Tugas Kecil IF2211 Strategi Algoritma

Laporan Implementasi Algoritma UCS dan A* untuk Menentukan Lintasan Terpendek

Oleh:

Naufal Syifa Firdaus / 13521050

Ghazi Akmal Fauzan / 13521058

K02

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

A. Deksripsi Persoalan

Persoalan utama yang harus diselesaikan dalam tugas kecil ini adalah penentuan lintasan terpendek dari suatu titik ke titik lain menggunakan algoritma UCS (Uniform cost search) dan A* (A star). Contoh pengaplikasiannya adalah menentukan rute jalan terpendek di Kota Bandung yang melewati lokasi tertentu dari lokasi awal ke lokasi akhir. Ketentuan program seperti dalam dokumen spesifikasi adalah sebagai berikut.

1. Program menerima input file graf (direpresentasikan sebagai matriks ketetanggaan berbobot), jumlah simpul minimal 8 buah.
2. Program dapat menampilkan peta/graf.
3. Program menerima input simpul asal dan simpul tujuan.
4. Program dapat menampilkan lintasan terpendek beserta jaraknya antara simpul asal dan simpul tujuan.
5. Antarmuka program bebas, apakah pakai GUI atau command line saja.

B. Algoritma Uniform Cost Search (UCS)

UCS adalah algoritma pencarian graf yang menemukan jalur terpendek dari node awal ke node tujuan dengan mempertimbangkan biaya setiap jalur. Algoritma ini menjelajahi graf dengan memperluas simpul dengan biaya kumulatif terendah (yaitu jalur dengan biaya total terkecil) di setiap langkah. Algoritma ini mempertahankan antrian prioritas (sering diimplementasikan menggunakan struktur data heap) untuk melacak node yang akan diperluas, dengan node biaya terendah yang diperluas terlebih dahulu. UCS memperluas node dan memperbarui biaya hingga node tujuan tercapai atau semua node yang dapat dijangkau telah dieksplorasi. Berikut adalah alur dari algoritma UCS yang kami implementasikan:

1. Menerima input berupa nodes (array of string) yang berisi nama-nama node, matrix (2D array of integer) yang berisi matriks representasi graf dengan bobot edge antar node, start (string) yang merupakan node awal, dan stop (string) yang merupakan node tujuan.
2. Membuat dictionary costs untuk menyimpan biaya untuk setiap node, diinisialisasi dengan nilai tak terhingga (inf) untuk semua node kecuali node awal yang diinisialisasi dengan nilai 0.
3. Membuat dictionary paths untuk menyimpan jalur ke setiap node, diinisialisasi dengan array kosong untuk semua node kecuali node awal yang diinisialisasi dengan array yang hanya berisi node awal.
4. Menginisialisasi variabel queue sebagai sebuah priority queue (antrian prioritas) yang akan menyimpan tuple berisi biaya dan node.
5. Menginisialisasi variabel nCalc sebagai 0 untuk menghitung jumlah node yang telah dihitung (visited).
6. Melakukan loop while selama queue tidak kosong, yang berarti masih ada node yang perlu diperiksa.
7. Mem-pop node dengan biaya terendah dari queue dan menyimpan biaya dan nama node tersebut ke dalam variabel cost dan current.

8. Jika current adalah node tujuan (stop), maka break loop karena sudah ditemukan jalur ke tujuan.
9. Mengecek apakah biaya yang ditemukan (cost) untuk current lebih besar dari biaya yang sudah ada (costs[current]). Jika ya, maka skip node ini karena sudah pernah dikunjungi dengan biaya yang lebih rendah sebelumnya.
10. Melakukan loop through semua node dalam nodes.
11. Jika ada koneksi antara current dan node lain (matrix[nodes.index(current)][i] != 0), maka menghitung biaya baru ke tetangga melalui current dengan menambahkan biaya dari matrix ke new_cost.
12. Jika new_cost lebih kecil dari biaya yang sudah ada (costs[nodes[i]]), maka update biaya dan jalur ke node tetangga tersebut (paths[nodes[i]]) dengan biaya dan jalur baru. Selanjutnya, mem-push node tetangga tersebut ke dalam queue dengan biaya baru (new_cost) untuk diperiksa lebih lanjut.
13. Setelah loop selesai, ekstrak jalur akhir (path) dan total biaya (totalCost) dari paths dan matrix menggunakan index dari nodes dan menjumlahkan biaya edge antar node di jalur tersebut.
14. Mengembalikan jalur akhir (path), total biaya (totalCost), dan jumlah node yang dihitung (nCalc) sebagai output dari fungsi UCS.

C. Algoritma A*

A* adalah algoritma pencarian heuristik yang merupakan perpanjangan dari UCS. Seperti UCS, algoritma ini menggunakan antrian prioritas untuk menjelajahi node, tetapi juga menggabungkan fungsi heuristik tambahan yang memperkirakan biaya dari node saat ini ke node tujuan. A* menggabungkan biaya jalur dari node awal ke node saat ini (g-cost) dengan perkiraan biaya dari node saat ini ke node tujuan (h-cost) untuk menentukan prioritas node dalam antrian prioritas. A* memilih node dengan biaya-f terendah (biaya-g + biaya-h) untuk ekspansi. Fungsi heuristik ini membantu A* untuk secara efisien menjelajahi jalur yang paling menjanjikan menuju simpul tujuan, menjadikannya lebih terinformasi dan seringkali lebih cepat daripada UCS dalam menemukan jalur terpendek. Berikut adalah alur dari algoritma A* yang kami implementasikan:

1. Program menerima input berupa matriks representasi graf yang berisi biaya (cost) untuk berpindah dari satu node ke node lain, titik awal (start), titik tujuan (stop), dan fungsi heuristic untuk mengestimasi biaya dari node saat ini ke node tujuan.
2. Program menginisialisasi variabel seperti antrian (queue) untuk menyimpan node yang akan dievaluasi, node awal dengan biaya g (cost) 0, nilai heuristic h dari node awal ke node tujuan, jumlah perhitungan (nCalc) awal 0, serta menyimpan jalur (path) awal yang hanya berisi node awal.
3. Program memasukkan node awal ke dalam antrian (queue) untuk dievaluasi.
4. Program melakukan loop while selama antrian (queue) tidak kosong.
5. Dalam setiap iterasi loop, program mengeluarkan (pop) tuple dengan nilai f (total estimasi biaya dari node awal ke node tujuan melalui node saat ini) terendah dari antrian (queue) sebagai node saat ini yang akan dievaluasi.

6. Jika node saat ini adalah node tujuan (stop), program menghentikan loop dan keluar dari loop while.
7. Jika node saat ini bukan node tujuan, program menandai node saat ini sebagai sudah dikunjungi (visited) dan memeriksa setiap node tetangga yang terhubung dengan node saat ini.
8. Untuk setiap node tetangga yang terhubung dengan node saat ini, program menghitung biaya baru (g) untuk mencapai node tetangga tersebut melalui node saat ini dengan menjumlahkan biaya (cost) dari node awal ke node saat ini dengan biaya (cost) dari node saat ini ke node tetangga tersebut.
9. Program juga menghitung estimasi biaya total (f) dari node awal ke node tujuan melalui node saat ini dengan menambahkan biaya baru (g) dengan nilai heuristic (h) dari node tetangga tersebut.
10. Jika nilai heuristic (h) untuk node tetangga tersebut belum pernah dihitung sebelumnya atau lebih besar dari nilai heuristic (h) yang baru dihitung, program memperbarui nilai heuristic (h) untuk node tetangga tersebut dengan nilai yang baru dihitung.
11. Setelah itu, program memasukkan node tetangga tersebut beserta informasi nilai f, nilai g, node saat ini, dan jalur yang sudah dilewati ke dalam antrian (queue) untuk dievaluasi pada iterasi selanjutnya.
12. Program terus melakukan langkah 5-11 selama antrian (queue) masih berisi node yang harus dievaluasi.
13. Setelah antrian (queue) kosong, program menghitung total biaya (totalCost) dari jalur yang ditemukan dengan menjumlahkan biaya (cost) dari node awal ke setiap node pada jalur yang ditemukan berdasarkan matriks yang diberikan.
14. Program juga mengembalikan jumlah perhitungan (nCalc) yang dilakukan oleh algoritma A* selama proses pencarian jalur.
15. Akhirnya, program mengembalikan jalur yang ditemukan (path), total biaya (totalCost), dan jumlah perhitungan (nCalc) sebagai output dari fungsi ini.

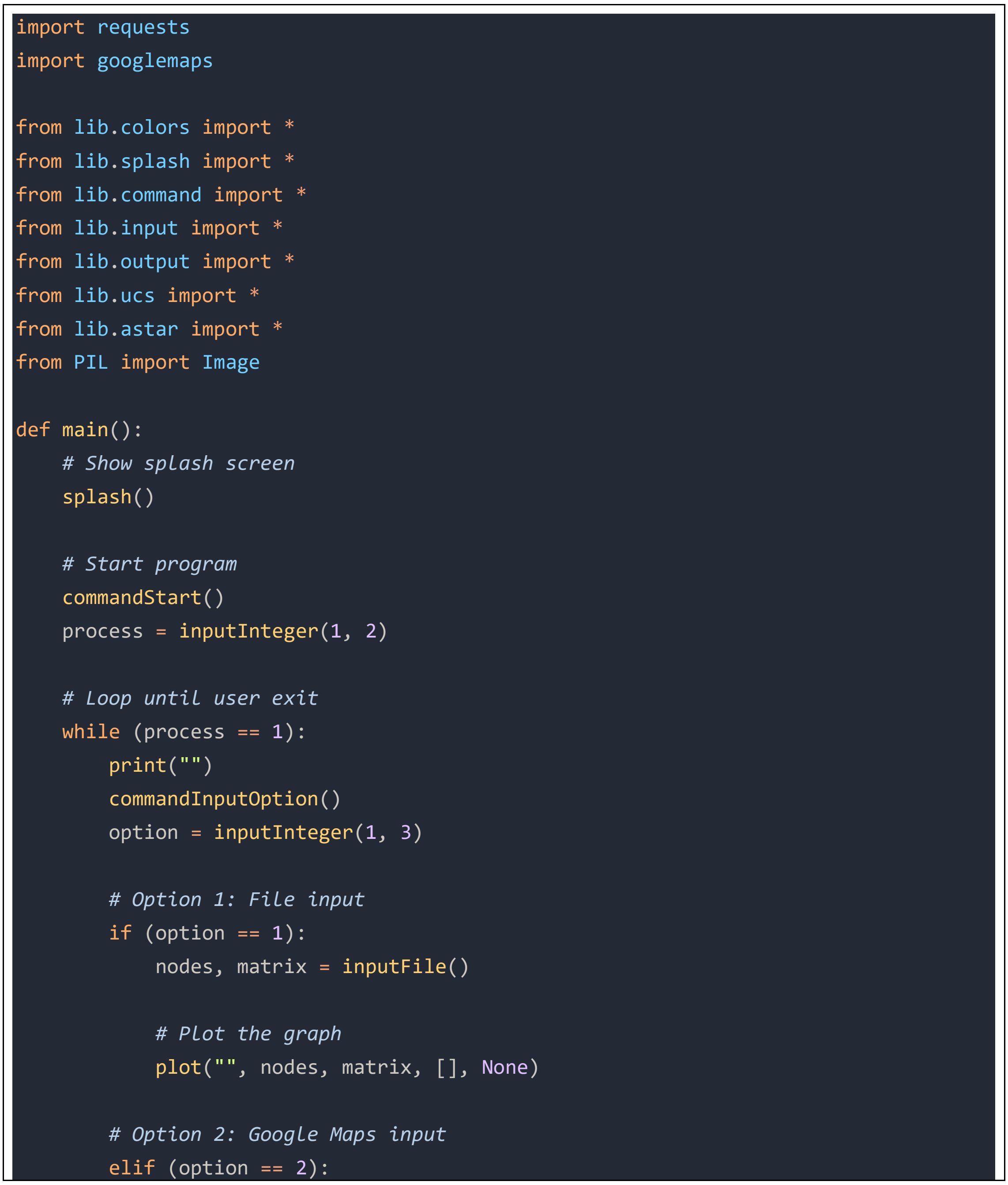
D. Google Maps API

Dalam pengerjaan tugas kecil berikut, disertakan bonus yang menyertakan penggunaan API Peta Google. Untuk menggunakan API tersebut, diperlukan aktivasi *key* yang berperan sebagai kunci akses agar permintaan data dan penggunaan API diizinkan oleh Google. Pada Google Cloud, kami membuat sebuah project baru yang memiliki layanan API. Project yang dibuat bersifat trial, sehingga fungsi Google Maps pada program tugas kecil tidak bisa digunakan lebih dari **90 hari** setelah tugas rilis. Setelah *key* didapatkan, maka layanan API Google telah dapat diakses dari platfrom yang mendukung.

