти у документації про поля моделі. У випадку, якщо вбудовані типи полів Django не задовольняють ваші потреби, ви можете легко створити власні поля, дотримуючись інструкцій у документації "Як створити власні поля моделі користувача".

Опції полів моделі

Кожне поле приймає специфічний набір аргументів, які описані в документації для полів моделі. Наприклад, для поля CharField (і його підкласів) потрібно вказати аргумент max_length, який визначає максимальну довжину поля VARCHAR у базі даних для зберігання даних.

Крім того, є загальні аргументи, які доступні для всіх типів полів. Вони є необов'язковими і повністю описані на сторінці опцій полів моделі. Ось короткий огляд найбільш часто використовуваних опцій:

- null: Якщо встановлено значення True, Django зберігатиме порожні значення у базі даних. За замовчуванням це значення встановлено на False.

- blank: Якщо встановлено значення True для поля, його можна залишити порожнім. За замовчуванням це значення встановлено на False. Варто зазначити, що ця опція відрізняється від опції null. Опція null застосовується лише до бази даних, тоді як опція blank використовується для перевірки поля. Якщо поле має значення blank=True, форма дозволить ввести порожнє значення. Якщо поле має значення blank=False, поле є обов'язковим.

- default: Значення, що використовується за замовчуванням для поля. Це може бути конкретне значення або функція/об'єкт, які викликаються при створенні нового об'єкта.

- help_text: Додатковий текст, що надає пояснення до поля при відображенні його в формі. Це корисно для документації, навіть якщо поле не використовується у формі.

- primary_key: Якщо поле має значення True, воно встановлюється як первинний ключ для моделі. Якщо для жодного з полів моделі не вказано primary_key=True, Django автоматично додасть поле IntegerField для зберігання первинного ключа. Тому, якщо ви не бажаєте перевизначати поведінку первинного ключа за замовчуванням, вам не потрібно встановлювати primary_key=True для будь-якого з полів. Додаткову інформацію можна знайти в розділі про автоматичні поля первинного ключа.

Перш за все, поле первинного ключа можна лише читати. Якщо ви зміните значення первинного ключа для існуючого об'єкта і збережете його, створиться новий об'єкт разом із старим.

Тепер щодо відношень. Очевидно, що перевага реляційних баз даних полягає в зв'язку таблиць між собою. Django пропонує способи визначення трьох найпоширеніших типів відносин з базою даних: багато-до-одного (many-to-one), багато-до-багатьох (many-to-many) і один-до-одного (one-to-one).

Відношення багато-до-одного визначається за допомогою поля django.db.models.ForeignKey. Ви використовуєте його так само, як будь-який інший тип поля, додаючи його як атрибут до класу вашої моделі. ForeignKey вимагає позиційного аргументу - класу, до якого відноситься модель.

Відношення багато-до-багатьох визначається за допомогою поля ManyToManyField. Ви використовуєте його так само, як будь-який інший тип

поля, додаючи його як атрибут до класу вашої моделі. ManyToManyField також вимагає позиційного аргументу - класу, до якого відноситься модель.

ManyToManyField вимагає передачі позиційного аргументу: класу, до якого відноситься модель. Так само, як з ForeignKey, ви можете створити рекурсивні відношення (коли об'єкт має зв'язок "багато хто до багатьох" з самим собою) або коли відносини з моделями ще не визначені.

Хоча передбачається, але не є обов'язковим, щоб ім'я ManyToManyField було множиною, яке описує набір пов'язаних об'єктів моделі. Важливо, що модель, яка має ManyToManyField, повинна бути вказана лише в одній з моделей, а не в обох. Зазвичай екземпляри ManyToManyField розміщуються в об'єкті, який редагується у формі.

Динамічний інтерфейс адміністратора в Django - це не просто інструмент для будівництва, а повноцінний готовий до використання будинок.

Після визначення ваших моделей, Django може автоматично створити професійний адміністративний інтерфейс - веб-сайт, який дозволяє автентифікованим користувачам додавати, змінювати та видаляти об'єкти. Щоб це стало можливим, вам потрібно лише зареєструвати вашу модель на сторінці адміністратора.

Головна ідея полягає в тому, що ваш веб-сайт може бути редагований персоналом, клієнтами або навіть вами самими, без необхідності розробки внутрішніх інтерфейсів лише для управління контентом.

