**Міністерство освіти і науки України**

**Національний університет «Запорізька Політехніка»**

Кафедра програмних засобів

**ЗВІТ**

з самостійної роботи

з дисципліни «Алгоритми та Структури Даних»

**Виконав:**

Студент групи КНТ-122 О. А. Онищенко

**Прийняли:**

Викладач: В. М. Льовкін

2023

Зміст

[Вступ 3](#_Toc153873686)

[Основна частина 4](#_Toc153873687)

[Стуктури даних 4](#_Toc153873688)

[Алгоритм пірамідального сортування (Heap Sort) 4](#_Toc153873689)

[Гешування та B-дерева 8](#_Toc153873690)

[Удосконалені методи розробки та аналізу 13](#_Toc153873691)

[Жадібні алгоритми 13](#_Toc153873692)

[Динамічне програмування 19](#_Toc153873693)

[Алгоритми роботи з графами 25](#_Toc153873694)

[Алгоритми обходу графа 25](#_Toc153873695)

[Алгоритми пошуку найкоротшого шляху 35](#_Toc153873696)

[Висновки 44](#_Toc153873697)

[Додатки 46](#_Toc153873698)

[Додаток А – Повний код програми 46](#_Toc153873699)

Вступ

Алгоритми та структури даних (АтаСД) мають фундаментальне значення для комп'ютерних наук, оскільки вони є основою для розробки ефективного програмного забезпечення та розв'язання складних задач. Сучасний стан АтаСД є досить розвиненим, і багато проблем вже практично вирішено. Однак у цій галузі все ще існують прогалини в знаннях, особливо в таких сферах, як квантові обчислення та алгоритми, пов'язані зі штучним інтелектом.

Провідні компанії та науковці в цій галузі постійно працюють над новими алгоритмами та структурами даних, прагнучи підвищити ефективність та вирішити нові проблеми. Вони також займаються викладанням і дослідженнями, роблячи свій внесок у світові тенденції вирішення проблем.

Ця робота є актуальною та необхідною у зв'язку зі зростанням складності завдань у різних галузях, таких як штучний інтелект, наука про дані та комп'ютерна графіка. Вона також є основою для виконання багатьох програмних та апаратних проєктів.

Мета цієї роботи - продемонструвати застосування різних концепцій АтаСД через поєднання демонстраційних прикладів і завдань. До них відносяться купа, зв'язаний список, геш-таблиця, B-дерево, кодування Гаффмана, жадібні алгоритми, динамічне програмування, алгоритми BFS, DFS, Дейкстри, Флойда-Уоршалла та Беллмана-Форда. Ця робота також може бути використана в інших роботах в галузі комп'ютерних наук, таких як розробка нових мов програмування, баз даних та програмних систем.

Основна частина

Стуктури даних

Алгоритм пірамідального сортування (Heap Sort)

Сортування купою - це алгоритм сортування на основі порівняння, який використовує структуру даних у вигляді купи. Він працює, розділяючи вхідні дані на відсортовану та невідсортовану області, та ітеративно зменшує невідсортовану область, виділяючи найбільший елемент і переміщуючи його до відсортованої області. Часова складність цього алгоритму становить O(n log n) у всіх випадках (найкращому, середньому та найгіршому), що робить його досить ефективним для великих наборів даних.

Для індивідуального завдання ми реалізували Heap Sort на Python, візуалізувавши процес сортування за допомогою графічного інтерфейсу. Програма дозволяла користувачеві вводити масив чисел, і відображала його до і після сортування. Реалізація алгоритму Heap Sort була виконана з використанням вбудованої в Python структури даних у вигляді списку. Результати показали, що алгоритм Heap Sort здатен ефективно сортувати масив за зростанням.

Програма також дозволила користувачеві порівняти ефективність алгоритму Heap Sort з іншими алгоритмами сортування, такими як Quick Sort і вбудована функція сортування Python. Результати показали, що алгоритм Heap Sort загалом був швидшим за інші два алгоритми, особливо для великих наборів даних.

Фрагменти коду та результати роботи алгоритмів наведено нижче у вигляді знімків з екрану.

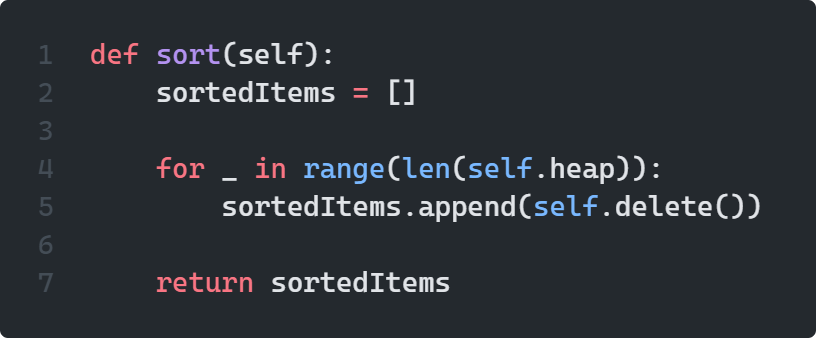


Рисунок 2.1 – Метод, що реалізує алгоритм пірамідального сортування у класі Heap



Рисунок 2.2 – Вирішення індивідуального завдання за допомогою пірамідального сортування

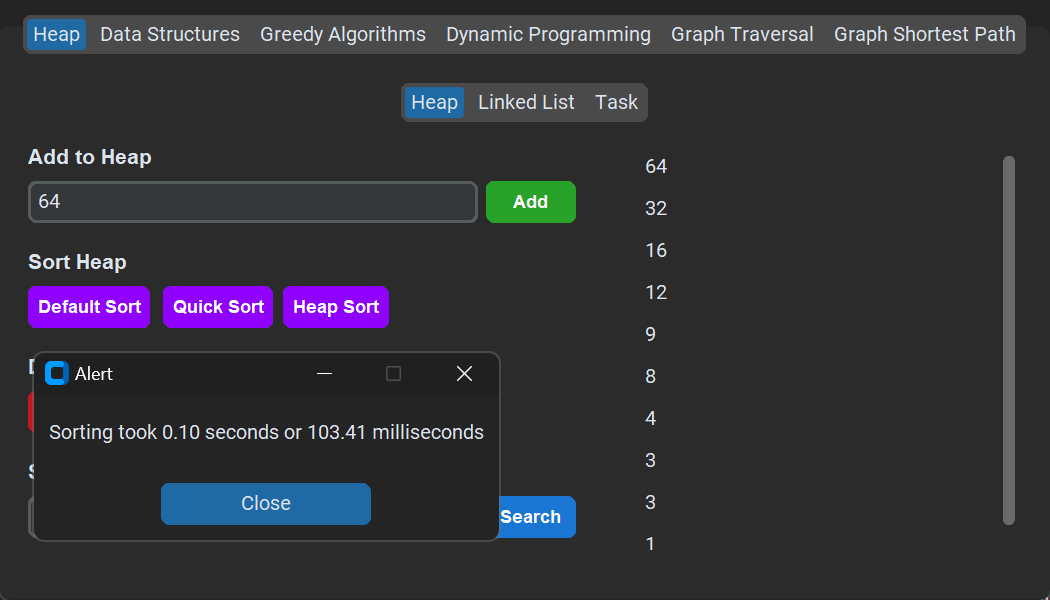


Рисунок 2.3 – Результати виконання пірамідального сортування на довільній купі

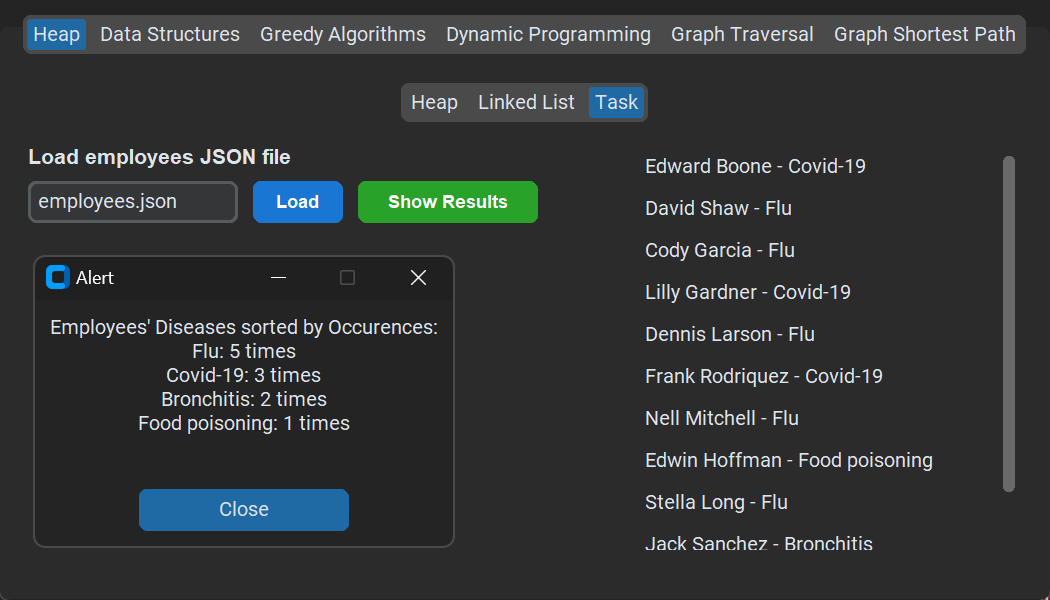


Рисунок 2.4 – Результати виконання індивідуального завдання на основі пірамідального сортування

Гешування та B-дерева

Гешування та B-дерева - це два різних типи структур даних, які можна використовувати для зберігання та пошуку даних. Гешування - це техніка, яка використовується для унікальної ідентифікації конкретного об'єкта з групи схожих об'єктів. Вона використовує геш-функцію для обчислення індексу в масиві комірок або слотів, з якого можна знайти потрібне значення.

B-дерева, з іншого боку, є типом самобалансуючої деревоподібної структури даних, яка підтримує відсортовані дані і дозволяє ефективно виконувати операції вставки, видалення та пошуку. Вони особливо корисні для систем з великими обсягами даних, які потребують швидкого доступу.

Для індивідуального завдання ми реалізували геш-таблицю та B-дерево на Python, знову ж таки з використанням графічного інтерфейсу для візуалізації. Програмне забезпечення дозволяло користувачеві вводити пари ключ-значення, і воно відображало таблицю або дерево до і після вставки. Результати показали, що обидві структури даних здатні ефективно зберігати та отримувати дані.

Що стосується індивідуальних завдань, то перше завдання передбачає створення геш-таблиці, яка використовує ланцюговий метод для вирішення колізій і функцію множення гешів. Геш-таблиця заповнюється на основі вибірки інформації з текстового файлу, який містить імена співробітників компанії та їхні посади. Потім визначається посада конкретного співробітника.

Друга задача передбачає формування дерева з відповідної інформації про абонентів, забезпечення пошуку інформації про абонента за його номером телефону та визначення кількості з'єднань для кожного з тарифів.

Наведені рішення показують, як ці задачі можуть бути реалізовані мовою Python. Перше рішення використовує геш-таблицю для зберігання даних про співробітників, а потім використовує функцію множення гешів для вирішення колізій. Друге рішення використовує B-дерево для зберігання даних про абонентів, а потім використовує функцію пошуку для пошуку абонента за його номером телефону. Кількість з'єднань для кожного тарифу визначається шляхом підрахунку кількості входжень кожного тарифу в масивах даних.

Фрагменти коду, що стосуються алгоритмів, та результати роботи наведено нижче у вигляді знімків з екрану.



Рисунок 2.5 – Імплементація методу гешування у класі HashTable

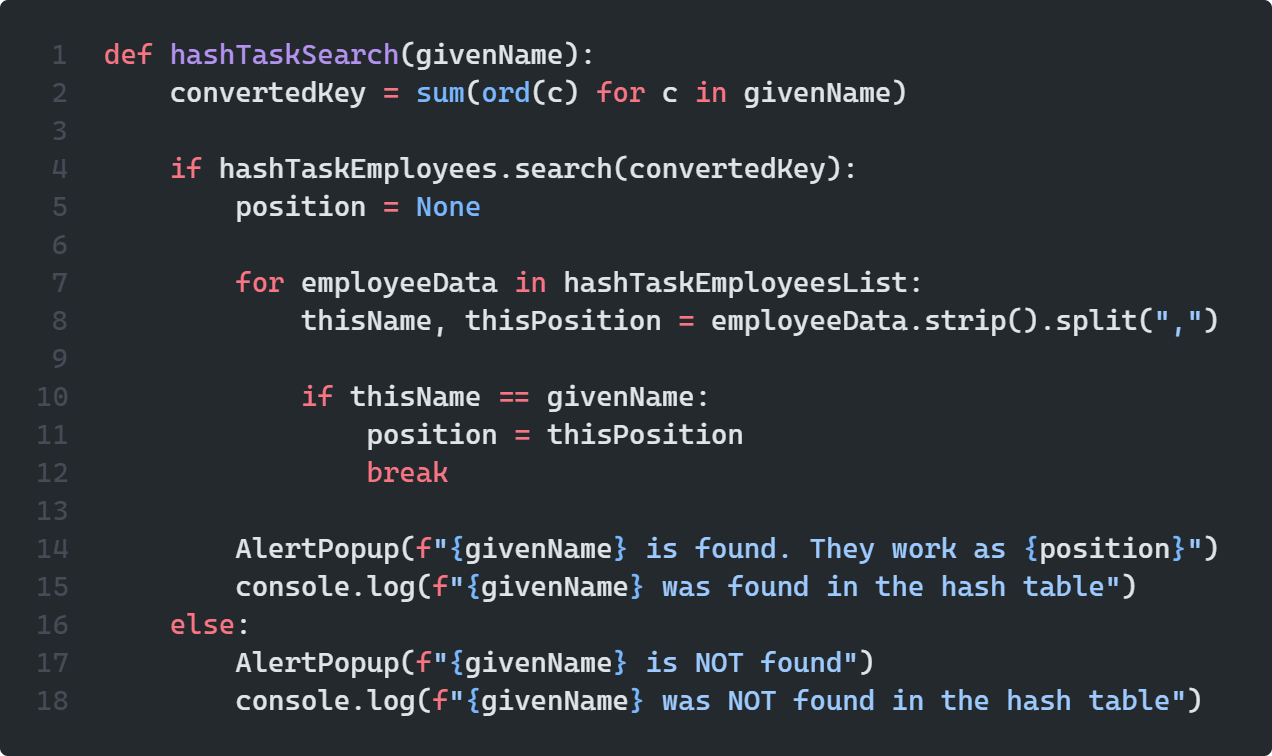


Рисунок 2.6 – Код, що виконує індивідуальне завдання на основі геш-таблиці

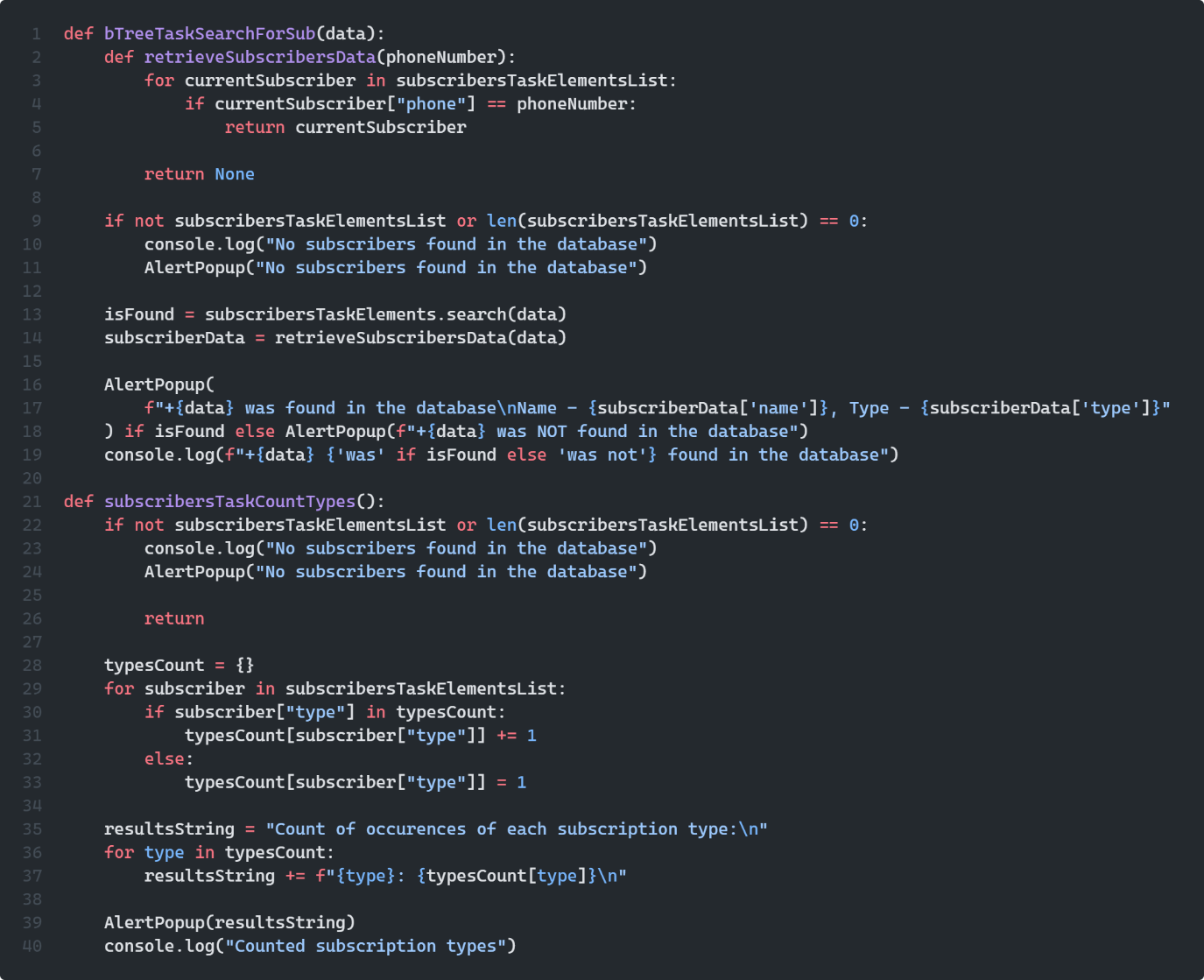


Рисунок 2.7 – Код, що виконує індивідуальне завдання на основі класу B-Tree

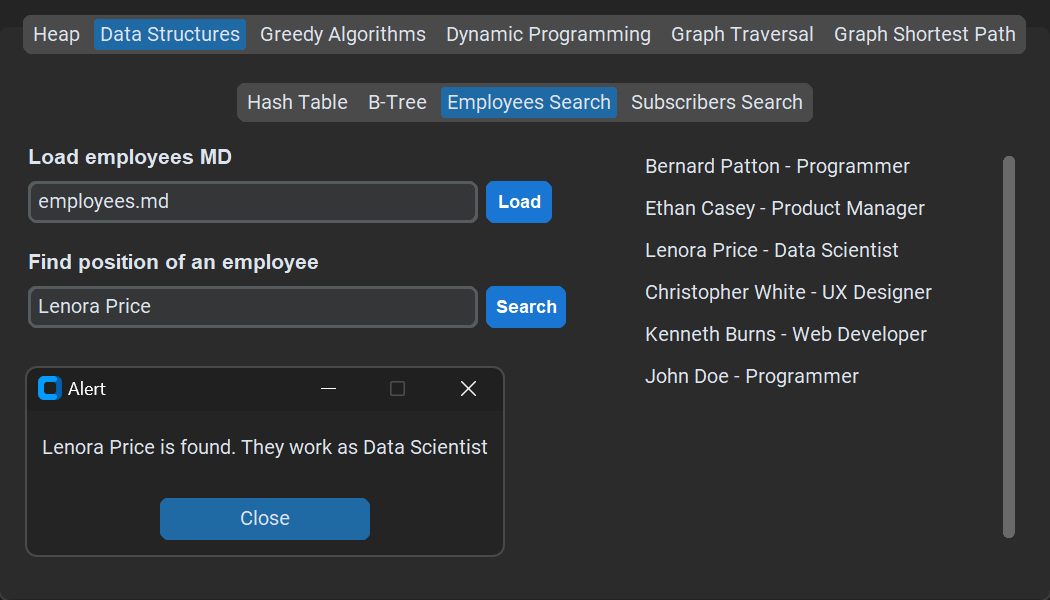


Рисунок 2.8 – Результати виконання індивідуального завдання на основі класу HashTable

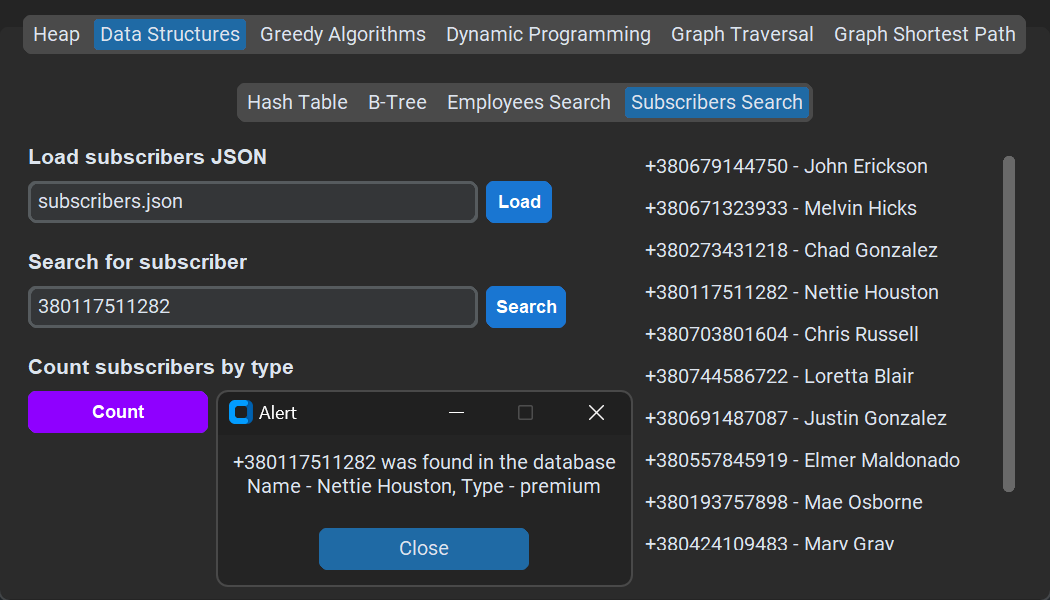


Рисунок 2.9 – Результати виконання індивідуального завдання на основі класу B-Tree

Удосконалені методи розробки та аналізу

Жадібні алгоритми

Жадібні алгоритми - це тип алгоритмічної парадигми, яка використовує евристичну схему розв'язання задач, що полягає у виборі локально оптимального рішення на кожному етапі з надією знайти глобальний оптимум. Метою цих алгоритмів є пошук найкращого рішення у найефективніший спосіб.

Для індивідуальної задачі з жадібним алгоритмом ми реалізували алгоритм Гаффмана для стиснення даних текстового файлу. Алгоритм Гаффмана - це жадібний алгоритм, який використовується для стиснення даних. Він працює шляхом присвоєння вхідним символам кодів змінної довжини, довжина яких залежить від частоти відповідних символів. Найчастіший символ отримує найменший код, а найбільш рідкісний символ - найбільший код.

Програма дозволяла користувачеві вводити текстовий файл, а потім стискала його за алгоритмом Гаффмана. Результати показали, що алгоритм Гаффмана ефективно стискає текстовий файл.

Друга задача стосувалась продавця у магазині, який повинен був вирішити, чи варто тримати товар поруч, щоб прискорити чергу покупців. Для прийняття цього рішення продавець використав жадібний алгоритм. Алгоритм працював, порівнюючи час, який знадобився б, щоб дістати товар, якого в даний момент немає в наявності (що сповільнило б чергу), з часом, який знадобився б, щоб дістати товар, який в даний момент є в наявності (що прискорило б чергу). Алгоритм вирішив тримати товар в наявності, якщо це призведе до прискорення черги.

Результати показали, що жадібний алгоритм зміг допомогти продавцю вирішити, які товари тримати в наявності, щоб прискорити чергу.

Фрагменти коду та результати виконання індивідуальних завдань наведено нижче у вигляді знімків з екрану.

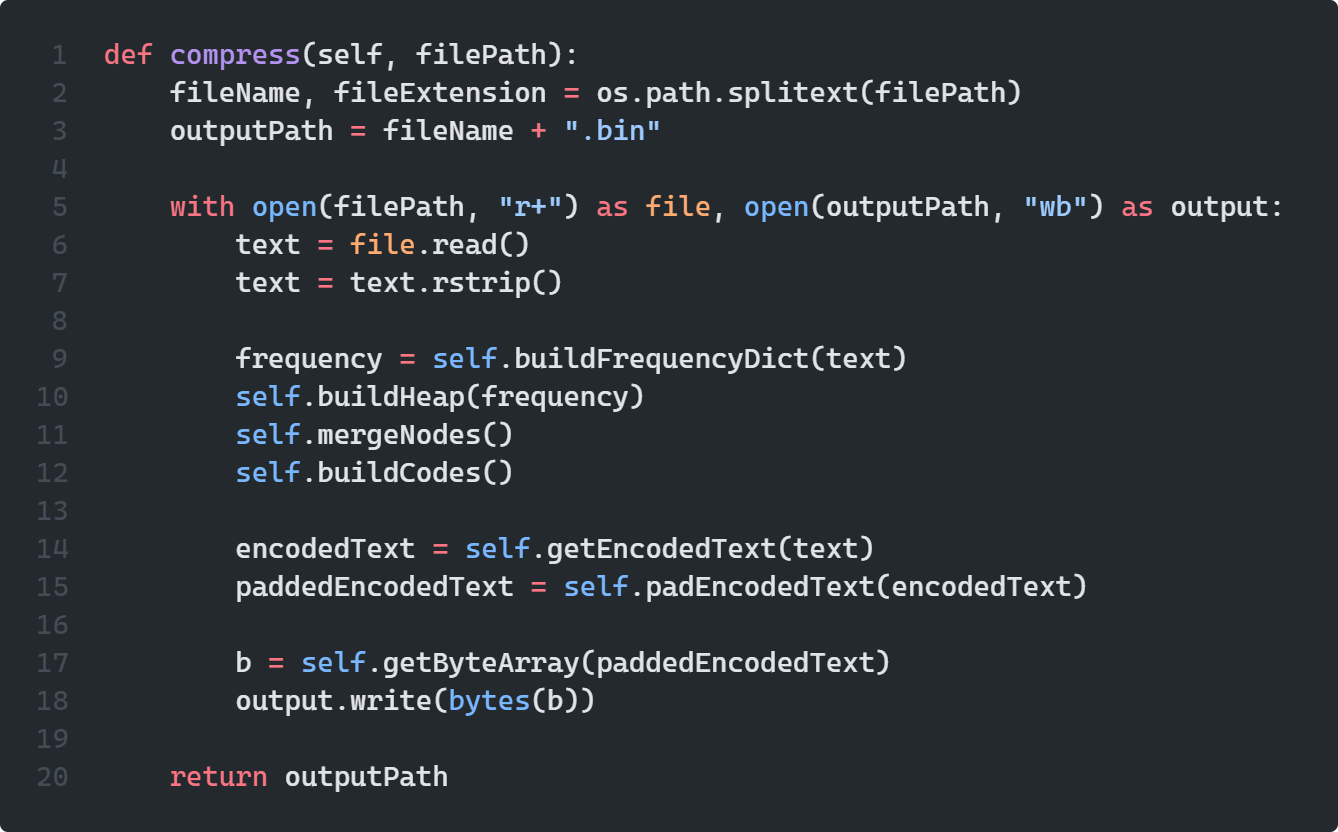


Рисунок 2.10 – Метод, що дозволяє стискати наданий текст у класі HuffmanCoding

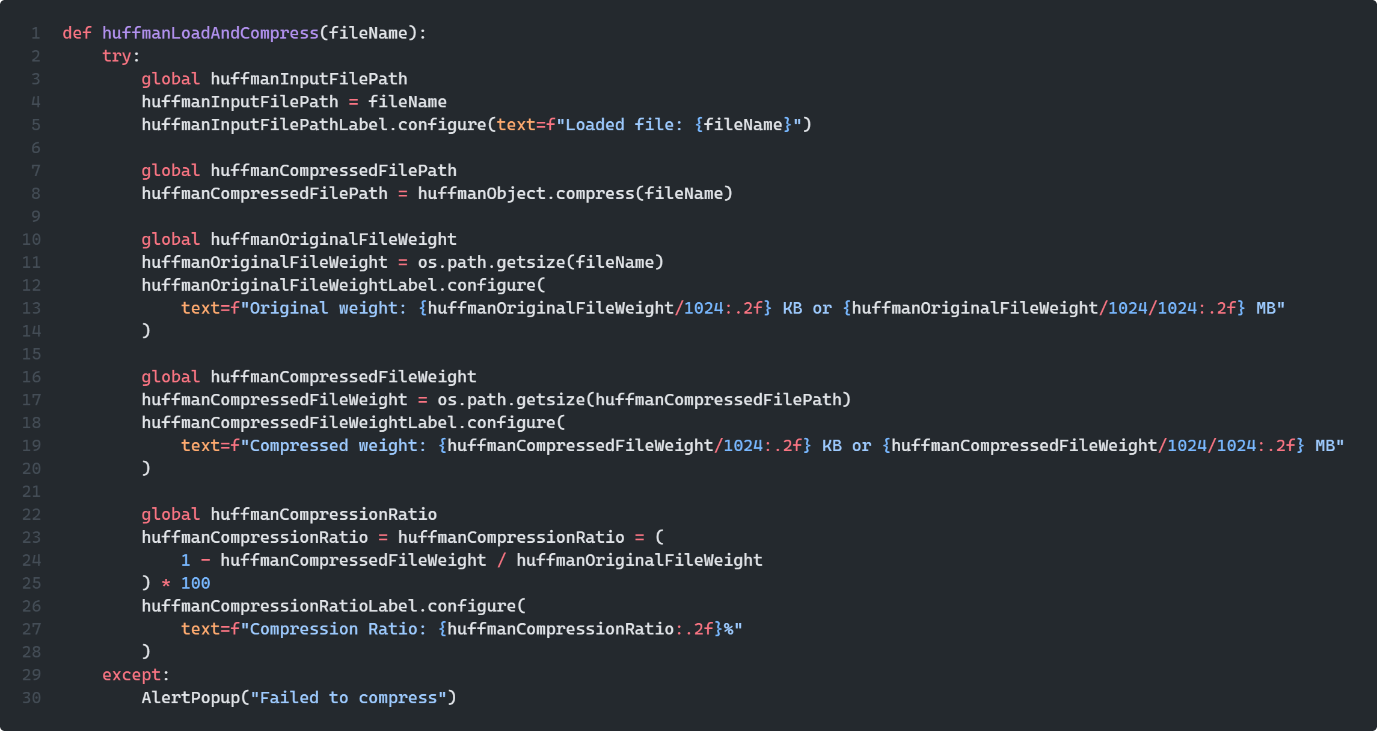


Рисунок 2.11 – Функція, що вирішує індивідуальне завдання на основі класу HuffmanCoding