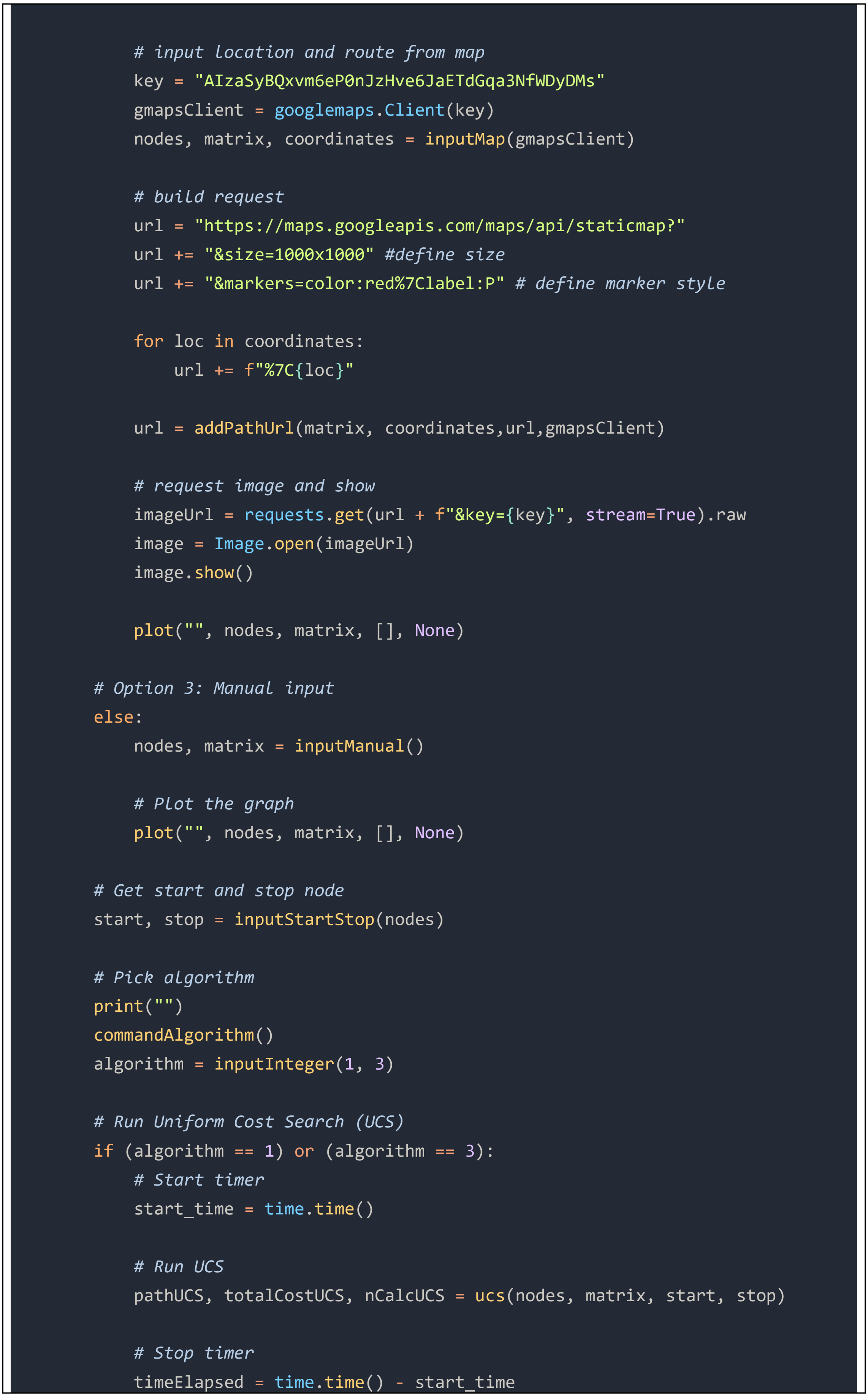
Sebagai alat bantu penggunaan API dibahasa pemrograman python, digunakan Python Client for Google Maps Services. Modul tersebut memungkinkan penggunanya untuk mengakses fitur yang disediakan API dengan sintaks yang lebih sederhana. Program mengimplementasikan modul untuk menghasilkan alamat dan koordinat lokasi serta menampilkan sebuah peta yang berisikan lokasi-lokasi yang dimasukan beserta rute nya. Program akan meminta pengguna untuk memasukan beberapa lokasi (simpul) lalu secara otomatis akan menghasilkan alamat lokasi yang paling cocok dengan masukan pengguna. Masukan dapat berupa nama tempat, alamat, ataupun *landmark*. Setelahnya pengguna diminta untuk mendefinisikan rute (sisi) antar lokasi tersebut. Hasilnya, program akan menampilkan peta lengkap dengan penanda dan rute yang mengikuti marka jalan. Diakhir program, akan ditampilkan peta dengan rute dengan jarak paling dekat antara lokasi awal dan akhir. Tampilan peta menggunakan *request* peta statik dan juga tipe bentuk *encoding* Polyline untuk rute jalan.

E. Source Program

1. main.py



```python
import requests
import googlemaps

from lib.colors import *
from lib.splash import *
from lib.command import *
from lib.input import *
from lib.output import *
from lib.ucs import *
from lib.astar import *
from PIL import Image

def main():
    # Show splash screen
    splash()

    # Start program
    commandStart()
    process = inputInteger(1, 2)

    # Loop until user exit
    while (process == 1):
        print("")
        commandInputOption()
        option = inputInteger(1, 3)

        # Option 1: File input
        if (option == 1):
            nodes, matrix = inputFile()

            # Plot the graph
            plot("", nodes, matrix, [], None)

        # Option 2: Google Maps input
        elif (option == 2):
```

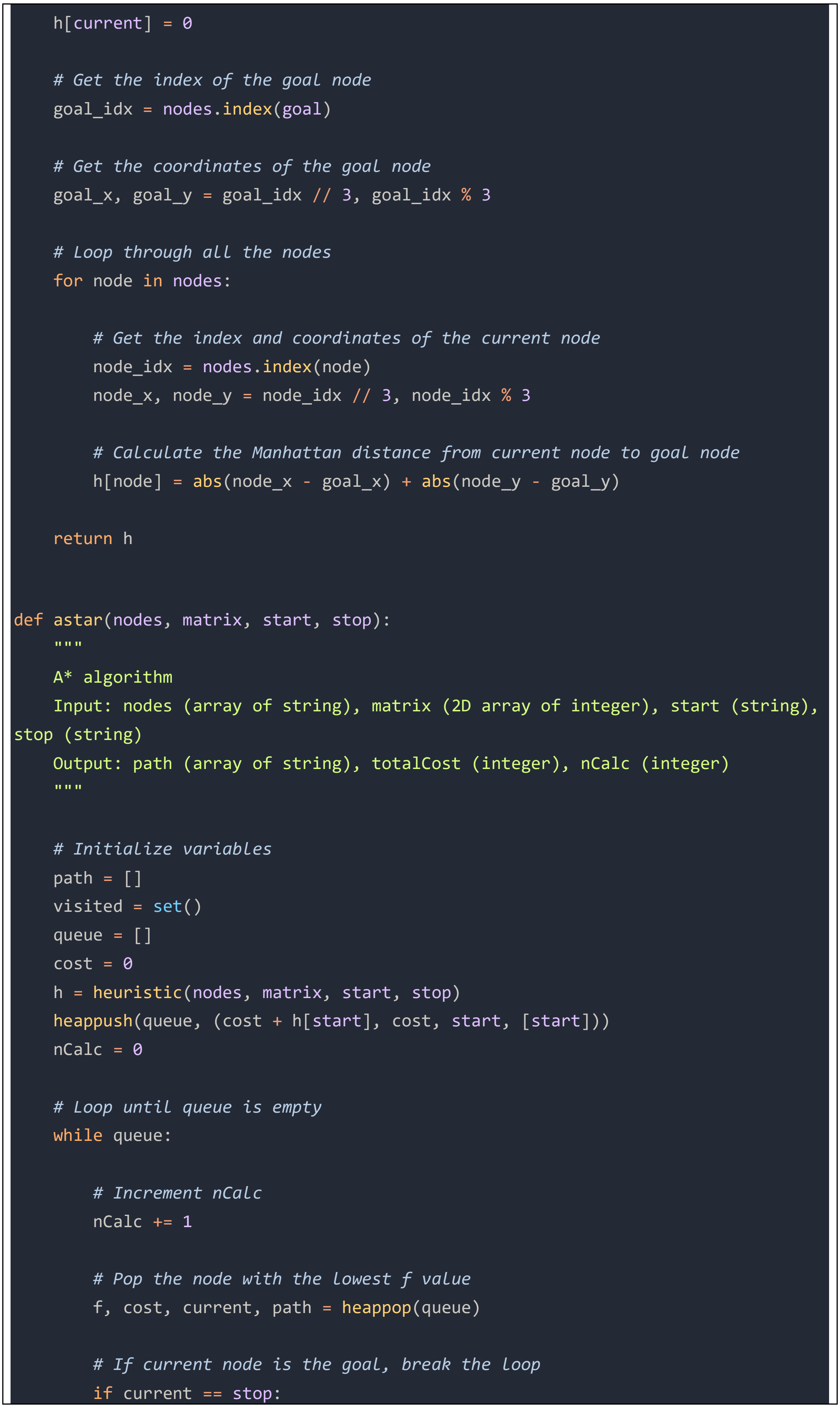


```python
        # input location and route from map
        key = "AIzaSyBQxvm6eP0nJzHve6JaETdGqa3NfWDyDMs"
        gmapsClient = googlemaps.Client(key)
        nodes, matrix, coordinates = inputMap(gmapsClient)

        # build request
        url = "https://maps.googleapis.com/maps/api/staticmap?"
        url += "&size=1000x1000" #define size
        url += "&markers=color:red%7Clabel:P" # define marker style

        for loc in coordinates:
            url += f"%7C{loc}"

        url = addPathUrl(matrix, coordinates,url,gmapsClient)

        # request image and show
        imageUrl = requests.get(url + f"&key={key}", stream=True).raw
        image = Image.open(imageUrl)
        image.show()

        plot("", nodes, matrix, [], None)

    # Option 3: Manual input
    else:
        nodes, matrix = inputManual()

        # Plot the graph
        plot("", nodes, matrix, [], None)

    # Get start and stop node
    start, stop = inputStartStop(nodes)

    # Pick algorithm
    print("")
    commandAlgorithm()
    algorithm = inputInteger(1, 3)

    # Run Uniform Cost Search (UCS)
    if (algorithm == 1) or (algorithm == 3):
        # Start timer
        start_time = time.time()

        # Run UCS
        pathUCS, totalCostUCS, nCalcUCS = ucs(nodes, matrix, start, stop)

        # Stop timer
        timeElapsed = time.time() - start_time
```

```
            # Print result
            printResult("UNIFORM COST SEARCH", start, stop, nCalcUCS, pathUCS, totalCostUCS, timeElapsed)

            # Download Image maps
            if(option == 2):
                path = convertPathToInt(pathUCS, nodes)
                url = addShortestPathUrl(coordinates,url,gmapsClient,path)
                imageUrl = requests.get(url + f"&key={key}", stream=True).raw
                imageUCS = Image.open(imageUrl)
                imageUCS.show()

            # Plot the graph
            plot("UNIFORM COST SEARCH", nodes, matrix, pathUCS, None)

        # Run A*
        if (algorithm == 2) or (algorithm == 3):
            # Start timer
            start_time = time.time()

            # Run A*
            pathAStar, totalCostAStar, nCalcAStar = astar(nodes, matrix, start, stop)

            # Stop timer
            timeElapsed = time.time() - start_time

            # Print result
            printResult("A*", start, stop, nCalcAStar, pathAStar, totalCostAStar, timeElapsed)

            # image map
            if(option == 2):
                path = convertPathToInt(pathAStar, nodes)
                url = addShortestPathUrl(coordinates,url,gmapsClient,path)
                imageUrl = requests.get(url + f"&key={key}", stream=True).raw
                imageAStar = Image.open(imageUrl)
                imageAStar.show()

            # Plot the graph
            plot("A*", nodes, matrix, pathAStar, None)

        # Save option
        print("")
        commandSave()
        save = inputInteger(1, 2)
        # Save to file
        if (save == 1):
            saveConfig = input(str(WHITE + "\nInput Filename: " + RESET))

            # Save Uniform Cost Search (UCS) result
            if (algorithm == 1) or (algorithm == 3):
                saveResult("UNIFORM COST SEARCH", start, stop, nCalcUCS, pathUCS,
totalCostUCS, timeElapsed, nodes, matrix, saveConfig, imageUCS, option)

            # Save A* result
            if (algorithm == 2) or (algorithm == 3):
                saveResult("A*", start, stop, nCalcAStar, pathAStar,
totalCostAStar, timeElapsed, nodes, matrix, saveConfig, imageAStar, option)

            # Display message file saved
            print(LIGHT_GREEN + "\nAdditional Information Added into txt File" +
RESET)
            print(LIGHT_GREEN + "File saved!" + RESET)

        # Try again option, continue loop if yes
        print(LIGHT_GREEN + "\nDo you want to try again?\n" + RESET)
        commandStart()
        process = inputInteger(1, 2)

    # Outside loop. Exit program
    print(LIGHT_GREEN + "\nThank you for using Shortest Path Solver!\n" +
RESET)

if __name__ == "__main__":
    main()
```

2. astar.py

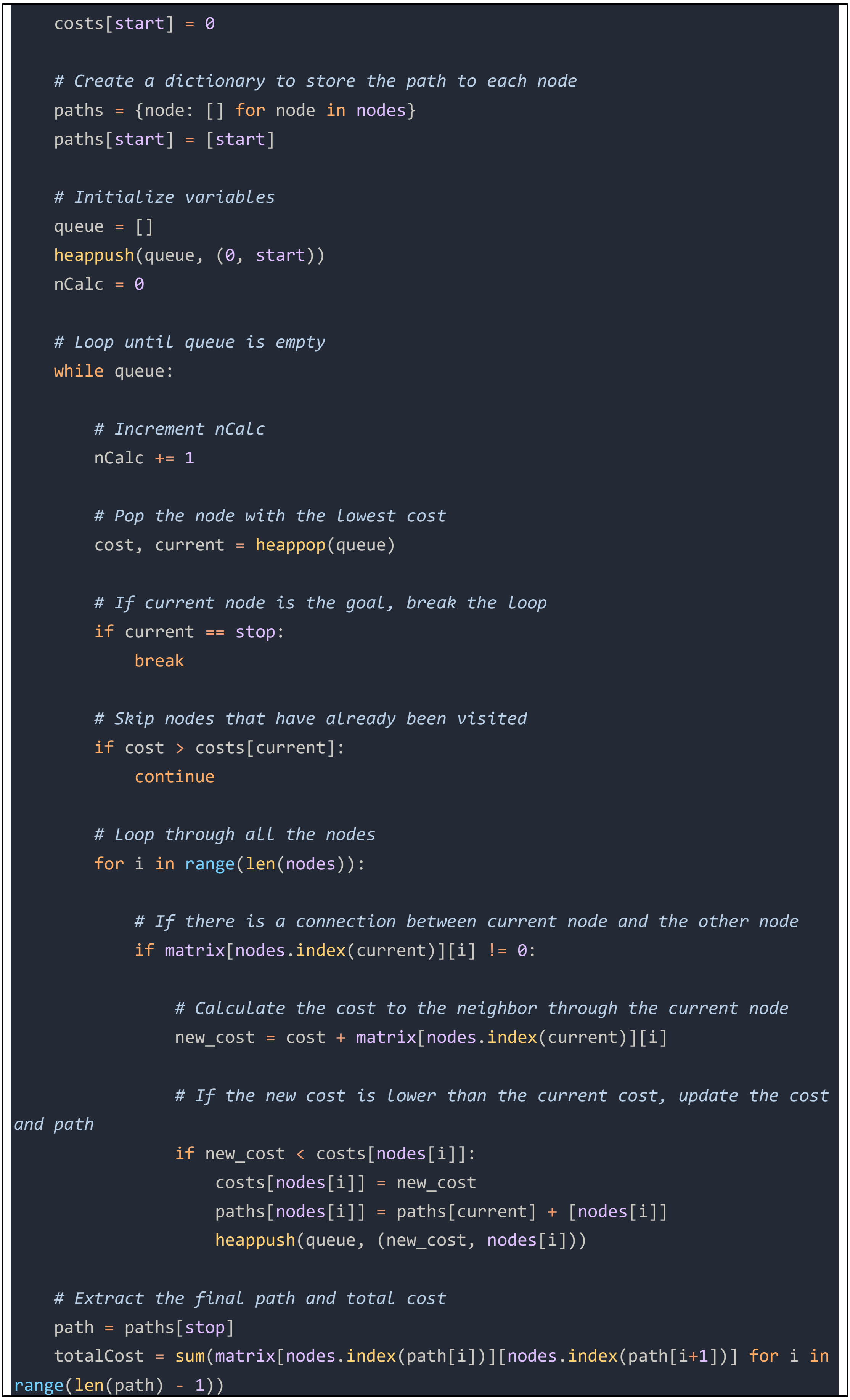
```
from heapq import heappush, heappop

def heuristic(nodes, matrix, current, goal):
    """
    Manhattan distance heuristic
    Input: nodes (array of string), matrix (2D array of integer), current
(string), goal (string)
    Output: h (dictionary)
    """

    # Initialize the heuristic dictionary with initial values as infinity for all
nodes
    h = {node: float('inf') for node in nodes}

    # Set the heuristic value for the current node to 0
```