Один з типових робочих процесів у Django - це створення моделей і швидке запуск адміністративного сайту, щоб співробітники або клієнти могли почати вводити дані. При цьому ви можете продовжувати працювати над публічною частиною вашого проекту.

Проектування URL

Одним з важливих аспектів високоякісного веб-додатку є чиста і елегантна структура URL. Django підтримує принципи красивого дизайну URL і уникне додавання зайвих елементів, таких як .php або .asp.

Для проектування URL-адрес в програмі вам потрібно створити модуль Python, відомий як URLconf. Цей модуль містить прості зіставлення між шаблонами URL і функціями зворотного виклику Python. URLconf також дозволяє відокремити URL-адреси від коду Python.

Написання представлень

Кожне представлення відповідає за виконання однієї з двох дій: повертає об'єкт HttpResponse з вмістом для запитуваної сторінки або генерує виняток, наприклад Http404. Все інше залежить від вашої власної логіки.

Зазвичай, представлення отримують дані залежно від параметрів, завантажують шаблон і відображають його з отриманими даними.

Шаблони в Django

У кожному веб-фреймворку необхідно мати зручний механізм для генерації HTML-файлів. У Django для цього використовуються шаблони - окремі HTML-файли, які взаємодіють між собою і містять базові логічні операції.

Шаблони можна використовувати для створення сайтів з великою кількістю сторінок, від сотень до мільйонів, з використанням мінімальної кількості коду.

Спочатку потрібно визначити, де розмістити шаблони всередині структури Django-проекту. Існують два підходи. За замовчуванням, завантажувач шаблонів Django проглядає кожен додаток, шукаючи пов'язані з ним шаблони. Однак така структура може бути дещо заплутаною: кожен додаток потребує окремої директорії "templates", ще однієї директорії зі своїм власним ім'ям та файлу шаблону.

Чому ми використовуємо цей, на перший погляд, повторюваний підхід? Коротка відповідь полягає в тому, що завантажувач шаблонів Django має точно знайти відповідний шаблон! Що станеться, якщо дві різні програми мають файли "home.html"? Використання такої структури гарантує відсутність конфліктів такого роду.

Існує ще один підхід до вирішення цього питання - створення однієї директорії "templates" на рівні проекту і розміщення всіх шаблонів там. Зробивши невеликі зміни у файлі "settings.py", можна вказати Django, щоб він шукав шаблони також у цій новій директорії.

3.2. Вибір середовища розробки

Редактор коду – це зручне програмне забезпечення, аналогічне стандартному редактору тексту. Відмінність полягає в тому, що кодові редактори адаптовані спеціально під написання рядків коду, а також мають ряд специфічних можливостей:

- підсвічування синтаксису, яке підсвічує ключові слова, функції, виділяє змінні;
- перевірка та автоматичне встановлення відступів, які використовуються в багатьох мовах програмування, але в Python це важливі атрибути синтаксису;
- автоформатування, що підкреслює неточності та звертає увагу програміста на помилку;
- редагування файлів та збереження проектів у потрібному розширенні.

IDE - це цілий програмний комплекс. Інтегроване середовище поєднує в собі відразу кілька важливих для комфортної розробки програмного забезпечення інструментів. У складі IDE можна побачити як вбудований кодовий редактор. У будь-якому інтегрованому середовищі передбачені:

- редактор коду;
- транслятор (компілятор чи інтерпретатор);
- статистичний аналізатор та відладчик;
- візуальний редактор;
- система роботи з файлами та управління версіями;

•ряд інших функцій, що спрощують процес програмування.

Редактор коду та IDE дуже відрізняються від стандартного редактора тексту. Редактор коду має меншу вагу та більш швидкісну роботу, але поступається IDE за функціоналом. Інтегроване середовище розробки відкриває перед програмістом більше можливостей, особливо за наявності детальніших знань. Одна IDE може підтримувати кілька мов програмування та мати багато можливостей. Тому середовище розробки займає більше місця, ніж редактор коду, але в його установку потрібно більше часу.

Набір інструментів та опцій різних IDE, які підтримують кодування на Python, відрізняється. Але ряд функцій є базовим - передбачений у будь-якому інтегрованому середовищі розробки для Python.

