Міністерство освіти та науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра Інформаційних Систем та Технологій

Теорія алгоритмів

Індивідуальне завдання

Виконав: студент 1 курсу

Білогуб К.С

Перевірила пос.

[Солдатова М. О.](http://epi.kpi.ua/Schedules/ViewSchedule.aspx?v=f1ad598f-3968-4de6-98a7-cfced073066c)

Київ – 2023

Індивідуальне завдання

1. Постановка задачі:

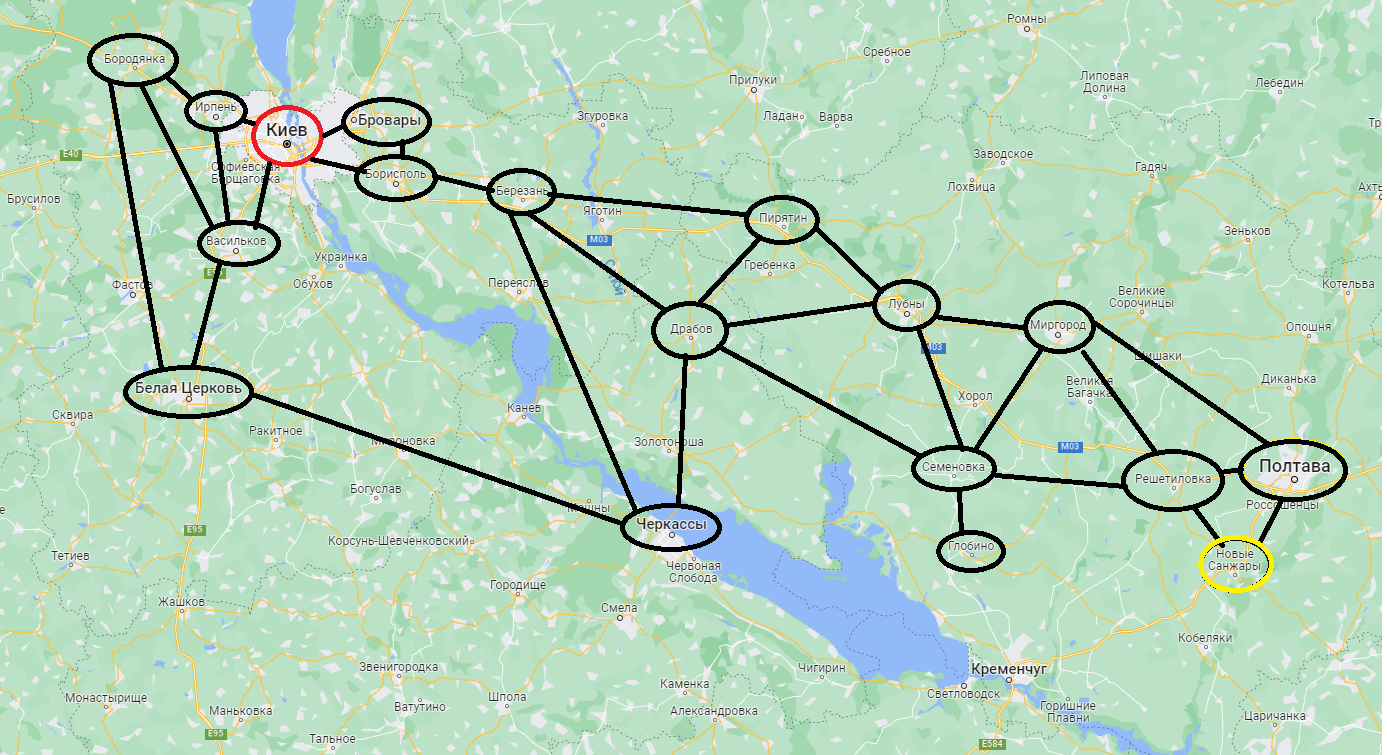
На основі карти України обрати частину території, яка включає в себе населений пункт проживання студента, населений пункт його рідних (не співпадає з попереднім) та ще декільканаселених пунктів навколо них. На основі виділеної території створити базовий граф, вершиниякого – населені пункти, ребра – дороги між ними. Кількість вершин повинна бути від 15 до 20, кількість ребер від 30 . Вагами ребер необхідно взяти:

a) довжину шляху між населеними пунктами в км,

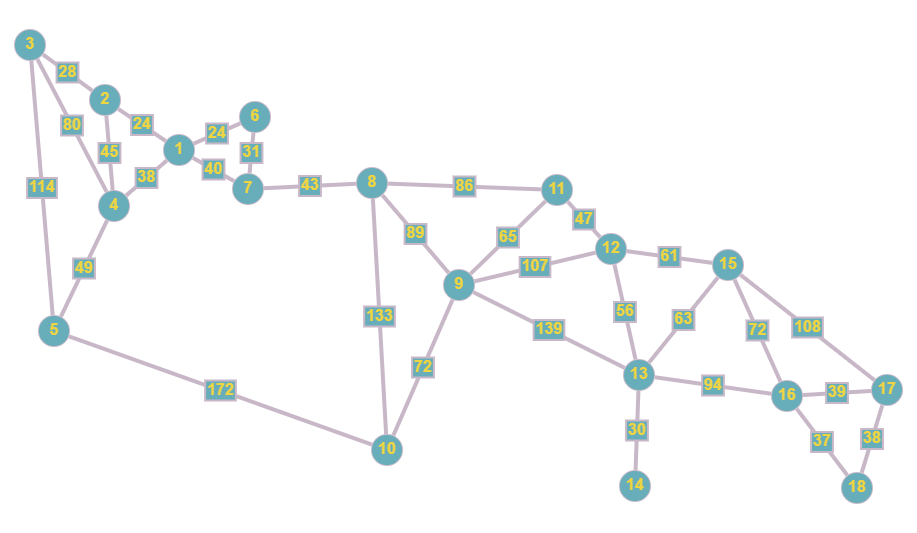
b) тип дороги, ставлячи йому відповідний пріоритет.

Для кожного населеного пункту визначити його координати та кількість населення.

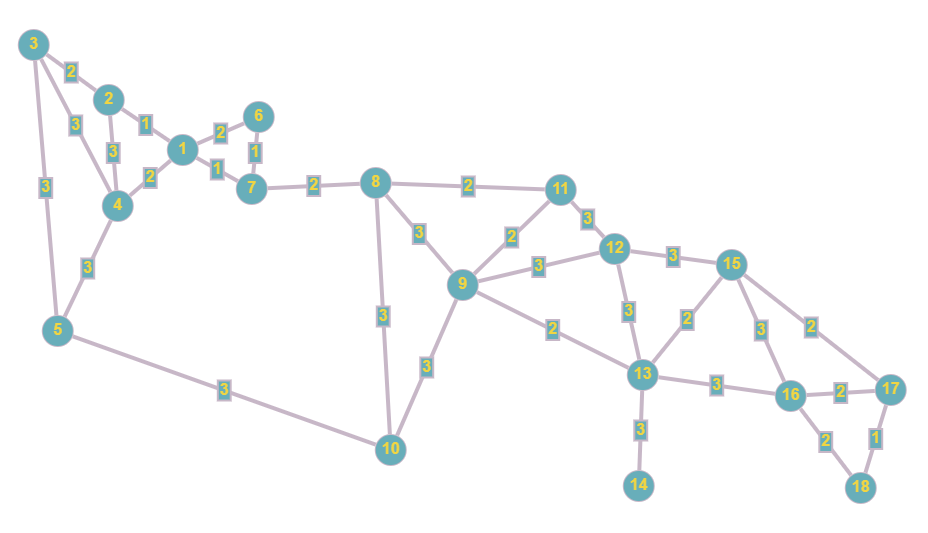
Демонстрація карти, на основі якої було побудовано граф, червоне коло – населений пункт проживання студента, жовте коло – населений пункт проживання його рідних



Базовий граф, в якому за ваги ребер взята довжина шляху між населеними пунктами в км:



Базовий граф, в якому за ваги ребер взят тип дороги (чим менше цифра, тим більша пріоритетність):



1 – Нове шосе

2 – Дорога регіонального значення

3 - Пересічена місцевість

Таблиця з показниками населених пунктів:



2. Завдання:

На основі базового графу необхідно розв’язати наступні завдання;

1. За допомогою одного з алгоритмів обходу графів обійти всі населені пункти на виділеній територій. Початком вважати населений пункт проживання студента, кінцевою точкою – пункт проживання його рідних. На довжину шляху та тип дороги не зважати.

Вибір алгоритму:

Для обходу всіх вершин графу може використовуватися BFS або DFS алгоритм. Обидва алгоритми мають складність O(V+E), де V - кількість вершин, а E - кількість ребер для найгіршого випадку, тому для цього завдання вибір конкретного з цих двох не є принциповим. Для цього завдання був обраний BFS алгоритм, який в реалізації не використовує рекурсію.

Опис алгоритму:

BFS, або пошук в ширину (Breadth-First Search), - це алгоритм обходу або пошуку в графі. Його основна ідея полягає в тому, що він спочатку відвідує всі вершини на одному рівні, перш ніж перейти до наступного рівня. Це робить BFS особливо корисним для знаходження найкоротшого шляху в невагомих графах або для знаходження всіх вершин, які можна досягти з початкової вершини.

Кілька основних характеристик BFS:

Охоплення всього графа: BFS відвідує всі вершини графа, якщо вони досяжні від початкової вершини.

Знаходження найкоротшого шляху: BFS може використовуватися для знаходження найкоротшого шляху в невагомих графах, оскільки він спочатку відвідує всі вершини на одному рівні, перш ніж перейти до наступного рівня.

Використання черги: BFS використовує структуру даних "черга" для відслідковування вершин для відвідування.

Пошук в ширину (BFS) працює наступним чином:

Ініціалізація: BFS починає свою роботу з початкової вершини (root node). Він створює структуру даних "черга" та додає початкову вершину до цієї черги.