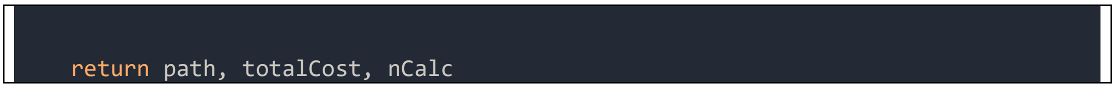
```
    h[current] = 0

    # Get the index of the goal node
    goal_idx = nodes.index(goal)

    # Get the coordinates of the goal node
    goal_x, goal_y = goal_idx // 3, goal_idx % 3

    # Loop through all the nodes
    for node in nodes:

        # Get the index and coordinates of the current node
        node_idx = nodes.index(node)
        node_x, node_y = node_idx // 3, node_idx % 3

        # Calculate the Manhattan distance from current node to goal node
        h[node] = abs(node_x - goal_x) + abs(node_y - goal_y)

    return h


def astar(nodes, matrix, start, stop):
    """
    A* algorithm
    Input: nodes (array of string), matrix (2D array of integer), start (string),
stop (string)
    Output: path (array of string), totalCost (integer), nCalc (integer)
    """

    # Initialize variables
    path = []
    visited = set()
    queue = []
    cost = 0
    h = heuristic(nodes, matrix, start, stop)
    heappush(queue, (cost + h[start], cost, start, [start]))
    nCalc = 0

    # Loop until queue is empty
    while queue:

        # Increment nCalc
        nCalc += 1

        # Pop the node with the lowest f value
        f, cost, current, path = heappop(queue)

        # If current node is the goal, break the loop
        if current == stop:
            break

        # If current node is not visited, add it to visited
        if current not in visited:
            visited.add(current)

            # Loop through all the nodes
            for i in range(len(nodes)):

                # If there is a connection between current node and the other node
                if matrix[nodes.index(current)][i] != 0:

                    # Calculate the new g value
                    new_cost = cost + matrix[nodes.index(current)][i]

                    # Calculate the new f value using the heuristic
                    new_f = new_cost + h[nodes[i]]

                    # Update the heuristic value for the next iteration
                    if nodes[i] not in h or new_f < h[nodes[i]]:
                        h[nodes[i]] = new_f

                    # Append the new node to queue
                    heappush(queue, (new_f, new_cost, nodes[i], path + [nodes[i]]))

    # Calculate total cost
    totalCost = 0
    for i in range(len(path) - 1):
        totalCost += matrix[nodes.index(path[i])][nodes.index(path[i + 1])]

    return path, totalCost, nCalc
```

3. ucs.py

```
from heapq import heappush, heappop

def ucs(nodes, matrix, start, stop):
    """
    Uniform Cost Search algorithm
    Input: nodes (array of string), matrix (2D array of integer), start (string), stop (string)
    Output: path (array of string), totalCost (integer), nCalc (integer)
    """

    # Create a dictionary to store the cost of each node
    costs = {node: float('inf') for node in nodes}
```

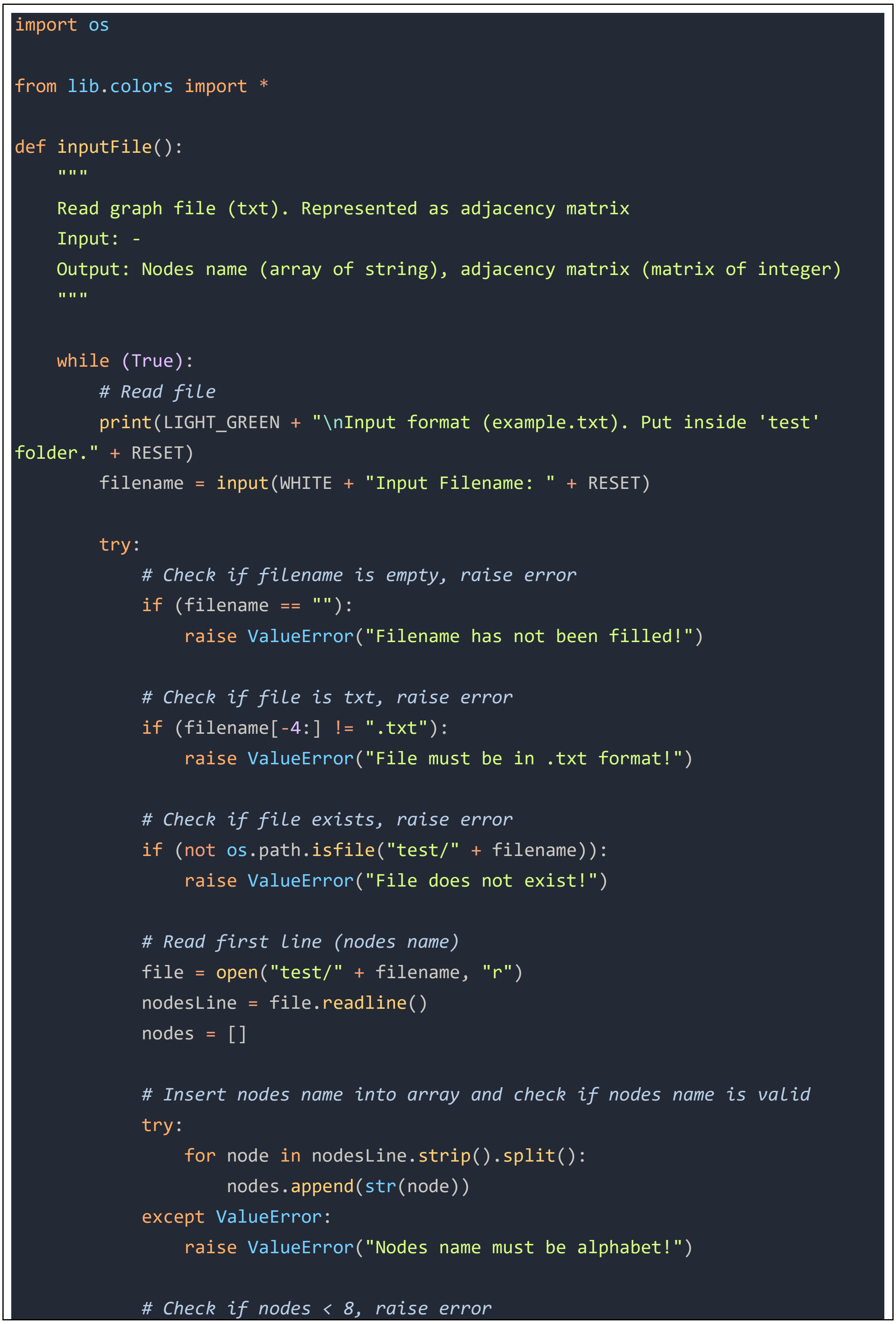


```
    costs[start] = 0

    # Create a dictionary to store the path to each node
    paths = {node: [] for node in nodes}
    paths[start] = [start]

    # Initialize variables
    queue = []
    heappush(queue, (0, start))
    nCalc = 0

    # Loop until queue is empty
    while queue:

        # Increment nCalc
        nCalc += 1

        # Pop the node with the lowest cost
        cost, current = heappop(queue)

        # If current node is the goal, break the loop
        if current == stop:
            break

        # Skip nodes that have already been visited
        if cost > costs[current]:
            continue

        # Loop through all the nodes
        for i in range(len(nodes)):

            # If there is a connection between current node and the other node
            if matrix[nodes.index(current)][i] != 0:

                # Calculate the cost to the neighbor through the current node
                new_cost = cost + matrix[nodes.index(current)][i]

                # If the new cost is lower than the current cost, update the cost
and path
                if new_cost < costs[nodes[i]]:
                    costs[nodes[i]] = new_cost
                    paths[nodes[i]] = paths[current] + [nodes[i]]
                    heappush(queue, (new_cost, nodes[i]))

    # Extract the final path and total cost
    path = paths[stop]
    totalCost = sum(matrix[nodes.index(path[i])][nodes.index(path[i+1])] for i in
range(len(path) - 1))
```

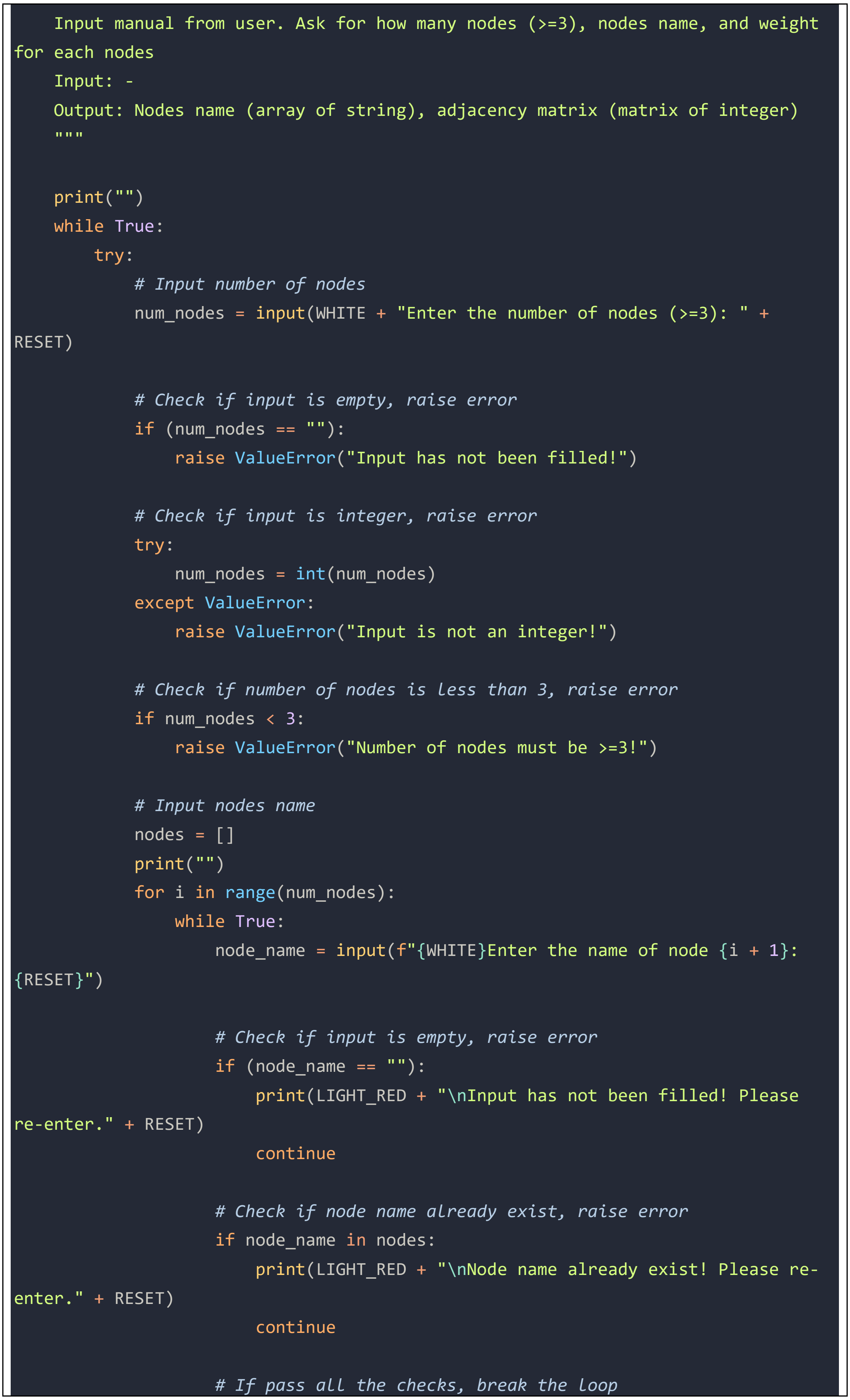


```
    return path, totalCost, nCalc
```