У IDE, комфортному для кодування на Python, є:

•Редактор коду з усіма його можливостями. Особливе значення в IDE для Python має компактне, але функціональне вікно для редагування тесту та підсвічування елементів синтаксису. Це дозволяє краще орієнтуватися в тексті та писати рядки коду швидше.

•Інструменти збирання та виконання команд. IDE дозволяє запускати код, написаний на Python, безпосередньо із середовища без допомоги сторонніх компіляторів.

•Статистичний аналізатор та відладчик. IDE дають можливості для пошуку помилок у коді Python. Наприклад, IDE можуть виконувати код Python за кроками або зупинити його виконання при досягненні певної точки.

•Кастомайзер. Багато IDE налаштовується під конкретного користувача. У деяких IDE доступний вибір теми, колірної гами, положення та розміру тексту, розташування вікон та гарячих клавіш. Для багатьох програмістів важлива темна тема, від якої менше втомлюються очі. Просунуті користувачі можуть підключити необхідні плагіни, встановити бібліотеку, фреймворки або додаткові розширення.

•Система управління проектами та версіями. Щоб написати будь-який програмний продукт на Python, потрібен час. Тому потрібно, щоб IDE могла зберегти код у поточному стані, а за необхідності повернутися до його попередньої версії. Багато IDE підтримують комплексну роботу з файлами, наприклад формування ієрархічної системи папок.

PyCharm

IDE, розроблена для Python і доступна на різних ОС - Microsoft Windows, Linux, MacOS. Безкоштовної версії достатньо для вирішення більшості завдань.

Переваги:

- безліч безкоштовних інтегрованих функцій;

- велике та товариське користувальницьке ком'юніті;

- можливість встановлення фреймворків, плагінів та доповнень;

- готовність до роботи.

Недоліки:

- повільний запуск;

- для серйозних проектів краще придбати платну версію;

- великий об’єм.

IDLE

Стандартний програмний комплекс, що за замовчуванням йде в комплекті з Python. IDE ідеальна для розуміння азів у програмуванні та розумінні мови. Підходить для початку роботи з Python, але для написання серйозних проектів її функціоналу недостатньо.

Переваги:

- встановлення «за замовчуванням»;

- наявність базових інструментів;

- зрозумілий інтерфейс;

- невеликий об’єм.

Недоліки:

- мінімальний набір функцій;

- обмежені можливості для кастомізації;

- низька продуктивність.

Microsoft Visual Code

Кросплатформний багатопрофільний редактор коду, розроблений Microsoft. ПЗ регулярно розширюється і доповнюється новими функціями. Зручний у використанні, працює без збоїв.

Переваги:

- вбудований маркетплейс із безліччю доповнень;

- легке встановлення Python або будь-якого іншого ЯП;

- можливість налагодження та запуску коду на деяких ЯП, система контролю версій;

- доступний безкоштовно.

Недоліки:

- повільна робота на старих комп'ютерах;

- не є окремим додатком, оскільки написаний на Electron;

- відносно великий об’єм, але все ще менший, ніж у однойменної IDE.

Було вибрано редактор коду Microsoft *Visual Code*.

3.3. Реалізація алгоритму

Почнемо зі створення проекту.

Для налаштування Django [9-16] використовується командний рядок. Командний рядок є потужним інструментом, який дозволяє виконувати операції через текст і широко використовується розробниками.

Використання віртуального оточення Python, такого як pipenv, є важливою складовою програмування на Python. Це ізольовані контейнери, які містять необхідні програмні інструменти для конкретного проекту. Вони особливо корисні, оскільки Python і Django за замовчуванням встановлюються в одну директорію.

У світі програмування існує безліч дискусій щодо вибору різних інструментів. Проте, коли мова йде про віртуальні оточення Python, спору не виникає. Ми повинні використовувати окреме віртуальне оточення для кожного проекту Python.

Для управління віртуальними оточеннями ми будемо використовувати Pipenv. Pipenv схожий на npm і yarn з екосистеми JavaScript/Node: він створює файл Pipfile, в якому знаходяться необхідні програмні інструменти, а також Pipfile.lock для забезпечення детермінованих збірок. "Детермінованість" означає, що при кожному завантаженні проекту в нове віртуальне оточення конфігурація залишатиметься незмінною.

Для встановлення Pipenv використовується pip. Виконайте команду:

python3 -m pip install pipenv

Тепер ми використовуємо Pipenv для встановлення Django.