Рисунок 2.12 – Функція, що дозволяє декодувати закодований текст у індивідуальному завдання на основі класу HuffmanCoding

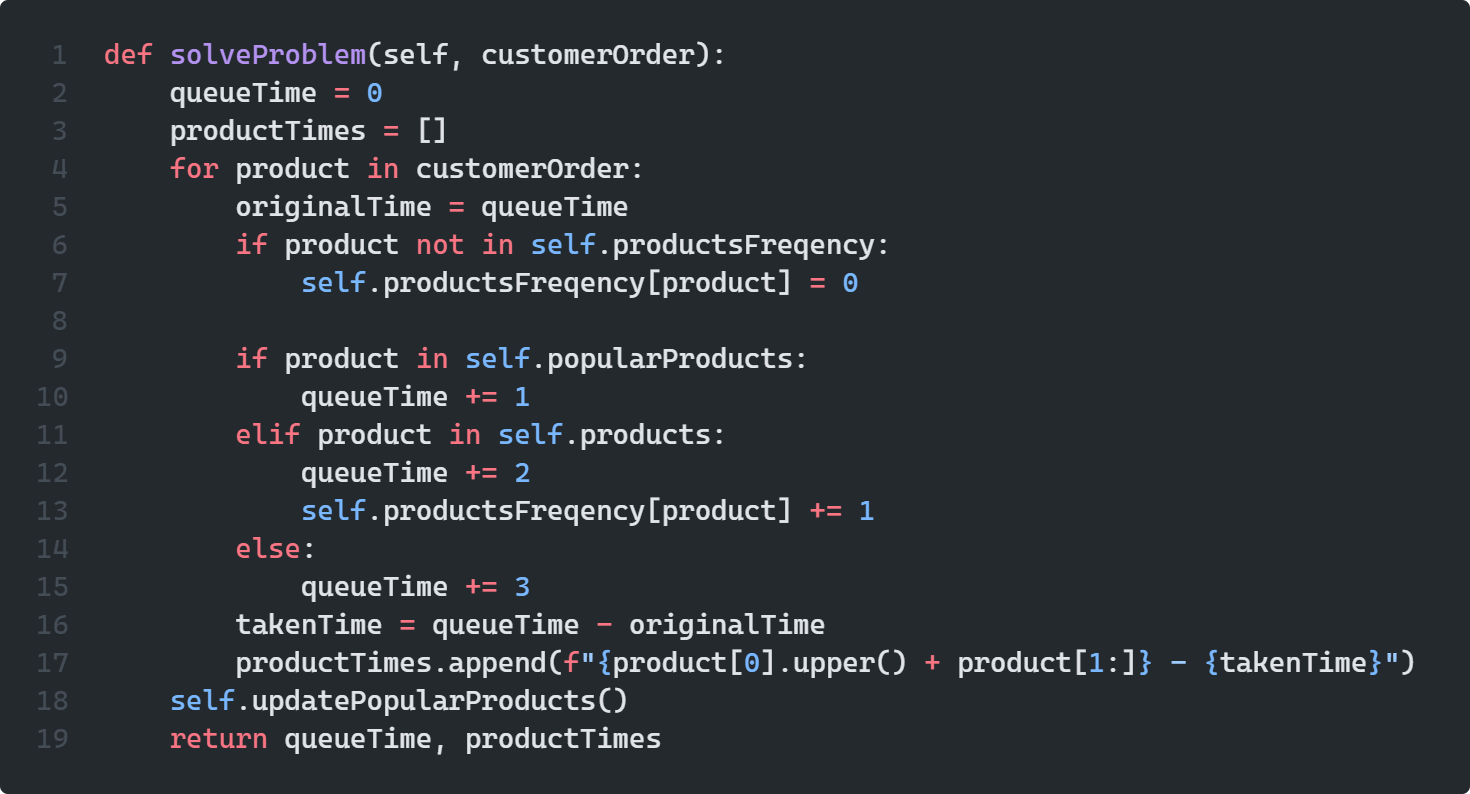


Рисунок 2.13 – Метод, що дозволяє вирішити індивідуальне завдання на основі класу Shopkeeper



Рисунок 2.14 – Функція, що дозволяє вирішити індивідуальне завдання на основі класу Shopkeeper

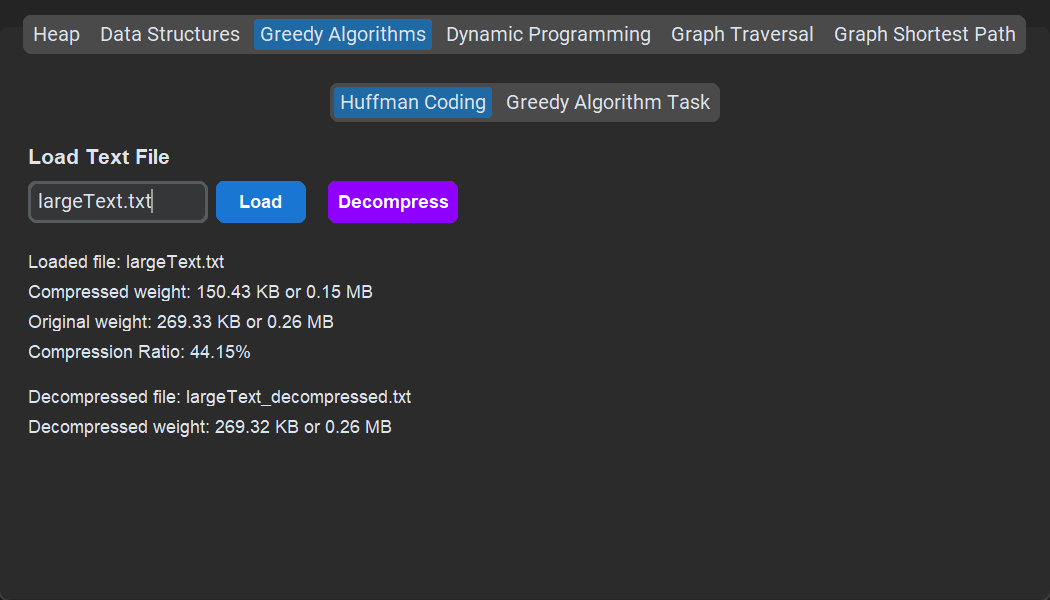


Рисунок 2.15 – Результати виконання індивідуального завдання на основі класу HuffmanCoding

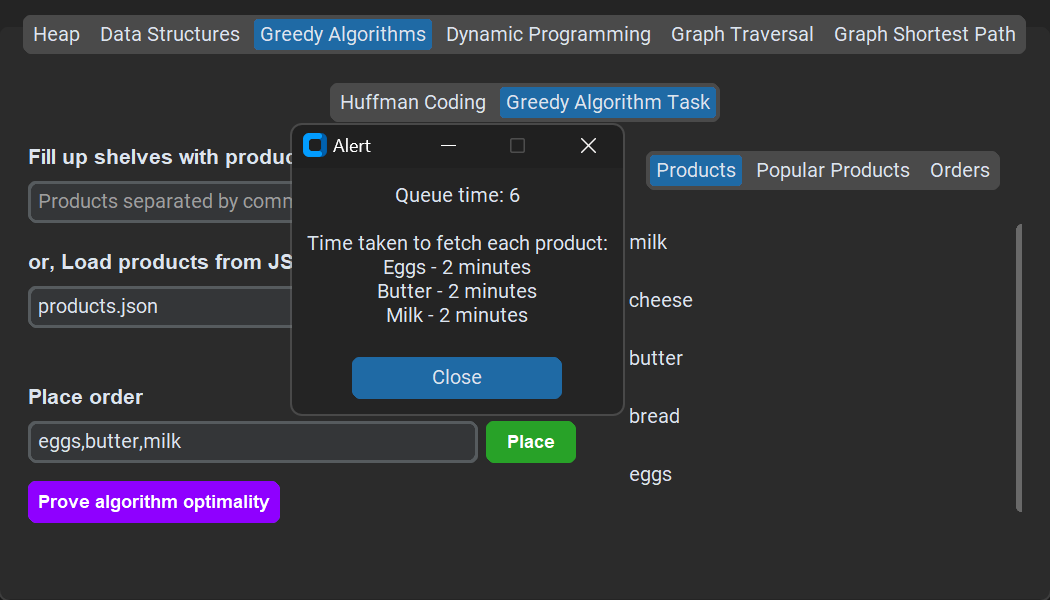


Рисунок 2.16 – Результати виконаня індивідуального завдання на основі класу Shopkeeper

Динамічне програмування

Динамічне програмування - це метод розв'язання складних задач шляхом розбиття їх на простіші підзадачі. Він застосовується до задач, що мають ознаки підзадач, що перекриваються, та оптимальної підструктури.

Для індивідуального завдання з динамічного програмування було реалізовано розв'язання задачі розподілу роботів на групи. Задача була вирішена з використанням підходу динамічного програмування, де для кожної можливої кількості роботів і груп розраховувалася кількість способів розбиття роботів на групи.

Програма дозволяла користувачеві вводити кількість роботів і кількість груп, а програма обчислювала кількість способів розбиття роботів на групи. Результати показали, що підхід динамічного програмування здатен ефективно вирішити проблему.

Друга задача полягала в розташуванні будинків на вулиці на основі заданої матриці. Задача була розв'язана за допомогою методу динамічного програмування, де для кожної можливої кількості будинків і типів будинків було розраховано кількість способів розміщення будинків.

Результати показали, що методи динамічного програмування дозволяють ефективно розв'язувати поставлені задачі.

Фрагменти коду та результати виконання індивідуальних завдань наведено нижче у вигляді знімків з екрану.

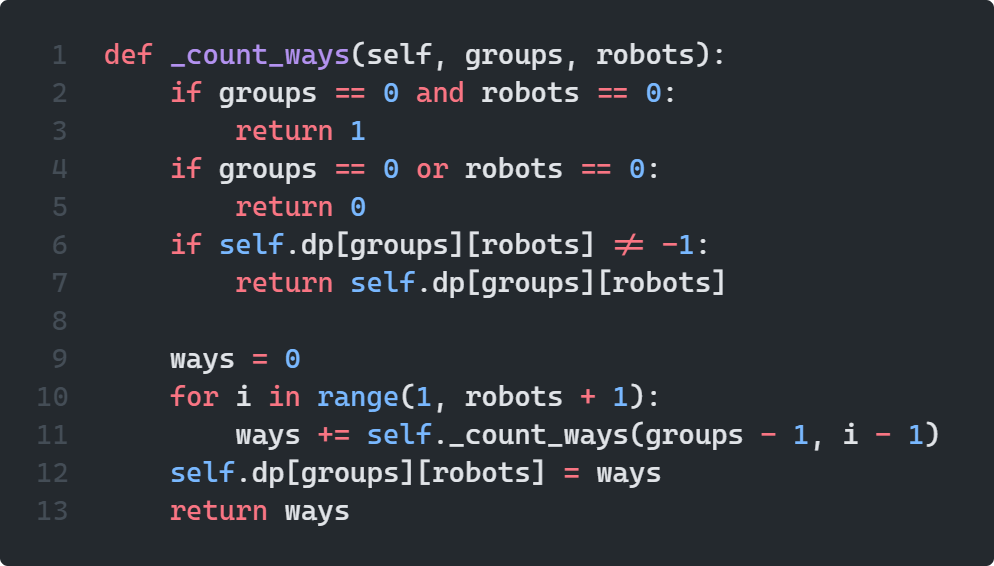


Рисунок 2.17 – Метод, що дозволяє вирішити індивідуальне завдання на основі класу RobotGroup



Рисунок 2.18 – Функція, що дозволяє користувачу вирішити індивідуальне завдання на основі класу RobotGroup

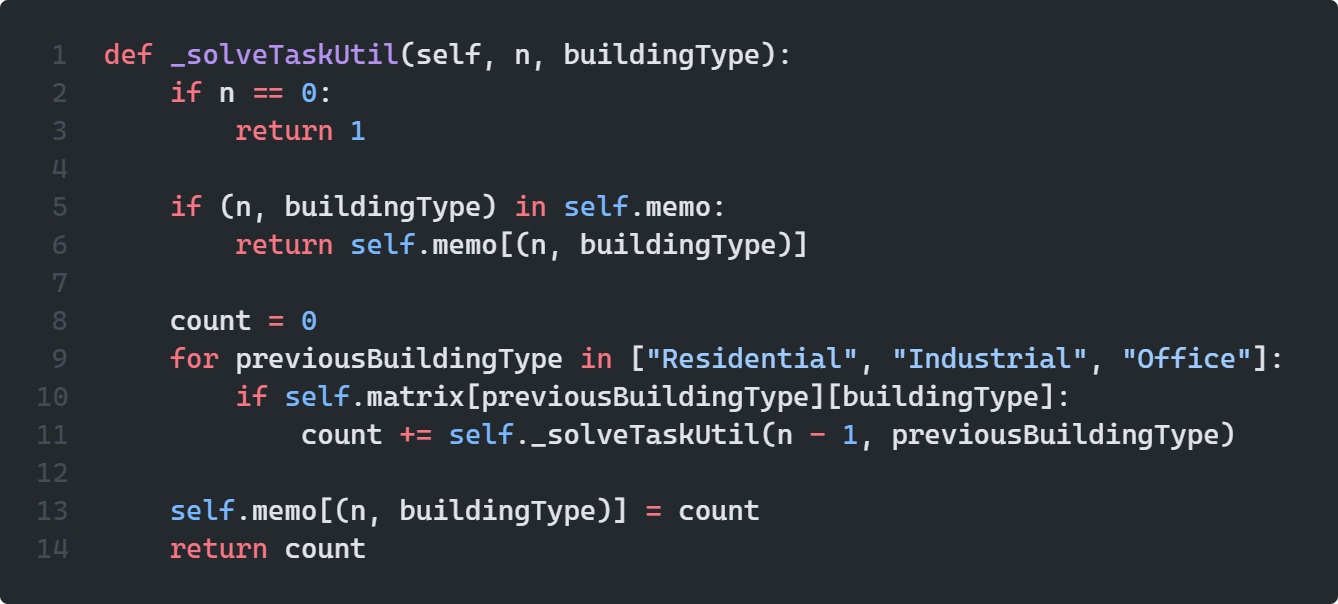


Рисунок 2.19 – Метод, що дозволяє вирішити індивідуальне завдання на основі класу BuildingArrangement



Рисунок 2.20 – Функція, що дозволяє користувачу вирішити індивідуальне завдання на основі класу BuildingArrangement

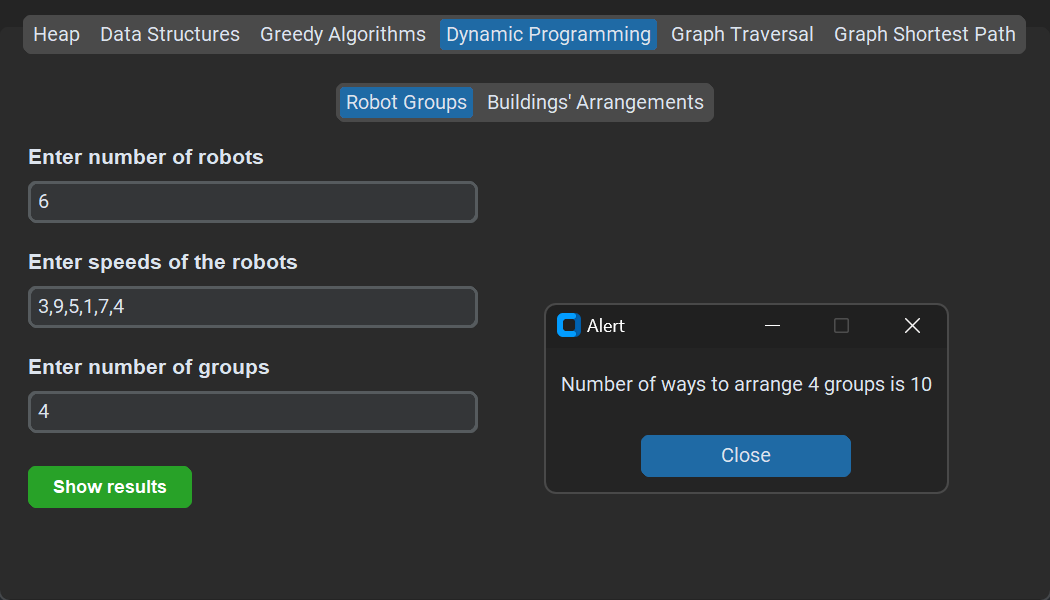


Рисунок 2.21 – Результати виконання індивідуального завдання на основі класу RobotGroup

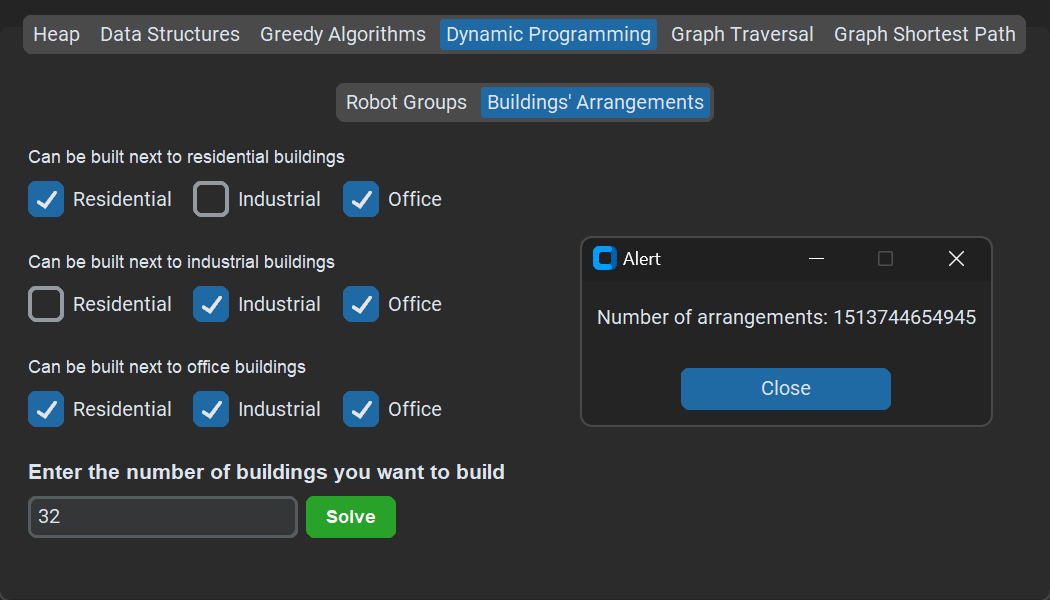


Рисунок 2.22 – Результати виконання індивідуального завдання на основі класу BuildingArrangement

Алгоритми роботи з графами

Алгоритми обходу графа

Алгоритми обходу графів призначені для відвідування кожної вершини графа один раз. Два найпоширеніші типи обходів - це пошук в ширину (BFS) та пошук в глибину (DFS). Ці алгоритми використовуються в багатьох застосунках, включаючи мережеву маршрутизацію, аналіз соціальних мереж та багато інших.

В алгоритмі BFS ми починаємо з обраної вершини (зазвичай кореневої) і відвідуємо всі сусідні з нею вершини. Потім для кожної з цих найближчих вершин ми відвідуємо їхніх невідвіданих сусідів, і так далі, поки не відвідаємо кожну вершину графа. Алгоритм BFS використовує структуру даних у вигляді черги для відстеження вершин, які потрібно відвідати.

DFS - ще один алгоритм обходу графа, який відвідує кожну вершину графа. У DFS ми починаємо з обраної вершини, потім проходимо якомога далі вздовж кожного шляху, перш ніж повернутися назад. DFS використовує стек, щоб не забути почати пошук з наступної вершини, коли на будь-якій ітерації виникає глухий кут.

Для окремих завдань ми реалізували ці алгоритми на Python, візуалізуючи процес за допомогою графічного інтерфейсу. Програма дозволяла користувачеві вводити граф, і вона відображала граф до і після застосування алгоритму BFS або DFS. Результати показали, що ці алгоритми здатні ефективно обходити граф.

Фрагменти коду та результати виконання індивідуальних завдань наведено нижче у вигляді знімків з екрану.

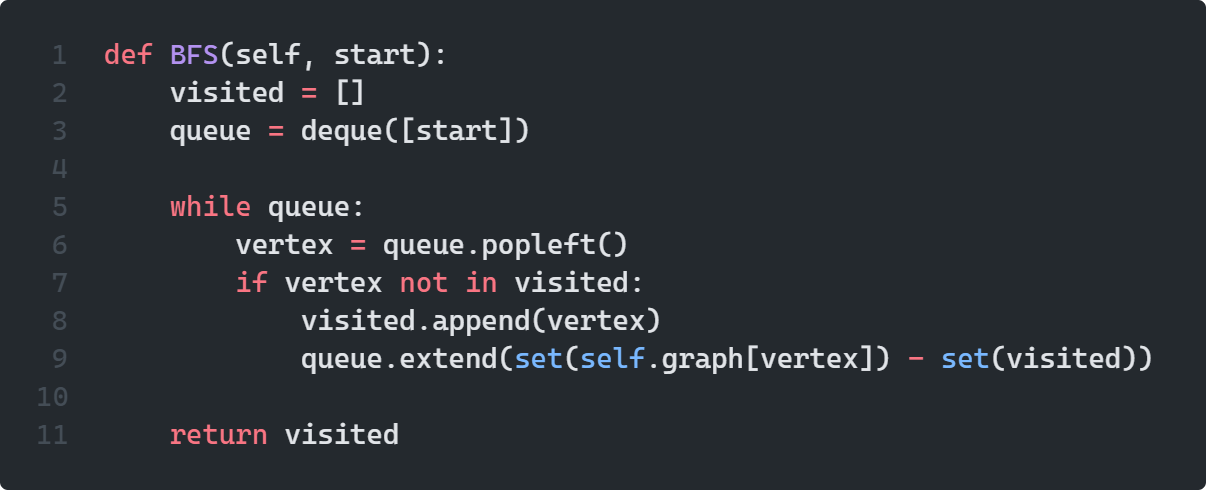


Рисунок 2.23 – Метод, що реалізує пошук в ширину у класі Graph



Рисунок 2.24 – Функція, що дозволяє виконувати пошук в ширину на основі класу Graph

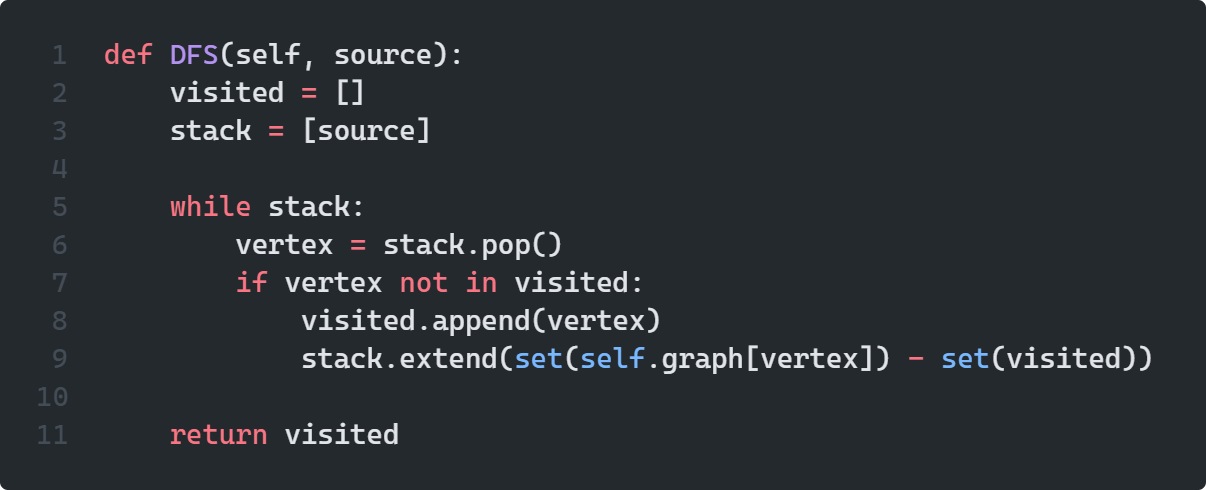


Рисунок 2.25 – Метод, що реалізує пошук в глибину на основі класу Graph



Рисунок 2.26 – Функція, що дозволяє виконувати пошук в глибину на основі методу з класу Graph

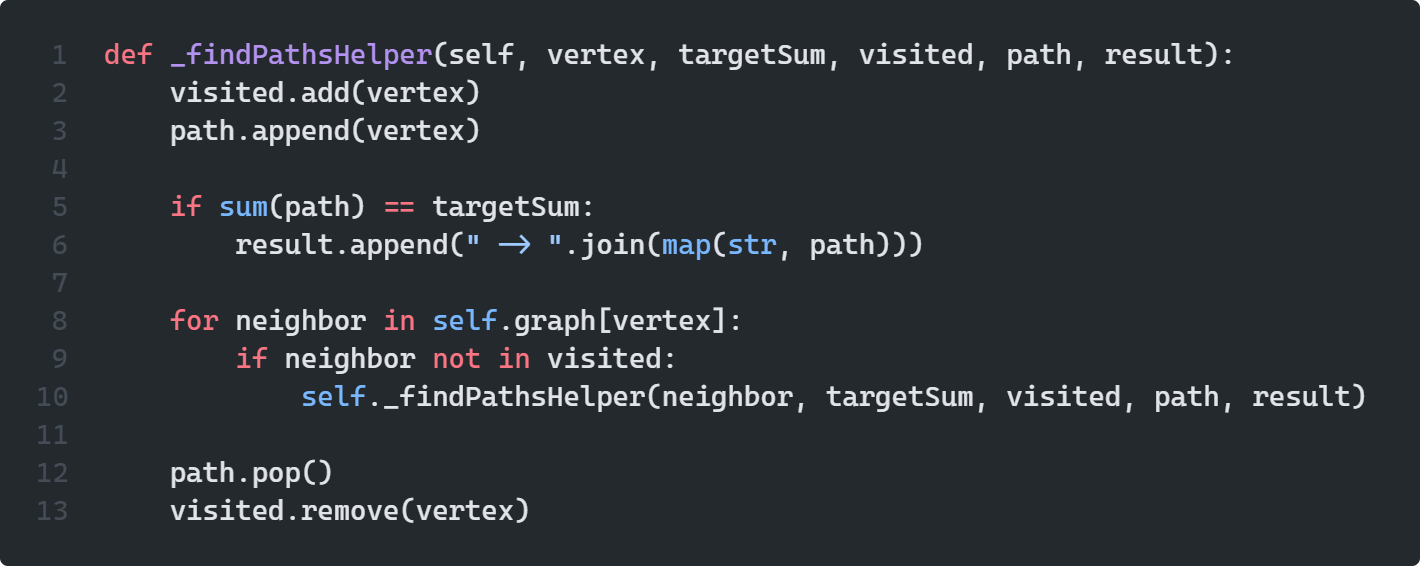


Рисунок 2.27 – Метод, що виконує індивідуальне завдання на основі класу SumGraph

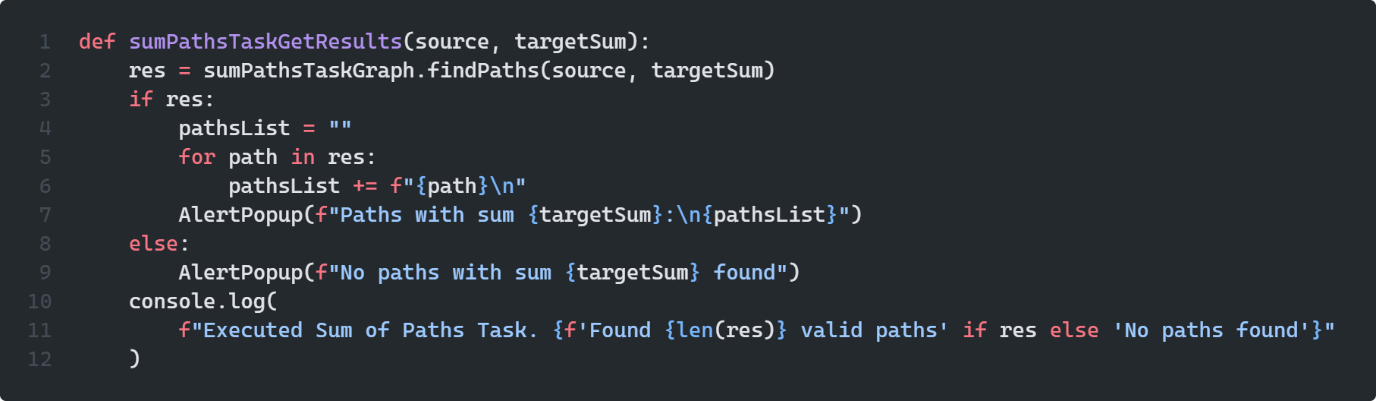


Рисунок 2.28 – Функція, що дозволяє виконати індивідуальне завдання на основі методу з класу SumGraph