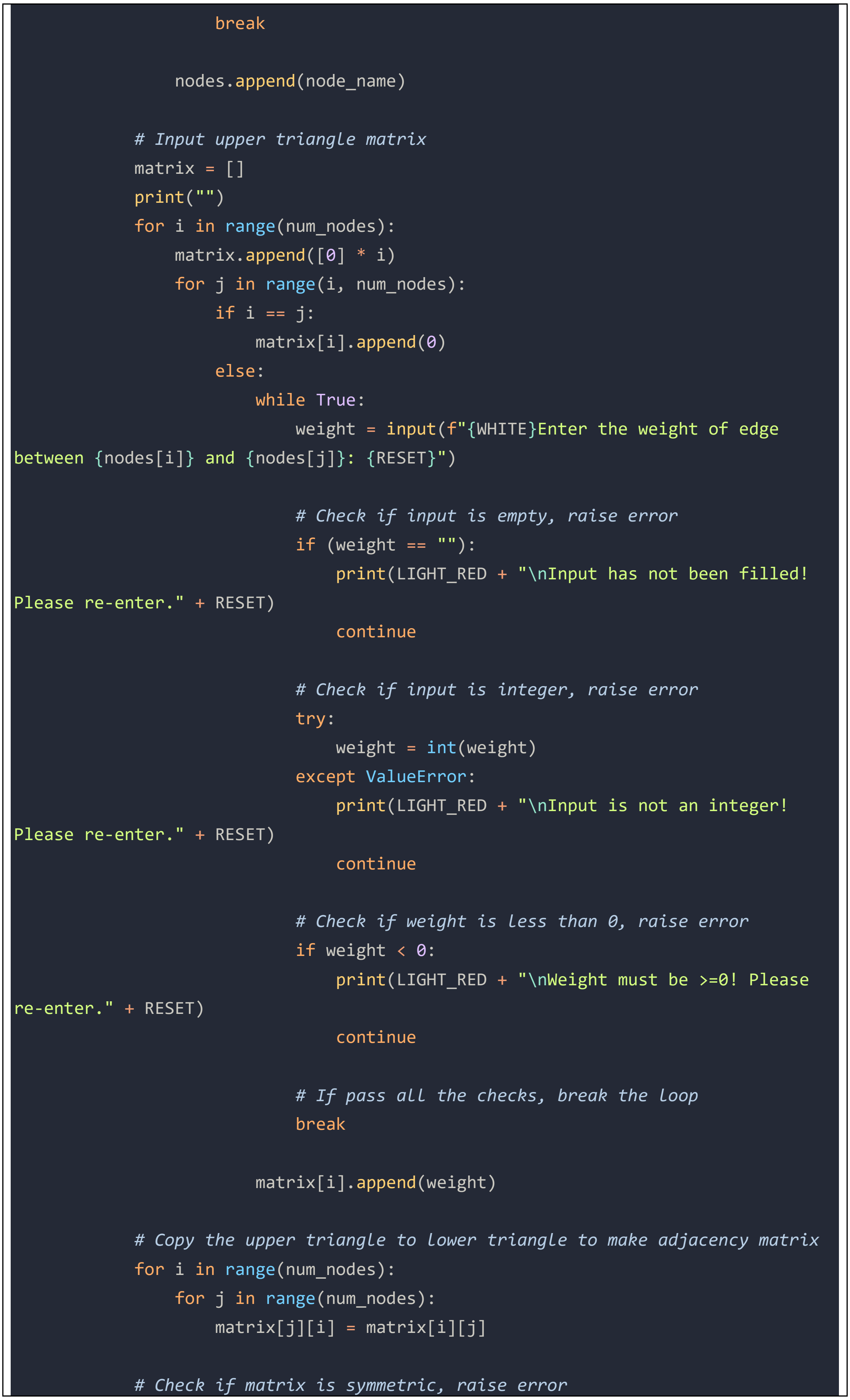
4. input.py



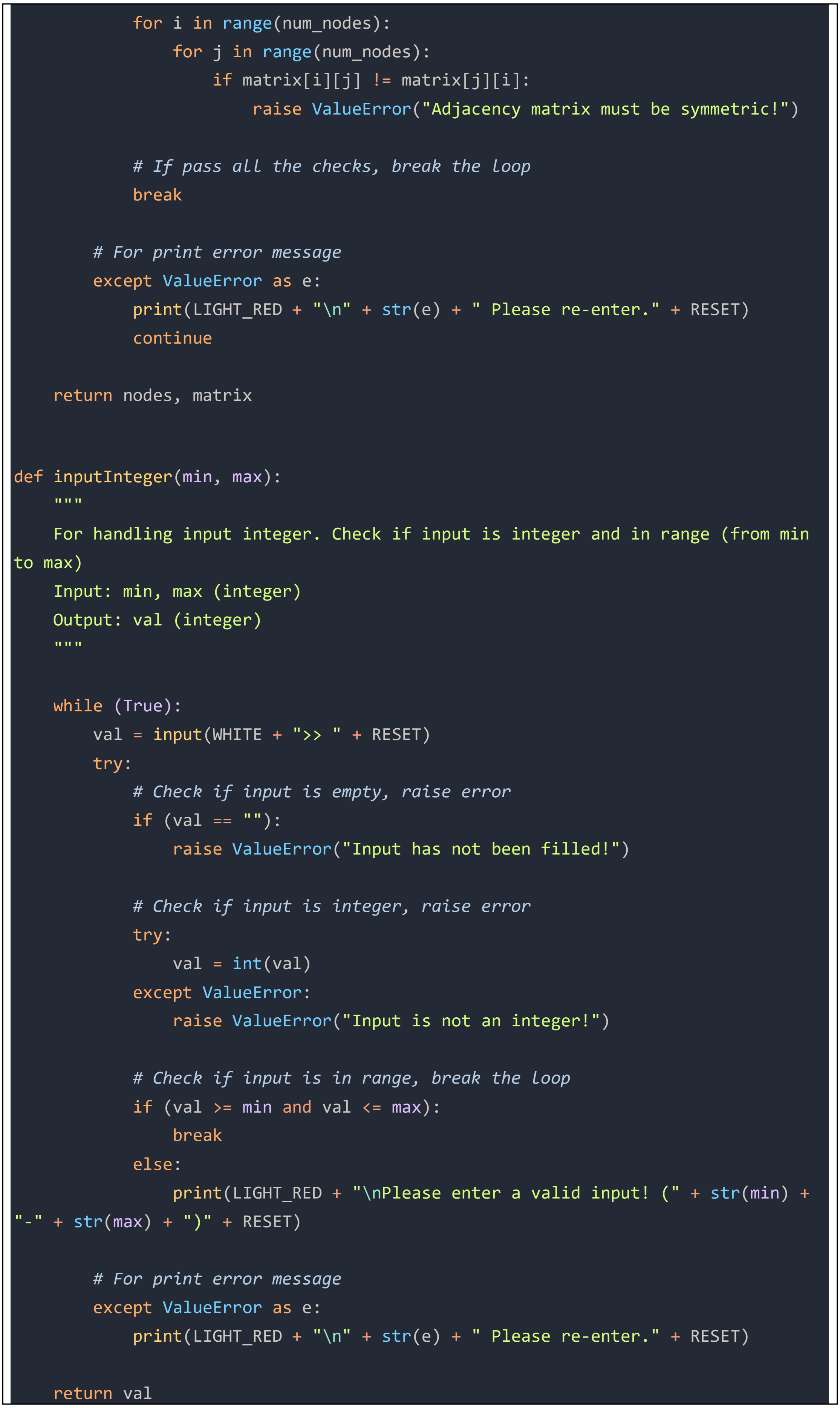
```
import os

from lib.colors import *

def inputFile():
    """
    Read graph file (txt). Represented as adjacency matrix
    Input: -
    Output: Nodes name (array of string), adjacency matrix (matrix of integer)
    """

    while (True):
        # Read file
        print(LIGHT_GREEN + "\nInput format (example.txt). Put inside 'test'
folder." + RESET)
        filename = input(WHITE + "Input Filename: " + RESET)

        try:
            # Check if filename is empty, raise error
            if (filename == ""):
                raise ValueError("Filename has not been filled!")

            # Check if file is txt, raise error
            if (filename[-4:] != ".txt"):
                raise ValueError("File must be in .txt format!")

            # Check if file exists, raise error
            if (not os.path.isfile("test/" + filename)):
                raise ValueError("File does not exist!")

            # Read first line (nodes name)
            file = open("test/" + filename, "r")
            nodesLine = file.readline()
            nodes = []

            # Insert nodes name into array and check if nodes name is valid
            try:
                for node in nodesLine.strip().split():
                    nodes.append(str(node))
            except ValueError:
                raise ValueError("Nodes name must be alphabet!")

            # Check if nodes < 8, raise error
            if (len(nodes) < 8):
                raise ValueError("Number of nodes must be at least 8!")

            # Check duplicate node name, raise error
            if (len(nodes) != len(set(nodes))):
                raise ValueError("Node name must be unique!")

            # Read the rest of the file (adjacency matrix)
            matrix = []
            for i in range(len(nodes)):
                line = file.readline()
                matrix.append(line.strip().split())

                # Check if matrix is not square (column), raise error
                if (len(matrix[i]) != len(nodes)):
                    raise ValueError("Adjacency matrix must be square!")

                # Check if matrix element is valid, raise error
                for j in range(len(nodes)):
                    try:
                        matrix[i][j] = int(matrix[i][j])
                    except ValueError:
                        raise ValueError("Adjacency matrix contains non-valid
elements!")

            # Check if matrix is not square (row), raise error
            with open("test/" + filename, "r") as f:
                if (len(f.readlines())-1 != len(nodes)):
                    raise ValueError("Adjacency matrix must be square!")

            # Check if matrix is symmetric, raise error
            for i in range(len(nodes)):
                for j in range(len(nodes)):
                    if (matrix[i][j] != matrix[j][i]):
                        raise ValueError("Adjacency matrix must be symmetric!")

            # If pass all the checks, break the loop
            break

        # For print error message
        except ValueError as e:
            print(LIGHT_RED + "\n" + str(e) + " Please re-enter." + RESET)
            continue

    return nodes, matrix


def inputManual():
    """
```



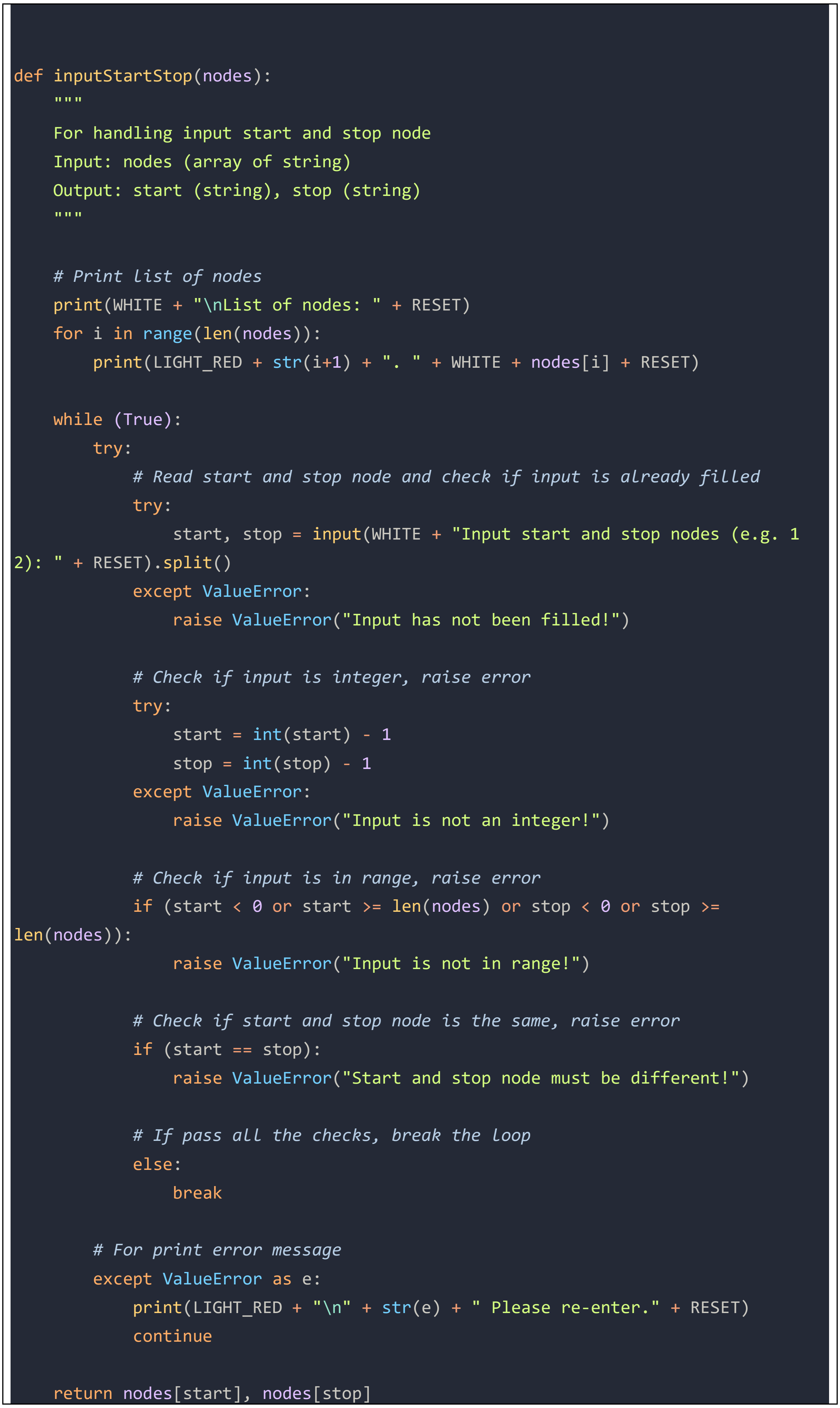
```python
    Input manual from user. Ask for how many nodes (>=3), nodes name, and weight for each nodes
    Input: -
    Output: Nodes name (array of string), adjacency matrix (matrix of integer)
    """

    print("")
    while True:
        try:
            # Input number of nodes
            num_nodes = input(WHITE + "Enter the number of nodes (>=3): " + RESET)

            # Check if input is empty, raise error
            if (num_nodes == ""):
                raise ValueError("Input has not been filled!")

            # Check if input is integer, raise error
            try:
                num_nodes = int(num_nodes)
            except ValueError:
                raise ValueError("Input is not an integer!")

            # Check if number of nodes is less than 3, raise error
            if num_nodes < 3:
                raise ValueError("Number of nodes must be >=3!")

            # Input nodes name
            nodes = []
            print("")
            for i in range(num_nodes):
                while True:
                    node_name = input(f"{WHITE}Enter the name of node {i + 1}: {RESET}")

                    # Check if input is empty, raise error
                    if (node_name == ""):
                        print(LIGHT_RED + "\nInput has not been filled! Please re-enter." + RESET)
                        continue

                    # Check if node name already exist, raise error
                    if node_name in nodes:
                        print(LIGHT_RED + "\nNode name already exist! Please re-enter." + RESET)
                        continue

                    # If pass all the checks, break the loop
```



```
                    break

                nodes.append(node_name)

            # Input upper triangle matrix
            matrix = []
            print("")
            for i in range(num_nodes):
                matrix.append([0] * i)
                for j in range(i, num_nodes):
                    if i == j:
                        matrix[i].append(0)
                    else:
                        while True:
                            weight = input(f"{WHITE}Enter the weight of edge
between {nodes[i]} and {nodes[j]}: {RESET}")

                            # Check if input is empty, raise error
                            if (weight == ""):
                                print(LIGHT_RED + "\nInput has not been filled!
Please re-enter." + RESET)
                                continue

                            # Check if input is integer, raise error
                            try:
                                weight = int(weight)
                            except ValueError:
                                print(LIGHT_RED + "\nInput is not an integer!
Please re-enter." + RESET)
                                continue

                            # Check if weight is less than 0, raise error
                            if weight < 0:
                                print(LIGHT_RED + "\nWeight must be >=0! Please
re-enter." + RESET)
                                continue

                            # If pass all the checks, break the loop
                            break

                        matrix[i].append(weight)

            # Copy the upper triangle to lower triangle to make adjacency matrix
            for i in range(num_nodes):
                for j in range(num_nodes):
                    matrix[j][i] = matrix[i][j]

            # Check if matrix is symmetric, raise error
```