Виконайте команду `pipenv install django`. Якщо ви переглянете вміст нашої папки, побачите, що з'явилися два нові файли: Pipfile та Pipfile.lock. Тепер у нас є повна інформація, необхідна для створення нового віртуального оточення, але наразі нічого не активовано. Ми виправимо це за допомогою команди `pipenv shell`.

Тепер ми створюємо новий проект Django з назвою `train_schedule` за допомогою наступної команди: `django-admin startproject train_schedule .`. Результатом виконання цієї команди буде створення дерева нових файлів і каталогів, а саме:

- `manage.py`: утиліта, яка забезпечує різні способи взаємодії з проектом.

- Внутрішній каталог `train_schedule/`: це Python-модуль нашого проекту. Ми будемо використовувати його назву для імпорту речей з цього модуля (наприклад, `train_schedule.urls`).

- `train_schedule/__init__.py`: це порожній файл, який вказує Python на те, що цей каталог повинен бути розглянутий як пакет Python.

- `train_schedule/settings.py`: конфігурація та налаштування проекту Django.

- `train_schedule/urls.py`: URL-вказівник проекту Django можна розглядати як "навігацію" або "структуру" вашого проекту.

- `train_schedule/asgi.py`: точка входу для веб-серверів, сумісних з ASGI, яка служить для обробки вашого проекту.

- `train_schedule/wsgi.py`: точка входу для веб-серверів, сумісних з WSGI, яка використовується для опрацювання вашого проекту.

Кожна програма, яку ви розробляєте в Django, має певну структуру каталогів, якої ви повинні дотримуватися. Для спрощення цього процесу Django постачається з утилітою, яка автоматично створює основну структуру каталогів для вашої програми.

Щоб створити нову програму, переконайтеся, що ви перебуваєте в тому самому каталозі, у якому знаходиться файл `manage.py`, і запишіть команду:

python manage.py startapp schedule

Це створить новий пакет Python з назвою "schedule" і відповідною структурою каталогів для вашої програми.

Це створить каталог, який виглядає так (рис 3.1):

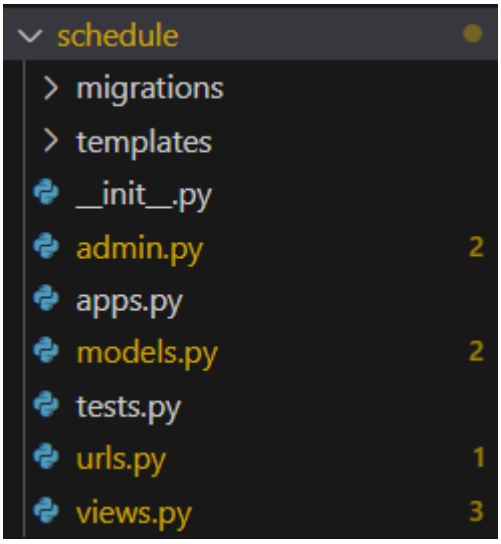

Рис 3.1. Каталог додатку «Розклад»

Переконфігуруємо налаштування бази даних:

Файл `settings.py` є стандартним модулем Python, в якому знаходяться змінні налаштувань для Django.

За замовчуванням, в конфігурації використовується SQLite.

Параметр `INSTALLED_APPS` містить перелік імен усіх програм Django, які активовані в цьому проекті Django. Програми можуть використовуватись у кількох проектах, і ви можете збирати та поширювати їх для використання іншими розробникам.

Зараз ми визначимо моделі, які представляють структуру нашої бази даних разом з додатковими метаданими.

Модель є одним і єдиним джерелом інформації про ваші дані. Вона встановлює основні поля та поведінку даних, які ви зберігаєте. Django пропагує принцип DRY (Don't Repeat Yourself), що означає, що ви визначаєте вашу модель даних в одному місці і маєте автоматичний доступ до цих даних через цю модель.

Це включає міграції - на відміну від інших фреймворків, таких як Ruby On Rails, міграції Django повністю виникають з файлів ваших моделей. Практично, це є історія, яку Django може виконувати, щоб оновити схему вашої бази даних, відповідно до вашої поточної моделі.

Кожна модель представляється класом, який є підкласом `django.db.models.Model`. У кожної моделі є кілька змінних класу, що представляють поля бази даних моделі.