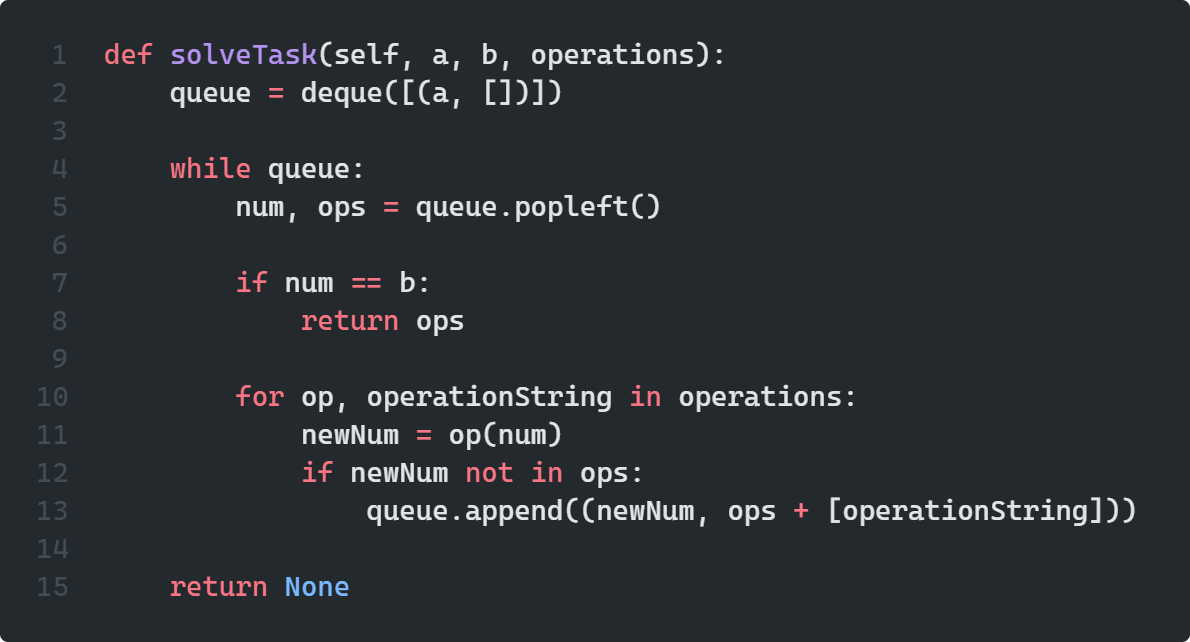


Рисунок 2.29 – Метод, що виконує індивідуальне завдання на основі класу MinOperationsGraph

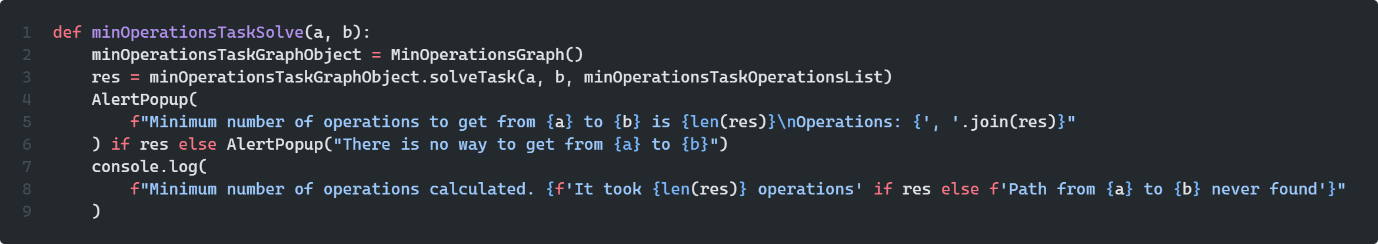


Рисунок 2.30 – Функція, що дозволяє виконати індивідуальне завдання на основі методу з класу MinOperationsGraph

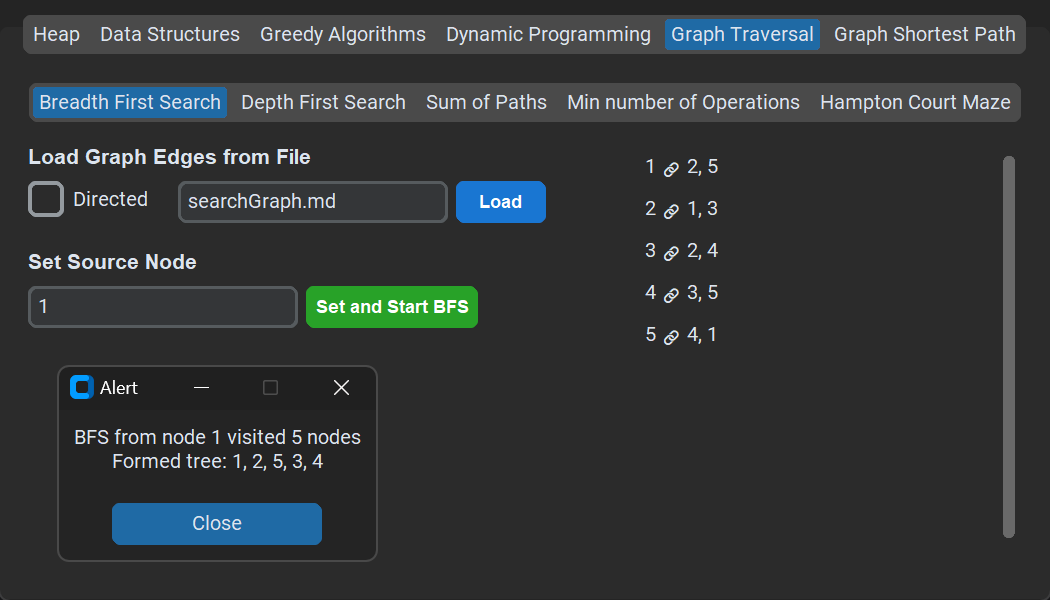


Рисунок 2.31 – Результати виконання пошуку в ширину

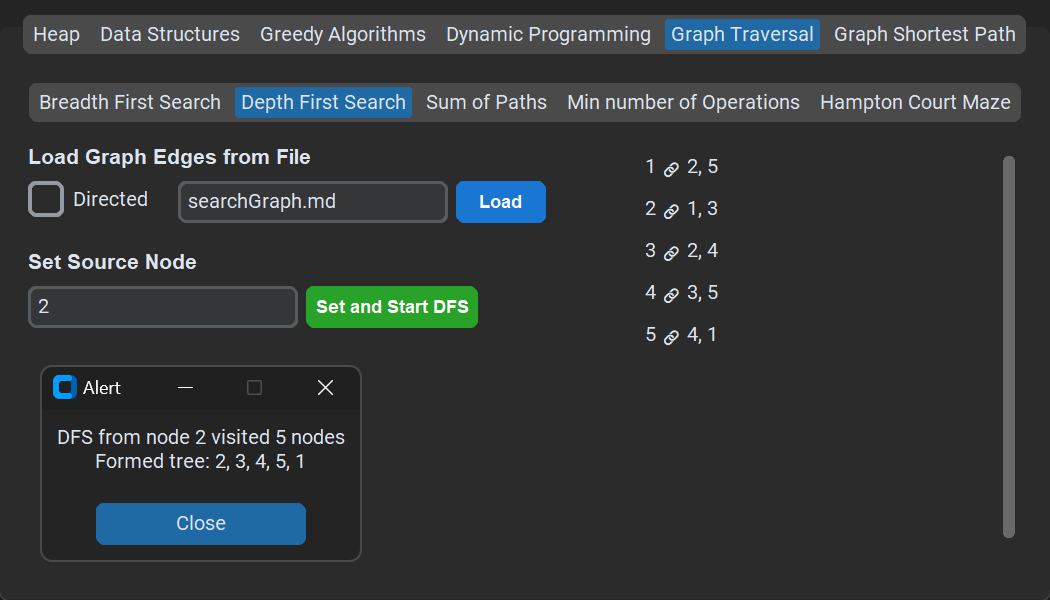


Рисунок 2.32 – Результати виконання пошуку в глибину

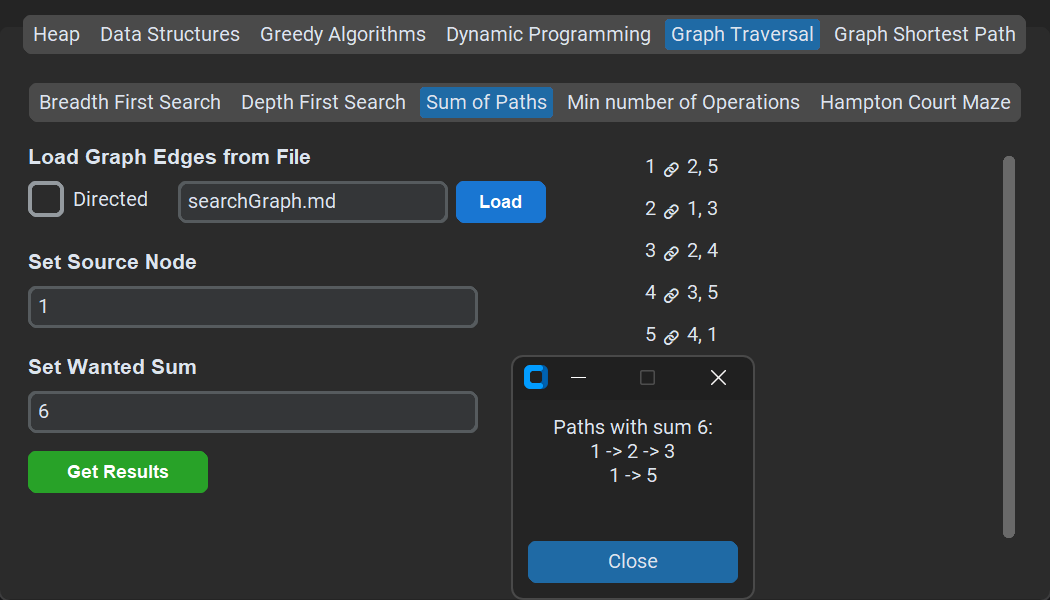


Рисунок 2.33 – Результати виконання індивідуального завдання

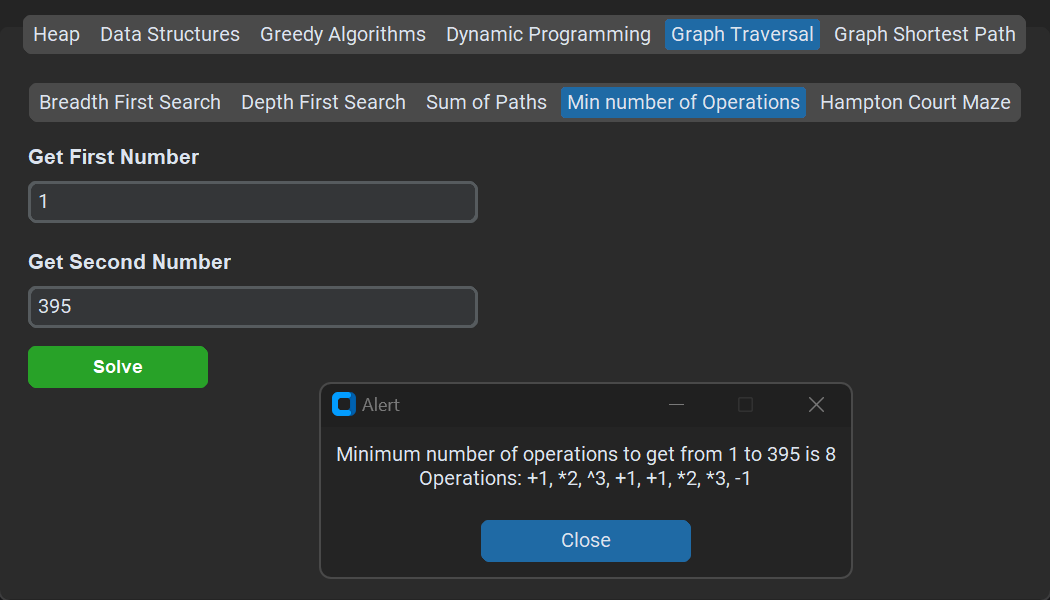


Рисунок 2.34 – Результати виконання індивідуального завдання

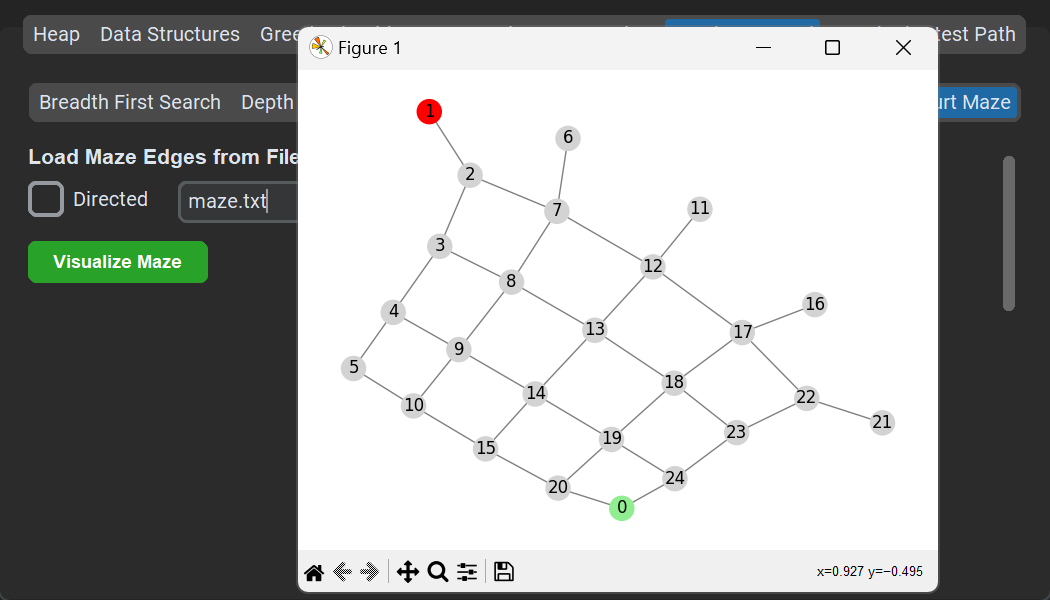


Рисунок 2.35 – Результати виконання індивідуального завдання

Алгоритми пошуку найкоротшого шляху

Алгоритми пошуку найкоротшого шляху призначені для пошуку найкоротшого шляху між двома вершинами графа. Найпоширенішими алгоритмами для цієї мети є алгоритм Дейкстри, алгоритм Беллмана-Форда та алгоритм Флойда-Уоршалла.

Алгоритм Дейкстри працює шляхом створення дерева найкоротших шляхів від початкової вершини до всіх інших точок графа. Алгоритм Беллмана-Форда працює шляхом ітеративного розслаблення ребер графа, що поступово призводить до знаходження найкоротшого шляху. Алгоритм Флойда-Уоршалла працює шляхом розбиття задачі на менші підзадачі, які потім вирішуються за принципом "знизу вгору".

Для окремих завдань ми реалізували ці алгоритми мовою Python, візуалізуючи процес за допомогою графічного інтерфейсу. Програма дозволяла користувачеві вводити граф, і вона відображала граф до і після застосування алгоритму найкоротшого шляху. Результати показали, що ці алгоритми здатні ефективно знаходити найкоротший шлях.

Фрагменти коду та результати виконання індивідуальних завдань наведено нижче у вигляді знімків з екрану.