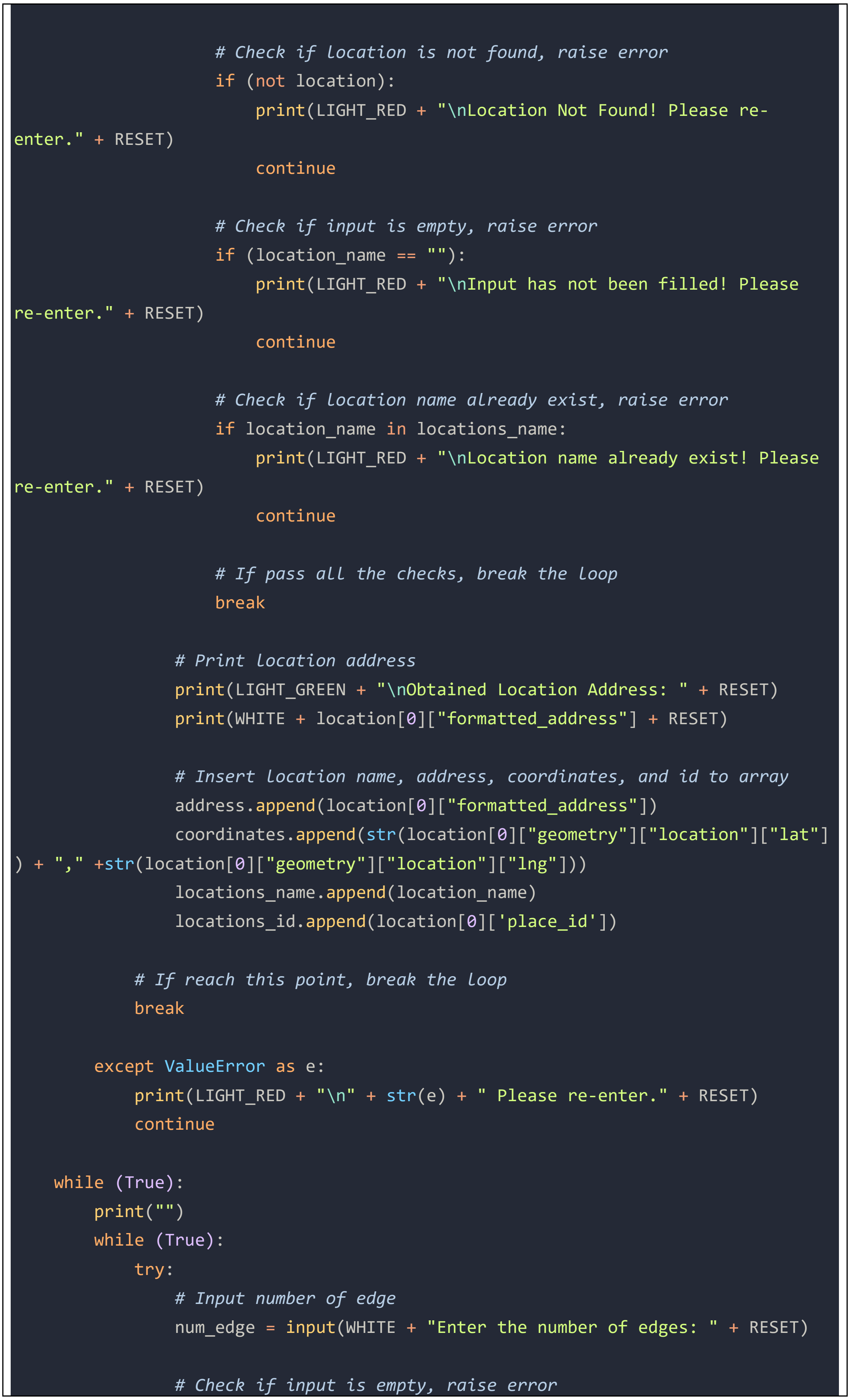


```
            for i in range(num_nodes):
                for j in range(num_nodes):
                    if matrix[i][j] != matrix[j][i]:
                        raise ValueError("Adjacency matrix must be symmetric!")

            # If pass all the checks, break the loop
            break

        # For print error message
        except ValueError as e:
            print(LIGHT_RED + "\n" + str(e) + " Please re-enter." + RESET)
            continue

    return nodes, matrix


def inputInteger(min, max):
    """
    For handling input integer. Check if input is integer and in range (from min
to max)
    Input: min, max (integer)
    Output: val (integer)
    """

    while (True):
        val = input(WHITE + ">> " + RESET)
        try:
            # Check if input is empty, raise error
            if (val == ""):
                raise ValueError("Input has not been filled!")

            # Check if input is integer, raise error
            try:
                val = int(val)
            except ValueError:
                raise ValueError("Input is not an integer!")

            # Check if input is in range, break the loop
            if (val >= min and val <= max):
                break
            else:
                print(LIGHT_RED + "\nPlease enter a valid input! (" + str(min) +
"-" + str(max) + ")" + RESET)

        # For print error message
        except ValueError as e:
            print(LIGHT_RED + "\n" + str(e) + " Please re-enter." + RESET)

    return val
```

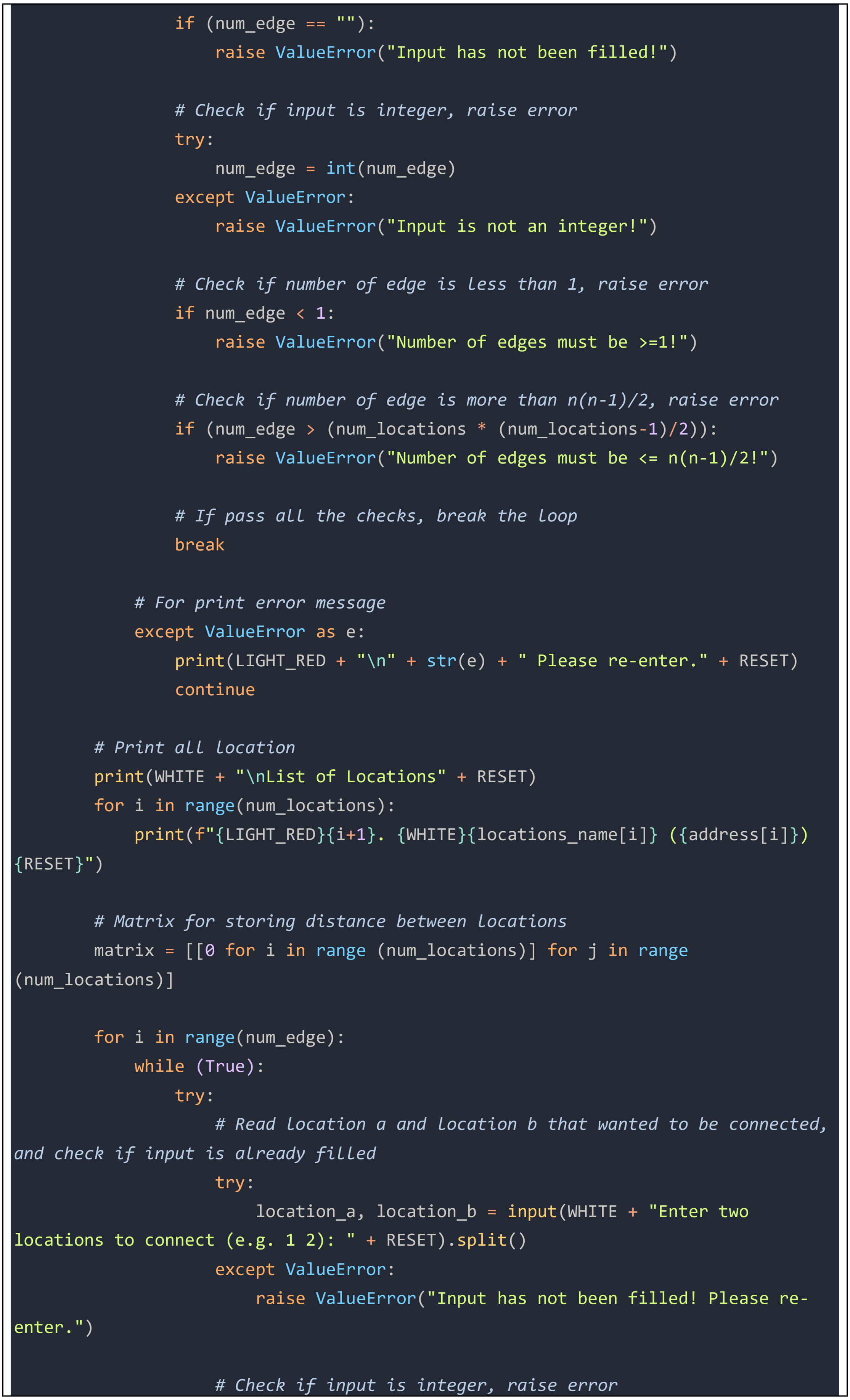


```
def inputStartStop(nodes):
    """
    For handling input start and stop node
    Input: nodes (array of string)
    Output: start (string), stop (string)
    """

    # Print list of nodes
    print(WHITE + "\nList of nodes: " + RESET)
    for i in range(len(nodes)):
        print(LIGHT_RED + str(i+1) + ". " + WHITE + nodes[i] + RESET)

    while (True):
        try:
            # Read start and stop node and check if input is already filled
            try:
                start, stop = input(WHITE + "Input start and stop nodes (e.g. 1
2): " + RESET).split()
            except ValueError:
                raise ValueError("Input has not been filled!")

            # Check if input is integer, raise error
            try:
                start = int(start) - 1
                stop = int(stop) - 1
            except ValueError:
                raise ValueError("Input is not an integer!")

            # Check if input is in range, raise error
            if (start < 0 or start >= len(nodes) or stop < 0 or stop >=
len(nodes)):
                raise ValueError("Input is not in range!")

            # Check if start and stop node is the same, raise error
            if (start == stop):
                raise ValueError("Start and stop node must be different!")

            # If pass all the checks, break the loop
            else:
                break

        # For print error message
        except ValueError as e:
            print(LIGHT_RED + "\n" + str(e) + " Please re-enter." + RESET)
            continue

    return nodes[start], nodes[stop]
```

```
def inputMap(gmapsClient):
    """
    For handling input map
    Input: gmapsClient (googlemaps.Client)
    Output: locations_name (array of string), matrix (matrix of integer),
coordinates (array of string)
    """

    print("")
    while True:
        try:
            # Input number of location
            num_locations = input(WHITE + "Enter the number of locations (>=3): "
+ RESET)

            # Check if input is empty, raise error
            if (num_locations == ""):
                raise ValueError("Input has not been filled!")

            # Check if input is integer, raise error
            try:
                num_locations = int(num_locations)
            except ValueError:
                raise ValueError("Input is not an integer!")

            # Check if number of location is less than 3, raise error
            if num_locations < 3:
                raise ValueError("Number of locations must be >=3!")

            print(LIGHT_GREEN + "\nInput a place, landmark, or address name.
Recommended to put specific address or full name.")
            print("Example: Institut Teknologi Bandung, Gedung Sate, Jl.Cisitu" +
RESET)

            # Array for storing location name, address, id, and coordinates
            locations_name = []
            locations_id = []
            address = []
            coordinates = []

            for i in range(num_locations):
                print("")
                while True:
                    # Input location name
                    location_name = input(f"{WHITE}Enter the name of location {i
+ 1}: {RESET}")
                    location = gmapsClient.geocode(address = location_name)
```

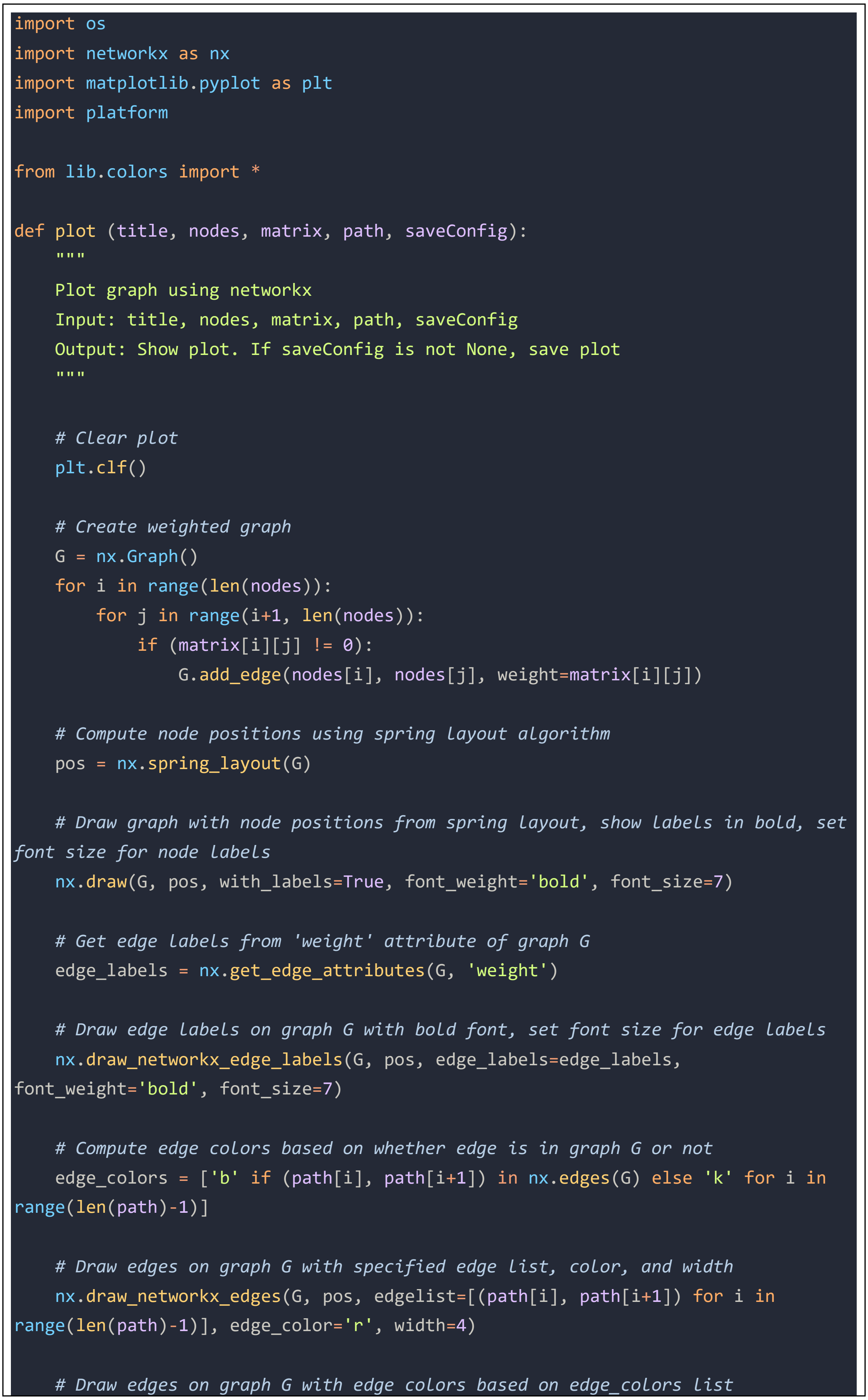


```python
                    # Check if location is not found, raise error
                    if (not location):
                        print(LIGHT_RED + "\nLocation Not Found! Please re-enter." + RESET)
                        continue

                    # Check if input is empty, raise error
                    if (location_name == ""):
                        print(LIGHT_RED + "\nInput has not been filled! Please re-enter." + RESET)
                        continue

                    # Check if location name already exist, raise error
                    if location_name in locations_name:
                        print(LIGHT_RED + "\nLocation name already exist! Please re-enter." + RESET)
                        continue

                    # If pass all the checks, break the loop
                    break

                # Print location address
                print(LIGHT_GREEN + "\nObtained Location Address: " + RESET)
                print(WHITE + location[0]["formatted_address"] + RESET)

                # Insert location name, address, coordinates, and id to array
                address.append(location[0]["formatted_address"])
                coordinates.append(str(location[0]["geometry"]["location"]["lat"]) + "," +str(location[0]["geometry"]["location"]["lng"]))
                locations_name.append(location_name)
                locations_id.append(location[0]['place_id'])

            # If reach this point, break the loop
            break

        except ValueError as e:
            print(LIGHT_RED + "\n" + str(e) + " Please re-enter." + RESET)
            continue

    while (True):
        print("")
        while (True):
            try:
                # Input number of edge
                num_edge = input(WHITE + "Enter the number of edges: " + RESET)

                # Check if input is empty, raise error
```