У Django кожне поле представлене об'єктом класу Field. Наприклад, для зберігання символьних даних використовується клас CharField, а для полів, що містять дату і час, використовується клас DateTimeField. Це дозволяє Django визначити тип даних, який зберігається в кожному полі.

Ім'я кожного екземпляра Field використовується у машинно-зручному форматі. Ви будете використовувати це ім'я у своєму коді Python, а ваша база даних використовуватиме його як ім'я стовпця.

Давайте створимо модель (див. рис. 3.2) і перевизначимо метод __str__(self) для зручного відображення моделі в адмін-панелі.

Рис 3.2. Модель «Розклад»

Тепер, коли ми створили нову модель бази даних, ми повинні створити нову міграцію та застосувати зміни до нашої бази даних.

Виконайте наступні команди:

python manage.py makemigrations

python manage.py migrate

Ці дві команди створять та застосують міграції, що внесуть зміни до структури бази даних.

Адміністративна частина Django

Створення адміністративного сайту для співробітників або клієнтів, які можуть додавати, змінювати та видаляти контент, може бути рутинною задачею, яка не вимагає великої творчості. Тому Django повністю автоматизує процес створення адміністративного інтерфейсу для моделей.

Django був розроблений зі сфокусованістю на новинах, з чітким розрізненням між "видавцями контенту" і "публічним" сайтом. Менеджери

сайту використовують систему для додавання новин, подій, результатів спортивних змагань тощо. Цей контент відображається на загальнодоступному сайті. Django вирішує проблему створення єдиного інтерфейсу для адміністраторів сайту, який дозволяє редагувати контент.

Адміністративна частина не призначена для використання відвідувачами сайту. Вона призначена для менеджерів.

Спочатку нам потрібно створити користувача, який зможе увійти на адміністративний сайт. Виконайте наступну команду:

python manage.py createsuperuser

Тепер нам залишилося зробити ще одну річ - повідомити адміністратору, що об'єкти типу Schedule мають мати доступ до адміністративного інтерфейсу. Для цього відредагуємо файл schedule/admin.py, додаючи імпорт моделі та реєстрацію

from .models import Schedule

admin.site.register(Schedule)

Далі, запустимо сервер розробки:

python manage.py runserver

Зараз відкрийте веб-браузер і введіть шлях `/admin/` на вашому локальному сервері, наприклад, http://127.0.0.1:8000/admin/.

Тепер ми можемо додавати нові елементи (наприклад, розклад руху поїздів) до таблиці (див. рис. 3.3), переглядати вміст таблиць (див. рис. 3.4) і змінювати існуючі записи (див. рис. 3.5).