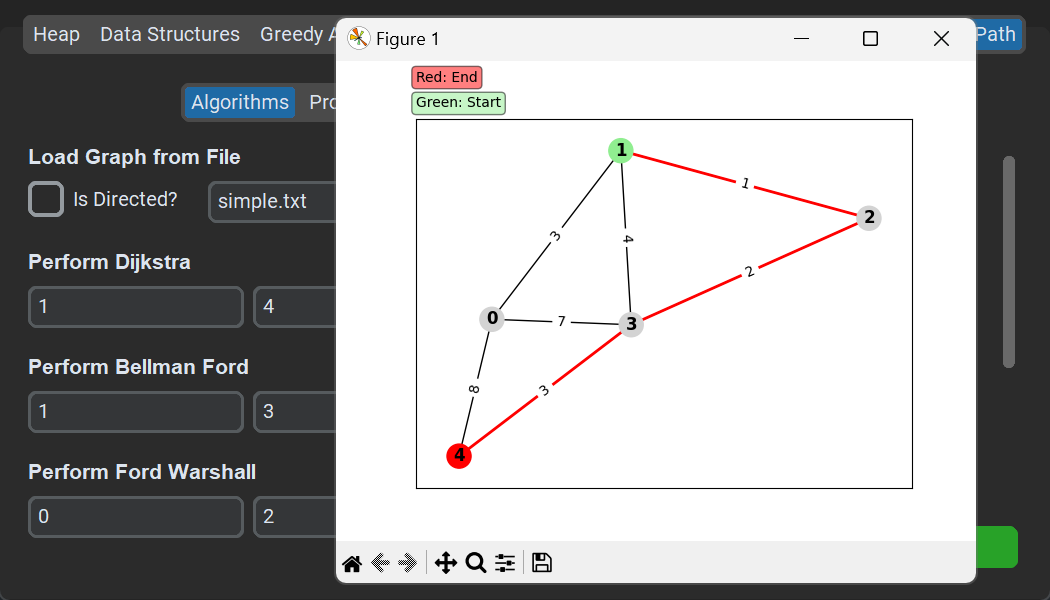


Рисунок 2.36 – Результат виконання алгоритму Дайкстри

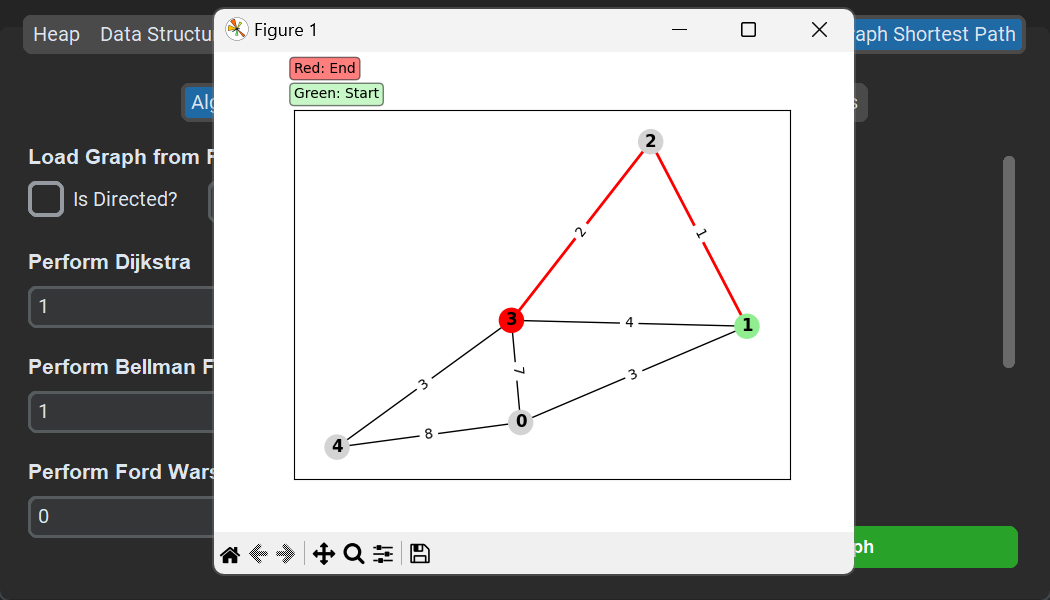


Рисунок 2.37 – Результат виконання алгоритму Белмана Форда

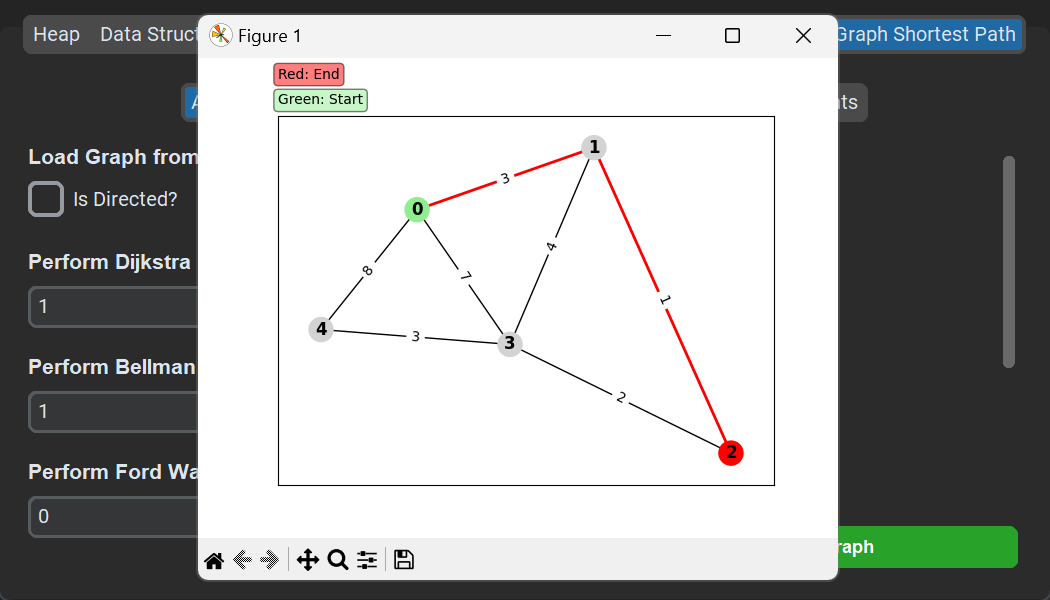


Рисунок 2.38 – Результат виконання алгоритму Флойда Уоршела

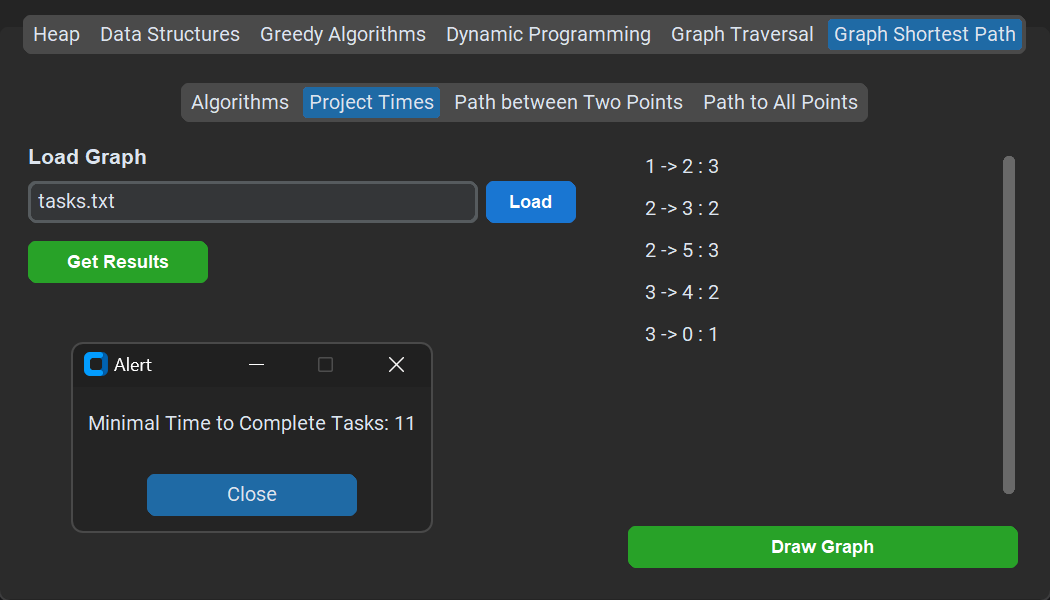


Рисунок 2.39 – Результати виконання індивідуального завдання

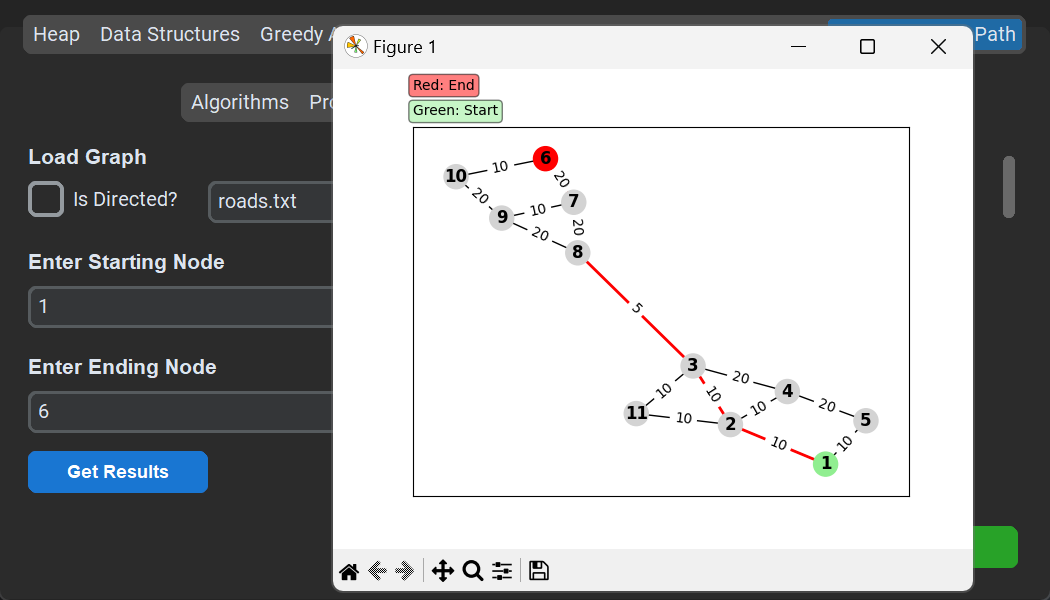


Рисунок 2.40 – Результати виконання індивідуального завдання

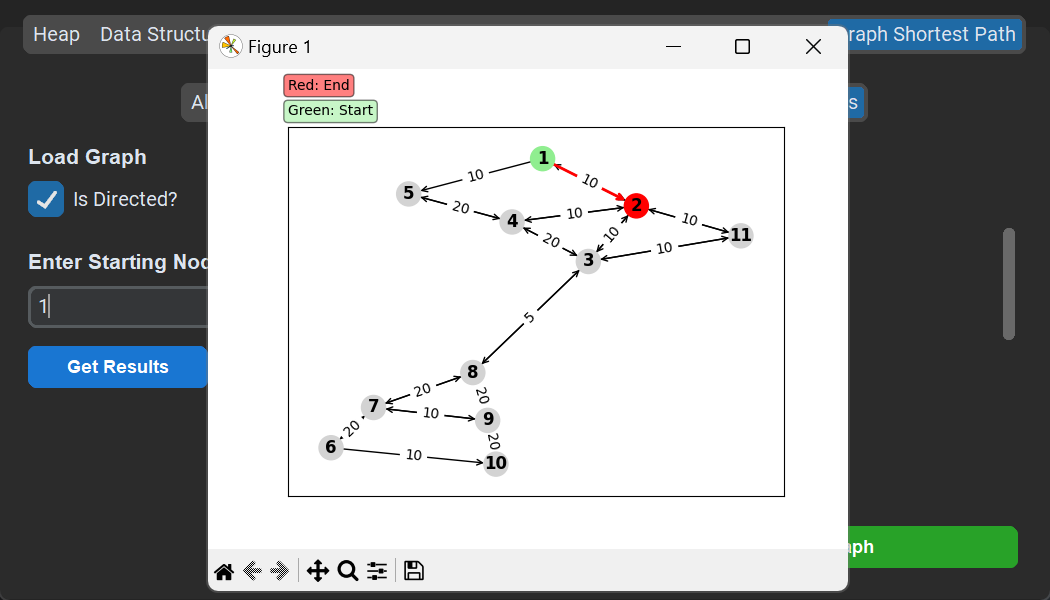


Рисунок 2.41 – Результати виконання індивідуального завдання

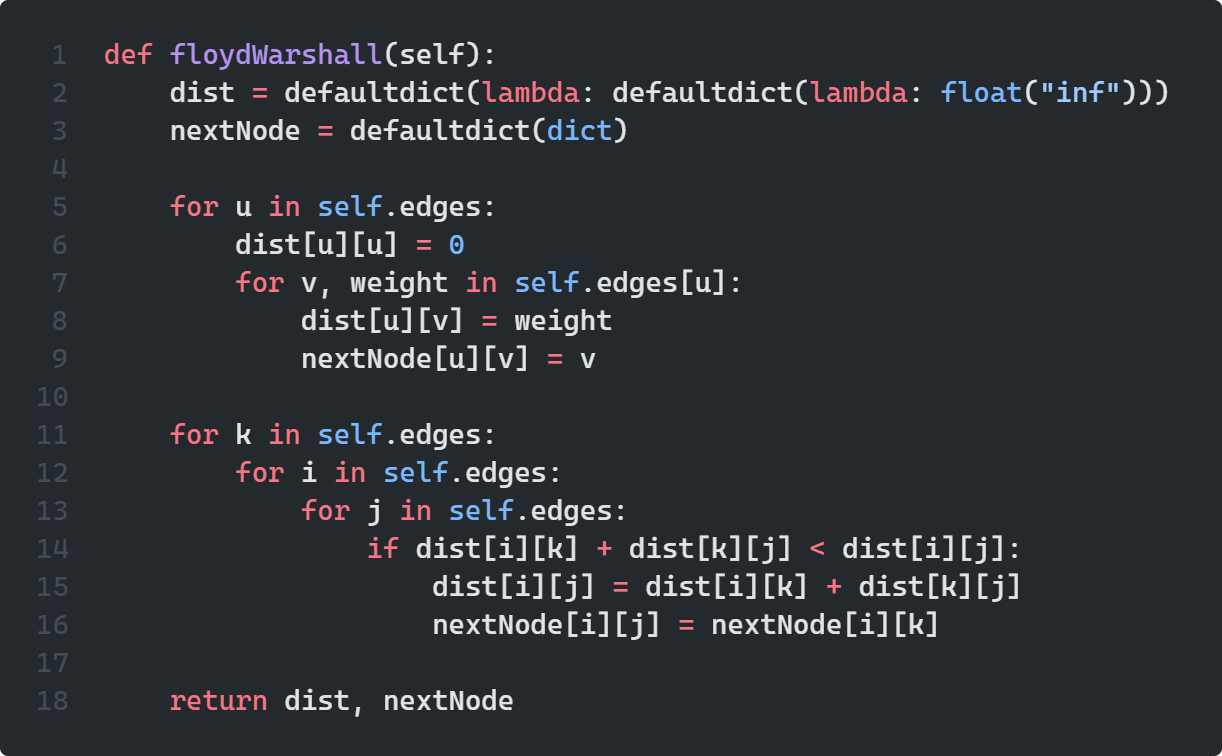
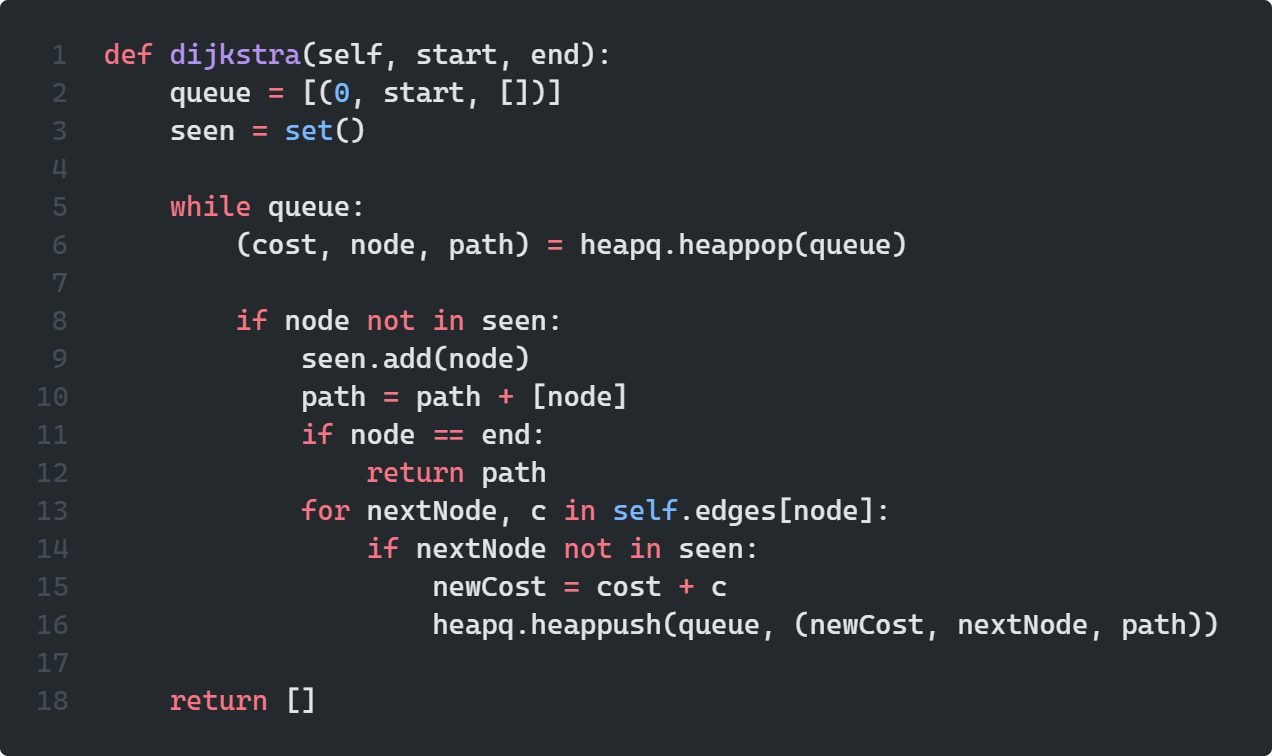


Рисунок 2.42 – Метод, що виконує алгоритм Флойда Уоршела

  
Рисунок 2.43 – Метод, що виконує алгоритм Дайкстри

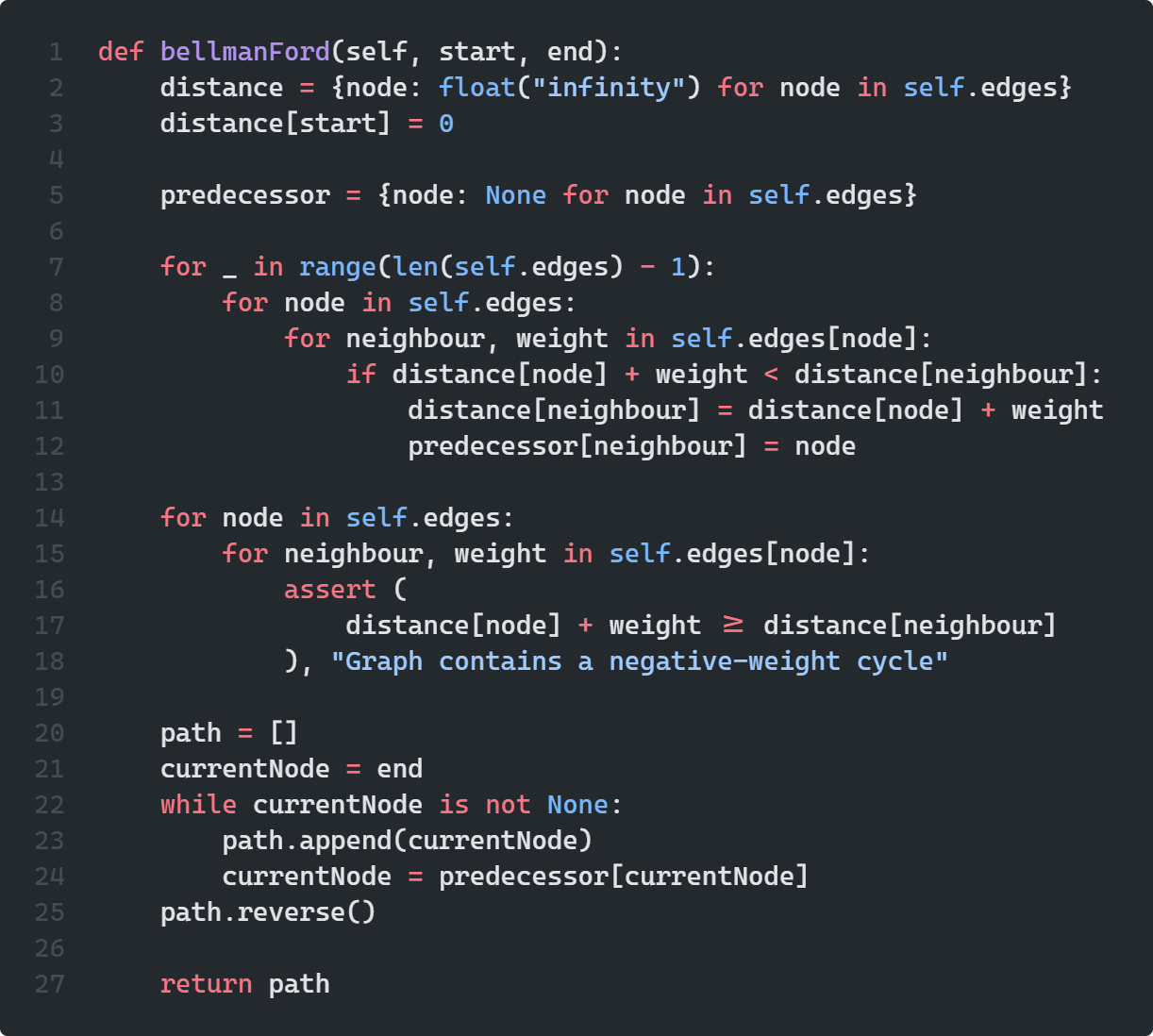


Рисунок 2.44 – Метод, що виконує алгоритм Белмана Форда

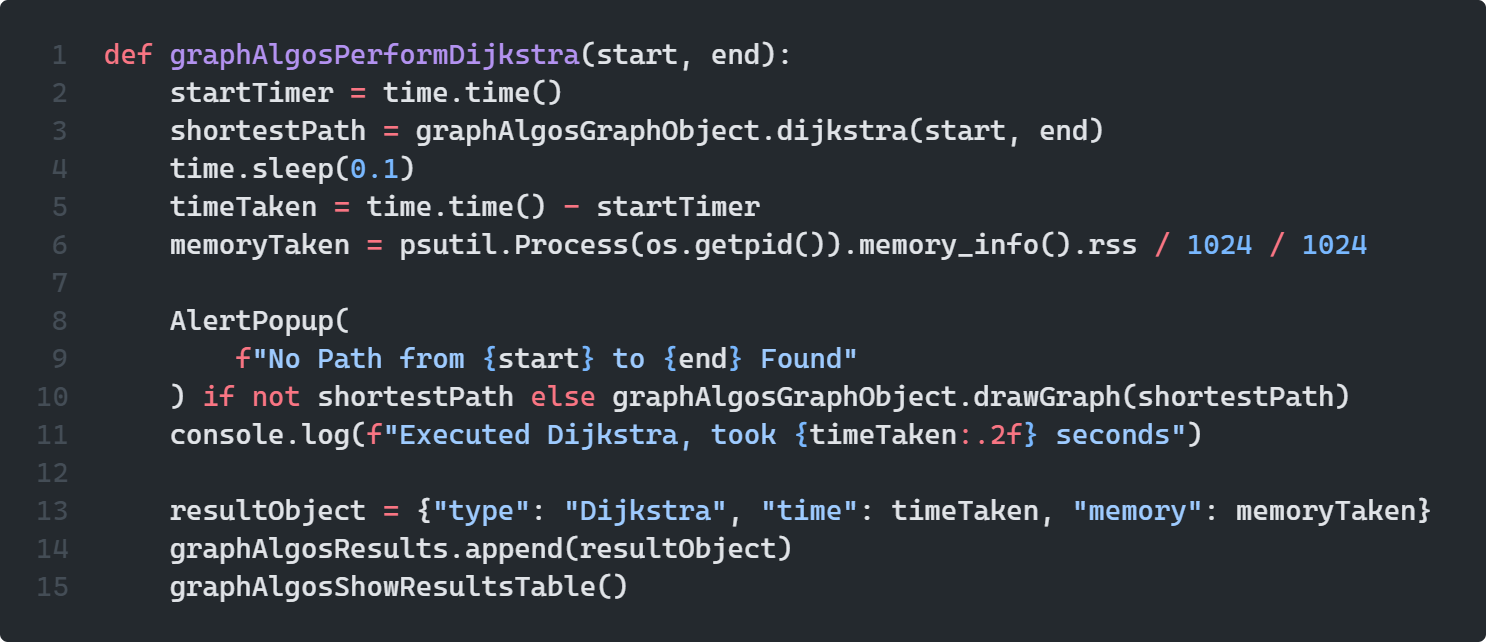


Рисунок 2.45 – Функція, що дозволяє виконати алгоритм Дайкстри

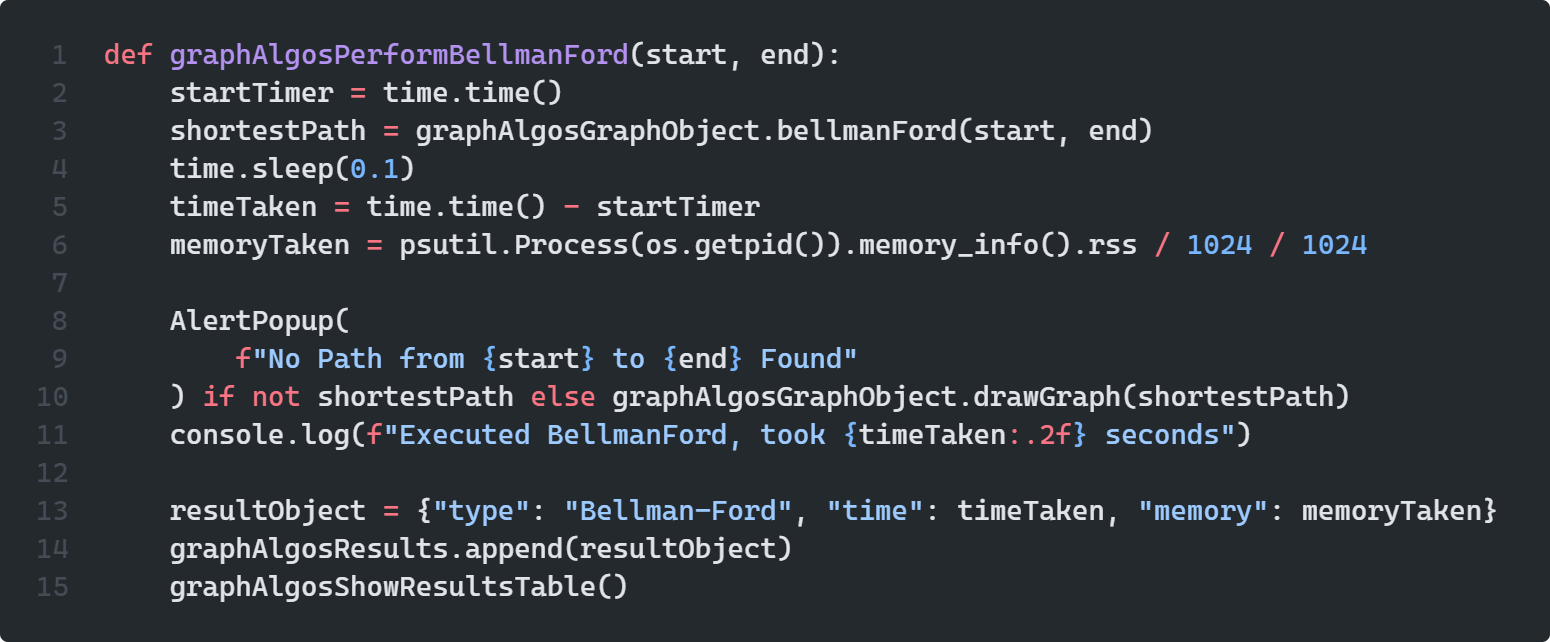


Рисунок 2.46 – Функція, що дозволяє виконати алгоритм Белмана Форда

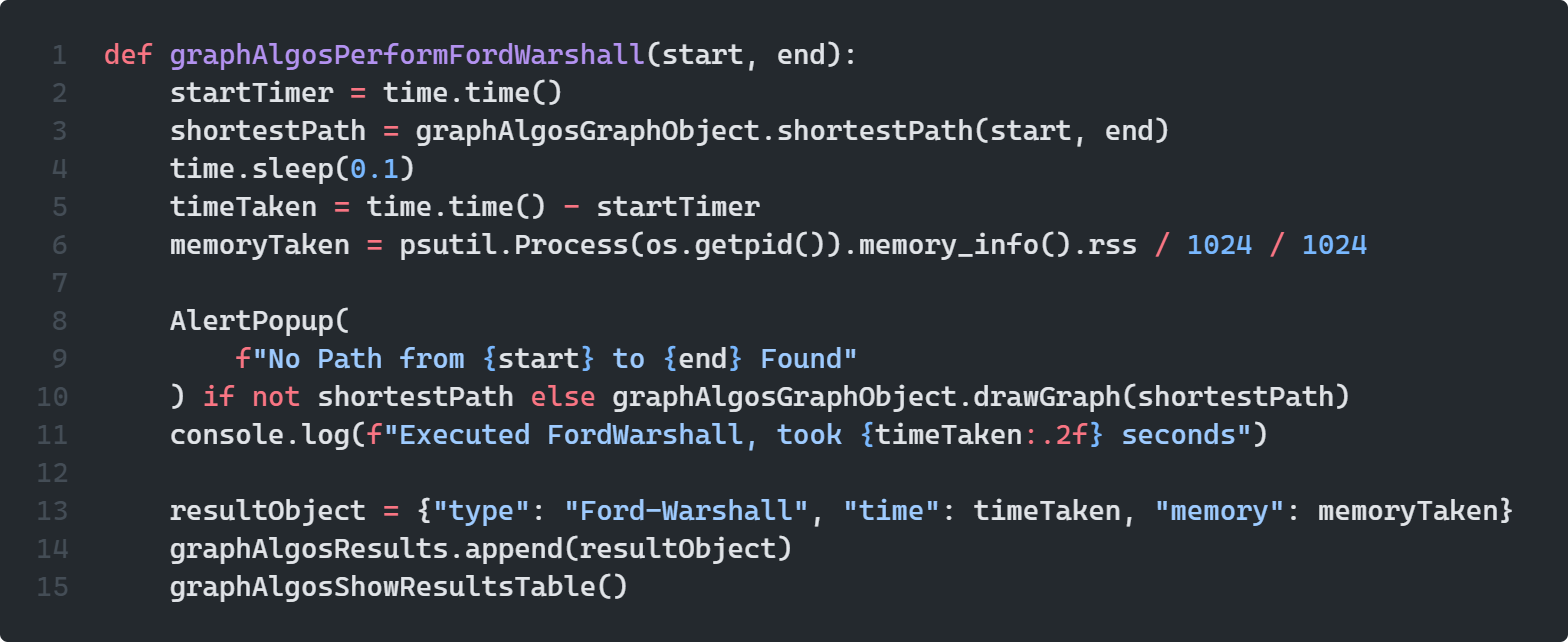


Рисунок 2.47 – Функція, що дозволяє виконати алгоритм Флойда Уоршела

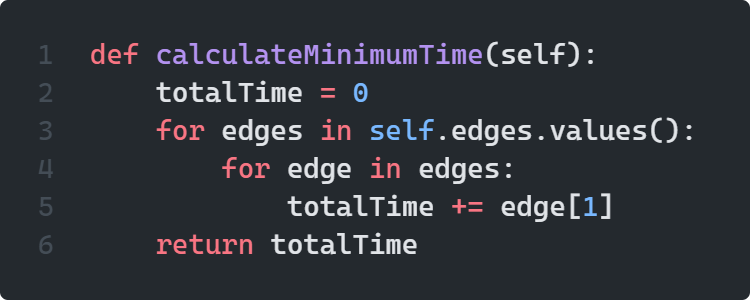


Рисунок 2.48 – Метод, що дозволяє виконати індивідуальне завдання

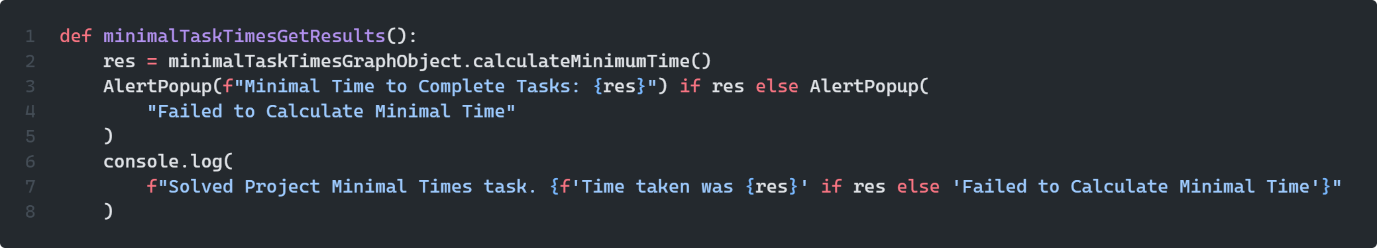


Рисунок 2.49 – Функція, що дозволяє виконати індивідуальне завдання

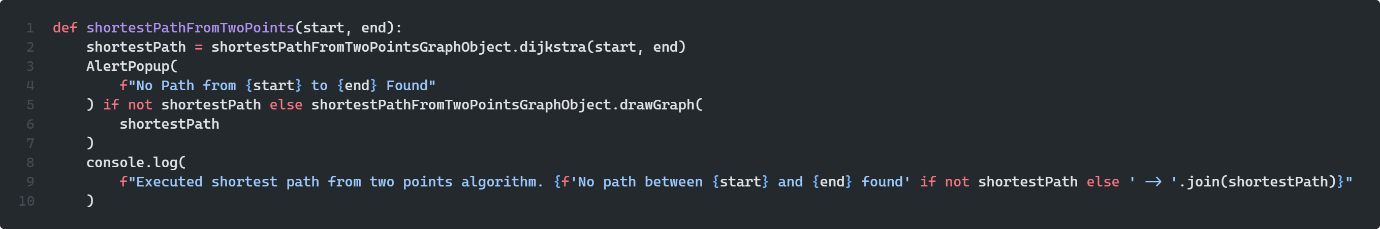


Рисунок 2.50 – Функція, що дозволяє виконати індивідуальне завдання

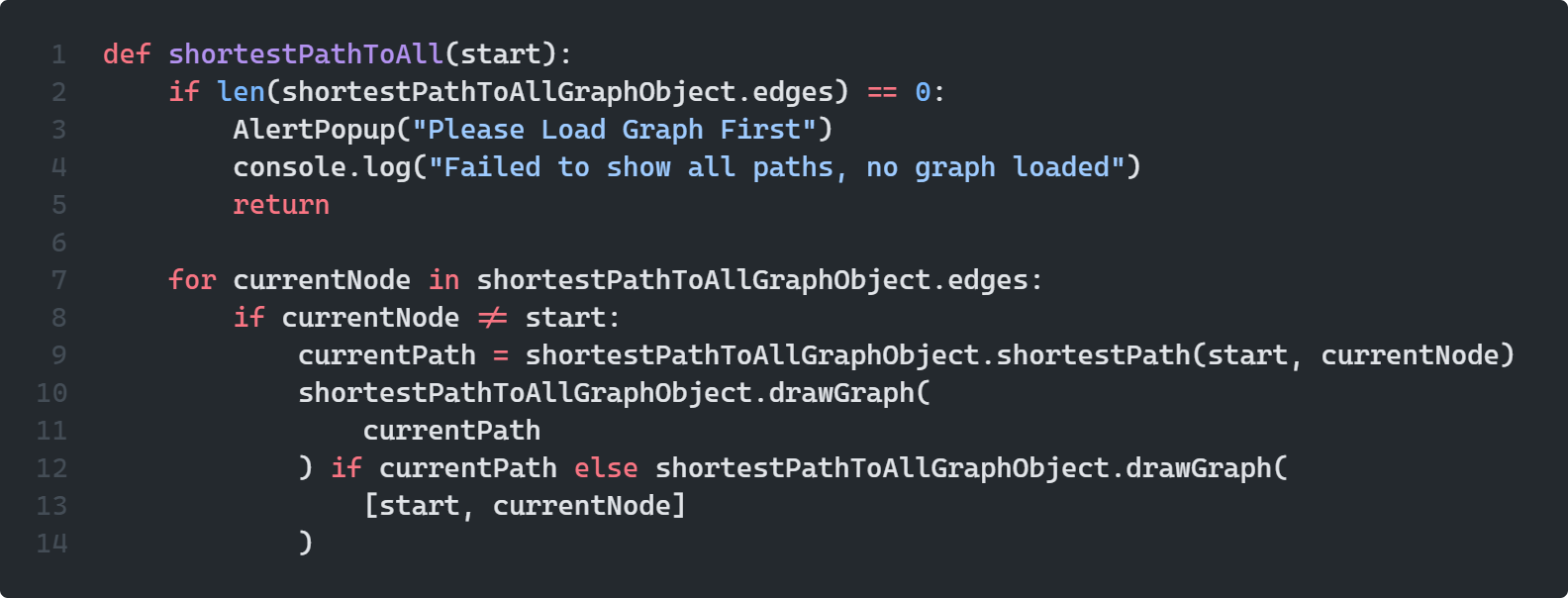


Рисунок 2.51 – Функція, що дозволяє виконати індивідуальне завдання

Висновки

Результати цієї роботи підтверджують, що розуміння та реалізація структур даних та алгоритмів є фундаментальною навичкою в інформатиці. Програма, розроблена в цьому проєкті, є комплексною демонстрацією різних структур даних, таких як купа, зв'язаний список, хеш-таблиця, B-дерево, та алгоритмів, таких як сортування купи, кодування Хаффмана, жадібні алгоритми, динамічне програмування, BFS, DFS, алгоритм Дейкстри, алгоритм Флойда-Уоршалла та алгоритм Беллмана-Форда.

Проєкт демонструє практичну реалізацію цих концепцій на мові Python, з акцентом на візуалізацію процесу за допомогою графічного інтерфейсу. Він показує, як ці алгоритми та структури даних можуть бути використані для ефективного вирішення складних завдань.

Наприклад, сортування купою демонструє ефективний спосіб сортування великих наборів даних з часовою складністю O(n log n). Аналогічно, використання хешування та B-дерев як структур даних забезпечило ефективні засоби для зберігання та пошуку даних.

Використання передових методів розробки та аналізу, таких як жадібні алгоритми та динамічне програмування, продемонструвало, як складні проблеми можна вирішувати, роблячи локально оптимальний вибір або розбиваючи їх на простіші підпроблеми.

Алгоритми обходу графів, такі як BFS і DFS, були реалізовані для відвідування кожної вершини графа, в той час як алгоритми пошуку найкоротшого шляху, такі як Дейкстри, Беллмана-Форда і Флойда-Уоршалла, були використані для пошуку найкоротшого шляху між вершинами графа.

Успішна реалізація та застосування цих структур даних і алгоритмів для розв'язання різноманітних задач підтверджує їх важливість в інформатиці. Проєкт також підкреслив необхідність вибору правильної структури даних або алгоритму, виходячи з поставленої задачі, для забезпечення ефективності.

Результати цієї роботи можуть бути використані в різних галузях комп'ютерних наук, включаючи розробку нових мов програмування, баз даних і програмних систем. Вони також можуть сприяти вивченню і дослідженням в області алгоритмів і структур даних.

На закінчення, цей проєкт підкреслює важливість глибокого розуміння алгоритмів і структур даних для всіх, хто прагне зробити кар'єру в галузі комп'ютерних наук або суміжних галузях.