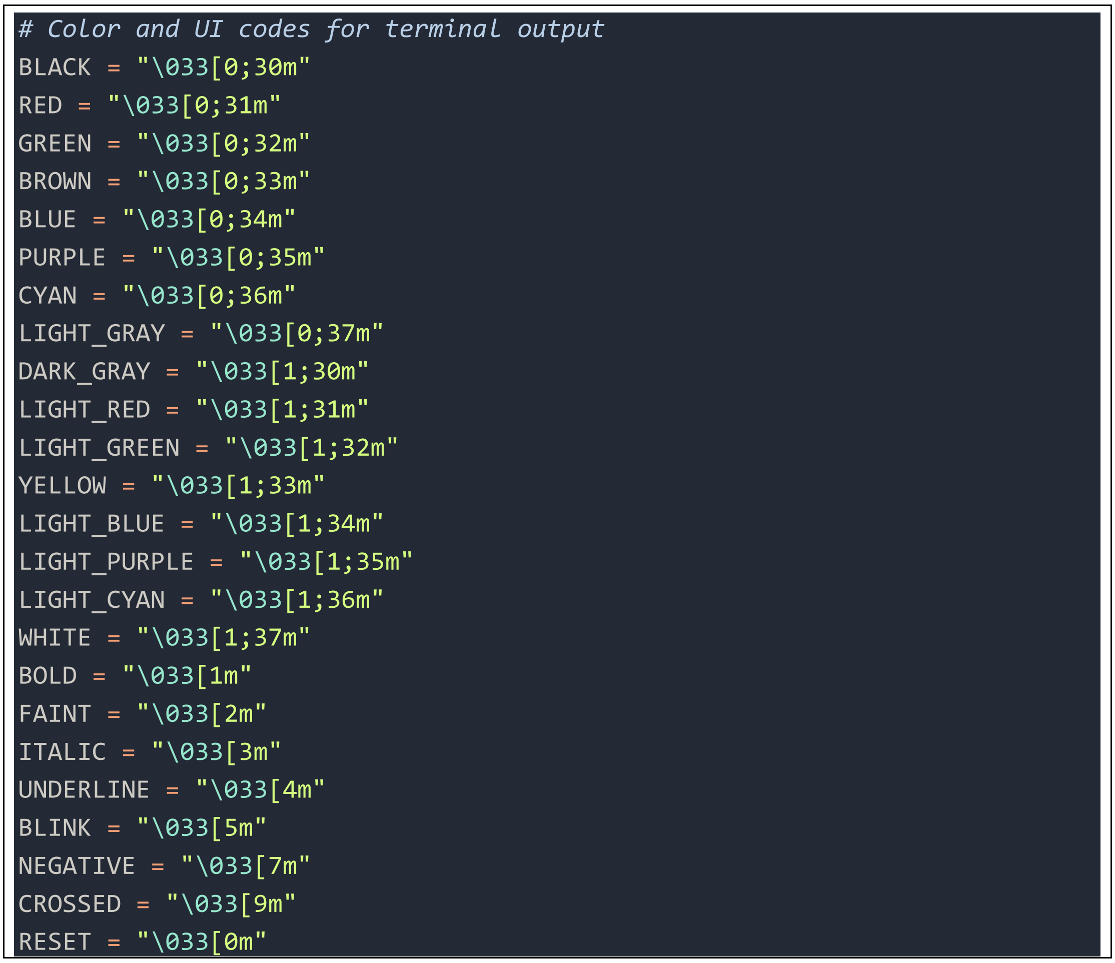
```
                if (num_edge == ""):
                    raise ValueError("Input has not been filled!")

                # Check if input is integer, raise error
                try:
                    num_edge = int(num_edge)
                except ValueError:
                    raise ValueError("Input is not an integer!")

                # Check if number of edge is less than 1, raise error
                if num_edge < 1:
                    raise ValueError("Number of edges must be >=1!")

                # Check if number of edge is more than n(n-1)/2, raise error
                if (num_edge > (num_locations * (num_locations-1)/2)):
                    raise ValueError("Number of edges must be <= n(n-1)/2!")

                # If pass all the checks, break the loop
                break

            # For print error message
            except ValueError as e:
                print(LIGHT_RED + "\n" + str(e) + " Please re-enter." + RESET)
                continue

        # Print all location
        print(WHITE + "\nList of Locations" + RESET)
        for i in range(num_locations):
            print(f"{LIGHT_RED}{i+1}. {WHITE}{locations_name[i]} ({address[i]})
{RESET}")

        # Matrix for storing distance between locations
        matrix = [[0 for i in range (num_locations)] for j in range
(num_locations)]

        for i in range(num_edge):
            while (True):
                try:
                    # Read location a and location b that wanted to be connected,
and check if input is already filled
                    try:
                        location_a, location_b = input(WHITE + "Enter two
locations to connect (e.g. 1 2): " + RESET).split()
                    except ValueError:
                        raise ValueError("Input has not been filled! Please re-
enter.")

                    # Check if input is integer, raise error
                    try:
                        location_a = int(location_a) - 1
                        location_b = int(location_b) - 1
                    except ValueError:
                        raise ValueError("Input is not an integer! Please re-enter.")

                    # Check if input is in range, raise error
                    if (location_a < 0 or location_a >= len(locations_name) or location_b < 0 or location_b >= len(locations_name)):
                        raise ValueError("Input is not in range! Please re-enter.")

                    # Check if input is already connected, raise error
                    if (matrix[location_a][location_b] != 0):
                        raise ValueError("Locations is already connected! Please re-enter.")

                    # Check if location_a and location_b is the same, raise error
                    if (location_a == location_b):
                        raise ValueError("Locations is the same! Please re-enter.")

                    # If pass all the checks, break the loop
                    break

                # For print error message
                except ValueError as e:
                    print(LIGHT_RED + "\n" + str(e) + RESET)
                    continue

            # Get distance between location_a and location_b
            distance = gmapsClient.directions(coordinates[location_a],coordinates[location_b])[0]['legs'][0]['distance']['value']

            # Insert distance to matrix
            matrix[location_a][location_b] = distance
            matrix[location_b][location_a] = distance

        break

    return locations_name, matrix, coordinates
```

5. output.py



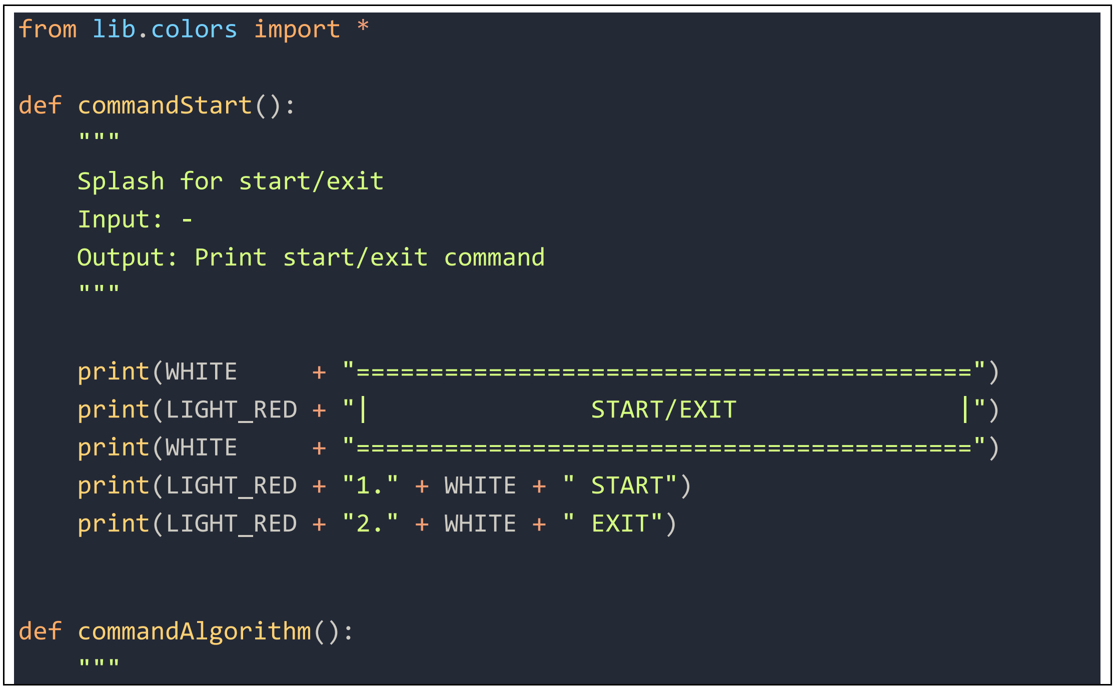
```
import os
import networkx as nx
import matplotlib.pyplot as plt
import platform

from lib.colors import *

def plot (title, nodes, matrix, path, saveConfig):
    """
    Plot graph using networkx
    Input: title, nodes, matrix, path, saveConfig
    Output: Show plot. If saveConfig is not None, save plot
    """

    # Clear plot
    plt.clf()

    # Create weighted graph
    G = nx.Graph()
    for i in range(len(nodes)):
        for j in range(i+1, len(nodes)):
            if (matrix[i][j] != 0):
                G.add_edge(nodes[i], nodes[j], weight=matrix[i][j])

    # Compute node positions using spring layout algorithm
    pos = nx.spring_layout(G)

    # Draw graph with node positions from spring layout, show labels in bold, set
font size for node labels
    nx.draw(G, pos, with_labels=True, font_weight='bold', font_size=7)

    # Get edge labels from 'weight' attribute of graph G
    edge_labels = nx.get_edge_attributes(G, 'weight')

    # Draw edge labels on graph G with bold font, set font size for edge labels
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels,
font_weight='bold', font_size=7)

    # Compute edge colors based on whether edge is in graph G or not
    edge_colors = ['b' if (path[i], path[i+1]) in nx.edges(G) else 'k' for i in
range(len(path)-1)]

    # Draw edges on graph G with specified edge list, color, and width
    nx.draw_networkx_edges(G, pos, edgelist=[(path[i], path[i+1]) for i in
range(len(path)-1)], edge_color='r', width=4)

    # Draw edges on graph G with edge colors based on edge_colors list
    nx.draw_networkx_edges(G, pos, edge_color=edge_colors)

    # Set title, set font size for title
    plt.suptitle(title, fontsize=10)

    # Show or save plot
    if saveConfig is None:
        plt.show()
    else:
        plt.savefig(saveConfig)


def printResult(algorithm, start, stop, nCalc, path, totalCost, time):
    """
    Print result of algorithm
    Input: algorithm, start, stop, nCalc, path, totalCost, time
    Output: Print result
    """

    if (algorithm == "UNIFORM COST SEARCH"):
        print(WHITE + "\n=============== " + LIGHT_RED + "UNIFORM COST SEARCH" +
WHITE + " ===============")
    else:
        print(WHITE + "\n====================== " + LIGHT_RED + "A*" + WHITE + "
=======================")
    print(WHITE + "Start node: " + YELLOW + str(start))
    print(WHITE + "Stop node: " + YELLOW + str(stop))
    print(WHITE + "Shortest path: " + YELLOW + " -> ".join(str(p) for p in path))
    print(WHITE + "Total cost: " + YELLOW + str(totalCost))
    print(WHITE + "Number of calculations: " + YELLOW + str(nCalc))
    print(WHITE + "Execution time: " + YELLOW + "{:.2f} ms".format(time * 1000))
    print(WHITE + "Processor: " + YELLOW + str(platform.processor()) + RESET)


def saveResult(algorithm, start, stop, nCalc, path, totalCost, time, nodes,
matrix, saveConfig, mapImage, option):
    """
    Save result of algorithm
    Input: algorithm, start, stop, nCalc, path, totalCost, time, nodes, matrix,
saveConfig
    Output: Save result
    """

    # Create folder if not exist
    if not os.path.exists("test"):
        os.mkdir("test")
    if not os.path.exists("test/" + saveConfig):
        os.mkdir("test/" + saveConfig)
    if algorithm == "A*":
        f = open("test/" + saveConfig + "/" + saveConfig + "AStar.txt", "w")

        # save image
        if(option == 2):
            mapImage.save("test/" + saveConfig + "/" + saveConfig +
"MapAStar.png")

        # Save plot
        plot(algorithm, nodes, matrix, path, "test/" + saveConfig + "/" +
saveConfig + "AStar.png")
    else:
        f = open("test/" + saveConfig + "/" + saveConfig + "UCS.txt", "w")

        # save image
        if(option == 2):
            mapImage.save("test/" + saveConfig + "/" + saveConfig + "MapUCS.png")

        # Save plot
        plot(algorithm, nodes, matrix, path, "test/" + saveConfig + "/" +
saveConfig + "UCS.png")

    # Save nodes
    f.write("Nodes:\n")
    for i in range(len(nodes)):
        f.write(str(nodes[i]) + "\n")
    f.write("\n")

    # Save matrix
    f.write("Matrix:\n")
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            f.write(str(matrix[i][j]) + " ")
        f.write("\n")

    # Save weight for each connected node
    f.write("\nWeight for each node:\n")
    for i in range(len(nodes)):
        for j in range(i+1, len(nodes)):
            if (matrix[i][j] != 0):
                f.write(str(nodes[i]) + " <-> " + str(nodes[j]) + " = " +
str(matrix[i][j]) + "\n")

    # Save result
    if (algorithm == "UNIFORM COST SEARCH"):
        f.write("\n=============== " + "UNIFORM COST SEARCH" + "
===============\n")
    else:
        f.write("\n======================= " + "A*" + "
========================\n")
    f.write("Start node: " + str(start) + "\n")
    f.write("Stop node: " + str(stop) + "\n")
    f.write("Shortest path: " + " -> ".join(str(p) for p in path) + "\n")
    f.write("Total cost: " + str(totalCost) + "\n")
    f.write("Number of calculations: " + str(nCalc) + "\n")
    f.write("Execution time: " + "{:.2f} ms".format(time * 1000) + "\n")
    f.write("Processor: " + str(platform.processor()))


def addPathUrl(matrix, coordinates, url, gmapsClient):
    pathurl = url

    # find relation
    for i in range (len(matrix)):
        for j in range (len(matrix)):
            if(i != j and i < j and matrix[i][j]):
                enc =
gmapsClient.directions(coordinates[i],coordinates[j])[0]["overview_polyline"]["po
ints"]
                pathurl += f"&path=color:blue%7Cenc:{enc}"

    return pathurl


def addShortestPathUrl(coordinates, url, gmapsClient, path):
    pathurl = url

    for i in range(len(path)-1):
        enc =
gmapsClient.directions(coordinates[path[i]],coordinates[path[i+1]])[0]["overview_
polyline"]["points"]
        pathurl += f"&path=color:0xff0000ff%7Cenc:{enc}"

    return pathurl


def convertPathToInt(path, nodes):
    intPath = []

    for el in path:
        intPath.append(nodes.index(el))

    return intPath
```