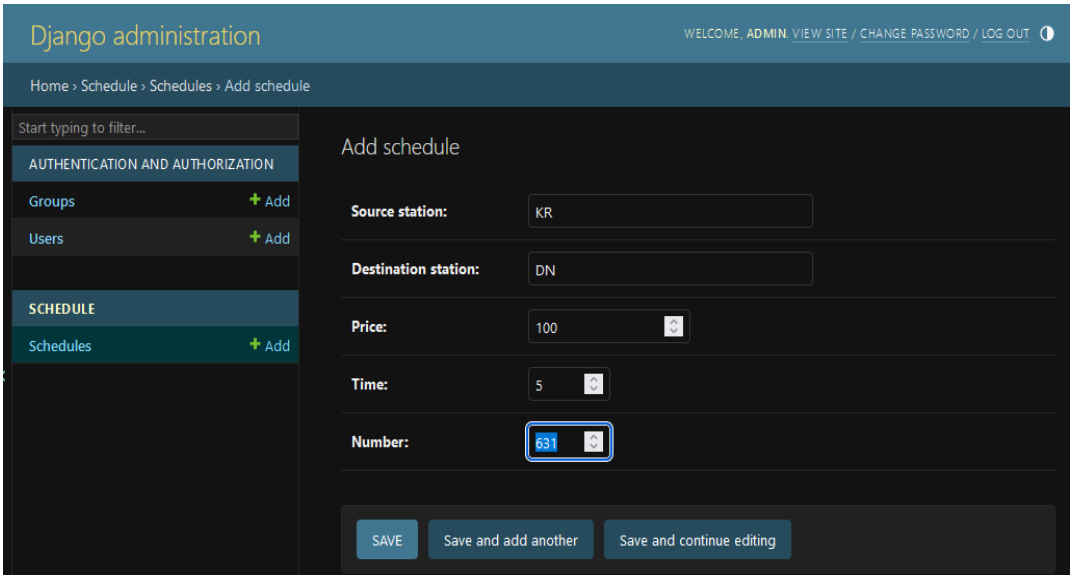

Рис. 3.3. Додавання об’єкта до таблиці

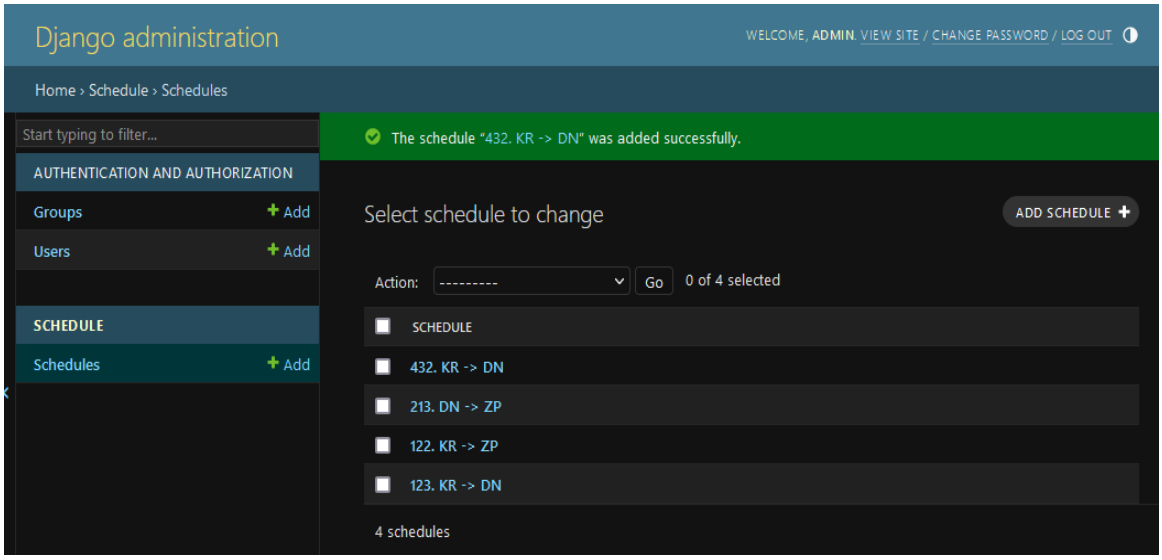

Рис. 3.4. Перегляд змісту таблиці

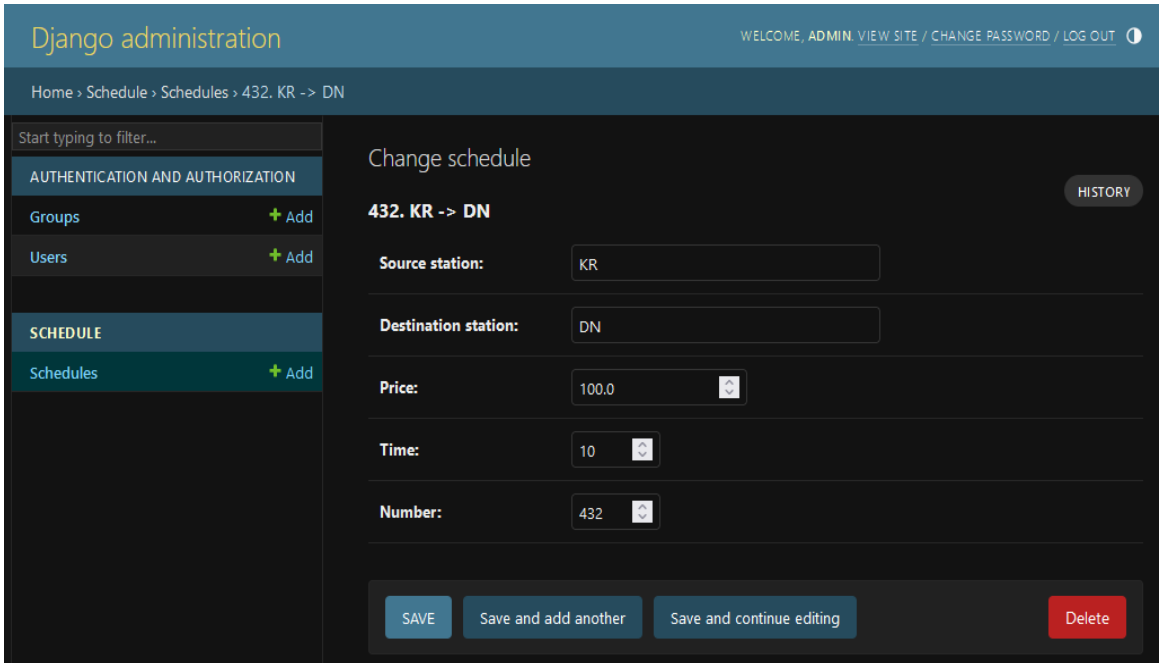

Рис.3.5. Зміна елемента таблиці

Залишилось зробити представлення, шаблони та налаштувати роути для нашого додатку.

Наш додаток буде складатись з двох сторінок: сторінки з формою для вибору станцій і сторінки з отриманими результатами.

Для цього нам потрібно створити два шаблони - файли home.html (див. рис. 3.6) і result.html (див. рис. 3.7).

```
<form action="{% url 'result' %}" method="post">
    {% csrf_token %}
    <label> звідки </label>
    <input type="text" name="source"/>
    <label> relb </label>
    <input type="text" name="destination"/>
    <input type="submit" value="Знайти маршрут">
</form>
```

Рис. 3.6. Шаблон форми

```
{% if alls %}
    <h3>Всі результати</h3>
    <ul>
    {% for item in alls %}
        <li>{{ item.source_station}} -> {{ item.destination_station}} ({{ item.price}} грн, {{item.time}} хв)</li>
    {% endfor %}
    </ul>
    <br>
    <h3>Оптимальний за часом без пересадок</h3>
    {{ time.source_station}} -> {{ time.destination_station}} ({{ time.price}} грн, {{time.time}} хв)
    <h3>Оптимальний за вартістю без пересадок</h3>
    {{ price.source_station}} -> {{ price.destination_station}} ({{ price.price}} грн, {{price.time}} хв)
    {% else %}
    <p>Маршрутів не знайдено</p>
{% endif %}
```

Рис. 3.7. Шаблон результату оптимального пошуку

Після цього створимо представлення, яке буде використовувати ці шаблони.

Представлення home буде відповідальне за відображення контенту на шаблоні home.html. Воно буде показувати форму для вибору станцій.

Представлення result отримуватиме POST-запити від форми на сторінці home.html. Воно буде проводити валідацію введених даних, виконувати запит до бази даних і знаходити оптимальний маршрут залежно від ціни та часу подорожі, як без пересадок, так і з однією пересадкою.