Відвідування вершини: BFS видаляє вершину з початку черги та "відвідує" її. "Відвідування" вершини означає, що алгоритм виконує певну дію з цією вершиною, наприклад, друкує її значення або перевіряє, чи є вона цільовою вершиною.

Додавання сусідів до черги: Після відвідування вершини, BFS додає всі її невідвідані сусіди до кінця черги.

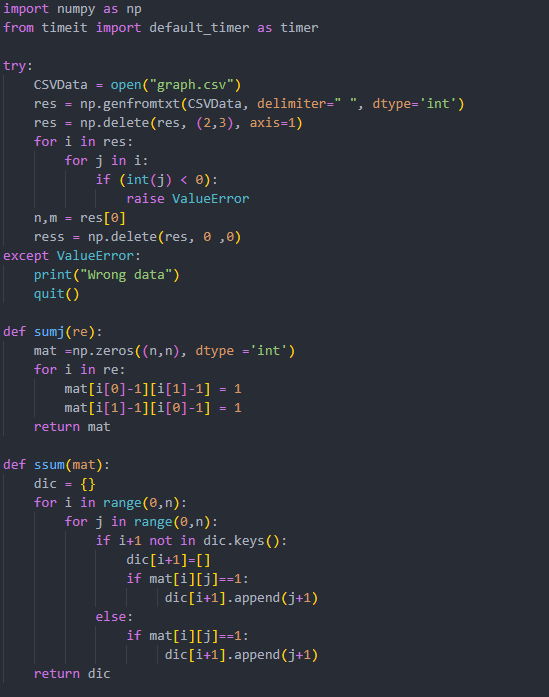
Повторення: Алгоритм продовжує цикл видалення вершини з початку черги, відвідування цієї вершини та додавання її невідвіданих сусідів до кінця черги, поки черга не стане порожньою.

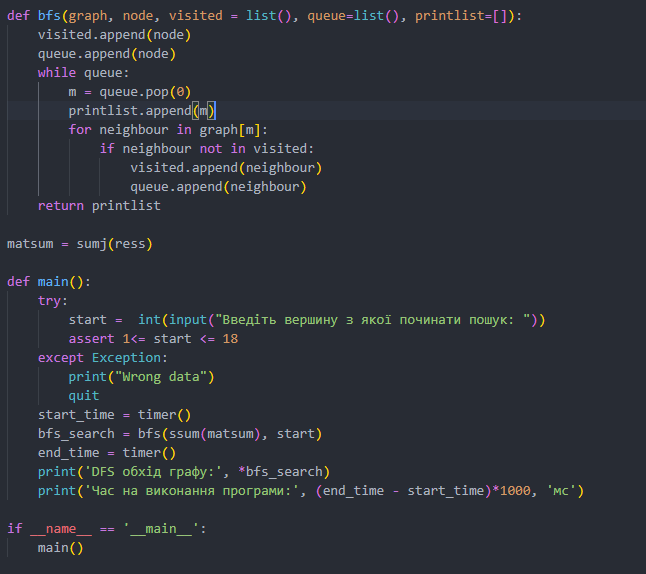
Кінець: Коли черга стає порожньою, це означає, що BFS відвідав всі вершини, які можна досягнути від початкової вершини, і алгоритм завершує свою роботу.

Особливості:

Важливо пам'ятати, що BFS відвідує вершини в порядку їх додавання до черги, тобто він спочатку відвідує всі вершини на одному рівні (вершини, які можна досягнути за один крок від початкової вершини), перш ніж перейти до вершин наступного рівня (вершини, які можна досягнути за два кроки від початкової вершини).

Реалізація BFS мовою Python 3.11.3:







**Таблиця тестування**

|  |  |  |  |
| --- | --- | --- | --- |
| Вхідні дані | | Результат | Призначення тесту |
| start | Graph.csv |
| 1 | 18 30 0 0  1 2 1 24  1 4 2 38  1 7 1 40  …….. | Відомий | Перевірка правильності роботи алгоритму та результату його роботи |
| 1<=Випадкове<=18 | Випадкове  Int, int, int, int | Невідомий, але легко обчислити | Перевірка правильності роботи алгоритму та результату його роботи |
| Випадкове<=1, Випадкове>=18 | - | Відомий | Реакція на невалідні дані |
| - | - | Відомий | Реакція на невалідні дані |
| 5 | 18 30 0 0  1 2 1hgh 24  1 4ds 2 38  1 7 1 40  ……… | Відомий | Реакція на невалідні дані |

1. За допомогою одного з алгоритмів знайти шлях від населеного пункту проживання студента до пункту проживання його рідних. На довжину шляху та тип дороги не зважати.

Вибір алгоритму:

Для цього завдання буде краще використати алгоритм DFS, бо він працює, спочатку відвідуючи якомога глибше від початкової вершини, перш ніж повертатися назад, тобто краще підходить для ситуацій, коли вершина, до якої треба знайти шлях знаходиться далеко від початкової (наш випадок). Як було зазначено вище, він має таку ж складність як і BFS: O(V+E), де V - кількість вершин, а E - кількість ребер для найгіршого випадку. При реалізації була використана рекурсія.