6. colors.py

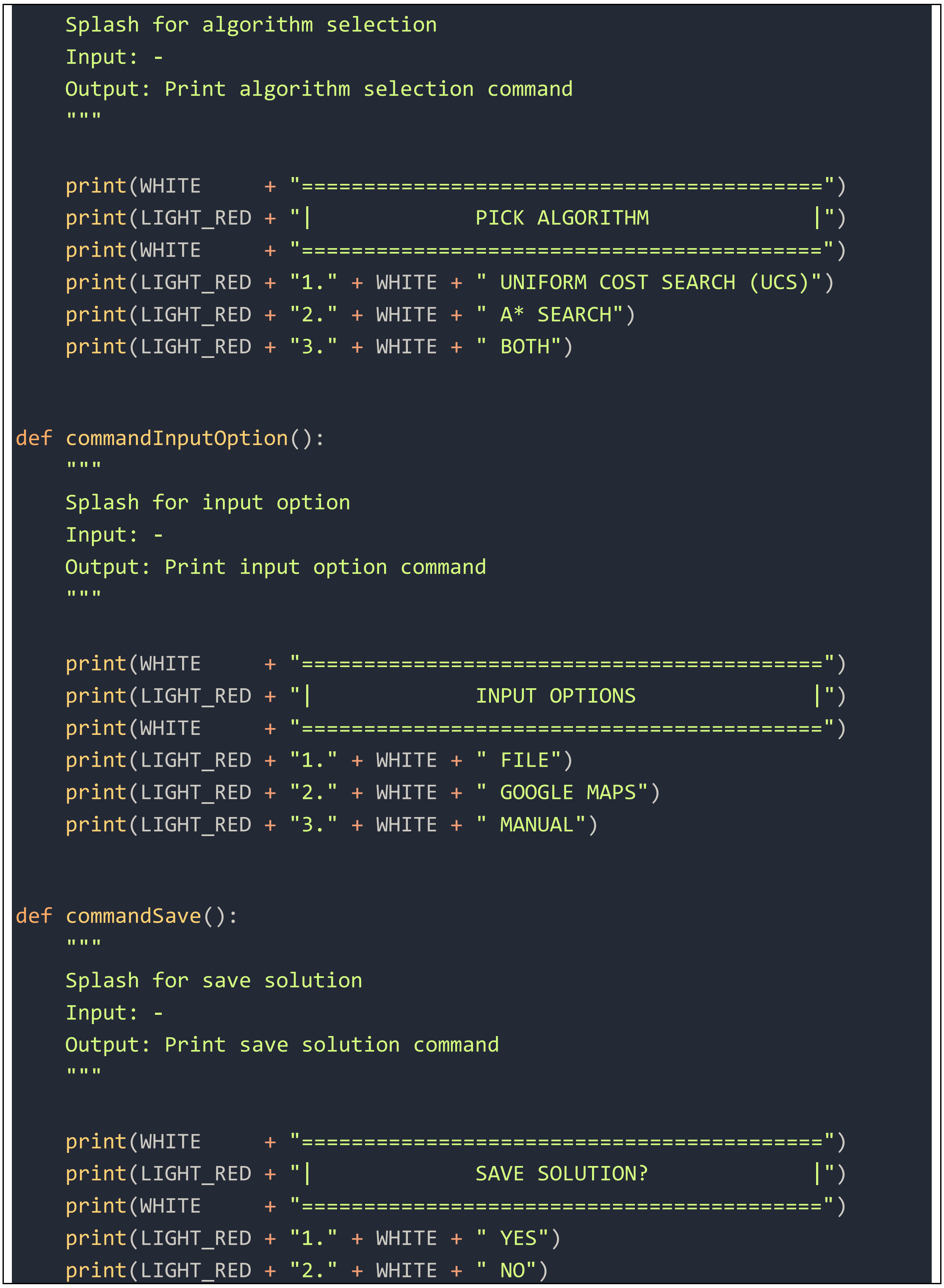


```
# Color and UI codes for terminal output
BLACK = "\033[0;30m"
RED = "\033[0;31m"
GREEN = "\033[0;32m"
BROWN = "\033[0;33m"
BLUE = "\033[0;34m"
PURPLE = "\033[0;35m"
CYAN = "\033[0;36m"
LIGHT_GRAY = "\033[0;37m"
DARK_GRAY = "\033[1;30m"
LIGHT_RED = "\033[1;31m"
LIGHT_GREEN = "\033[1;32m"
YELLOW = "\033[1;33m"
LIGHT_BLUE = "\033[1;34m"
LIGHT_PURPLE = "\033[1;35m"
LIGHT_CYAN = "\033[1;36m"
WHITE = "\033[1;37m"
BOLD = "\033[1m"
FAINT = "\033[2m"
ITALIC = "\033[3m"
UNDERLINE = "\033[4m"
BLINK = "\033[5m"
NEGATIVE = "\033[7m"
CROSSED = "\033[9m"
RESET = "\033[0m"
```

7. command.py

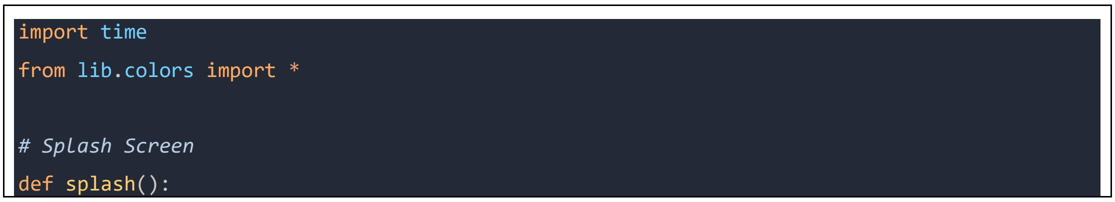


```
from lib.colors import *

def commandStart():
    """
    Splash for start/exit
    Input: -
    Output: Print start/exit command
    """

    print(WHITE     + "==========================================")
    print(LIGHT_RED + "|               START/EXIT               |")
    print(WHITE     + "==========================================")
    print(LIGHT_RED + "1." + WHITE + " START")
    print(LIGHT_RED + "2." + WHITE + " EXIT")


def commandAlgorithm():
    """
```

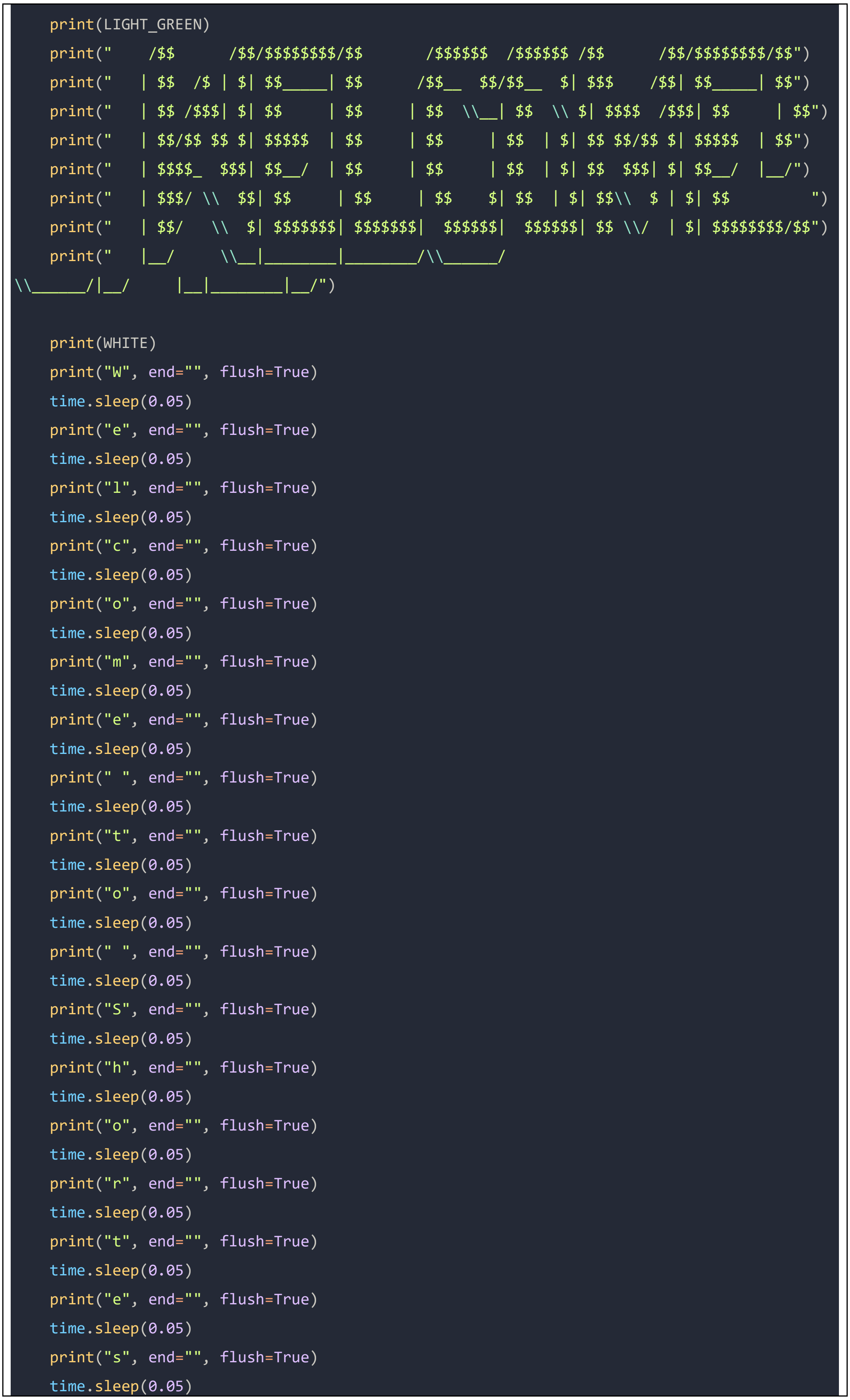


```
    Splash for algorithm selection
    Input: -
    Output: Print algorithm selection command
    """

    print(WHITE     + "==========================================")
    print(LIGHT_RED + "|             PICK ALGORITHM             |")
    print(WHITE     + "==========================================")
    print(LIGHT_RED + "1." + WHITE + " UNIFORM COST SEARCH (UCS)")
    print(LIGHT_RED + "2." + WHITE + " A* SEARCH")
    print(LIGHT_RED + "3." + WHITE + " BOTH")


def commandInputOption():
    """
    Splash for input option
    Input: -
    Output: Print input option command
    """

    print(WHITE     + "==========================================")
    print(LIGHT_RED + "|             INPUT OPTIONS              |")
    print(WHITE     + "==========================================")
    print(LIGHT_RED + "1." + WHITE + " FILE")
    print(LIGHT_RED + "2." + WHITE + " GOOGLE MAPS")
    print(LIGHT_RED + "3." + WHITE + " MANUAL")


def commandSave():
    """
    Splash for save solution
    Input: -
    Output: Print save solution command
    """

    print(WHITE     + "==========================================")
    print(LIGHT_RED + "|             SAVE SOLUTION?             |")
    print(WHITE     + "==========================================")
    print(LIGHT_RED + "1." + WHITE + " YES")
    print(LIGHT_RED + "2." + WHITE + " NO")
```