Після виконання пошуку, представлення result відобразить результати на шаблоні result.html.

Додатково, ми передбачаємо, що можлива лише одна пересадка в маршруті.

Висновки до розділу 3

У розділі 3 було розглянуто реалізацію алгоритму для веб-додатку "Розклад". Було вирішено розробити веб-додаток на мові програмування Python з використанням фреймворку Django. Django був обраний через його швидкість та простоту виконання загальних завдань веб-розробки.

Для розробки веб-додатку було створено дві сторінки: сторінку з формою для вибору станцій та сторінку з отриманими результатами. Для цього було створено два шаблони - файли home.html та result.html.

Для зберігання даних було використано об'єктно-реляційне відображення Django, яке дозволяє описувати структуру бази даних у вигляді Python-коду. Кожна модель відображається в таблиці бази даних, а кожне поле моделі представляє поле бази даних.

Для відображення контенту на шаблоні було створено два представлення: home та result. Представлення home відповідає за відображення форми для вибору станцій, а представлення result отримує POST-запити від форми на сторінці home.html, проводить валідацію введених даних, виконує запит до бази даних та знаходить оптимальний маршрут залежно від ціни та часу подорожі. У розділі також було розглянуто переконфігурування налаштувань бази даних та параметр INSTALLED_APPS, який містить перелік імен усіх програм Django, які активовані в цьому проекті Django.

Отже, розділ 3 - це опис реалізації алгоритму для веб-додатку "Розклад" з використанням мови програмування Python та фреймворку Django.

ВИСНОВКИ

В результаті виконаної роботи досягнуто поставленої мети. Розроблено алгоритм пошуку залізничного маршруту пасажирських перевезень.