Опис алгоритму:

DFS, або пошук в глибину (Depth-First Search), - це алгоритм обходу або пошуку в графі, який працює, спочатку відвідуючи якомога глибше від початкової вершини, перш ніж повертатися назад.

Основні характеристики DFS:

Охоплення всього графа: як і BFS, DFS відвідує всі вершини графа, якщо вони досяжні від початкової вершини.

Використання стеку: DFS використовує структуру даних "стек" для відслідковування вершин для відвідування.

Відвідування глибоких вершин спочатку: відмінність між DFS і BFS полягає в тому, що DFS спочатку відвідує глибокі вершини (тобто вершини, які знаходяться далеко від початкової вершини).

Пошук в глибину (DFS) працює наступним чином:

Ініціалізація: DFS починає свою роботу з початкової вершини (root node). Він створює структуру даних "стек" та додає початкову вершину до цього стеку.

Відвідування вершини: DFS видаляє вершину з верхівки стеку та "відвідує" її. "Відвідування" вершини означає, що алгоритм виконує певну дію з цією вершиною, наприклад, друкує її значення або перевіряє, чи є вона цільовою вершини.

Додавання сусідів до стеку: Після відвідування вершини, DFS додає всі її невідвідані сусіди до верхівки стеку.

Повторення: Алгоритм продовжує цикл видалення вершини з верхівки стеку, відвідування цієї вершини та додавання її невідвіданих сусідів до верхівки стеку, поки стек не стане порожнім.

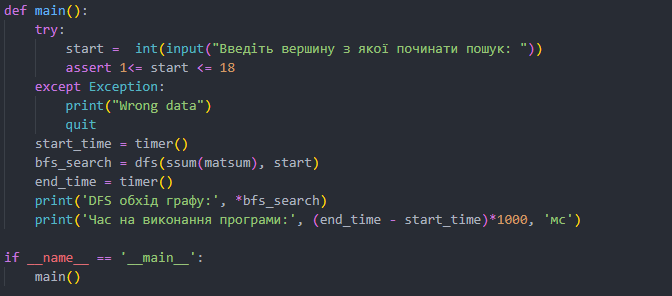
Кінець: Коли стек стає порожнім, це означає, що DFS відвідав всі вершини, які можна досягнути від початкової вершини, і алгоритм завершує свою роботу.

Особливості:

Важливо зазначити, що DFS відвідує вершини в порядку їх додавання до стеку, тобто він спочатку відвідує "глибокі" вершини (вершини, які знаходяться далеко від початкової вершини), перш ніж повертатися до менш глибоких вершин.

Реалізація DFS мовою Python 3.11.3:







**Таблиця тестування**

|  |  |  |  |
| --- | --- | --- | --- |
| Вхідні дані | | Результат | Призначення тесту |
| start | Graph.csv |
| 1 | 18 30 0 0  1 2 1 24  1 4 2 38  1 7 1 40  …….. | Відомий | Перевірка правильності роботи алгоритму та результату його роботи |
| 1<=Випадкове<=18 | Випадкове  Int, int, int, int | Невідомий, але легко обчислити | Перевірка правильності роботи алгоритму та результату його роботи |
| Випадкове<=1, Випадкове>=18 | - | Відомий | Реакція на невалідні дані |
| - | - | Відомий | Реакція на невалідні дані |
| 5 | 18 30 0 0  1 2 1hgh 24  1 4ds 2 38  1 7 1 40  ……… | Відомий | Реакція на невалідні дані |

1. За допомогою одного з алгоритмів знайти найкоротший шлях для двох випадків обирання критерію проходження (вага a) та b)) від населеного пункту проживання студента до пункту проживання його рідних. Порівняти результати.

Вибір алгоритму:

Для пошуку найкоротшого шляху був обраний алгоритм Дейкстри, який використовує таблиці суміжності, в яких, якщо вершини мають зв’язок, ставиться вага ребра, якщо не мають – 0. Найгірша швидкодія цього алгоритму O(V2), де V - кількість вершин.

Опис алгоритму:

Алгоритм Дейкстри - це алгоритм для знаходження найкоротшого шляху від однієї вершини (початкової) до всіх інших вершин в графі з вагами. Граф повинен бути зважений (тобто кожному ребру призначається певне числове значення, або "вага") і без циклів з від'ємною вагою.

Основні характеристики:

Жадібний алгоритм: Алгоритм Дейкстри є жадібним алгоритмом. Це означає, що він приймає найкраще локальне рішення на кожному кроці з надією, що ці локальні оптимуми приведуть до глобального оптимуму.

Працює лише з невід'ємними вагами: Алгоритм Дейкстри працює коректно лише тоді, коли всі ваги ребер у графі невід'ємні. Якщо граф містить ребра з від'ємною вагою, може виникнути проблема з циклами від'ємної ваги.

Одноетапний алгоритм: Одним з ключових аспектів алгоритму Дейкстри є те, що він одноетапний, тобто він не переглядає ребра чи вершини більше одного разу.

Кроки алгоритму:

Ініціалізація: На початку всі вершини мають "вартість" або "відстань" від початкової вершини, яка дорівнює нескінченності, крім початкової вершини, вартість якої дорівнює нулю.