Додатки

Додаток А – Повний код програми

from customtkinter import \*

import time

import json

import heapq

import os

import random

import sys

import psutil

import matplotlib.pyplot as plt

import networkx as nx

from collections import defaultdict, deque

from rich import print

from rich.console import Console

from rich.theme import Theme

from rich.table import Table

from rich.traceback import install

from rich.markdown import Markdown as md

install()

consoleTheme = Theme(

{

"warning": "bold yellow",

"error": "bold red",

"success": "bold green",

"info": "bold blue",

}

)

console = Console(theme=consoleTheme)

class AlertPopup(CTkToplevel):

def \_\_init\_\_(self, message: str, title: str = "Alert"):

super().\_\_init\_\_()

self.resizable(False, False)

self.title(title)

label = CTkLabel(master=self, text=message)

label.pack(padx=10, pady=10, anchor="center")

button = CTkButton(master=self, text="Close", command=self.close\_dialog)

button.pack(padx=10, pady=10, anchor="center")

self.grab\_set()

self.lift()

self.bind("<Escape>", self.close\_dialog)

def close\_dialog(self, event=None):

self.destroy()

app = CTk()

app.title("Data Structures and Algorithms")

app.geometry("700x400")

app.resizable(False, False)

appTabsContainer = CTkTabview(app)

appTabsContainer.pack(expand=True, fill="both")

appTabsContainer.add("Heap")

appTabsContainer.add("Data Structures")

appTabsContainer.add("Greedy Algorithms")

appTabsContainer.add("Dynamic Programming")

appTabsContainer.add("Graph Traversal")

appTabsContainer.add("Graph Shortest Path")

tabHeap = appTabsContainer.tab("Heap")

tabDataStructures = appTabsContainer.tab("Data Structures")

tabGreedyAlgorithms = appTabsContainer.tab("Greedy Algorithms")

tabDynamicProgramming = appTabsContainer.tab("Dynamic Programming")

tabGraphTraversalAlgorithms = appTabsContainer.tab("Graph Traversal")

tabGraphShortestPathsAlgorithms = appTabsContainer.tab("Graph Shortest Path")

class Heap:

def \_\_init\_\_(self):

self.heap = []

def insert(self, value):

self.heap.append(value)

self.\_heapifyUp(givenIndex=len(self.heap) - 1)

def delete(self):

if len(self.heap) == 0:

return None

self.\_swap(0, len(self.heap) - 1)

root = self.heap.pop()

self.\_heapifyDown(parentIndex=0)

return root

def sort(self):

sortedItems = []

for \_ in range(len(self.heap)):

sortedItems.append(self.delete())

return sortedItems

def search(self, value):

for index, item in enumerate(self.heap):

if item == value:

return index

return -1

def buildHeap(self, arr):

self.heap = arr

start = len(arr) // 2

for i in reversed(range(start + 1)):

self.\_heapifyDown(parentIndex=i)

return self

def \_heapifyUp(self, givenIndex):

parentIndex = (givenIndex - 1) // 2

if parentIndex >= 0 and self.heap[givenIndex] > self.heap[parentIndex]:

self.\_swap(parentIndex, givenIndex)

self.\_heapifyUp(givenIndex=parentIndex)

def \_heapifyDown(self, parentIndex):

leftChildIndex = 2 \* parentIndex + 1

rightChildIndex = 2 \* parentIndex + 2

largest = parentIndex

if (

leftChildIndex < len(self.heap)

and self.heap[leftChildIndex] > self.heap[largest]

):

largest = leftChildIndex

if (

rightChildIndex < len(self.heap)

and self.heap[rightChildIndex] > self.heap[largest]

):

largest = rightChildIndex

if largest != parentIndex:

self.\_swap(parentIndex, largest)

self.\_heapifyDown(parentIndex=largest)

def \_swap(self, i, j):

self.heap[i], self.heap[j] = self.heap[j], self.heap[i]

def updateHeapElementsContainer():

for widget in heapElementsContainer.winfo\_children():

widget.destroy()

for heapNode in heapElements.heap:

CTkLabel(heapElementsContainer, text=heapNode).pack(padx=5, anchor="w")

def addHeapElement(element):

heapElements.insert(element)

updateHeapElementsContainer()

def deleteHeapElement():

heapElements.delete()

updateHeapElementsContainer()

def quickSortUtil(arr):

n = len(arr)

if n < 2:

return arr

pivot = arr[n // 2]

less = [x for x in arr if x < pivot]

equal = [x for x in arr if x == pivot]

more = [x for x in arr if x > pivot]

return quickSortUtil(more) + equal + quickSortUtil(less)

def sortHeap(sortingType: str):

global heapSortingResults

startTimer = time.time()

heapElements.heap = (

heapElements.sort()

if sortingType == "Heap Sort"

else quickSortUtil(heapElements.heap)

if sortingType == "Quick Sort"

else sorted(heapElements.heap, reverse=True)

)

time.sleep(0.1)

sortingTime = time.time() - startTimer

updateHeapElementsContainer()

sortingMemory = sys.getsizeof(heapElements.heap)

AlertPopup(

f"Sorting took {sortingTime:.2f} seconds or {sortingTime\*1000:.2f} milliseconds"

)

console.log(

f"Completed sorting heap using {'Heap Sort' if sortingType == 'Heap Sort' else 'Quick Sort' if sortingType == 'Quick Sort' else 'Default Sort'}, took {sortingTime:.2f} seconds"

)

sortingResults = {"type": sortingType, "time": sortingTime, "memory": sortingMemory}

heapSortingResults.append(sortingResults)

def showResultsTable():

sortingResultsTable = Table(

title="Sorting Results",

)

sortingResultsTable.add\_column("Sorting Type", style="white")

sortingResultsTable.add\_column("Time - [cyan]seconds[/]")

sortingResultsTable.add\_column("Memory - [cyan]bytes[/]")

sortingTypesCount = {result["type"]: 0 for result in heapSortingResults}

averageTimes = {result["type"]: 0 for result in heapSortingResults}

averageMemories = {result["type"]: 0 for result in heapSortingResults}

for result in heapSortingResults:

sortingTypesCount[result["type"]] += 1

averageTimes[result["type"]] += result["time"]

averageMemories[result["type"]] += result["memory"]

for sortingType in averageTimes:

averageTimes[sortingType] /= sortingTypesCount[sortingType]

averageMemories[sortingType] /= sortingTypesCount[sortingType]

for sortingType in sortingTypesCount:

sortingResultsTable.add\_row(

sortingType,

f"{averageTimes[sortingType]:.2f}",

f"{averageMemories[sortingType]:.2f}",

)

console.print(sortingResultsTable)

showResultsTable()

def searchHeapElement(value):

index = heapElements.search(value)

if index == -1:

AlertPopup(f"{value} not found")

else:

AlertPopup(f"{value} found at index {index}")

console.log(

f"{value} searched in heap, {f'Found at index {index}' if index != -1 else 'Not Found'}"

)

def saveHeapOnExit():

if len(heapElements.heap) == 0:

return

with open("heap.json", "w") as file:

json.dump(heapElements.heap, file)

def loadHeapOnStart():

try:

with open("heap.json", "r") as file:

heapArray = json.load(file)

heapElements.buildHeap(arr=heapArray)

updateHeapElementsContainer()

except:

AlertPopup("Failed to load heap")

tabHeap\_SubTabsContainer = CTkTabview(tabHeap)

tabHeap\_SubTabsContainer.add("Heap")

tabHeap\_SubTabsContainer.add("Linked List")

tabHeap\_SubTabsContainer.add("Task")

tabHeap\_SubTabsContainer.pack(fill="both", expand=True)

subtabHeap = tabHeap\_SubTabsContainer.tab("Heap")

subtabLinkedList = tabHeap\_SubTabsContainer.tab("Linked List")

subtabHeapTask = tabHeap\_SubTabsContainer.tab("Task")

heapElementsContainer = CTkScrollableFrame(subtabHeap, width=240, height=270)

heapElementsContainer.place(x=400, y=0)

linkedListElementsContainer = CTkScrollableFrame(

subtabLinkedList, width=240, height=270

)

linkedListElementsContainer.place(x=400, y=0)

heapTaskElementsContainer = CTkScrollableFrame(subtabHeapTask, width=240, height=270)

heapTaskElementsContainer.place(x=400, y=0)

addHeapElementHeading = CTkLabel(

subtabHeap, text="Add to Heap", font=("Arial", 14, "bold")

)

addHeapElementHeading.place(x=0, y=0)

addHeapElementInput = CTkEntry(

subtabHeap, placeholder\_text="Enter element...", width=300

)

addHeapElementInput.place(x=0, y=30)

addHeapElementButton = CTkButton(

subtabHeap,

text="Add",

command=lambda: addHeapElement(int(addHeapElementInput.get()))

if addHeapElementInput.get()

else AlertPopup("Please enter an element to add"),

width=60,

fg\_color="#28A228",

hover\_color="#1F7D1F",

text\_color="white",

font=("Arial", 12, "bold"),

)

addHeapElementButton.place(x=305, y=30)

sortHeapHeading = CTkLabel(subtabHeap, text="Sort Heap", font=("Arial", 14, "bold"))

sortHeapHeading.place(x=0, y=70)

sortHeapButton\_DefaultSort = CTkButton(

subtabHeap,

text="Default Sort",

command=lambda: sortHeap("Default Sort")

if len(heapElements.heap) > 0

else AlertPopup("Heap is empty, so there is nothing to sort"),

width=60,

text\_color="white",

font=("Arial", 12, "bold"),

fg\_color="#8F00FF",

hover\_color="#7500D1",

)

sortHeapButton\_DefaultSort.place(x=0, y=100)

sortHeapButton\_QuickSort = CTkButton(

subtabHeap,

text="Quick Sort",

command=lambda: sortHeap("Quick Sort")

if len(heapElements.heap) > 0

else AlertPopup("Heap is empty, so there is nothing to sort"),

width=60,

text\_color="white",

font=("Arial", 12, "bold"),

fg\_color="#8F00FF",

hover\_color="#7500D1",

)

sortHeapButton\_QuickSort.place(x=90, y=100)

sortHeapButton\_HeapSort = CTkButton(

subtabHeap,

text="Heap Sort",

command=lambda: sortHeap("Heap Sort")

if len(heapElements.heap) > 0

else AlertPopup("Heap is empty, so there is nothing to sort"),

width=60,

text\_color="white",

font=("Arial", 12, "bold"),

fg\_color="#8F00FF",

hover\_color="#7500D1",

)

sortHeapButton\_HeapSort.place(x=170, y=100)

deleteFromHeapHeading = CTkLabel(

subtabHeap, text="Delete from Heap", font=("Arial", 14, "bold")

)

deleteFromHeapHeading.place(x=0, y=140)

deleteFromHeapButton = CTkButton(

subtabHeap,

text="Delete Element",

command=lambda: deleteHeapElement()

if len(heapElements.heap) > 0

else AlertPopup("Heap is already empty"),

width=120,

fg\_color="#BF181D",

hover\_color="#961316",

text\_color="white",

font=("Arial", 12, "bold"),

)

deleteFromHeapButton.place(x=0, y=170)

searchInHeapHeading = CTkLabel(

subtabHeap, text="Search in Heap", font=("Arial", 14, "bold")

)

searchInHeapHeading.place(x=0, y=210)

searchInHeapInput = CTkEntry(subtabHeap, placeholder\_text="Enter element...", width=300)

searchInHeapInput.place(x=0, y=240)

searchInHeapButton = CTkButton(

subtabHeap,

text="Search",

command=lambda: searchHeapElement(int(searchInHeapInput.get()))

if searchInHeapInput.get()

else AlertPopup("Please enter an element to search"),

width=60,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

)

searchInHeapButton.place(x=305, y=240)

heapElements = Heap()

heapSortingResults = []

loadHeapOnStart()

class DoublyLinkedList:

class LinkedListNode:

def \_\_init\_\_(self, data=None):

self.data = data

self.next = None

self.prev = None

def \_\_init\_\_(self):

self.head = None

def append(self, data):

if self.head is None:

self.head = self.LinkedListNode(data)

else:

currentNode = self.head

while currentNode.next is not None:

currentNode = currentNode.next

newNode = self.LinkedListNode(data)

currentNode.next = newNode

newNode.prev = currentNode

def search(self, data) -> bool:

currentNode = self.head

while currentNode is not None:

if currentNode.data == data:

return True

currentNode = currentNode.next

return False

def delete(self, data) -> bool:

currentNode = self.head

while currentNode is not None:

if currentNode.data == data:

if currentNode.prev is not None:

currentNode.prev.next = currentNode.next

if currentNode.next is not None:

currentNode.next.prev = currentNode.prev

if currentNode == self.head:

self.head = currentNode.next

return True

currentNode = currentNode.next

return False

def getList(self):

list = []

currentNode = self.head

while currentNode is not None:

list.append(currentNode.data)

currentNode = currentNode.next

return list

def buildList(self, inputList):

for element in inputList:

self.append(element)

def updateLinkedListElementsContainer():

for widget in linkedListElementsContainer.winfo\_children():

widget.destroy()

for node in linkedListElements.getList():

CTkLabel(linkedListElementsContainer, text=node).pack(padx=5, anchor="w")

def addLinkedListNode(data):

if linkedListElements.search(data):

AlertPopup(f"{data} already exists in linked list")

return

linkedListElements.append(data)

updateLinkedListElementsContainer()

def deleteLinkedListNode(data):

if not linkedListElements.delete(data):

AlertPopup(f"{data} is NOT found in a list")

return

updateLinkedListElementsContainer()

def searchLinkedListNode(data):

if linkedListElements.search(data):

AlertPopup(f"{data} is found in a list")

else:

AlertPopup(f"{data} is NOT found in a list")

def saveLinkedListOnExit():

if len(linkedListElements.getList()) == 0:

return

with open("linkedList.json", "w") as f:

json.dump(linkedListElements.getList(), f)

def loadLinkedListOnStart():

try:

with open("linkedList.json", "r") as f:

list = json.load(f)

linkedListElements.buildList(list)

updateLinkedListElementsContainer()

except:

AlertPopup("Failed to load Linked List data")

addLinkedListNodeHeading = CTkLabel(

subtabLinkedList, text="Add Linked List Node", font=("Arial", 14, "bold")

)

addLinkedListNodeHeading.place(x=0, y=0)

addLinkedListNodeInput = CTkEntry(

subtabLinkedList, placeholder\_text="Node value...", width=300

)

addLinkedListNodeInput.place(x=0, y=30)

addLinkedListNodeButton = CTkButton(

subtabLinkedList,

text="Add",

command=lambda: addLinkedListNode(int(addLinkedListNodeInput.get()))

if addLinkedListNodeInput.get()

else AlertPopup("Please enter the value to add"),

width=45,

fg\_color="#28A228",

hover\_color="#1F7D1F",

text\_color="white",

font=("Arial", 12, "bold"),

)

addLinkedListNodeButton.place(x=305, y=30)

deleteHeapNodeHeading = CTkLabel(

subtabLinkedList, text="Delete Linked List Node", font=("Arial", 14, "bold")

)

deleteHeapNodeHeading.place(x=0, y=70)

deleteLinkedListNodeInput = CTkEntry(

subtabLinkedList, placeholder\_text="Node...", width=300

)

deleteLinkedListNodeInput.place(x=0, y=100)

deleteLinkedListNodeButton = CTkButton(

subtabLinkedList,

text="Delete",

command=lambda: deleteLinkedListNode(int(deleteLinkedListNodeInput.get()))

if deleteLinkedListNodeInput.get()

else AlertPopup("Please enter the value to delete"),

width=45,

fg\_color="#D32F2F",

hover\_color="#B71C1C",

text\_color="white",

font=("Arial", 12, "bold"),

)

deleteLinkedListNodeButton.place(x=305, y=100)

searchLinkedListNodeHeading = CTkLabel(

subtabLinkedList, text="Search Linked List Node", font=("Arial", 14, "bold")

)

searchLinkedListNodeHeading.place(x=0, y=140)

searchLinkedListNodeInput = CTkEntry(

subtabLinkedList, placeholder\_text="Node...", width=300

)

searchLinkedListNodeInput.place(x=0, y=170)

searchLinkedListNodeButton = CTkButton(

subtabLinkedList,

text="Search",

command=lambda: searchLinkedListNode(int(searchLinkedListNodeInput.get()))

if searchLinkedListNodeInput.get()

else AlertPopup("Please enter the value to search for"),

width=45,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

)

searchLinkedListNodeButton.place(x=305, y=170)

linkedListElements = DoublyLinkedList()

loadLinkedListOnStart()

def updateHeapTaskElementsContainer():

for widget in heapTaskElementsContainer.winfo\_children():

widget.destroy()

for employee in heapTaskEmployeesData:

CTkLabel(

heapTaskElementsContainer,

text=f"{employee['name']} - {employee['disease']}",

).pack(padx=5, anchor="w")

def heapTaskLoadEmployeesData(filename: str):

try:

with open(filename, "r", encoding="utf-8") as file:

data = json.load(file)

global heapTaskEmployeesData

heapTaskEmployeesData = data["employees"]

except:

AlertPopup(f"Failed to load data from {filename}")

updateHeapTaskElementsContainer()

def heapTaskShowResults():

def countDiseaseCases():

diseasesCount = dict()

for employee in heapTaskEmployeesData:

if employee["disease"] not in diseasesCount:

diseasesCount[employee["disease"]] = 1

else:

diseasesCount[employee["disease"]] += 1

return diseasesCount

def sortDiseasesList(arr):

if len(arr) == 0:

return arr

sortedArr = Heap().buildHeap(arr).sort()

return sortedArr

diseasesCount = countDiseaseCases()

diseasesCountList = [

(occurences, name) for name, occurences in diseasesCount.items()

]

diseasesCountList = sortDiseasesList(diseasesCountList)

resultsString = f"Employees' Diseases sorted by Occurences:\n"

for occurence, name in diseasesCountList:

currentDisease = f"{name}: {occurence} times\n"

resultsString += currentDisease

AlertPopup(resultsString)

console.log("Heap Sort Task Completed")

heapTaskLoadEmployeesDataHeading = CTkLabel(

subtabHeapTask, text="Load employees JSON file", font=("Arial", 14, "bold")

)

heapTaskLoadEmployeesDataHeading.place(x=0, y=0)

heapTaskLoadEmployeesDataInput = CTkEntry(

subtabHeapTask, placeholder\_text="Filename...", width=140

)

heapTaskLoadEmployeesDataInput.place(x=0, y=30)

heapTaskLoadEmployeesDataButton = CTkButton(

subtabHeapTask,

text="Load",

width=60,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: heapTaskLoadEmployeesData(heapTaskLoadEmployeesDataInput.get())

if heapTaskLoadEmployeesDataInput.get()

else AlertPopup("Please enter the filename to load data from"),

)

heapTaskLoadEmployeesDataButton.place(x=150, y=30)

heapTaskShowResultsButton = CTkButton(

subtabHeapTask,

text="Show Results",

width=120,

fg\_color="#28A228",

hover\_color="#1F7D1F",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: heapTaskShowResults()

if len(heapTaskEmployeesData) > 0

else AlertPopup("Please load employees data first"),

)

heapTaskShowResultsButton.place(x=220, y=30)

heapTaskEmployeesData = []

class HashTable:

def \_\_init\_\_(self, size=15):

self.size = size

self.table = [[] for \_ in range(self.size)]

def hash(self, key):

return int((key \* ((5\*\*0.5 - 1) / 2) % 1) \* self.size)

def insert(self, key, value):

self.table[self.hash(key)].append((key, value))

def delete(self, key):

chain = self.table[self.hash(key)]

for index, keyValuePair in enumerate(chain):

if keyValuePair[0] == key:

del chain[index]

return True

return False

def search(self, key):

chain = self.table[self.hash(key)]

for keyValuePair in chain:

if keyValuePair[0] == key:

return True

return False

def updateHashTableElementsContainer():

for widget in hashTableElementsContainer.winfo\_children():

widget.destroy()

for pair in hashTableElementsList:

key, value = pair.split(" ")

CTkLabel(hashTableElementsContainer, text=f"{key} - {value}").pack(

padx=5, anchor="w"

)

def addHashTableElement(keyValuePair):

hashTableElementsList.append(keyValuePair)

key, value = keyValuePair.split(" ")

key = sum(ord(c) for c in key)

hashTableElements.insert(key, value)

updateHashTableElementsContainer()

def deleteHashTableKey(key):

keysList = [keyValuePair.split(" ")[0] for keyValuePair in hashTableElementsList]

if key not in keysList:

AlertPopup(f"{key} is not in the dictionary")

return

for el in hashTableElementsList:

if el.split(" ")[0] == key:

hashTableElementsList.remove(el)

break

convertedKey = sum(ord(c) for c in key)

\_ = hashTableElements.delete(convertedKey)

updateHashTableElementsContainer()

def searchHashTableKey(key):

convertedKey = sum(ord(c) for c in key)

res = hashTableElements.search(convertedKey)

AlertPopup(f"{key} is in the dictionary") if res else AlertPopup(

f"{key} is NOT in the dictionary"

)

def saveHashTableOnExit():

if len(hashTableElementsList) == 0:

return

with open("hashTable.json", "w") as f:

json.dump(hashTableElementsList, f)

def loadHashTableOnStart():

try:

with open("hashTable.json", "r") as f:

res = json.load(f)

for pair in res:

addHashTableElement(pair)

except:

AlertPopup("Failed to load Hash Table data")

tabDataStructures\_SubTabsContainer = CTkTabview(

tabDataStructures, width=210, height=290

)

tabDataStructures\_SubTabsContainer.add("Hash Table")

tabDataStructures\_SubTabsContainer.add("B-Tree")

tabDataStructures\_SubTabsContainer.add("Employees Search")

tabDataStructures\_SubTabsContainer.add("Subscribers Search")

tabDataStructures\_SubTabsContainer.pack(fill="both", expand=True)

subtabHashTable = tabDataStructures\_SubTabsContainer.tab("Hash Table")

subtabBTree = tabDataStructures\_SubTabsContainer.tab("B-Tree")

subtabBTreeTask = tabDataStructures\_SubTabsContainer.tab("Subscribers Search")

subtabHashTableTask = tabDataStructures\_SubTabsContainer.tab("Employees Search")

hashTableElementsContainer = CTkScrollableFrame(subtabHashTable, width=240, height=270)

hashTableElementsContainer.place(x=400, y=0)

bTreeElementsContainer = CTkScrollableFrame(subtabBTree, width=240, height=270)

bTreeElementsContainer.place(x=400, y=0)

hashTaskElementsContainer = CTkScrollableFrame(

subtabHashTableTask, width=240, height=270

)

hashTaskElementsContainer.place(x=400, y=0)

bTreeTaskElementsContainer = CTkScrollableFrame(subtabBTreeTask, width=240, height=270)

bTreeTaskElementsContainer.place(x=400, y=0)

addHashTableElementHeading = CTkLabel(

subtabHashTable, text="Add Key-Value Pair", font=("Arial", 14, "bold")

)

addHashTableElementHeading.place(x=0, y=0)

addHashTableElementInput = CTkEntry(

subtabHashTable, placeholder\_text="Key & Val...", width=300

)

addHashTableElementInput.place(x=0, y=30)

addHashTableElementButton = CTkButton(

subtabHashTable,

text="Add",

width=45,

fg\_color="#28A228",

hover\_color="#1F7D1F",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: addHashTableElement(addHashTableElementInput.get())

if addHashTableElementInput.get()

else AlertPopup("Please enter key and value to add"),

)

addHashTableElementButton.place(x=305, y=30)

deleteHashTableElementHeading = CTkLabel(

subtabHashTable, text="Delete Hash Table Key", font=("Arial", 14, "bold")

)

deleteHashTableElementHeading.place(x=0, y=70)

deleteHashTableElementInput = CTkEntry(

subtabHashTable, placeholder\_text="Key...", width=300

)

deleteHashTableElementInput.place(x=0, y=100)

deleteHashTableElementButton = CTkButton(

subtabHashTable,

text="Delete",

width=45,

fg\_color="#D32F2F",

hover\_color="#B71C1C",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: deleteHashTableKey(deleteHashTableElementInput.get())

if deleteHashTableElementInput.get()

else AlertPopup("Please enter key to delete"),

)

deleteHashTableElementButton.place(x=305, y=100)

searchHashTableElementHeading = CTkLabel(

subtabHashTable, text="Search Hash Table Key", font=("Arial", 14, "bold")

)

searchHashTableElementHeading.place(x=0, y=140)

searchHashTableElementInput = CTkEntry(

subtabHashTable, placeholder\_text="Key...", width=300

)

searchHashTableElementInput.place(x=0, y=170)

searchHashTableElementButton = CTkButton(

subtabHashTable,

text="Search",

width=45,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: searchHashTableKey(searchHashTableElementInput.get())

if searchHashTableElementInput.get()

else AlertPopup("Please enter key to search"),

)

searchHashTableElementButton.place(x=305, y=170)

hashTableElementsList = []

hashTableElements = HashTable()

loadHashTableOnStart()

class BTreeNode:

def \_\_init\_\_(self, leaf=True):

self.leaf = leaf

self.keys = []

self.children = []

class BTree:

def \_\_init\_\_(self, t=3):

self.root = BTreeNode()

self.t = t

def search(self, key, currentNode=None):

currentNode = currentNode or self.root

index = 0

while index < len(currentNode.keys) and key > currentNode.keys[index]:

index += 1

if index < len(currentNode.keys) and key == currentNode.keys[index]:

return True

elif currentNode.leaf:

return False

else:

return self.search(key, currentNode.children[index])

def insert(self, key):

root = self.root

if len(root.keys) == (2 \* self.t) - 1:

newRoot = BTreeNode(leaf=False)

newRoot.children.append(self.root)

self.\_splitChild(newRoot, 0)

self.root = newRoot

self.\_insertNotFull(newRoot, key)

else:

self.\_insertNotFull(root, key)

def \_insertNotFull(self, currentNode, key):

index = len(currentNode.keys) - 1

if currentNode.leaf:

currentNode.keys.append(0)

while index >= 0 and key < currentNode.keys[index]:

currentNode.keys[index + 1] = currentNode.keys[index]

index -= 1

currentNode.keys[index + 1] = key

else:

while index >= 0 and key < currentNode.keys[index]:

index -= 1

index += 1

if len(currentNode.children[index].keys) == (2 \* self.t) - 1:

self.\_splitChild(currentNode, index)

if key > currentNode.keys[index]:

index += 1

self.\_insertNotFull(currentNode.children[index], key)

def \_splitChild(self, parent, index):

t = self.t

child = parent.children[index]

newNode = BTreeNode(leaf=child.leaf)

parent.children.insert(index + 1, newNode)

parent.keys.insert(index, child.keys[t - 1])

newNode.keys = child.keys[t:]

child.keys = child.keys[: t - 1]

if not child.leaf:

newNode.children = child.children[t:]

child.children = child.children[:t]

def delete(self, key):

root = self.root

self.\_delete(root, key)

if len(root.keys) == 0 and not root.leaf:

self.root = root.children[0]

def \_delete(self, parent, key):

t = self.t

index = 0

while index < len(parent.keys) and key > parent.keys[index]:

index += 1

if parent.leaf:

if index < len(parent.keys) and key == parent.keys[index]:

parent.keys.pop(index)

else:

AlertPopup(f"Key {key} was not found")

else:

if index < len(parent.keys) and key == parent.keys[index]:

self.\_deleteInternalNode(parent, index)

else:

if len(parent.children[index].keys) >= t:

self.\_delete(parent.children[index], key)

else:

if index > 0 and len(parent.children[index - 1].keys) >= t:

self.\_borrowFromPrevious(parent, index)

elif (

index < len(parent.children) - 1

and len(parent.children[index + 1].keys) >= t

):

self.\_borrowFromNext(parent, index)

else:

self.\_merge(parent, index)

self.\_delete(parent.children[index], key)

def \_deleteInternalNode(self, parent, index):

t = self.t

key = parent.keys[index]

if len(parent.children[index].keys) >= t:

predecessor = self.\_getPredecessor(parent.children[index])

parent.keys[index] = predecessor

self.\_delete(parent.children[index], predecessor)

elif len(parent.children[index + 1].keys) >= t:

successor = self.\_getSuccessor(parent.children[index + 1])

parent.keys[index] = successor

self.\_delete(parent.children[index + 1], successor)

else:

self.\_merge(parent, index)

self.\_delete(parent.children[index], key)

def \_getPredecessor(self, parent):

while not parent.leaf:

parent = parent.children[-1]

return parent.keys[-1]

def \_getSuccessor(self, parent):

while not parent.leaf:

parent = parent.children[0]

return parent.keys[0]

def \_borrowFromPrevious(self, parent, index):

child = parent.children[index]

sibling = parent.children[index - 1]

child.keys.insert(0, parent.keys[index - 1])

parent.keys[index - 1] = sibling.keys.pop()

if not child.leaf:

child.children.insert(0, sibling.children.pop())

def \_borrowFromNext(self, parent, index):

child = parent.children[index]

sibling = parent.children[index + 1]

child.keys.append(parent.keys[index])

parent.keys[index] = sibling.keys.pop(0)

if not child.leaf:

child.children.append(sibling.children.pop(0))

def \_merge(self, parent, index):

child = parent.children[index]

sibling = parent.children[index + 1]

child.keys.append(parent.keys[index])

child.keys.extend(sibling.keys)

if not child.leaf:

child.children.extend(sibling.children)

parent.keys.pop(index)

parent.children.pop(index + 1)

def getList(self, node=None):

node = node or self.root

keys = []

for child in node.children:

keys.extend(self.getList(child))

keys.extend(node.keys)

return keys

def addBTreeNode(data):

bTreeElements.insert(data)

bTreeElementsList.append(data)

updateBTreeElementsContainer()

def deleteBTreeNode(data):

if data not in bTreeElementsList:

AlertPopup(f"{data} is NOT in the B-Tree")

return

bTreeElements.delete(data)

bTreeElementsList.remove(data)

updateBTreeElementsContainer()

def searchBTreeNode(data):

isFound = bTreeElements.search(data)

AlertPopup(f"{data} is in the B-Tree") if isFound else AlertPopup(

f"{data} is NOT in the B-Tree"

)

def updateBTreeElementsContainer():

for widget in bTreeElementsContainer.winfo\_children():

widget.destroy()

for node in bTreeElementsList:

CTkLabel(bTreeElementsContainer, text=node).pack(padx=5, anchor="w")

def saveBTreeOnExit():

if len(bTreeElementsList) == 0:

return

with open("bTree.json", "w") as file:

json.dump(bTreeElementsList, file)

def loadBTreeOnStart():

try:

with open("bTree.json", "r") as file:

res = json.load(file)

for element in res:

addBTreeNode(element)

except:

AlertPopup("Failed to load B-Tree elements")

addBTreeNodeHeading = CTkLabel(

subtabBTree, text="Add B-Tree Node", font=("Arial", 14, "bold")

)

addBTreeNodeHeading.place(x=0, y=0)

addBTreeNodeInput = CTkEntry(subtabBTree, placeholder\_text="Node...", width=300)

addBTreeNodeInput.place(x=0, y=30)

addBTreeNodeButton = CTkButton(

subtabBTree,

text="Add",

command=lambda: addBTreeNode(addBTreeNodeInput.get())

if addBTreeNodeInput.get()

else AlertPopup("Please enter a value to add"),

width=45,

fg\_color="#28A228",

hover\_color="#1F7D1F",

text\_color="white",

font=("Arial", 12, "bold"),

)

addBTreeNodeButton.place(x=305, y=30)

deleteBTreeNodeHeading = CTkLabel(

subtabBTree, text="Delete B-Tree Node", font=("Arial", 14, "bold")

)

deleteBTreeNodeHeading.place(x=0, y=70)

deleteBTreeNodeInput = CTkEntry(subtabBTree, placeholder\_text="Node...", width=300)

deleteBTreeNodeInput.place(x=0, y=100)

deleteBTreeNodeButton = CTkButton(

subtabBTree,

text="Delete",

command=lambda: deleteBTreeNode(deleteBTreeNodeInput.get())

if deleteBTreeNodeInput.get()

else AlertPopup("Please enter a value to delete"),

width=45,

fg\_color="#D32F2F",

hover\_color="#B71C1C",

text\_color="white",

font=("Arial", 12, "bold"),

)

deleteBTreeNodeButton.place(x=305, y=100)

searchBTreeNodeHeading = CTkLabel(

subtabBTree, text="Search B-Tree Node", font=("Arial", 14, "bold")

)

searchBTreeNodeHeading.place(x=0, y=140)

searchBTreeNodeInput = CTkEntry(subtabBTree, placeholder\_text="Node...", width=300)

searchBTreeNodeInput.place(x=0, y=170)

searchBTreeNodeButton = CTkButton(

subtabBTree,

text="Search",

command=lambda: searchBTreeNode(searchBTreeNodeInput.get())

if searchBTreeNodeInput.get()

else AlertPopup("Please enter a value to search"),

width=45,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

)

searchBTreeNodeButton.place(x=305, y=170)

bTreeElementsList = []

bTreeElements = BTree()

loadBTreeOnStart()

def updateBTreeTaskElementsContainer():

for widget in bTreeTaskElementsContainer.winfo\_children():

widget.destroy()

for subscriber in subscribersTaskElementsList:

CTkLabel(

bTreeTaskElementsContainer,

text=f"+{subscriber['phone']} - {subscriber['name']}",

).pack(padx=5, anchor="w")

def bTreeTaskSearchForSub(data):

def retrieveSubscribersData(phoneNumber):

for currentSubscriber in subscribersTaskElementsList:

if currentSubscriber["phone"] == phoneNumber:

return currentSubscriber

return None

if not subscribersTaskElementsList or len(subscribersTaskElementsList) == 0:

console.log("No subscribers found in the database")

AlertPopup("No subscribers found in the database")

isFound = subscribersTaskElements.search(data)

subscriberData = retrieveSubscribersData(data)

AlertPopup(

f"+{data} was found in the database\nName - {subscriberData['name']}, Type - {subscriberData['type']}"

) if isFound else AlertPopup(f"+{data} was NOT found in the database")

console.log(f"+{data} {'was' if isFound else 'was not'} found in the database")

def loadBTreeTaskData(fileName):

subscribersDataHolder = []

try:

with open(fileName, "r") as file:

subscribersDataHolder = json.load(file)

except:

AlertPopup(f"Failed to load subscribers' data from {fileName}")

console.log(f"Failed to load subscribers' data from {fileName}")

for element in subscribersDataHolder:

subscribersTaskElements.insert(element["phone"])

for element in subscribersDataHolder:

subscribersTaskElementsList.append(

{

"phone": element["phone"],

"name": element["name"],

"type": element["type"],

}

)

updateBTreeTaskElementsContainer()

console.log(f"Loaded subscribers data from {fileName}")

def subscribersTaskCountTypes():

if not subscribersTaskElementsList or len(subscribersTaskElementsList) == 0:

console.log("No subscribers found in the database")

AlertPopup("No subscribers found in the database")

return

typesCount = {}

for subscriber in subscribersTaskElementsList:

if subscriber["type"] in typesCount:

typesCount[subscriber["type"]] += 1

else:

typesCount[subscriber["type"]] = 1

resultsString = "Count of occurences of each subscription type:\n"

for type in typesCount:

resultsString += f"{type}: {typesCount[type]}\n"

AlertPopup(resultsString)

console.log("Counted subscription types")

loadBTreeTaskDataHeading = CTkLabel(

subtabBTreeTask, text="Load subscribers JSON", font=("Arial", 14, "bold")

)

loadBTreeTaskDataHeading.place(x=0, y=0)

loadBTreeTaskDataInput = CTkEntry(

subtabBTreeTask, placeholder\_text="Filename...", width=300

)

loadBTreeTaskDataInput.place(x=0, y=30)

loadBTreeTaskDataButton = CTkButton(

subtabBTreeTask,

text="Load",

width=45,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: loadBTreeTaskData(loadBTreeTaskDataInput.get())

if loadBTreeTaskDataInput.get()

else AlertPopup("Input box is empty"),

)

loadBTreeTaskDataButton.place(x=305, y=30)

bTreeTaskSearchForSubHeading = CTkLabel(

subtabBTreeTask, text="Search for subscriber", font=("Arial", 14, "bold")

)

bTreeTaskSearchForSubHeading.place(x=0, y=70)

bTreeTaskSearchForSubInput = CTkEntry(

subtabBTreeTask, placeholder\_text="Number...", width=300

)

bTreeTaskSearchForSubInput.place(x=0, y=100)

bTreeTaskSearchForSubButton = CTkButton(

subtabBTreeTask,

text="Search",

width=45,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: bTreeTaskSearchForSub(bTreeTaskSearchForSubInput.get())

if bTreeTaskSearchForSubInput.get()

else AlertPopup("Input box is empty"),

)

bTreeTaskSearchForSubButton.place(x=305, y=100)

subscribersTaskCountTypesHeading = CTkLabel(

subtabBTreeTask,

text="Count subscribers by type",

font=("Arial", 14, "bold"),

)

subscribersTaskCountTypesHeading.place(x=0, y=140)

subscribersTaskCountTypesButton = CTkButton(

subtabBTreeTask,

text="Count",

width=120,

text\_color="white",

font=("Arial", 12, "bold"),

fg\_color="#8F00FF",

hover\_color="#7500D1",

command=lambda: subscribersTaskCountTypes(),

)

subscribersTaskCountTypesButton.place(x=0, y=170)

subscribersTaskElementsList = []

subscribersTaskElements = BTree()

def updateHashTaskElementsContainer():

for widget in hashTaskElementsContainer.winfo\_children():

widget.destroy()

for pair in hashTaskEmployeesList:

name, position = pair.strip().split(",")

CTkLabel(hashTaskElementsContainer, text=f"{name} - {position}").pack(

padx=5, anchor="w"

)

def hashTaskLoadData(fileName):

try:

with open(fileName, "r") as file:

for line in file:

name, position = line.strip().split(",")

key = sum(ord(c) for c in name)

hashTaskEmployees.insert(key, (name, position))

hashTaskEmployeesList.append(line)

updateHashTaskElementsContainer()

console.log(f"Loaded employees data from {fileName}")

except:

AlertPopup(f"Failed to load data from {fileName}")

console.log(f"Failed to load employees data from {fileName}")

def hashTaskSearch(givenName):

convertedKey = sum(ord(c) for c in givenName)

if hashTaskEmployees.search(convertedKey):

position = None

for employeeData in hashTaskEmployeesList:

thisName, thisPosition = employeeData.strip().split(",")

if thisName == givenName:

position = thisPosition

break

AlertPopup(f"{givenName} is found. They work as {position}")

console.log(f"{givenName} was found in the hash table")

else:

AlertPopup(f"{givenName} is NOT found")

console.log(f"{givenName} was NOT found in the hash table")

hashTaskLoadDataHeading = CTkLabel(

subtabHashTableTask, text="Load employees MD", font=("Arial", 14, "bold")

)

hashTaskLoadDataHeading.place(x=0, y=0)

hashTaskLoadDataInput = CTkEntry(

subtabHashTableTask, placeholder\_text="Filename...", width=300

)

hashTaskLoadDataInput.place(x=0, y=30)

hashTaskLoadDataButton = CTkButton(

subtabHashTableTask,

text="Load",

width=45,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: hashTaskLoadData(hashTaskLoadDataInput.get())

if hashTaskLoadDataInput.get()

else AlertPopup("Please enter a filename to load data from"),

)

hashTaskLoadDataButton.place(x=305, y=30)

hashTaskSearchHeading = CTkLabel(

subtabHashTableTask, text="Find position of an employee", font=("Arial", 14, "bold")

)

hashTaskSearchHeading.place(x=0, y=70)

hashTaskSearchInput = CTkEntry(

subtabHashTableTask, placeholder\_text="Employee...", width=300

)

hashTaskSearchInput.place(x=0, y=100)

hashTaskSearchButton = CTkButton(

subtabHashTableTask,

text="Search",

width=45,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: hashTaskSearch(hashTaskSearchInput.get())

if hashTaskSearchInput.get()

else AlertPopup("Please enter an employee name to search for"),

)

hashTaskSearchButton.place(x=305, y=100)

hashTaskEmployeesList = []

hashTaskEmployees = HashTable()

class HuffmanCoding:

def \_\_init\_\_(self):

self.heap = []

self.codes = {}

self.reverseMapping = {}

class HeapNode:

def \_\_init\_\_(self, char, freq):

self.char = char

self.freq = freq

self.left = None

self.right = None

def \_\_lt\_\_(self, other):

return self.freq < other.freq

def buildFrequencyDict(self, text):

frequency = {}

for char in text:

if char not in frequency:

frequency[char] = 0

frequency[char] += 1

return frequency

def buildHeap(self, frequencies):

for key in frequencies:

node = self.HeapNode(key, frequencies[key])

heapq.heappush(self.heap, node)

def mergeNodes(self):

while len(self.heap) > 1:

child = heapq.heappop(self.heap)

mergeWith = heapq.heappop(self.heap)

merged = self.HeapNode(None, child.freq + mergeWith.freq)

merged.left = child

merged.right = mergeWith

heapq.heappush(self.heap, merged)

def \_buildCodesHelper(self, root, currentCode):

if root is None:

return

if root.char is not None:

self.codes[root.char] = currentCode

self.reverseMapping[currentCode] = root.char

return

self.\_buildCodesHelper(root.left, currentCode + "0")

self.\_buildCodesHelper(root.right, currentCode + "1")

def buildCodes(self):

root = heapq.heappop(self.heap)

currentCode = ""

self.\_buildCodesHelper(root, currentCode)

def getEncodedText(self, text):

encodedText = ""

for char in text:

encodedText += self.codes[char]

return encodedText

def padEncodedText(self, encodedText):

extraPadding = 8 - len(encodedText) % 8

for \_ in range(extraPadding):

encodedText += "0"

paddingInfo = f"{extraPadding:08b}"

encodedText = paddingInfo + encodedText

return encodedText

def getByteArray(self, paddedEncodedText):

if len(paddedEncodedText) % 8 != 0:

console.log("Encoded text not padded properly")

exit(0)

b = bytearray()

for i in range(0, len(paddedEncodedText), 8):

byte = paddedEncodedText[i : i + 8]

b.append(int(byte, 2))

return b

def compress(self, filePath):

fileName, fileExtension = os.path.splitext(filePath)

outputPath = fileName + ".bin"

with open(filePath, "r+") as file, open(outputPath, "wb") as output:

text = file.read()

text = text.rstrip()

frequency = self.buildFrequencyDict(text)

self.buildHeap(frequency)

self.mergeNodes()

self.buildCodes()

encodedText = self.getEncodedText(text)

paddedEncodedText = self.padEncodedText(encodedText)

b = self.getByteArray(paddedEncodedText)

output.write(bytes(b))

return outputPath

def removePadding(self, paddedEncodedText):

paddedInfo = paddedEncodedText[:8]

extraPadding = int(paddedInfo, 2)

paddedEncodedText = paddedEncodedText[8:]

encodedText = paddedEncodedText[: -1 \* extraPadding]

return encodedText

def decodeText(self, encodedText):

currentCode = ""

decodedText = ""

for bit in encodedText:

currentCode += bit

if currentCode in self.reverseMapping:

character = self.reverseMapping[currentCode]

decodedText += character

currentCode = ""

return decodedText

def decompress(self, filePath):

fileName, fileExtension = os.path.splitext(filePath)

outputPath = fileName + "\_decompressed.txt"

with open(filePath, "rb") as file, open(outputPath, "w") as output:

bitString = ""

byte = file.read(1)

while byte:

byte = ord(byte)

bits = bin(byte)[2:].rjust(8, "0")

bitString += bits

byte = file.read(1)

encodedText = self.removePadding(bitString)

decodedText = self.decodeText(encodedText)

output.write(decodedText)

return outputPath

greedyAlgosTabsContainer = CTkTabview(tabGreedyAlgorithms)

greedyAlgosTabsContainer.add("Huffman Coding")

greedyAlgosTabsContainer.add("Greedy Algorithm Task")

greedyAlgosTabsContainer.pack(fill="both", expand=True)

huffmanCodingTab = greedyAlgosTabsContainer.tab("Huffman Coding")

greedyTaskTab = greedyAlgosTabsContainer.tab("Greedy Algorithm Task")

def huffmanLoadAndCompress(fileName):

try:

global huffmanInputFilePath

huffmanInputFilePath = fileName

huffmanInputFilePathLabel.configure(text=f"Loaded file: {fileName}")

global huffmanCompressedFilePath

huffmanCompressedFilePath = huffmanObject.compress(fileName)

global huffmanOriginalFileWeight

huffmanOriginalFileWeight = os.path.getsize(fileName)

huffmanOriginalFileWeightLabel.configure(

text=f"Original weight: {huffmanOriginalFileWeight/1024:.2f} KB or {huffmanOriginalFileWeight/1024/1024:.2f} MB"

)

global huffmanCompressedFileWeight

huffmanCompressedFileWeight = os.path.getsize(huffmanCompressedFilePath)

huffmanCompressedFileWeightLabel.configure(

text=f"Compressed weight: {huffmanCompressedFileWeight/1024:.2f} KB or {huffmanCompressedFileWeight/1024/1024:.2f} MB"

)

global huffmanCompressionRatio

huffmanCompressionRatio = huffmanCompressionRatio = (

1 - huffmanCompressedFileWeight / huffmanOriginalFileWeight

) \* 100

huffmanCompressionRatioLabel.configure(

text=f"Compression Ratio: {huffmanCompressionRatio:.2f}%"

)

except:

AlertPopup("Failed to compress")

def huffmanDecompress():

try:

global huffmanDecompressedFilePath

huffmanDecompressedFilePath = huffmanObject.decompress(

huffmanCompressedFilePath

)

huffmanDecompressedFilePathLabel.configure(

text=f"Decompressed file: {huffmanDecompressedFilePath}"

)

huffmanDecompressedFileWeightLabel.configure(

text=f"Decompressed weight: {os.path.getsize(huffmanDecompressedFilePath)/1024:.2f} KB or {os.path.getsize(huffmanDecompressedFilePath)/1024/1024:.2f} MB"

)

except:

AlertPopup("Failed to decompress")

huffmanLoadFileHeading = CTkLabel(

huffmanCodingTab, text="Load Text File", font=("Arial", 14, "bold")

)

huffmanLoadFileHeading.place(x=0, y=0)

huffmanLoadFileInput = CTkEntry(

huffmanCodingTab, placeholder\_text="Filename...", width=120

)

huffmanLoadFileInput.place(x=0, y=30)

huffmanLoadFileButton = CTkButton(

huffmanCodingTab,

text="Load",

width=60,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: huffmanLoadAndCompress(huffmanLoadFileInput.get())

if huffmanLoadFileInput.get()

else AlertPopup("Input box is empty"),

)

huffmanLoadFileButton.place(x=125, y=30)

huffmanDecompressFileButton = CTkButton(

huffmanCodingTab,

text="Decompress",

width=80,

fg\_color="#8F00FF",

hover\_color="#7500D1",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: huffmanDecompress()

if huffmanCompressedFilePath

else AlertPopup("No file loaded"),

)

huffmanDecompressFileButton.place(x=200, y=30)

huffmanInputFilePathLabel = CTkLabel(

huffmanCodingTab,

text="No file loaded",

font=("Arial", 12),

)

huffmanInputFilePathLabel.place(x=0, y=70)

huffmanCompressedFileWeightLabel = CTkLabel(

huffmanCodingTab,

text="File weight: ",

font=("Arial", 12),

)

huffmanCompressedFileWeightLabel.place(x=0, y=90)

huffmanOriginalFileWeightLabel = CTkLabel(

huffmanCodingTab,

text="Original file weight: ",

font=("Arial", 12),

)

huffmanOriginalFileWeightLabel.place(x=0, y=110)

huffmanCompressionRatioLabel = CTkLabel(

huffmanCodingTab,

text="Compression ratio: ",

font=("Arial", 12),

)

huffmanCompressionRatioLabel.place(x=0, y=130)

huffmanDecompressedFilePathLabel = CTkLabel(

huffmanCodingTab,

text="No file decompressed",

font=("Arial", 12),

)

huffmanDecompressedFilePathLabel.place(x=0, y=160)

huffmanDecompressedFileWeightLabel = CTkLabel(

huffmanCodingTab,

text="Decompressed file weight: ",

font=("Arial", 12),

)

huffmanDecompressedFileWeightLabel.place(x=0, y=180)

huffmanObject = HuffmanCoding()

huffmanInputFilePath = None

huffmanCompressedFilePath = None

huffmanDecompressedFilePath = None

huffmanOriginalFileWeight = None

huffmanCompressedFileWeight = None

huffmanCompressionRatio = None

class Shopkeeper:

def \_\_init\_\_(self, products):

self.products = products

self.popularProducts = set()

self.productsFreqency = {product: 0 for product in products}

def solveProblem(self, customerOrder):

queueTime = 0

productTimes = []

for product in customerOrder:

originalTime = queueTime

if product not in self.productsFreqency:

self.productsFreqency[product] = 0

if product in self.popularProducts:

queueTime += 1

elif product in self.products:

queueTime += 2

self.productsFreqency[product] += 1

else:

queueTime += 3

takenTime = queueTime - originalTime

productTimes.append(f"{product[0].upper() + product[1:]} - {takenTime}")

self.updatePopularProducts()

return queueTime, productTimes

def updatePopularProducts(self):

mostPopularProduct = max(self.productsFreqency, key=self.productsFreqency.get)

self.popularProducts.add(mostPopularProduct)

del self.productsFreqency[mostPopularProduct]

greedyTaskUpdatePopularProductsContainer()

def updateProducts(self, newProducts):

self.products = newProducts

greedyTaskUpdateProductsContainer()

def proveOptimality(self, customerOrder):

greedyQueueTime, \_ = self.solveProblem(customerOrder)

optimalShopkeeper = Shopkeeper(self.products)

optimalShopkeeper.popularProducts = self.popularProducts.copy()

optimalShopkeeper.productsFreqency = self.productsFreqency.copy()

for product in customerOrder:

if product not in optimalShopkeeper.productsFreqency:

optimalShopkeeper.productsFreqency[product] = 0

if product not in optimalShopkeeper.popularProducts:

optimalShopkeeper.popularProducts.add(product)

del optimalShopkeeper.productsFreqency[product]

optimalQueueTime, \_ = optimalShopkeeper.solveProblem(customerOrder)

AlertPopup(

"The greedy choice is an optimal solution."

) if greedyQueueTime == optimalQueueTime else AlertPopup(

"The greedy choice is a part of some optimal solution."

)

console.log(

"Proved greedy algorithm optimality"

if greedyQueueTime == optimalQueueTime

else "Proved greedy algorithm partial optimality"

)

def greedyTaskUpdatePopularProductsContainer():

for widget in greedyTaskPopularProductsContainer.winfo\_children():

widget.destroy()

for product in greedyTaskShop.popularProducts:

CTkLabel(

greedyTaskPopularProductsContainer,

text=product,

).pack(padx=5, pady=5, anchor="w")

def greedyTaskUpdateProductsContainer():

for widget in greedyTaskProductsContainer.winfo\_children():

widget.destroy()

for product in greedyTaskShop.products:

CTkLabel(

greedyTaskProductsContainer,

text=product,

).pack(padx=5, pady=5, anchor="w")

def greedyTaskUpdateOrdersContainer():

for widget in greedyTaskOrdersContainer.winfo\_children():

widget.destroy()

for order in greedyTaskOrders:

orderString = ", ".join(order)

CTkLabel(

greedyTaskOrdersContainer,

text=orderString,

).pack(padx=5, pady=5, anchor="w")

def greedyTaskLoadProductsFromFile(fileName):

with open(fileName, "r") as file:

products = json.load(file)

greedyTaskShop.updateProducts(products)

def greedyTaskPlaceOrder(order):

global greedyTaskResults

order = order.split(",")

startTimer = time.time()

queueTime, productTimes = greedyTaskShop.solveProblem(order)

time.sleep(0.1)

timeTaken = time.time() - startTimer

memoryTaken = psutil.Process(os.getpid()).memory\_info().rss / 1024 / 1024

productTimesString = "\nTime taken to fetch each product:"

for product in productTimes:

productTimesString += f"\n{product} minutes"

AlertPopup(f"Queue time: {queueTime}\n{productTimesString}")

console.log(f"Placed order to shopkeeper, took {timeTaken:.2f} seconds")

greedyTaskOrders.append(order)

greedyTaskUpdateOrdersContainer()

resultObjects = {"time": timeTaken, "memory": memoryTaken}

greedyTaskResults.append(resultObjects)

def showResultsTable():

resultsTable = Table(title="Shopkeeper Results")

resultsTable.add\_column("Iteration Number", style="white")

resultsTable.add\_column("Time - [cyan]seconds[/]")

resultsTable.add\_column("Memory - [cyan]bytes[/]")

averageTime = sum(result["time"] for result in greedyTaskResults) / len(

greedyTaskResults

)

averageMemory = sum(result["memory"] for result in greedyTaskResults) / len(

greedyTaskResults

)

resultsTable.add\_row(

"[bold]Average[/]",

f"[bold]{averageTime:.2f}[/]",

f"[bold]{averageMemory:.2f}[/]",

)

for i, result in enumerate(greedyTaskResults):

resultsTable.add\_row(

f"{i+1}",

f"{result['time']:.2f}",

f"{result['memory']:.2f}",

)

console.print(resultsTable)

showResultsTable()

def greedyTaskProveOptimality():

if len(greedyTaskOrders) > 0:

greedyTaskShop.proveOptimality(greedyTaskOrders[0])

else:

orderSize = max(1, len(greedyTaskShop.products) // 2)

randomOrder = random.sample(greedyTaskShop.products, orderSize)

greedyTaskShop.proveOptimality(randomOrder)

greedyTaskTabsContainer = CTkTabview(greedyTaskTab)

greedyTaskTabsContainer.add("Products")

greedyTaskTabsContainer.add("Popular Products")

greedyTaskTabsContainer.add("Orders")

greedyTaskTabsContainer.place(x=380, y=0)

greedyTaskProductsContainer = CTkScrollableFrame(

greedyTaskTabsContainer.tab("Products")

)

greedyTaskProductsContainer.pack(side="left", fill="both", expand=True)

greedyTaskPopularProductsContainer = CTkScrollableFrame(

greedyTaskTabsContainer.tab("Popular Products")

)

greedyTaskPopularProductsContainer.pack(side="left", fill="both", expand=True)

greedyTaskOrdersContainer = CTkScrollableFrame(greedyTaskTabsContainer.tab("Orders"))

greedyTaskOrdersContainer.pack(side="left", fill="both", expand=True)

greedyTaskAddProductsHeading = CTkLabel(

greedyTaskTab,

text="Fill up shelves with products",

font=("Arial", 14, "bold"),

)

greedyTaskAddProductsHeading.place(x=0, y=0)

greedyTaskAddProductsInput = CTkEntry(

greedyTaskTab,

placeholder\_text="Products separated by comma...",

width=300,

)

greedyTaskAddProductsInput.place(x=0, y=30)

greedyTaskAddProductsButton = CTkButton(

greedyTaskTab,

text="Add",

width=60,

command=lambda: greedyTaskShop.updateProducts(

greedyTaskAddProductsInput.get().split(",")