В результаті виконаної роботи можна зробити такі висновки:

1. Виконаний аналіз методів пошуку шляхів в графах показав їх загальність. Вони не враховують специфіку можливих типів даних, та інших особливостей конкретних задач

2. Розроблено структури даних для організації зберігання даних графа. Вони містять всі необхідні для алгоритму дані, визначено їх типи та склад.

3. Було проведено аналіз існуючих методів розв'язання даної проблеми та визначено, що найбільш ефективним є використання алгоритму Дейкстри. Розроблено алгоритм пошуку оптимального за вибраним параметром маршруту. Його суть полягає в послідовному виборі оптимальних маршрутів без пересадок, з однією пересадкою та оптимальний за параметром.

4. Виконано аналіз складності розробленого алгоритму. Вона складає O(|(V| + |E|) log (|(V|)).

5. Для реалізації даного алгоритму було обрано мову програмування Python та фреймворк Django.

Отже, в результаті виконання даної роботи було успішно реалізовано веб-додаток "Розклад", який дозволяє користувачам знаходити оптимальний маршрут між станціями залізничного транспорту залежно від ціни та часу подорожі. Використання мови програмування Python та фреймворку Django дозволило реалізувати дану задачу швидко та ефективно.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Пошук шляху. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Пошук_шляху.
2. Bellman R. On a routing problem. Quarterly of Applied Mathematics. 1958. Vol. 16, no. 1. P. 87–90. URL: https://doi.org/10.1090/qam/102435.
3. Bellman R. On a routing problem. Quarterly of Applied Mathematics. 1958. Vol. 16, no. 1. P. 87–90. URL: https://doi.org/10.1090/qam/102435.
4. Data Structures and Algorithms: Dijkstra's Algorithm. School of Computer Science - The University of Auckland. URL: https://www.cs.auckland.ac.nz/software/AlgAnim/dijkstra.html.
5. Теорія графів.: навч. посіб. для здобувачів ступеня бакалавра за освітньою програмою «Комп’ютерний моніторинг та геометричне моделювання процесів і систем» спеціальності 122 «Комп’ютерні науки»/ І.М. Кузьменко; КПІ ім. Ігоря Сікорського. Київ: КПІ ім. Ігоря Сікорського, 2020. 71 с.
6. Ніколаєва К.В., Койбічук В.В. Дискретний аналіз. Графи та їх застосування в економіці: Навчально-методичний посібник. Суми: УАБС НБУ, 2007. 84 с.
7. Трохимчук Р.М. ТЕОРІЯ ГРАФІВ Навчальний посібник для студентів факультету кібернетики. К.: РВЦ “Київський університет”, 1998. 43 с.
8. Караванова Т. Теорія алгоритмів. Частина 2. Обчислювальні алгоритми : навч. посіб. Чернівці : Чернівец. нац. ун-т ім. Ю. Федьков., 2022. 288 с.
9. Shaw B. Web Development with Django : Learn to build modern web applications with a Python-based framework, 2021. 826с.
10. Мізюк О. Путівник мовою програмування Python. URL: https://pythonguide.rozh2sch.org.ua/.
11. Онищенко В., Довженко Т. Спеціалізовані мови програмування. К : Держ. ун-т телекомунікацій, 2019. 146 с.

12. Підручник з Python. Python documentation. URL: https://docs.python.org/uk/3/tutorial/index.html.
13. Посібник з Django. URL: https://krypton.com.ua/tutorial/basic_django/.
14. Посібник по Django для початківців – Частина 1. Codeguida. URL: https://codeguida.com/post/1039.
15. Томка Ю., Талах М., Ушенко Ю. Python та Django Full Stack веб-розробка. Чернівці : Чернів. нац. ун-т ім. Ю.Федьков., 2022. 248 с.
16. Django Підручник. Уроки для початківців. W3Schools українською. HTML CSS JavaScript PHP. W3Schools українською. Уроки для початківців. URL: https://w3schoolsua.github.io/django/index.html.
17. Гришанович Т. Курс лекцій з дисципліни «Алгоритми та структури даних» для студентів спеціальності 014 Середня освіта. Інформатика. Луцьк : ВНУ ім. Лесі Українки, 2021. 110 с. URL: https://evnuir.vnu.edu.ua/bitstream/123456789/19978/1/kurs_hryshanovych.pdf.