Перевірка сусідів: Алгоритм вибирає вершину з найменшою вартістю, що ще не була оброблена, та оновлює вартості її сусідів. Вартість сусіда оновлюється, якщо вартість поточної вершини плюс вага ребра, що веде до сусіда, менша за поточну вартість сусіда.

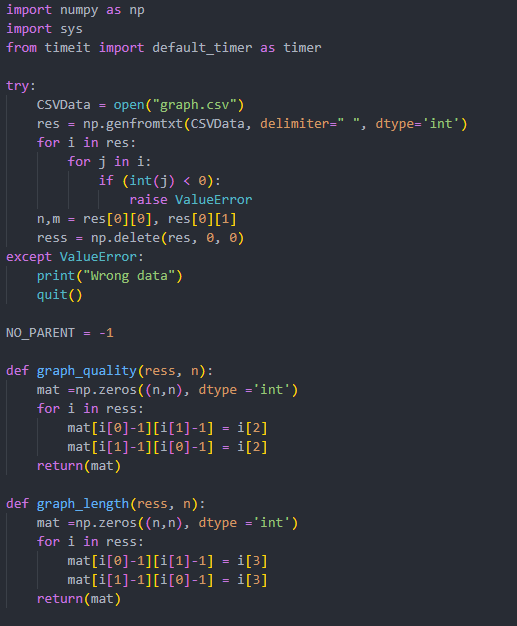
Повторення: Алгоритм повторює цей процес (вибирає вершину з найменшою вартістю, що ще не була оброблена, та оновлює вартості її сусідів), поки не будуть оброблені всі вершини.

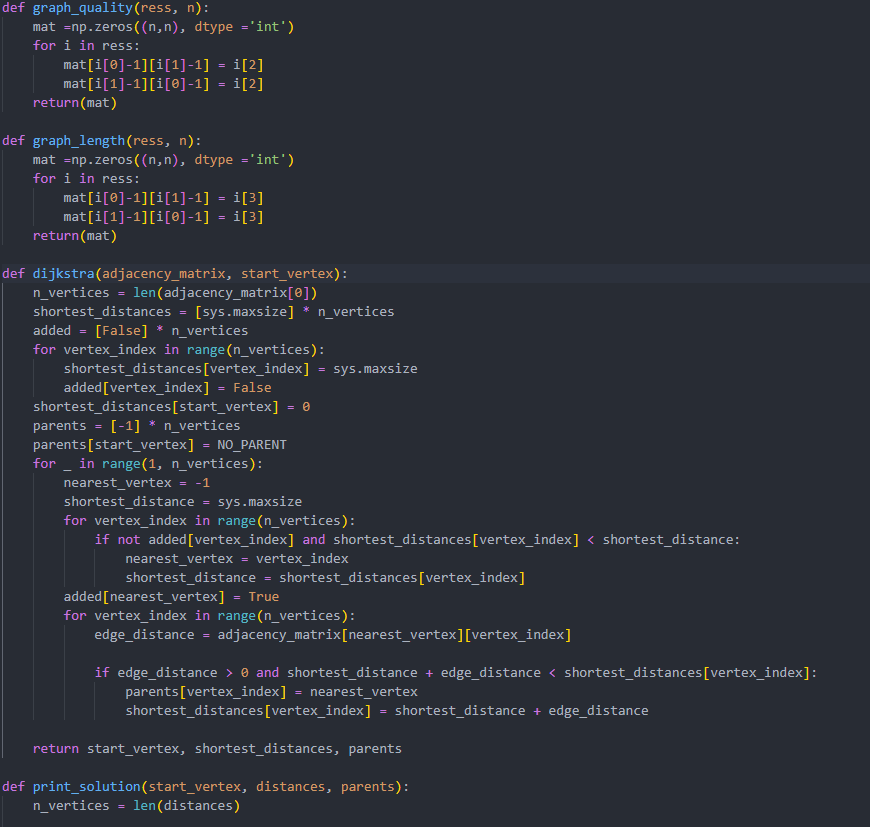
Кінець: Коли всі вершини оброблено, алгоритм завершено. Вартість кожної вершини відображає довжину найкоротшого шляху від початкової вершини до цієї вершини.