)

if greedyTaskAddProductsInput.get()

else AlertPopup("Enter products first"),

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

)

greedyTaskAddProductsButton.place(x=305, y=30)

greedyTaskLoadProductsHeading = CTkLabel(

greedyTaskTab,

text="or, Load products from JSON file",

font=("Arial", 14, "bold"),

)

greedyTaskLoadProductsHeading.place(x=0, y=70)

greedyTaskLoadProductsInput = CTkEntry(

greedyTaskTab,

placeholder\_text="Path to JSON file...",

width=250,

)

greedyTaskLoadProductsInput.place(x=0, y=100)

greedyTaskLoadProductsButton = CTkButton(

greedyTaskTab,

text="Load from JSON",

width=110,

command=lambda: greedyTaskLoadProductsFromFile(greedyTaskLoadProductsInput.get())

if greedyTaskLoadProductsInput.get()

else AlertPopup("Enter path first"),

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

)

greedyTaskLoadProductsButton.place(x=255, y=100)

greedyTaskPlaceOrderHeading = CTkLabel(

greedyTaskTab,

text="Place order",

font=("Arial", 14, "bold"),

)

greedyTaskPlaceOrderHeading.place(x=0, y=160)

greedyTaskPlaceOrderInput = CTkEntry(

greedyTaskTab,

placeholder\_text="Products separated by comma...",

width=300,

)

greedyTaskPlaceOrderInput.place(x=0, y=190)

greedyTaskPlaceOrderButton = CTkButton(

greedyTaskTab,

text="Place",

width=60,

fg\_color="#28A228",

hover\_color="#1F7D1F",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: greedyTaskPlaceOrder(greedyTaskPlaceOrderInput.get())

if greedyTaskPlaceOrderInput.get()

else AlertPopup("Enter products first"),

)

greedyTaskPlaceOrderButton.place(x=305, y=190)

greedyTaskProveOptimalityButton = CTkButton(

greedyTaskTab,

text="Prove algorithm optimality",

width=150,

text\_color="white",

font=("Arial", 12, "bold"),

fg\_color="#8F00FF",

hover\_color="#7500D1",

command=lambda: greedyTaskProveOptimality()

if greedyTaskShop.products

else AlertPopup("Enter products first"),

)

greedyTaskProveOptimalityButton.place(x=0, y=230)

greedyTaskOrders = []

greedyTaskShop = Shopkeeper([])

greedyTaskResults = []

class RobotGroup:

def \_\_init\_\_(self, robotsCount, speeds, groupsCount):

self.robotsCount = robotsCount

self.speeds = speeds

self.groupsCount = groupsCount

self.dp = [[-1] \* (robotsCount + 1) for \_ in range(groupsCount + 1)]

def countWays(self):

return self.\_countWaysHelper(self.groupsCount, self.robotsCount)

def \_countWaysHelper(self, groups, robots):

if groups == 0 and robots == 0:

return 1

if groups == 0 or robots == 0:

return 0

if self.dp[groups][robots] != -1:

return self.dp[groups][robots]

ways = 0

for i in range(1, robots + 1):

ways += self.\_countWaysHelper(groups - 1, i - 1)

self.dp[groups][robots] = ways

return ways

def solveRobotsTask(robotsCount, speeds, groupsCount):

global robotsTaskResults

robotGroup = RobotGroup(robotsCount, speeds, groupsCount)

startTimer = time.time()

res = robotGroup.countWays()

time.sleep(0.1)

timeTaken = time.time() - startTimer

AlertPopup(f"Number of ways to arrange {groupsCount} groups is {res}")

console.log(f"Completed robots DP task in {timeTaken:.2f} seconds")

resultsObject = {

"time": timeTaken,

"memory": sys.getsizeof(robotGroup.dp),

}

robotsTaskResults.append(resultsObject)

def showResultsTable():

resultsTable = Table(

title="Robots DP Task Results",

)

resultsTable.add\_column("Attempt Number", style="white")

resultsTable.add\_column("Time - [cyan]seconds[/]")

resultsTable.add\_column("Memory - [cyan]bytes[/]")

averageTimes = []

averageMemories = []

for result in robotsTaskResults:

averageTimes.append(result["time"])

averageMemories.append(result["memory"])

averageTime = sum(averageTimes) / len(averageTimes)

averageMemory = sum(averageMemories) / len(averageMemories)

resultsTable.add\_row(

"[bold]Average[/]",

f"[bold]{averageTime:.2f}[/]",

f"[bold]{averageMemory:.2f}[/]",

)

for result in robotsTaskResults:

resultsTable.add\_row(

str(robotsTaskResults.index(result) + 1),

f"{result['time']:.2f}",

f"{result['memory']:.2f}",

)

console.print(resultsTable)

showResultsTable()

dynamicProgrammingTabsContainer = CTkTabview(tabDynamicProgramming)

dynamicProgrammingTabsContainer.add("Robot Groups")

dynamicProgrammingTabsContainer.add("Buildings' Arrangements")

dynamicProgrammingTabsContainer.pack(expand=True, fill="both")

robotsTaskTab = dynamicProgrammingTabsContainer.tab("Robot Groups")

buildingsTaskTab = dynamicProgrammingTabsContainer.tab("Buildings' Arrangements")

robotsTaskGetRobotsCountHeading = CTkLabel(

robotsTaskTab,

text="Enter number of robots",

font=("Arial", 14, "bold"),

)

robotsTaskGetRobotsCountHeading.place(x=0, y=0)

robotsTaskGetRobotsCountInput = CTkEntry(

robotsTaskTab,

placeholder\_text="Number of robots...",

width=300,

)

robotsTaskGetRobotsCountInput.place(x=0, y=30)

robotsTaskGetRobotsSpeedsHeading = CTkLabel(

robotsTaskTab,

text="Enter speeds of the robots",

font=("Arial", 14, "bold"),

)

robotsTaskGetRobotsSpeedsHeading.place(x=0, y=70)

robotsTaskGetRobotsSpeedsInput = CTkEntry(

robotsTaskTab,

placeholder\_text="Speeds of the robots separated by comma...",

width=300,

)

robotsTaskGetRobotsSpeedsInput.place(x=0, y=100)

robotsTaskGetGroupsCountHeading = CTkLabel(

robotsTaskTab,

text="Enter number of groups",

font=("Arial", 14, "bold"),

)

robotsTaskGetGroupsCountHeading.place(x=0, y=140)

robotsTaskGetGroupsCountInput = CTkEntry(

robotsTaskTab,

placeholder\_text="Number of groups...",

width=300,

)

robotsTaskGetGroupsCountInput.place(x=0, y=170)

robotsTaskShowResultsButton = CTkButton(

robotsTaskTab,

text="Show results",

width=110,

text\_color="white",

font=("Arial", 12, "bold"),

fg\_color="#28A228",

hover\_color="#1F7D1F",

command=lambda: solveRobotsTask(

int(robotsTaskGetRobotsCountInput.get()),

list(map(int, robotsTaskGetRobotsSpeedsInput.get().split(","))),

int(robotsTaskGetGroupsCountInput.get()),

)

if robotsTaskGetRobotsCountInput.get()

and robotsTaskGetRobotsSpeedsInput.get()

and robotsTaskGetGroupsCountInput.get()

else AlertPopup("Fill in all the fields first"),

)

robotsTaskShowResultsButton.place(x=0, y=220)

robotsTaskResults = []

class BuildingArrangement:

def \_\_init\_\_(self, matrix):

self.matrix = matrix

self.memo = {}

def solveTask(self, buildingsCount):

return self.\_solveTaskUtil(buildingsCount, "Residential")

def \_solveTaskUtil(self, n, buildingType):

if n == 0:

return 1

if (n, buildingType) in self.memo:

return self.memo[(n, buildingType)]

count = 0

for previousBuildingType in ["Residential", "Industrial", "Office"]:

if self.matrix[previousBuildingType][buildingType]:

count += self.\_solveTaskUtil(n - 1, previousBuildingType)

self.memo[(n, buildingType)] = count

return count

def buildingsTaskSolve(

residentialToResidential,

residentialToIndustrial,

residentialToOffice,

industrialToResidential,

industrialToIndustrial,

industrialToOffice,

officeToResidential,

officeToIndustrial,

officeToOffice,

buildingsCount,

):

global buildingsTaskResults

matrix = {

"Residential": {

"Residential": residentialToResidential,

"Industrial": residentialToIndustrial,

"Office": residentialToOffice,

},

"Industrial": {

"Residential": industrialToResidential,

"Industrial": industrialToIndustrial,

"Office": industrialToOffice,

},

"Office": {

"Residential": officeToResidential,

"Industrial": officeToIndustrial,

"Office": officeToOffice,

},

}

arrangements = BuildingArrangement(matrix)

startTimer = time.time()

try:

res = arrangements.solveTask(int(buildingsCount))

except:

AlertPopup("Woah... That is too many buildings")

console.log(

"Failed to solve buildings arrangement due to maximum recursion depth exceeded"

)

return

time.sleep(0.1)

timeTaken = time.time() - startTimer

AlertPopup(f"Number of arrangements: {res}")

console.log(f"Solved buildings arrangement in {timeTaken:.2f} seconds")

resultObject = {"time": timeTaken, "memory": sys.getsizeof(arrangements.memo)}

buildingsTaskResults.append(resultObject)

def showResultsTable():

resultsTable = Table(title="Buildings Arrangements Results")

resultsTable.add\_column("Iteration Number", style="white")

resultsTable.add\_column("Time - [cyan]seconds[/]")

resultsTable.add\_column("Memory - [cyan]bytes[/]")

averageTimes = []

averageMemories = []

for result in buildingsTaskResults:

averageTimes.append(result["time"])

averageMemories.append(result["memory"])

averageTime = sum(averageTimes) / len(averageTimes)

averageMemory = sum(averageMemories) / len(averageMemories)

resultsTable.add\_row(

"[bold]Average[/]",

f"[bold]{averageTime:.2f}[/]",

f"[bold]{averageMemory:.2f}[/]",

)

for result in buildingsTaskResults:

resultsTable.add\_row(

f"{buildingsTaskResults.index(result) + 1}",

f"{result['time']:.2f}",

f"{result['memory']:.2f}",

)

console.print(resultsTable)

showResultsTable()

buildingsTaskGetMatrixForResidentialHeading = CTkLabel(

buildingsTaskTab,

text="Can be built next to residential buildings",

font=("Arial", 12),

)

buildingsTaskGetMatrixForResidentialHeading.place(x=0, y=0)

buildingsTaskGetInputResidentialToResidential = CTkCheckBox(

buildingsTaskTab, text="Residential", onvalue=1, offvalue=0

)

buildingsTaskGetInputResidentialToResidential.place(x=0, y=30)

buildingsTaskGetInputResidentialToIndustrial = CTkCheckBox(

buildingsTaskTab, text="Industrial", onvalue=1, offvalue=0

)

buildingsTaskGetInputResidentialToIndustrial.place(x=110, y=30)

buildingsTaskGetInputResidentialToOffice = CTkCheckBox(

buildingsTaskTab, text="Office", onvalue=1, offvalue=0

)

buildingsTaskGetInputResidentialToOffice.place(x=210, y=30)

buildingsTaskGetMatrixForIndustrialHeading = CTkLabel(

buildingsTaskTab,

text="Can be built next to industrial buildings",

font=("Arial", 12),

)

buildingsTaskGetMatrixForIndustrialHeading.place(x=0, y=70)

buildingsTaskGetInputIndustrialToResidential = CTkCheckBox(

buildingsTaskTab, text="Residential", onvalue=1, offvalue=0

)

buildingsTaskGetInputIndustrialToResidential.place(x=0, y=100)

buildingsTaskGetInputIndustrialToIndustrial = CTkCheckBox(

buildingsTaskTab, text="Industrial", onvalue=1, offvalue=0

)

buildingsTaskGetInputIndustrialToIndustrial.place(x=110, y=100)

buildingsTaskGetInputIndustrialToOffice = CTkCheckBox(

buildingsTaskTab, text="Office", onvalue=1, offvalue=0

)

buildingsTaskGetInputIndustrialToOffice.place(x=210, y=100)

buildingsTaskGetMatrixForOfficeHeading = CTkLabel(

buildingsTaskTab,

text="Can be built next to office buildings",

font=("Arial", 12),

)

buildingsTaskGetMatrixForOfficeHeading.place(x=0, y=140)

buildingsTaskGetInputOfficeToResidential = CTkCheckBox(

buildingsTaskTab, text="Residential", onvalue=1, offvalue=0

)

buildingsTaskGetInputOfficeToResidential.place(x=0, y=170)

buildingsTaskGetInputOfficeToIndustrial = CTkCheckBox(

buildingsTaskTab, text="Industrial", onvalue=1, offvalue=0

)

buildingsTaskGetInputOfficeToIndustrial.place(x=110, y=170)

buildingsTaskGetInputOfficeToOffice = CTkCheckBox(

buildingsTaskTab, text="Office", onvalue=1, offvalue=0

)

buildingsTaskGetInputOfficeToOffice.place(x=210, y=170)

buildingsTaskGetResultsHeading = CTkLabel(

buildingsTaskTab,

text="Enter the number of buildings you want to build",

font=("Arial", 14, "bold"),

)

buildingsTaskGetResultsHeading.place(x=0, y=210)

buildingsTaskGetResultsInput = CTkEntry(

buildingsTaskTab,

placeholder\_text="Number of Buildings...",

width=180,

)

buildingsTaskGetResultsInput.place(x=0, y=240)

buildingsTaskGetResultsButton = CTkButton(

buildingsTaskTab,

text="Solve",

width=60,

fg\_color="#28A228",

hover\_color="#1F7D1F",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: buildingsTaskSolve(

buildingsTaskGetInputResidentialToResidential.get(),

buildingsTaskGetInputResidentialToIndustrial.get(),

buildingsTaskGetInputResidentialToOffice.get(),

buildingsTaskGetInputIndustrialToResidential.get(),

buildingsTaskGetInputIndustrialToIndustrial.get(),

buildingsTaskGetInputIndustrialToOffice.get(),

buildingsTaskGetInputOfficeToResidential.get(),

buildingsTaskGetInputOfficeToIndustrial.get(),

buildingsTaskGetInputOfficeToOffice.get(),

buildingsTaskGetResultsInput.get(),

)

if buildingsTaskGetResultsInput.get()

else AlertPopup("Enter the number of buildings you want to build"),

)

buildingsTaskGetResultsButton.place(x=185, y=240)

buildingsTaskResults = []

class BFSGraph:

def \_\_init\_\_(self, directed=False):

self.graph = defaultdict(list)

self.directed = directed

def addEdge(self, u, v):

self.graph[u].append(v)

if not self.directed:

self.graph[v].append(u)

def BFS(self, start):

visited = []

queue = deque([start])

while queue:

vertex = queue.popleft()

if vertex not in visited:

visited.append(vertex)

queue.extend(set(self.graph[vertex]) - set(visited))

return visited

def getList(self):

list = []

for key, value in self.graph.items():

chainString = (

f"{key} 🔗 {', '.join(map(str, value))}"

if not self.directed

else f"{key} → {', '.join(map(str, value))}"

)

list.append(chainString)

return list

graphTraversalTabsContainer = CTkTabview(

tabGraphTraversalAlgorithms,

)

graphTraversalTabsContainer.add("Breadth First Search")

graphTraversalTabsContainer.add("Depth First Search")

graphTraversalTabsContainer.add("Sum of Paths")

graphTraversalTabsContainer.add("Min number of Operations")

graphTraversalTabsContainer.add("Hampton Court Maze")

graphTraversalTabsContainer.pack(fill="both", expand=True)

bfsTab = graphTraversalTabsContainer.tab("Breadth First Search")

dfsTab = graphTraversalTabsContainer.tab("Depth First Search")

pathsSumTab = graphTraversalTabsContainer.tab("Sum of Paths")

operationsTab = graphTraversalTabsContainer.tab("Min number of Operations")

mazeTab = graphTraversalTabsContainer.tab("Hampton Court Maze")

def bfsDemoUpdateEdgesContainer():

for widget in bfsEdgesContainer.winfo\_children():

widget.destroy()

for edge in BFSGraphObject.getList():

CTkLabel(

bfsEdgesContainer,

text=edge,

).pack(padx=5, anchor="w")

def bfsDemoLoadGraphEdges(fileName, isDirected):

global BFSGraphObject

BFSGraphObject = BFSGraph(isDirected)

with open(fileName, "r") as file:

for line in file:

u, v = map(int, line.strip().split())

BFSGraphObject.addEdge(u, v)

bfsDemoUpdateEdgesContainer()

def bfsDemoStartBFS(sourceNode):

if not BFSGraphObject:

AlertPopup("Load graph edges first")

return

global bfsResults

startTimer = time.time()

visited = BFSGraphObject.BFS(sourceNode)

time.sleep(0.1)

timeTaken = time.time() - startTimer

memoryTaken = psutil.Process(os.getpid()).memory\_info().rss / 1024 / 1024

nodesList = ", ".join(map(str, visited))

AlertPopup(

f"BFS from node {sourceNode} visited {len(visited)} nodes\nFormed tree: {nodesList}"

)

console.log(f"Executed BFS, took {timeTaken:.2f} seconds")

resultObject = {"time": timeTaken, "memory": memoryTaken}

bfsResults.append(resultObject)

def showResultsTable():

resultsTable = Table(title="BFS Results")

resultsTable.add\_column("Iteration Number", style="white")

resultsTable.add\_column("Time - [cyan]seconds[/]")

resultsTable.add\_column("Memory - [cyan]bytes[/]")

averageTime = sum([result["time"] for result in bfsResults]) / len(bfsResults)

averageMemory = sum([result["memory"] for result in bfsResults]) / len(

bfsResults

)

resultsTable.add\_row(

"[bold]Average[/]",

f"[bold]{averageTime:.2f}[/]",

f"[bold]{averageMemory:.2f}[/]",

)

for i, result in enumerate(bfsResults):

resultsTable.add\_row(

f"{i+1}", f"{result['time']:.2f}", f"{result['memory']:.2f}"

)

console.print(resultsTable)

showResultsTable()

bfsEdgesContainer = CTkScrollableFrame(bfsTab, width=240, height=270)

bfsEdgesContainer.place(x=400, y=0)

bfsDemoLoadGraphEdgesHeading = CTkLabel(

bfsTab,

text="Load Graph Edges from File",

font=("Arial", 14, "bold"),

)

bfsDemoLoadGraphEdgesHeading.place(x=0, y=0)

bfsDemoIsDirected = CTkCheckBox(

bfsTab,

text="Directed",

onvalue=1,

offvalue=0,

)

bfsDemoIsDirected.place(x=0, y=30)

bfsDemoFileNameInput = CTkEntry(

bfsTab,

placeholder\_text="Graph edges file...",

width=180,

)

bfsDemoFileNameInput.place(x=100, y=30)

bfsDemoLoadFileButton = CTkButton(

bfsTab,

text="Load",

width=60,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: bfsDemoLoadGraphEdges(

bfsDemoFileNameInput.get(), bfsDemoIsDirected.get()

),

)

bfsDemoLoadFileButton.place(x=285, y=30)

bfsDemoSetInitialNodeHeading = CTkLabel(

bfsTab,

text="Set Source Node",

font=("Arial", 14, "bold"),

)

bfsDemoSetInitialNodeHeading.place(x=0, y=70)

bfsDemoSetInitialNodeInput = CTkEntry(

bfsTab,

placeholder\_text="Source Node...",

width=180,

)

bfsDemoSetInitialNodeInput.place(x=0, y=100)

bfsDemoSetInitialNodeButton = CTkButton(

bfsTab,

text="Set and Start BFS",

width=100,

fg\_color="#28A228",

hover\_color="#1F7D1F",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: bfsDemoStartBFS(int(bfsDemoSetInitialNodeInput.get()))

if bfsDemoSetInitialNodeInput.get()

and bfsDemoSetInitialNodeInput.get().isdigit()

and int(bfsDemoSetInitialNodeInput.get()) in BFSGraphObject.graph

else AlertPopup("Enter a valid source node"),

)

bfsDemoSetInitialNodeButton.place(x=185, y=100)

BFSGraphObject = None

bfsResults = []

class DFSGraph:

def \_\_init\_\_(self, directed=False):

self.graph = defaultdict(list)

self.directed = directed

def addEdge(self, u, v):

self.graph[u].append(v)

if not self.directed:

self.graph[v].append(u)

def DFS(self, source):

visited = []

stack = [source]

while stack:

vertex = stack.pop()

if vertex not in visited:

visited.append(vertex)

stack.extend(set(self.graph[vertex]) - set(visited))

return visited

def getList(self):

list = []

for key, value in self.graph.items():

chainString = (

f"{key} 🔗 {', '.join(map(str, value))}"

if not self.directed

else f"{key} → {', '.join(map(str, value))}"

)

list.append(chainString)

return list

def dfsDemoUpdateEdgesContainer():

for item in dfsEdgesContainer.winfo\_children():

item.destroy()

for item in DFSGraphObject.getList():

CTkLabel(

dfsEdgesContainer,

text=item,

).pack(padx=5, anchor="w")

def dfsDemoLoadGraphEdges(fileName, isDirected):

global DFSGraphObject

DFSGraphObject = DFSGraph(isDirected)

with open(fileName, "r") as file:

for line in file:

u, v = map(int, line.strip().split())

DFSGraphObject.addEdge(u, v)

dfsDemoUpdateEdgesContainer()

def dfsDemoStartDFS(source):

if not DFSGraphObject:

AlertPopup("Load Graph Edges first")

global dfsResults

startTimer = time.time()

visited = DFSGraphObject.DFS(source)

time.sleep(0.1)

timeTaken = time.time() - startTimer

memoryTaken = psutil.Process(os.getpid()).memory\_info().rss / 1024 / 1024

nodesList = ", ".join(map(str, visited))

AlertPopup(

f"DFS from node {source} visited {len(visited)} nodes\nFormed tree: {nodesList}"

)

console.log(f"Executed DFS, took {timeTaken:.2f} seconds")

resultObject = {"time": timeTaken, "memory": memoryTaken}

dfsResults.append(resultObject)

def showResultsTable():

resultsTable = Table(title="DFS Results")

resultsTable.add\_column("Iteration Number", style="white")

resultsTable.add\_column("Time - [cyan]seconds[/]")

resultsTable.add\_column("Memory - [cyan]bytes[/]")

averageTime = sum([result["time"] for result in dfsResults]) / len(dfsResults)

averageMemory = sum([result["memory"] for result in dfsResults]) / len(

dfsResults

)

resultsTable.add\_row(

"[bold]Average[/]",

f"[bold]{averageTime:.2f}[/]",

f"[bold]{averageMemory:.2f}[/]",

)

for i, result in enumerate(dfsResults):

resultsTable.add\_row(

f"{i+1}",

f"{result['time']:.2f}",

f"{result['memory']:.2f}",

)

console.print(resultsTable)

showResultsTable()

dfsEdgesContainer = CTkScrollableFrame(dfsTab, width=240, height=270)

dfsEdgesContainer.place(x=400, y=0)

dfsDemoLoadGraphEdgesHeading = CTkLabel(

dfsTab,

text="Load Graph Edges from File",

font=("Arial", 14, "bold"),

)

dfsDemoLoadGraphEdgesHeading.place(x=0, y=0)

dfsDemoIsDirected = CTkCheckBox(

dfsTab,

text="Directed",

onvalue=1,

offvalue=0,

)

dfsDemoIsDirected.place(x=0, y=30)

dfsDemoFileNameInput = CTkEntry(

dfsTab,

placeholder\_text="Graph edges file...",

width=180,

)

dfsDemoFileNameInput.place(x=100, y=30)

dfsDemoLoadFileButton = CTkButton(

dfsTab,

text="Load",

width=60,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: dfsDemoLoadGraphEdges(

dfsDemoFileNameInput.get(), dfsDemoIsDirected.get()

),

)

dfsDemoLoadFileButton.place(x=285, y=30)

dfsDemoSetInitialNodeHeading = CTkLabel(

dfsTab,

text="Set Source Node",

font=("Arial", 14, "bold"),

)

dfsDemoSetInitialNodeHeading.place(x=0, y=70)

dfsDemoSetInitialNodeInput = CTkEntry(

dfsTab,

placeholder\_text="Source Node...",

width=180,

)

dfsDemoSetInitialNodeInput.place(x=0, y=100)

dfsDemoSetInitialNodeButton = CTkButton(

dfsTab,

text="Set and Start DFS",

width=100,

fg\_color="#28A228",

hover\_color="#1F7D1F",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: dfsDemoStartDFS(int(dfsDemoSetInitialNodeInput.get()))

if dfsDemoSetInitialNodeInput.get()

and dfsDemoSetInitialNodeInput.get().isdigit()

and int(dfsDemoSetInitialNodeInput.get()) in DFSGraphObject.graph

else AlertPopup("Enter a valid source node"),

)

dfsDemoSetInitialNodeButton.place(x=185, y=100)

DFSGraphObject = None

dfsResults = []

class SumGraph:

def \_\_init\_\_(self, directed=False):

self.graph = defaultdict(list)

self.directed = directed

def addEdge(self, u, v):

self.graph[u].append(v)

if not self.directed:

self.graph[v].append(u)

def findPaths(self, start, targetSum):

visited = set()

path = []

result = []

self.\_findPathsHelper(start, targetSum, visited, path, result)

return result

def \_findPathsHelper(self, vertex, targetSum, visited, path, result):

visited.add(vertex)

path.append(vertex)

if sum(path) == targetSum:

result.append(" -> ".join(map(str, path)))

for neighbor in self.graph[vertex]:

if neighbor not in visited:

self.\_findPathsHelper(neighbor, targetSum, visited, path, result)

path.pop()

visited.remove(vertex)

def getList(self):

list = []

for key, value in self.graph.items():

chainString = (

f"{key} 🔗 {', '.join(map(str, value))}"

if not self.directed

else f"{key} → {', '.join(map(str, value))}"

)

list.append(chainString)

return list

def sumPathsTaskLoadGraphEdges(fileName, isDirected):

global sumPathsTaskGraph

sumPathsTaskGraph = SumGraph(isDirected)

with open(fileName, "r") as file:

for line in file:

u, v = map(int, line.strip().split())

sumPathsTaskGraph.addEdge(u, v)

sumPathsTaskUpdateEdgesContainer()

def sumPathsTaskUpdateEdgesContainer():

for widget in sumPathsTaskEdgesContainer.winfo\_children():

widget.destroy()

for edge in sumPathsTaskGraph.getList():

CTkLabel(

sumPathsTaskEdgesContainer,

text=edge,

).pack(padx=5, anchor="w")

def sumPathsTaskGetResults(source, targetSum):

res = sumPathsTaskGraph.findPaths(source, targetSum)

if res:

pathsList = ""

for path in res:

pathsList += f"{path}\n"

AlertPopup(f"Paths with sum {targetSum}:\n{pathsList}")

else:

AlertPopup(f"No paths with sum {targetSum} found")

console.log(

f"Executed Sum of Paths Task. {f'Found {len(res)} valid paths' if res else 'No paths found'}"

)

sumPathsTaskEdgesContainer = CTkScrollableFrame(pathsSumTab, width=240, height=270)

sumPathsTaskEdgesContainer.place(x=400, y=0)

sumPathsTaskLoadGraphEdgesHeading = CTkLabel(

pathsSumTab,

text="Load Graph Edges from File",

font=("Arial", 14, "bold"),

)

sumPathsTaskLoadGraphEdgesHeading.place(x=0, y=0)

sumPathsTaskIsDirected = CTkCheckBox(

pathsSumTab,

text="Directed",

onvalue=1,

offvalue=0,

)

sumPathsTaskIsDirected.place(x=0, y=30)

sumPathsTaskFileNameInput = CTkEntry(

pathsSumTab,

placeholder\_text="Graph edges file...",

width=180,

)

sumPathsTaskFileNameInput.place(x=100, y=30)

sumPathsTaskLoadFileButton = CTkButton(

pathsSumTab,

text="Load",

width=60,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: sumPathsTaskLoadGraphEdges(

sumPathsTaskFileNameInput.get(), sumPathsTaskIsDirected.get()

),

)

sumPathsTaskLoadFileButton.place(x=285, y=30)

sumPathsTaskSetInitialNodeHeading = CTkLabel(

pathsSumTab,

text="Set Source Node",

font=("Arial", 14, "bold"),

)

sumPathsTaskSetInitialNodeHeading.place(x=0, y=70)

sumPathsTaskSetInitialNodeInput = CTkEntry(

pathsSumTab,

placeholder\_text="Source Node...",

width=300,

)

sumPathsTaskSetInitialNodeInput.place(x=0, y=100)

sumPathsTaskSetWantedSumHeading = CTkLabel(

pathsSumTab,

text="Set Wanted Sum",

font=("Arial", 14, "bold"),

)

sumPathsTaskSetWantedSumHeading.place(x=0, y=140)

sumPathsTaskSetWantedSumInput = CTkEntry(

pathsSumTab,

placeholder\_text="Wanted Sum...",

width=300,

)

sumPathsTaskSetWantedSumInput.place(x=0, y=170)

sumPathsTaskGetResultsButton = CTkButton(

pathsSumTab,

text="Get Results",

width=120,

fg\_color="#28A228",

hover\_color="#1F7D1F",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: sumPathsTaskGetResults(

int(sumPathsTaskSetInitialNodeInput.get()),

int(sumPathsTaskSetWantedSumInput.get()),

)

if sumPathsTaskSetInitialNodeInput.get() and sumPathsTaskSetWantedSumInput.get()

else AlertPopup("Fill in both fields"),

)

sumPathsTaskGetResultsButton.place(x=0, y=210)

SumPathsTaskGraphObject = None

class MinOperationsGraph:

def \_\_init\_\_(self, directed=False):

self.graph = defaultdict(list)

self.directed = directed

def solveTask(self, a, b, operations):

queue = deque([(a, [])])

while queue:

num, ops = queue.popleft()

if num == b:

return ops

for op, operationString in operations:

newNum = op(num)

if newNum not in ops:

queue.append((newNum, ops + [operationString]))

return None

def minOperationsTaskSolve(a, b):

minOperationsTaskGraphObject = MinOperationsGraph()

res = minOperationsTaskGraphObject.solveTask(a, b, minOperationsTaskOperationsList)

AlertPopup(

f"Minimum number of operations to get from {a} to {b} is {len(res)}\nOperations: {', '.join(res)}"

) if res else AlertPopup("There is no way to get from {a} to {b}")

console.log(

f"Minimum number of operations calculated. {f'It took {len(res)} operations' if res else f'Path from {a} to {b} never found'}"

)

minOperationsTaskGetFirstNumberHeading = CTkLabel(

operationsTab,

text="Get First Number",

font=("Arial", 14, "bold"),

)

minOperationsTaskGetFirstNumberHeading.place(x=0, y=0)

minOperationsTaskGetFirstNumberInput = CTkEntry(

operationsTab,

placeholder\_text="First Number...",

width=300,

)

minOperationsTaskGetFirstNumberInput.place(x=0, y=30)

minOperationsTaskGetFirstNumberHeading = CTkLabel(

operationsTab,

text="Get Second Number",

font=("Arial", 14, "bold"),

)

minOperationsTaskGetFirstNumberHeading.place(x=0, y=70)

minOperationsTaskGetSecondNumberInput = CTkEntry(

operationsTab,

placeholder\_text="Second Number...",

width=300,

)

minOperationsTaskGetSecondNumberInput.place(x=0, y=100)

minOperationsTaskSolveButton = CTkButton(

operationsTab,

text="Solve",

width=120,

fg\_color="#28A228",

hover\_color="#1F7D1F",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: minOperationsTaskSolve(

int(minOperationsTaskGetFirstNumberInput.get()),

int(minOperationsTaskGetSecondNumberInput.get()),

)

if minOperationsTaskGetFirstNumberInput.get()

and minOperationsTaskGetSecondNumberInput.get()

else AlertPopup("Fill in both fields"),

)

minOperationsTaskSolveButton.place(x=0, y=140)

minOperationsTaskOperationsList = [

(lambda x: x + 1, "+1"),

(lambda x: x - 1, "-1"),

(lambda x: x \* 2, "\*2"),

(lambda x: x // 2, "/2"),

(lambda x: x \* 3, "\*3"),

(lambda x: x // 3, "/3"),

(lambda x: x\*\*2, "^2"),

(lambda x: x\*\*3, "^3"),

]

class MazeGraph:

def \_\_init\_\_(self, directed=False):

self.graph = defaultdict(list)

self.directed = directed

def addEdge(self, u, v):

self.graph[u].append(v)

if not self.directed:

self.graph[v].append(u)

def drawGraph(self):

G = nx.Graph() if not self.directed else nx.DiGraph()

for u, edges in self.graph.items():

for v in edges:

G.add\_edge(u, v)

nodeColors = [

"red" if node == 1 else "lightgreen" if node == 0 else "lightgray"

for node in G.nodes()

]

nx.draw(G, with\_labels=True, node\_color=nodeColors, edge\_color="gray")

plt.show()

def getList(self):

list = []

for key, value in self.graph.items():

chainString = (

f"{key} 🔗 {', '.join(map(str, value))}"

if not self.directed

else f"{key} → {', '.join(map(str, value))}"

)

list.append(chainString)

return list

def mazeTaskUpdateElementsContainer():

for widget in mazeTaskElementsContainer.winfo\_children():

widget.destroy()

for edge in mazeTaskGraphObject.getList():

CTkLabel(

mazeTaskElementsContainer,

text=edge,

).pack(padx=5, anchor="w")

def mazeTaskLoadGraphEdges(fileName, isDirected):

global mazeTaskGraphObject

mazeTaskGraphObject = MazeGraph(isDirected)

with open(fileName, "r") as file:

for line in file:

u, v = map(int, line.strip().split())

mazeTaskGraphObject.addEdge(u, v)

mazeTaskUpdateElementsContainer()

def mazeTaskBuildGraph():

if not mazeTaskGraphObject:

AlertPopup("Load maze edges first")

return

mazeTaskGraphObject.drawGraph()

mazeTaskElementsContainer = CTkScrollableFrame(mazeTab, width=240, height=270)

mazeTaskElementsContainer.place(x=400, y=0)

mazeTaskLoadGraphEdgesHeading = CTkLabel(

mazeTab,

text="Load Maze Edges from File",

font=("Arial", 14, "bold"),

)

mazeTaskLoadGraphEdgesHeading.place(x=0, y=0)

mazeTaskIsDirected = CTkCheckBox(

mazeTab,

text="Directed",

onvalue=1,

offvalue=0,

)

mazeTaskIsDirected.place(x=0, y=30)

mazeTaskFileNameInput = CTkEntry(

mazeTab,

placeholder\_text="Maze edges file...",

width=180,

)

mazeTaskFileNameInput.place(x=100, y=30)

mazeTaskLoadFileButton = CTkButton(

mazeTab,

text="Load",

width=60,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: mazeTaskLoadGraphEdges(

mazeTaskFileNameInput.get(), mazeTaskIsDirected.get()

),

)

mazeTaskLoadFileButton.place(x=285, y=30)

mazeTaskSolveTask = CTkButton(

mazeTab,

text="Visualize Maze",

width=120,

fg\_color="#28A228",

hover\_color="#1F7D1F",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: mazeTaskBuildGraph()

if mazeTaskGraphObject

else AlertPopup("Load maze edges first"),

)

mazeTaskSolveTask.place(x=0, y=70)

mazeTaskGraphObject = None

class PathsGraph:

def \_\_init\_\_(self, directed=False):

self.edges = defaultdict(list)

self.directed = directed

def addEdge(self, u, v, weight):

self.edges[u].append((v, weight))

if not self.directed:

self.edges[v].append((u, weight))

def drawGraph(self, shortest\_path=None):

G = nx.Graph() if not self.directed else nx.DiGraph()

for u, edges in self.edges.items():

for v, weight in edges:

G.add\_edge(u, v, weight=weight)

pos = nx.spring\_layout(G, k=0.5)

if shortest\_path:

nodeColors = [

"red"

if node == shortest\_path[-1]

else "lightgreen"

if node == shortest\_path[0]

else "lightgray"

for node in G.nodes()

]

nx.draw\_networkx\_nodes(G, pos, node\_color=nodeColors)

plt.text(

0.00,

1.13,

"Red: End",

transform=plt.gca().transAxes,

fontsize=10,

verticalalignment="top",

bbox=dict(boxstyle="round", facecolor="red", alpha=0.5),

)

plt.text(

0.00,

1.06,

"Green: Start",

transform=plt.gca().transAxes,

fontsize=10,

verticalalignment="top",

bbox=dict(boxstyle="round", facecolor="lightgreen", alpha=0.5),

)

else:

nx.draw\_networkx\_nodes(G, pos, node\_color="lightblue")

nx.draw\_networkx\_edges(

G, pos, arrowstyle="->"

) if self.directed else nx.draw\_networkx\_edges(G, pos)

nx.draw\_networkx\_labels(G, pos, font\_weight="bold")

if shortest\_path:

edges = [

(shortest\_path[i], shortest\_path[i + 1])

for i in range(len(shortest\_path) - 1)

]

nx.draw\_networkx\_edges(

G, pos, edgelist=edges, edge\_color="red", arrowstyle="->", width=2

) if self.directed else nx.draw\_networkx\_edges(

G, pos, edgelist=edges, edge\_color="red", width=2

)

labels = nx.get\_edge\_attributes(G, "weight")

nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=labels)

plt.show()

def floydWarshall(self):

dist = defaultdict(lambda: defaultdict(lambda: float("inf")))

nextNode = defaultdict(dict)

for u in self.edges:

dist[u][u] = 0

for v, weight in self.edges[u]:

dist[u][v] = weight

nextNode[u][v] = v

for k in self.edges:

for i in self.edges:

for j in self.edges:

if dist[i][k] + dist[k][j] < dist[i][j]:

dist[i][j] = dist[i][k] + dist[k][j]

nextNode[i][j] = nextNode[i][k]

return dist, nextNode

def shortestPath(self, start, end):

distances, nextNode = self.floydWarshall()

if nextNode[start][end] is None:

console.log("No path from {start} to {end} found")

return None

path = [start]

while start != end:

start = nextNode[start][end]

path.append(start)

return path

def dijkstra(self, start, end):

queue = [(0, start, [])]

seen = set()

while queue:

(cost, node, path) = heapq.heappop(queue)

if node not in seen:

seen.add(node)

path = path + [node]

if node == end:

return path

for nextNode, c in self.edges[node]:

if nextNode not in seen:

newCost = cost + c

heapq.heappush(queue, (newCost, nextNode, path))

return []

def bellmanFord(self, start, end):

distance = {node: float("infinity") for node in self.edges}

distance[start] = 0

predecessor = {node: None for node in self.edges}

for \_ in range(len(self.edges) - 1):

for node in self.edges:

for neighbour, weight in self.edges[node]:

if distance[node] + weight < distance[neighbour]:

distance[neighbour] = distance[node] + weight

predecessor[neighbour] = node

for node in self.edges:

for neighbour, weight in self.edges[node]:

assert (

distance[node] + weight >= distance[neighbour]

), "Graph contains a negative-weight cycle"

path = []

currentNode = end

while currentNode is not None:

path.append(currentNode)

currentNode = predecessor[currentNode]

path.reverse()

return path

def getList(self):

list = []

for key, value in self.edges.items():

for node in value:

list.append(

f"{key} -> {node[0]} : {node[1]}"

) if self.directed else list.append(f"{key} 🔗 {node[0]} : {node[1]}")

return list

graphPathsTabsContainer = CTkTabview(

tabGraphShortestPathsAlgorithms,

)

graphPathsTabsContainer.add("Algorithms")

graphPathsTabsContainer.add("Project Times")

graphPathsTabsContainer.add("Path between Two Points")

graphPathsTabsContainer.add("Path to All Points")

graphPathsTabsContainer.pack(expand=True, fill="both")

graphPathAlogsTab = graphPathsTabsContainer.tab("Algorithms")

graphProjectTimesTab = graphPathsTabsContainer.tab("Project Times")

graphPathFromTwoPointsTab = graphPathsTabsContainer.tab("Path between Two Points")

graphPathFromAllPointsTab = graphPathsTabsContainer.tab("Path to All Points")

def graphAlgosLoadGraph(fileName, isDirected):

global graphAlgosGraphObject

graphAlgosGraphObject = PathsGraph(isDirected)

with open(fileName, "r") as file:

for line in file:

u, v, w = map(int, line.strip().split())

graphAlgosGraphObject.addEdge(u, v, int(w))

graphAlgosUpdateElementsContainer()

def graphAlgosUpdateElementsContainer():

for widget in graphAlgosElementsContainer.winfo\_children():

widget.destroy()

for edge in graphAlgosGraphObject.getList():

CTkLabel(

graphAlgosElementsContainer,

text=edge,

).pack(padx=5, anchor="w")

def graphAlgosShowResultsTable():

resultsTable = Table(title="Graph Shortest Path Algorithms Results")

resultsTable.add\_column("Type", style="white")

resultsTable.add\_column("Time - [cyan]seconds[/]")

resultsTable.add\_column("Memory - [cyan]bytes[/]")

eachTypeCount = {result["type"]: 0 for result in graphAlgosResults}

averageTimes = {result["type"]: 0 for result in graphAlgosResults}

averageMemories = {result["type"]: 0 for result in graphAlgosResults}

for result in graphAlgosResults:

eachTypeCount[result["type"]] += 1

averageTimes[result["type"]] += result["time"]

averageMemories[result["type"]] += result["memory"]

for type in eachTypeCount:

averageTimes[type] /= eachTypeCount[type]

averageMemories[type] /= eachTypeCount[type]

for type in eachTypeCount:

resultsTable.add\_row(

type,

f"{averageTimes[type]:.2f}",

f"{averageMemories[type]:.2f}",

)

console.print(resultsTable)

def graphAlgosPerformDijkstra(start, end):

startTimer = time.time()

shortestPath = graphAlgosGraphObject.dijkstra(start, end)

time.sleep(0.1)

timeTaken = time.time() - startTimer

memoryTaken = psutil.Process(os.getpid()).memory\_info().rss / 1024 / 1024

AlertPopup(

f"No Path from {start} to {end} Found"

) if not shortestPath else graphAlgosGraphObject.drawGraph(shortestPath)

console.log(f"Executed Dijkstra, took {timeTaken:.2f} seconds")

resultObject = {"type": "Dijkstra", "time": timeTaken, "memory": memoryTaken}

graphAlgosResults.append(resultObject)

graphAlgosShowResultsTable()

def graphAlgosPerformBellmanFord(start, end):

startTimer = time.time()

shortestPath = graphAlgosGraphObject.bellmanFord(start, end)

time.sleep(0.1)

timeTaken = time.time() - startTimer

memoryTaken = psutil.Process(os.getpid()).memory\_info().rss / 1024 / 1024

AlertPopup(

f"No Path from {start} to {end} Found"

) if not shortestPath else graphAlgosGraphObject.drawGraph(shortestPath)

console.log(f"Executed BellmanFord, took {timeTaken:.2f} seconds")

resultObject = {"type": "Bellman-Ford", "time": timeTaken, "memory": memoryTaken}

graphAlgosResults.append(resultObject)

graphAlgosShowResultsTable()

def graphAlgosPerformFloydWarshall(start, end):

startTimer = time.time()

shortestPath = graphAlgosGraphObject.shortestPath(start, end)

time.sleep(0.1)

timeTaken = time.time() - startTimer

memoryTaken = psutil.Process(os.getpid()).memory\_info().rss / 1024 / 1024

AlertPopup(

f"No Path from {start} to {end} Found"

) if not shortestPath else graphAlgosGraphObject.drawGraph(shortestPath)

console.log(f"Executed FloydWarshall, took {timeTaken:.2f} seconds")

resultObject = {"type": "Floyd-Warshall", "time": timeTaken, "memory": memoryTaken}

graphAlgosResults.append(resultObject)

graphAlgosShowResultsTable()

graphAlgosElementsContainer = CTkScrollableFrame(

graphPathAlogsTab, width=240, height=240

)

graphAlgosElementsContainer.place(x=400, y=0)

graphAlgosDrawGraphButton = CTkButton(

graphPathAlogsTab,

text="Draw Graph",

width=260,

fg\_color="#28A228",

hover\_color="#1F7D1F",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: graphAlgosGraphObject.drawGraph()

if graphAlgosGraphObject

else AlertPopup("Please Load Graph First"),

)

graphAlgosDrawGraphButton.place(x=400, y=260)

graphAlgosLoadGraphHeading = CTkLabel(

graphPathAlogsTab, text="Load Graph from File", font=("Arial", 14, "bold")

)

graphAlgosLoadGraphHeading.place(x=0, y=0)

graphAlgosIsDirected = CTkCheckBox(

graphPathAlogsTab,

text="Is Directed?",

onvalue=True,

offvalue=False,

)

graphAlgosIsDirected.place(x=0, y=30)

graphAlgosLoadGraphInput = CTkEntry(

graphPathAlogsTab, width=180, placeholder\_text="Graph File Path..."

)

graphAlgosLoadGraphInput.place(x=120, y=30)

graphAlgosLoadGraphButton = CTkButton(

graphPathAlogsTab,

text="Load",

width=60,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: graphAlgosLoadGraph(

graphAlgosLoadGraphInput.get(), graphAlgosIsDirected.get()

)

if graphAlgosLoadGraphInput.get()

else AlertPopup("Please Enter Graph File Path"),

)

graphAlgosLoadGraphButton.place(x=305, y=30)

graphAlgosPerformDijkstraHeading = CTkLabel(

graphPathAlogsTab, text="Perform Dijkstra", font=("Arial", 14, "bold")

)

graphAlgosPerformDijkstraHeading.place(x=0, y=70)

graphAlgosPerformDijkstraStart = CTkEntry(

graphPathAlogsTab, width=145, placeholder\_text="Start Point"

)

graphAlgosPerformDijkstraStart.place(x=0, y=100)

graphAlgosPerformDijkstraEnd = CTkEntry(

graphPathAlogsTab, width=145, placeholder\_text="End Point"

)

graphAlgosPerformDijkstraEnd.place(x=150, y=100)

graphAlgosPerformDijkstraButton = CTkButton(

graphPathAlogsTab,

text="Run",

width=60,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: graphAlgosPerformDijkstra(

int(graphAlgosPerformDijkstraStart.get()),

int(graphAlgosPerformDijkstraEnd.get()),

)

if graphAlgosPerformDijkstraStart.get() and graphAlgosPerformDijkstraEnd.get()

else AlertPopup("Please Enter Start and End Points"),

)

graphAlgosPerformDijkstraButton.place(x=305, y=100)

graphAlgosPerformBellmanFordHeading = CTkLabel(

graphPathAlogsTab, text="Perform Bellman Ford", font=("Arial", 14, "bold")

)

graphAlgosPerformBellmanFordHeading.place(x=0, y=140)

graphAlgosPerformBellmanFordStart = CTkEntry(

graphPathAlogsTab, width=145, placeholder\_text="Start Point"

)

graphAlgosPerformBellmanFordStart.place(x=0, y=170)

graphAlgosPerformBellmanFordEnd = CTkEntry(

graphPathAlogsTab, width=145, placeholder\_text="End Point"

)

graphAlgosPerformBellmanFordEnd.place(x=150, y=170)

graphAlgosPerformBellmanFordButton = CTkButton(

graphPathAlogsTab,

text="Run",

width=60,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: graphAlgosPerformBellmanFord(

int(graphAlgosPerformBellmanFordStart.get()),

int(graphAlgosPerformBellmanFordEnd.get()),

)

if graphAlgosPerformBellmanFordStart.get() and graphAlgosPerformBellmanFordEnd.get()

else AlertPopup("Please Enter Start and End Points"),

)

graphAlgosPerformBellmanFordButton.place(x=305, y=170)

graphAlgosPerformFloydWarshallHeading = CTkLabel(

graphPathAlogsTab, text="Perform Floyd Warshall", font=("Arial", 14, "bold")

)

graphAlgosPerformFloydWarshallHeading.place(x=0, y=210)

graphAlgosPerformFloydWarshallStart = CTkEntry(

graphPathAlogsTab, width=145, placeholder\_text="Start Point"

)

graphAlgosPerformFloydWarshallStart.place(x=0, y=240)

graphAlgosPerformFloydWarshallEnd = CTkEntry(

graphPathAlogsTab, width=145, placeholder\_text="End Point"

)

graphAlgosPerformFloydWarshallEnd.place(x=150, y=240)

graphAlgosPerformFloydWarshallButton = CTkButton(

graphPathAlogsTab,

text="Run",

width=60,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: graphAlgosPerformFloydWarshall(

int(graphAlgosPerformFloydWarshallStart.get()),

int(graphAlgosPerformFloydWarshallEnd.get()),

)

if graphAlgosPerformFloydWarshallStart.get()

and graphAlgosPerformFloydWarshallEnd.get()

else AlertPopup("Please Enter Start and End Points"),

)

graphAlgosPerformFloydWarshallButton.place(x=305, y=240)

graphAlgosGraphObject = None

graphAlgosResults = []

class TasksGraph:

def \_\_init\_\_(self, directed=False):

self.edges = defaultdict(list)

self.directed = directed

def addEdge(self, u, v, weight):

self.edges[u].append((v, weight))

if not self.directed:

self.edges[v].append((u, weight))

def drawGraph(self, tasks=None):

G = nx.Graph()

for u, edges in self.edges.items():

for v, weight in edges:

G.add\_edge(u, v, weight=weight)

pos = nx.spring\_layout(G, k=0.15)

if tasks:

nodeColors = [

"red"

if node == tasks[-1]

else "lightgreen"

if node == tasks[0]

else "lightgray"

for node in G.nodes()

]

nx.draw\_networkx\_nodes(G, pos, node\_color=nodeColors)

plt.text(

0.00,

1.13,

"Red: Last Task",

transform=plt.gca().transAxes,

fontsize=10,

verticalalignment="top",

bbox=dict(boxstyle="round", facecolor="red", alpha=0.5),

)

plt.text(

0.00,

1.06,

"Green: First Task",

transform=plt.gca().transAxes,

fontsize=10,

verticalalignment="top",

bbox=dict(boxstyle="round", facecolor="lightgreen", alpha=0.5),

)

else:

nx.draw\_networkx\_nodes(G, pos, node\_color="lightblue")

nx.draw\_networkx\_edges(G, pos)

nx.draw\_networkx\_labels(G, pos, font\_weight="bold")

labels = nx.get\_edge\_attributes(G, "weight")

nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=labels)

plt.show()

def calculateMinimumTime(self):

totalTime = 0

for edges in self.edges.values():

for edge in edges:

totalTime += edge[1]

return totalTime

def getList(self):

list = []

for task, subtasks in self.edges.items():

for subtask in subtasks:

list.append(f"{task} -> {subtask[0]} : {subtask[1]}")

return list

def minimalTaskTimesUpdateElementsContainer():

for widget in minimalTaskTimesElementsContainer.winfo\_children():

widget.destroy()

for task in minimalTaskTimesGraphObject.getList():

CTkLabel(

minimalTaskTimesElementsContainer,

text=task,

).pack(padx=5, anchor="w")

def minimalTaskTimesLoadGraph(fileName):

global minimalTaskTimesGraphObject

minimalTaskTimesGraphObject = TasksGraph(True)

with open(fileName, "r") as file:

for line in file:

try:

u, v, w = line.strip().split()

minimalTaskTimesGraphObject.addEdge(int(u), int(v), int(w))

except ValueError:

console.log(f"Skipping line {line}")

minimalTaskTimesUpdateElementsContainer()

def minimalTaskTimesGetResults():

res = minimalTaskTimesGraphObject.calculateMinimumTime()

AlertPopup(f"Minimal Time to Complete Tasks: {res}") if res else AlertPopup(

"Failed to Calculate Minimal Time"

)

console.log(

f"Solved Project Minimal Times task. {f'Time taken was {res}' if res else 'Failed to Calculate Minimal Time'}"

)

minimalTaskTimesElementsContainer = CTkScrollableFrame(

graphProjectTimesTab, width=240, height=240

)

minimalTaskTimesElementsContainer.place(x=400, y=0)

minimalTaskTimesDrawGraphButton = CTkButton(

graphProjectTimesTab,

text="Draw Graph",

width=260,

fg\_color="#28A228",

hover\_color="#1F7D1F",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: minimalTaskTimesGraphObject.drawGraph([1, 0])

if minimalTaskTimesGraphObject

else AlertPopup("Please Load Graph First"),

)

minimalTaskTimesDrawGraphButton.place(x=400, y=260)

minimalTaskTimesLoadGraphHeading = CTkLabel(

graphProjectTimesTab, text="Load Graph", font=("Arial", 14, "bold")

)

minimalTaskTimesLoadGraphHeading.place(x=0, y=0)

minimalTaskTimesLoadGraphInput = CTkEntry(

graphProjectTimesTab,

width=300,

placeholder\_text="Graph File Path...",

)

minimalTaskTimesLoadGraphInput.place(x=0, y=30)

minimalTaskTimesLoadGraphButton = CTkButton(

graphProjectTimesTab,

text="Load",

width=60,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: minimalTaskTimesLoadGraph(minimalTaskTimesLoadGraphInput.get())

if minimalTaskTimesLoadGraphInput.get()

else AlertPopup("Please Enter Graph File Path"),

)

minimalTaskTimesLoadGraphButton.place(x=305, y=30)

minimalTaskTimesGetResultsButton = CTkButton(

graphProjectTimesTab,

text="Get Results",

width=120,

fg\_color="#28A228",

hover\_color="#1F7D1F",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: minimalTaskTimesGetResults()

if minimalTaskTimesGraphObject

else AlertPopup("Please Load Graph First"),

)

minimalTaskTimesGetResultsButton.place(x=0, y=70)

minimalTaskTimesGraphObject = None

def shortestPathFromTwoPointsLoadGraph(fileName, isDirected):

global shortestPathFromTwoPointsGraphObject

shortestPathFromTwoPointsGraphObject = PathsGraph(isDirected)

with open(fileName, "r") as file:

for line in file:

try:

u, v, w = line.strip().split()

shortestPathFromTwoPointsGraphObject.addEdge(int(u), int(v), int(w))

except ValueError:

console.log(f"Skipping line {line}")

shortestPathFromTwoPointsUpdateElementsContainer()

def shortestPathFromTwoPointsUpdateElementsContainer():

for widget in shortestPathFromTwoPointsElementsContainer.winfo\_children():

widget.destroy()

for task in shortestPathFromTwoPointsGraphObject.getList():

CTkLabel(

shortestPathFromTwoPointsElementsContainer,

text=task,

).pack(padx=5, anchor="w")

def shortestPathFromTwoPoints(start, end):

shortestPath = shortestPathFromTwoPointsGraphObject.dijkstra(start, end)

AlertPopup(

f"No Path from {start} to {end} Found"

) if not shortestPath else shortestPathFromTwoPointsGraphObject.drawGraph(

shortestPath

)

console.log(

f"Executed shortest path from two points algorithm. {f'No path between {start} and {end} found' if not shortestPath else ' -> '.join(shortestPath)}"

)

shortestPathFromTwoPointsElementsContainer = CTkScrollableFrame(

graphPathFromTwoPointsTab, width=240, height=240

)

shortestPathFromTwoPointsElementsContainer.place(x=400, y=0)

shortestPathFromTwoPointsDrawGraphButton = CTkButton(

graphPathFromTwoPointsTab,

text="Draw Graph",

width=260,

fg\_color="#28A228",

hover\_color="#1F7D1F",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: shortestPathFromTwoPointsGraphObject.drawGraph()

if shortestPathFromTwoPointsGraphObject

else AlertPopup("Please Load Graph First"),

)

shortestPathFromTwoPointsDrawGraphButton.place(x=400, y=260)

shortestPathFromTwoPointsLoadGraphHeading = CTkLabel(

graphPathFromTwoPointsTab, text="Load Graph", font=("Arial", 14, "bold")

)

shortestPathFromTwoPointsLoadGraphHeading.place(x=0, y=0)

shortestPathFromTwoPointsIsDirected = CTkCheckBox(

graphPathFromTwoPointsTab,

text="Is Directed?",

onvalue=True,

offvalue=False,

)

shortestPathFromTwoPointsIsDirected.place(x=0, y=30)

shortestPathFromTwoPointsLoadGraphInput = CTkEntry(

graphPathFromTwoPointsTab,

width=180,

placeholder\_text="Graph File Path...",

)

shortestPathFromTwoPointsLoadGraphInput.place(x=120, y=30)

shortestPathFromTwoPointsLoadGraphButton = CTkButton(

graphPathFromTwoPointsTab,

text="Load",

width=60,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: shortestPathFromTwoPointsLoadGraph(

shortestPathFromTwoPointsLoadGraphInput.get(),

shortestPathFromTwoPointsIsDirected.get(),

)

if shortestPathFromTwoPointsLoadGraphInput.get()

else AlertPopup("Please Enter Graph File Path"),

)

shortestPathFromTwoPointsLoadGraphButton.place(x=305, y=30)

shortestPathFromTwoPointsGetStartHeading = CTkLabel(

graphPathFromTwoPointsTab, text="Enter Starting Node", font=("Arial", 14, "bold")

)

shortestPathFromTwoPointsGetStartHeading.place(x=0, y=70)

shortestPathFromTwoPointsGetStartInput = CTkEntry(

graphPathFromTwoPointsTab,

width=300,

placeholder\_text="Start Point...",

)

shortestPathFromTwoPointsGetStartInput.place(x=0, y=100)

shortestPathFromTwoPointsGetEndHeading = CTkLabel(

graphPathFromTwoPointsTab, text="Enter Ending Node", font=("Arial", 14, "bold")

)

shortestPathFromTwoPointsGetEndHeading.place(x=0, y=140)

shortestPathFromTwoPointsGetEndInput = CTkEntry(

graphPathFromTwoPointsTab,

width=300,

placeholder\_text="End Point...",

)

shortestPathFromTwoPointsGetEndInput.place(x=0, y=170)

shortestPathFromTwoPointsPerformButton = CTkButton(

graphPathFromTwoPointsTab,

text="Get Results",

width=120,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: shortestPathFromTwoPoints(

int(shortestPathFromTwoPointsGetStartInput.get()),

int(shortestPathFromTwoPointsGetEndInput.get()),

)

if shortestPathFromTwoPointsGetStartInput.get()

and shortestPathFromTwoPointsGetEndInput.get()

else AlertPopup("Please Enter Start and End Points"),

)

shortestPathFromTwoPointsPerformButton.place(x=0, y=210)

shortestPathFromTwoPointsGraphObject = None

def shortestPathToAllLoadGraph(fileName, isDirected):

global shortestPathToAllGraphObject

shortestPathToAllGraphObject = PathsGraph(isDirected)

with open(fileName, "r") as file:

for line in file:

try:

u, v, w = line.strip().split()

shortestPathToAllGraphObject.addEdge(int(u), int(v), int(w))

except ValueError:

console.log(f"Skipping line {line}")

shortestPathToAllUpdateElementsContainer()

def shortestPathToAllUpdateElementsContainer():

for widget in shortestPathToAllElementsContainer.winfo\_children():

widget.destroy()

for element in shortestPathToAllGraphObject.getList():

CTkLabel(shortestPathToAllElementsContainer, text=element).pack(

padx=5, anchor="w"

)

def shortestPathToAll(start):

if len(shortestPathToAllGraphObject.edges) == 0:

AlertPopup("Please Load Graph First")

console.log("Failed to show all paths, no graph loaded")

return

for currentNode in shortestPathToAllGraphObject.edges:

if currentNode != start:

currentPath = shortestPathToAllGraphObject.shortestPath(start, currentNode)

shortestPathToAllGraphObject.drawGraph(

currentPath

) if currentPath else shortestPathToAllGraphObject.drawGraph(

[start, currentNode]

)

shortestPathToAllElementsContainer = CTkScrollableFrame(

graphPathFromAllPointsTab,

width=240,

height=240,

)

shortestPathToAllElementsContainer.place(x=400, y=0)

shortestPathToAllDrawGraphButton = CTkButton(

graphPathFromAllPointsTab,

text="Draw Graph",

width=260,

fg\_color="#28A228",

hover\_color="#1F7D1F",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: shortestPathToAllGraphObject.drawGraph()

if shortestPathToAllGraphObject

else AlertPopup("Please Load Graph First"),

)

shortestPathToAllDrawGraphButton.place(x=400, y=260)

shortestPathToAllLoadGraphHeading = CTkLabel(

graphPathFromAllPointsTab, text="Load Graph", font=("Arial", 14, "bold")

)

shortestPathToAllLoadGraphHeading.place(x=0, y=0)

shortestPathToAllIsDirected = CTkCheckBox(

graphPathFromAllPointsTab,

text="Is Directed?",

onvalue=True,

offvalue=False,

)

shortestPathToAllIsDirected.place(x=0, y=30)

shortestPathToAllLoadGraphInput = CTkEntry(

graphPathFromAllPointsTab,

width=180,

placeholder\_text="Graph File Path...",

)

shortestPathToAllLoadGraphInput.place(x=120, y=30)

shortestPathToAllLoadGraphButton = CTkButton(

graphPathFromAllPointsTab,

text="Load",

width=60,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: shortestPathToAllLoadGraph(

shortestPathToAllLoadGraphInput.get(),

shortestPathToAllIsDirected.get(),

)

if shortestPathToAllLoadGraphInput.get()

else AlertPopup("Please enter the graph file path"),

)

shortestPathToAllLoadGraphButton.place(x=305, y=30)

shortestPathToAllGetStartHeading = CTkLabel(

graphPathFromAllPointsTab, text="Enter Starting Node", font=("Arial", 14, "bold")

)

shortestPathToAllGetStartHeading.place(x=0, y=70)

shortestPathToAllGetStartInput = CTkEntry(

graphPathFromAllPointsTab,

width=300,

placeholder\_text="Start Point...",

)

shortestPathToAllGetStartInput.place(x=0, y=100)

shortestPathToAllPerformButton = CTkButton(

graphPathFromAllPointsTab,

text="Get Results",

width=120,

fg\_color="#1976D2",

hover\_color="#0D47A1",

text\_color="white",

font=("Arial", 12, "bold"),

command=lambda: shortestPathToAll(

int(shortestPathToAllGetStartInput.get()),

)

if shortestPathToAllGetStartInput.get()

else AlertPopup("Please enter the starting node"),

)

shortestPathToAllPerformButton.place(x=0, y=140)

shortestPathToAllGraphObject = None

app.mainloop()

saveHeapOnExit()

saveLinkedListOnExit()

saveHashTableOnExit()

saveBTreeOnExit()