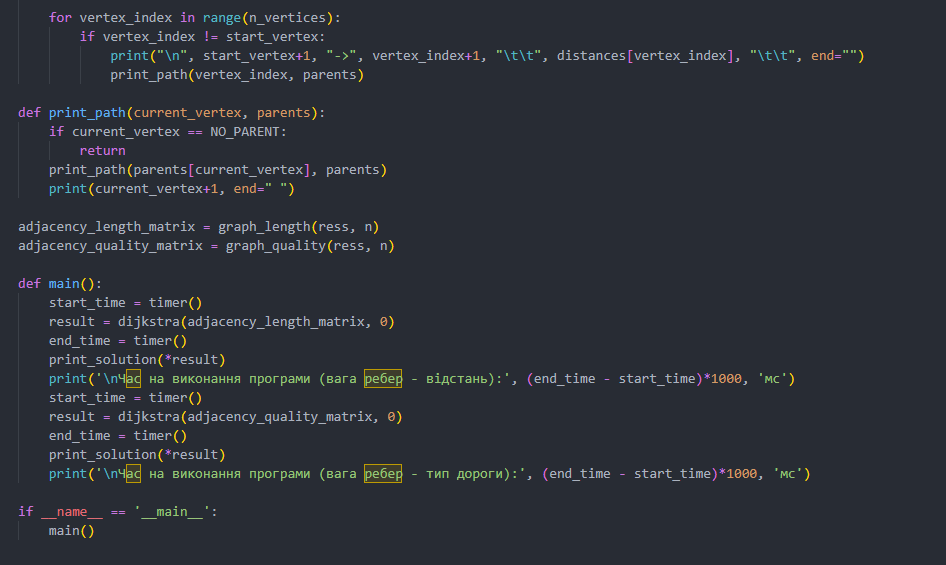
Особливості:

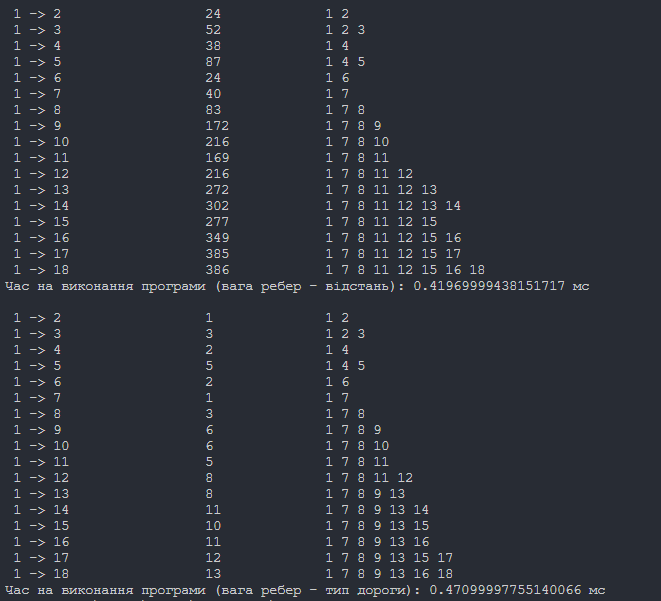
Слід зауважити, що алгоритм Дейкстри використовує жадібний підхід, тому він не завжди знаходить найкоротший шлях в графах з від'ємними вагами. Для таких графів використовують алгоритм Беллмана-Форда або алгоритм Джонсона.

Реалізація алгоритму Дейкстри мовою Python 3.11.3:









Як бачимо, незважаючи на те, що початкова та остання вершини однакові в обох випадках, шляхи відрізняються, тобто характеристики доріг впливають на вибір найкоротшого шляху.

**Таблиця тестування**

|  |  |  |
| --- | --- | --- |
| Результат | | Призначення тесту |
| Graph.csv |  |  |
| 18 30 0 0  1 2 1 24  1 4 2 38  1 7 1 40  …….. | Відомий | Перевірка правильності роботи алгоритму та результату його роботи |
| - | Відомий | Реакція на невалідні дані |
| 18 30 0 0  1 2 1hgh 24  1 4ds 2 38  1 7 1 40  ……… | Відомий | Реакція на невалідні дані |

1. Зробити сортування населених пунктів на основі відомих алгоритмів за їх координатами із заходу на схід, та кількістю населення з більшого к меншому.

Вибір алгоритму:

Для сортування за координатами був обраний Quick sort, оскільки ей алгоритм гарно працює на невеликих обсягах хаотичних даних без повторів, крім того вони мають тип float тому алгоритми по типу Counting sort не підходять для такого набору даних, до того задача не потребує збереження положення елементів в масиві, тобто не вимагає стабільного алгоритму сортування. Швидке сортування має найкращу швидкодію O(nlogn) і найгішру O(n^2).

Опис алгоритму:

Квіксорт — це алгоритм сортування, який використовує стратегію "розділяй і володарюй". Цей алгоритм був розроблений Брітанським інформатиком Тоні Хоаром в 1959 році і вважається одним з найшвидших алгоритмів сортування.

Основні характеристики квіксорту:

Ефективність: Квіксорт є одним з найшвидших алгоритмів сортування в середньому випадку з часом виконання O(n log n).

Місце: Квіксорт є алгоритмом сортування "на місці", тобто він не вимагає додаткового місця поза вхідним масивом.

Нестабільність: Квіксорт є "нестабільним" алгоритмом сортування, тобто він може змінювати відносний порядок однакових елементів.

Рекурсивний: Квіксорт є рекурсивним алгоритмом, і його можна легко реалізувати за допомогою рекурсії. Однак для дуже великих наборів даних це може призвести до переповнення стеку.

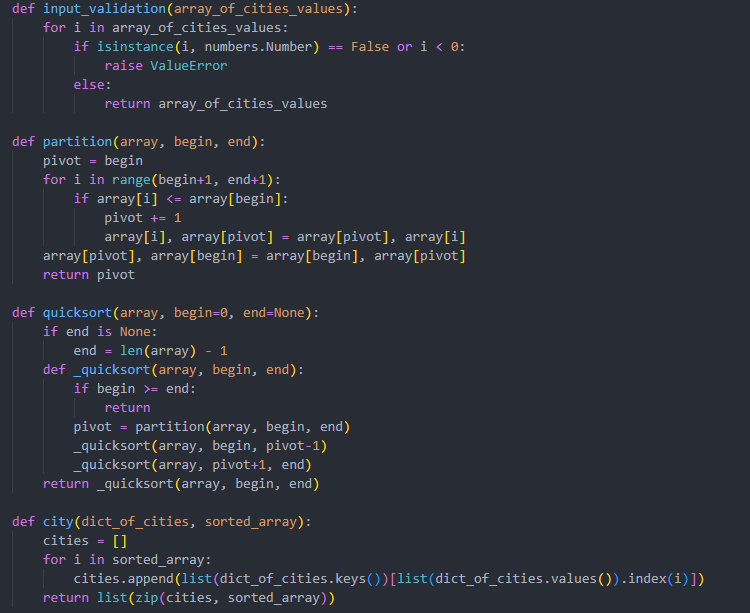
Кроки алгоритму:

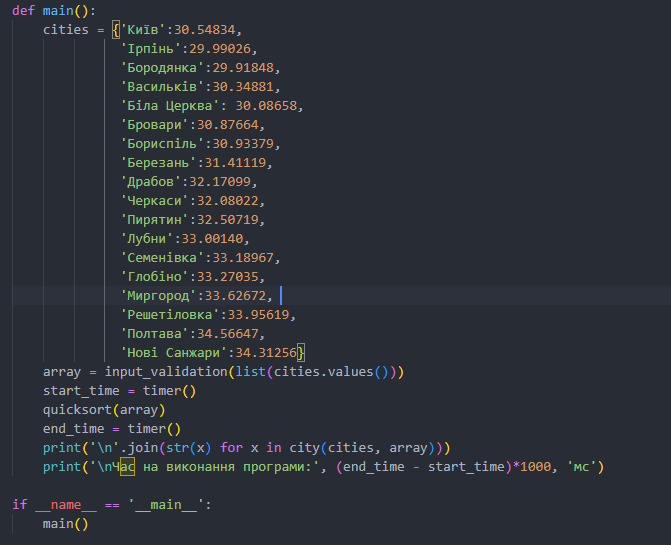
Вибір опорного елементу: Вибирається опорний елемент з масиву. Вибір опорного елементу може бути випадковим, або можна використовувати конкретні евристики (наприклад, середнє значення).

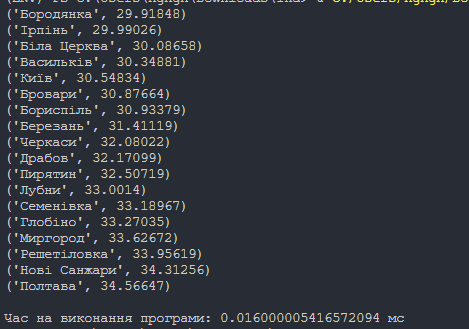
Партіція: Масив розбивається на дві частини таким чином, що елементи менші за опорний елемент знаходяться зліва від нього, а елементи, більші за опорний елемент, — справа.

Рекурсивне застосування: Квіксорт рекурсивно застосовується до двох підмасивів, отриманих після партіції.

Реалізація Quick sort мовою Python 3.11.3:







**Таблиця тестування**

|  |  |  |
| --- | --- | --- |
| Вхідні дані | Результат | Призначення тесту |
| 30.54834, 29.99026, 29.91848  ….. | Відомий | Перевірка роботи алгоритму |
| Випадкове | Невідомий, але легко обчислити | Перевірка роботи алгоритму |
| 1,50.45adas,30.52,3000000dsadd  2,50.25,28.65,266106 | Відомий | Реакція на невалідні дані |
| - | Відомий | Реакція на невалідні дані |

Вибір алгоритму:

Для сортування міст за кількістю населення був обраний LSD алгоритм, оскільки LSD швидше, ніж MSD, коли є фіксована довжина. MSD занадто повільний для невеликих файлів і вимагає величезної кількості рекурсивних викликів для невеликих файлів. Найгірша швидкодія O(w\*n), де n - кількість ключів, а w - довжина ключа.

Опис алгоритму:

LSD Radix Sort, або "цифрове сортування за меншим розрядом", - це алгоритм сортування, який обробляє цілочисельні ключі за окремими цифрами, які обробляються в порядку від меншого (найменш значущого) розряду до більшого (найбільш значущого). Цей метод сортування приймає на вхід послідовність чисел і виконує їх сортування.

Основні характеристики алгоритму LSD Radix Sort:

Стабільність: Radix Sort є стабільним алгоритмом сортування, що означає, що він зберігає відносний порядок однакових елементів.

Ін-плейс: LSD Radix Sort не є in-place алгоритмом, тому що він вимагає додаткового простору для "ведер".

Сортування за основою: Radix Sort сортує числа, обробляючи індивідуальні цифри, починаючи від найменш значущого розряду і закінчуючи найбільш значущим. Основа сортування може бути будь-якою, найчастіше використовують десяткову систему.

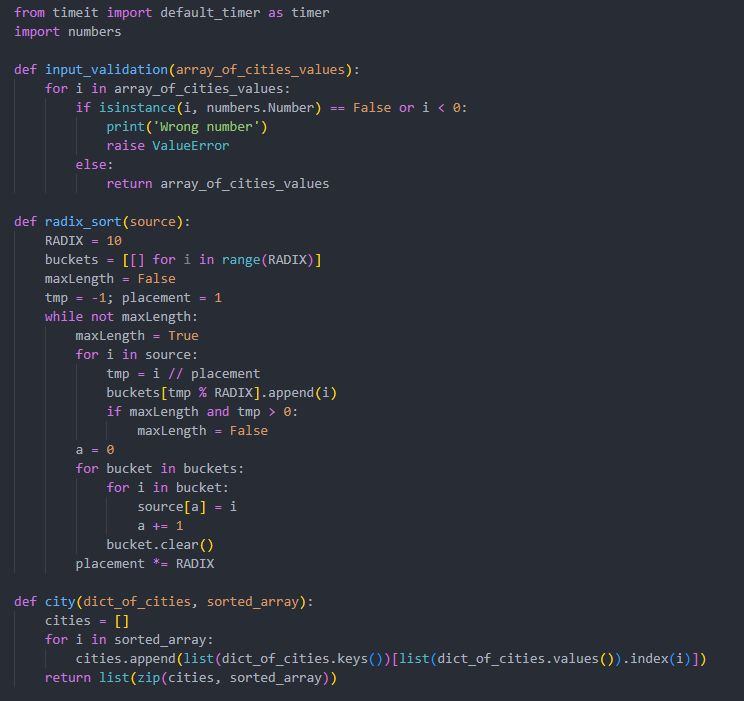
Кроки алгоритму:

Покладіть кожне число у відповідний "ведро" відповідно до його меншого розряду (наприклад, якщо менший розряд - 1, число 1 йде до першого ведра, 11 - до другого ведра і т.д.).

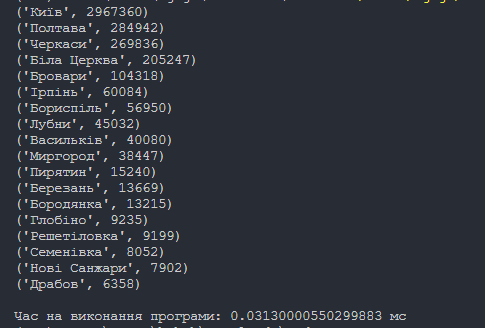
Заберіть числа з відер у порядку від нульового до найбільшого ведра, щоб отримати нову послідовність чисел.

Повторіть ці кроки для кожного наступного більшого розряду, поки не буде оброблено всі розряди всіх чисел.

Реалізація LSD Radix sort мовою Python 3.11.3:







**Таблиця тестування**

|  |  |  |
| --- | --- | --- |
| Вхідні дані | Результат | Призначення тесту |
| 2967360, 60084, 13215  ….. | Відомий | Перевірка роботи алгоритму |
| Випадкове | Невідомий, але легко обчислити | Перевірка роботи алгоритму |
| 1,50.45adas,30.52,3000000dsadd  2,50.25,28.65,266106 | Відомий | Реакція на невалідні дані |
| - | Відомий | Реакція на невалідні дані |

Висновок

Для виконання цієї роботи було створено програму, яка використовує граф для представлення дорожньої мережі між різними населеними пунктами України. Використовуючи цей граф, програма здатна вирішувати різні завдання, пов'язані зі знаходженням шляхів між різними містами.

Одне з головних завдань полягало в знаходженні найкоротшого шляху між двома точками за допомогою алгоритму Дейкстри. Цей алгоритм було обрано через його ефективність та можливість знаходити найкоротші шляхи в графах з невід'ємними вагами. Завдання було виконано двічі, з використанням двох різних метрик: довжини дороги в кілометрах і типу дороги.

Також програма була використана для обходу всіх населених пунктів на виділеній території. Для цього можна було використати алгоритми обходу графів, такі як DFS або BFS.

Додатково, програма використовувала алгоритм сортування (Quicksort) для сортування населених пунктів за їх координатами із заходу на схід, та LSD для сортування за кількістю населення з більшого к меншому.

Складність обраних алгоритмів була обчислена і обгрунтована. Для алгоритму Дейкстри складність становить O(V2), де V - кількість вершин. Для алгоритму квіксорт у середньому випадку складність становить O(n log n), де n - кількість елементів для сортування, для DFS та BFS - O(V+E), де V - кількість вершин, а E - кількість ребер для найгіршого випадку, для LSD - O(w\*n), де n - кількість ключів, а w - довжина ключа.

Час виконання кожної частини програмного забезпечення був виміряний і записаний.