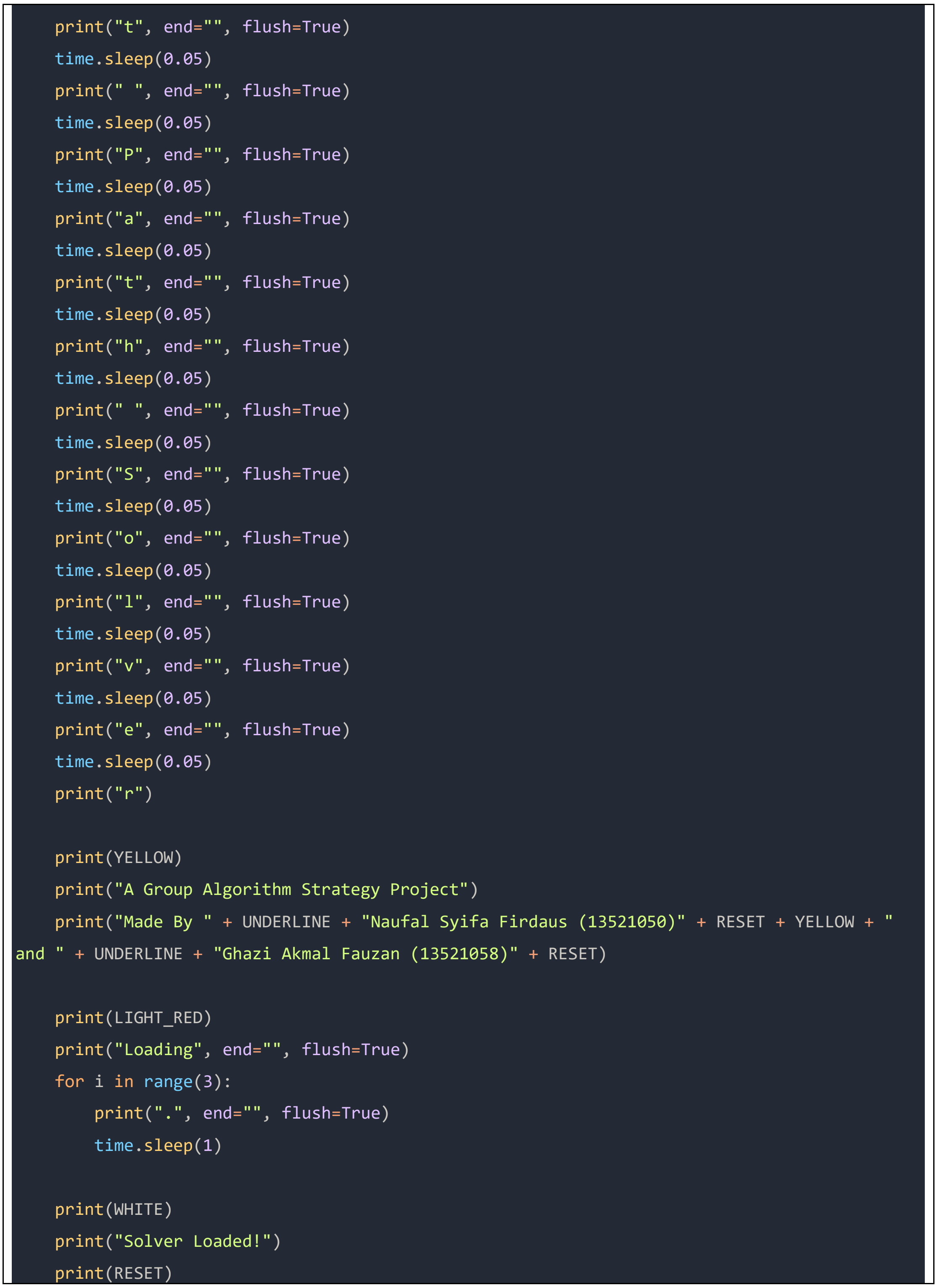
8. splash.py



```
import time
from lib.colors import *

# Splash Screen
def splash():
```



```
    print(LIGHT_GREEN)
    print("    /$$      /$$/$$$$$$$$/$$       /$$$$$$  /$$$$$$ /$$      /$$/$$$$$$$$/$$")
    print("   | $$  /$ | $| $$_____| $$      /$$__  $$/$$__  $| $$$    /$$| $$_____| $$")
    print("   | $$ /$$$| $| $$     | $$     | $$  \\__| $$  \\ $| $$$$  /$$$| $$     | $$")
    print("   | $$/$$ $$ $| $$$$$  | $$     | $$     | $$  | $| $$ $$/$$ $| $$$$$  | $$")
    print("   | $$$$_  $$$| $$__/  | $$     | $$     | $$  | $| $$  $$$| $| $$__/  |__/")
    print("   | $$$/ \\  $$| $$     | $$     | $$    $| $$  | $| $$\\  $ | $| $$         ")
    print("   | $$/   \\  $| $$$$$$$| $$$$$$$|  $$$$$$|  $$$$$$| $$ \\/  | $| $$$$$$$$/$$")
    print("   |__/     \\__|________|________/\\______/
\\______/|__/     |__|________|__/")

    print(WHITE)
    print("W", end="", flush=True)
    time.sleep(0.05)
    print("e", end="", flush=True)
    time.sleep(0.05)
    print("l", end="", flush=True)
    time.sleep(0.05)
    print("c", end="", flush=True)
    time.sleep(0.05)
    print("o", end="", flush=True)
    time.sleep(0.05)
    print("m", end="", flush=True)
    time.sleep(0.05)
    print("e", end="", flush=True)
    time.sleep(0.05)
    print(" ", end="", flush=True)
    time.sleep(0.05)
    print("t", end="", flush=True)
    time.sleep(0.05)
    print("o", end="", flush=True)
    time.sleep(0.05)
    print(" ", end="", flush=True)
    time.sleep(0.05)
    print("S", end="", flush=True)
    time.sleep(0.05)
    print("h", end="", flush=True)
    time.sleep(0.05)
    print("o", end="", flush=True)
    time.sleep(0.05)
    print("r", end="", flush=True)
    time.sleep(0.05)
    print("t", end="", flush=True)
    time.sleep(0.05)
    print("e", end="", flush=True)
    time.sleep(0.05)
    print("s", end="", flush=True)
    time.sleep(0.05)
```



```
    print("t", end="", flush=True)
    time.sleep(0.05)
    print(" ", end="", flush=True)
    time.sleep(0.05)
    print("P", end="", flush=True)
    time.sleep(0.05)
    print("a", end="", flush=True)
    time.sleep(0.05)
    print("t", end="", flush=True)
    time.sleep(0.05)
    print("h", end="", flush=True)
    time.sleep(0.05)
    print(" ", end="", flush=True)
    time.sleep(0.05)
    print("S", end="", flush=True)
    time.sleep(0.05)
    print("o", end="", flush=True)
    time.sleep(0.05)
    print("l", end="", flush=True)
    time.sleep(0.05)
    print("v", end="", flush=True)
    time.sleep(0.05)
    print("e", end="", flush=True)
    time.sleep(0.05)
    print("r")

    print(YELLOW)
    print("A Group Algorithm Strategy Project")
    print("Made By " + UNDERLINE + "Naufal Syifa Firdaus (13521050)" + RESET + YELLOW + " and " + UNDERLINE + "Ghazi Akmal Fauzan (13521058)" + RESET)

    print(LIGHT_RED)
    print("Loading", end="", flush=True)
    for i in range(3):
        print(".", end="", flush=True)
        time.sleep(1)

    print(WHITE)
    print("Solver Loaded!")
    print(RESET)
```

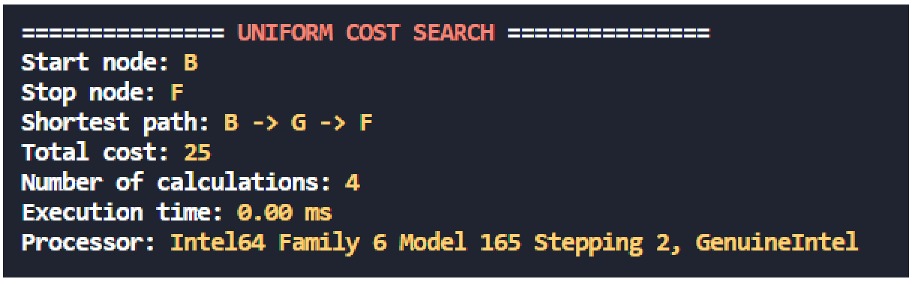
F. Eksperimen Test Case

a. graph1.txt

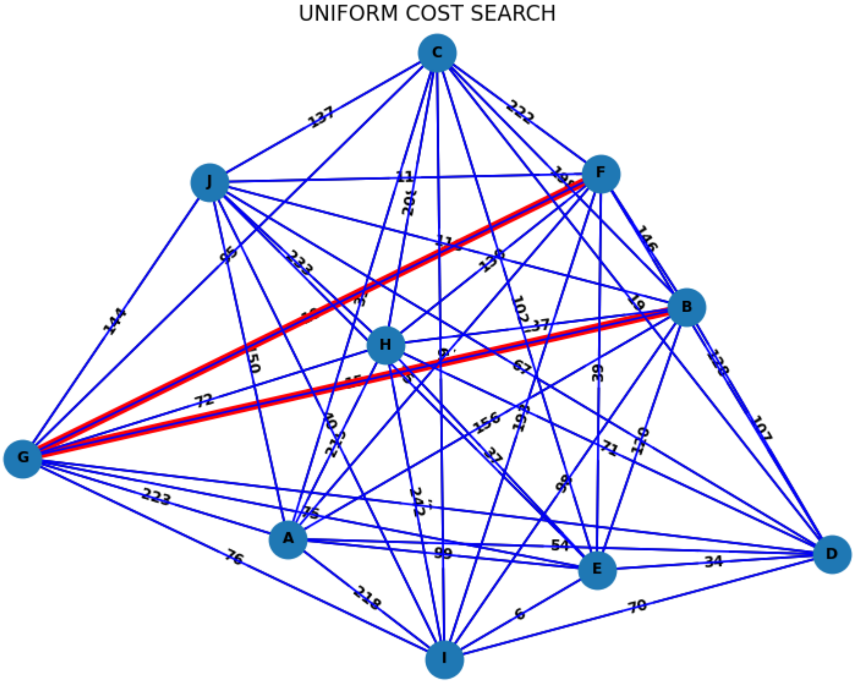
input

```
A    B    C    D    E    F    G    H    I    J
  0 156  39  54  99 102 223 219 218 150
156   0 198 107 120 146  15 137  98 118
 39 198   0  19 102 222  95 208  61 137
 54 107  19   0  34 128   6  71  70  67
 99 120 102  34   0  39  75  37   6  35
102 146 222 128  39   0  10 130 193  11
223  15  95   6  75  10   0  72  76 144
219 137 208  71  37 130  72   0 242 233
218  98  61  70   6 193  76 242   0  40
150 118 137  67  35  11 144 233  40   0
```

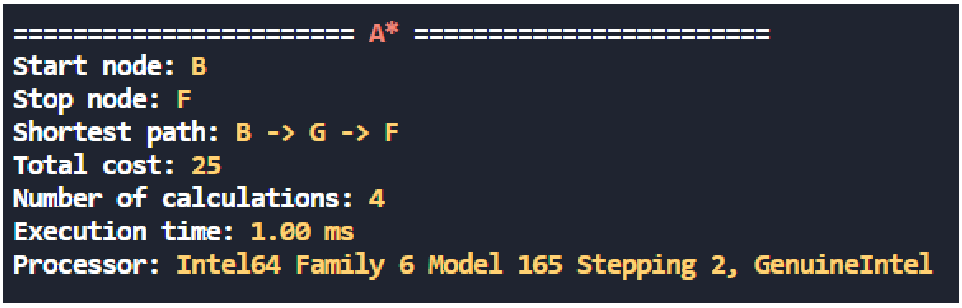
output UCS



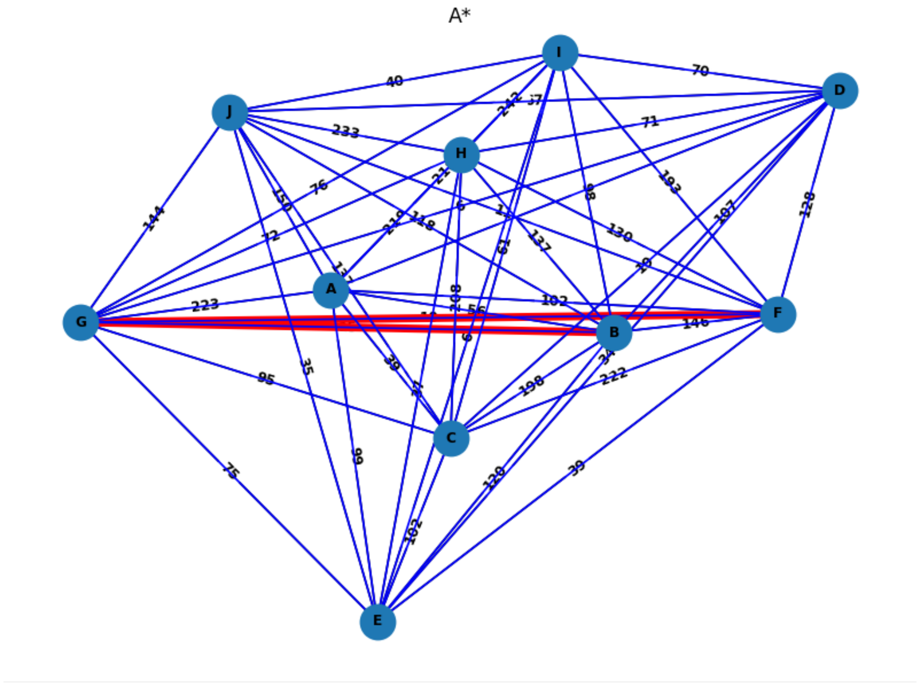
```
=============== UNIFORM COST SEARCH ===============
Start node: B
Stop node: F
Shortest path: B -> G -> F
Total cost: 25
Number of calculations: 4
Execution time: 0.00 ms
Processor: Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
```



output Astar



```
======================= A* ========================
Start node: B
Stop node: F
Shortest path: B -> G -> F
Total cost: 25
Number of calculations: 4
Execution time: 1.00 ms
Processor: Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
```

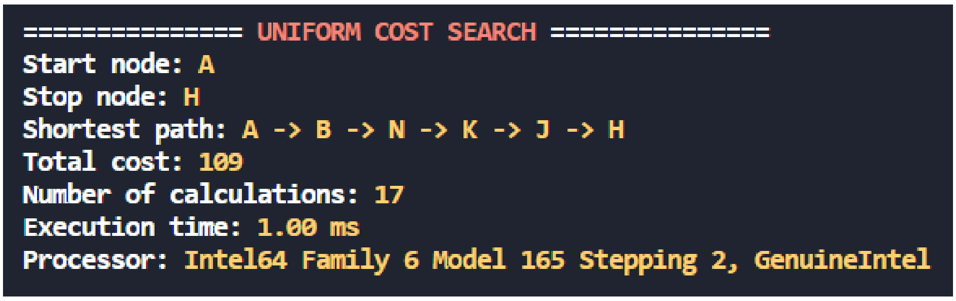


b. graph2.txt

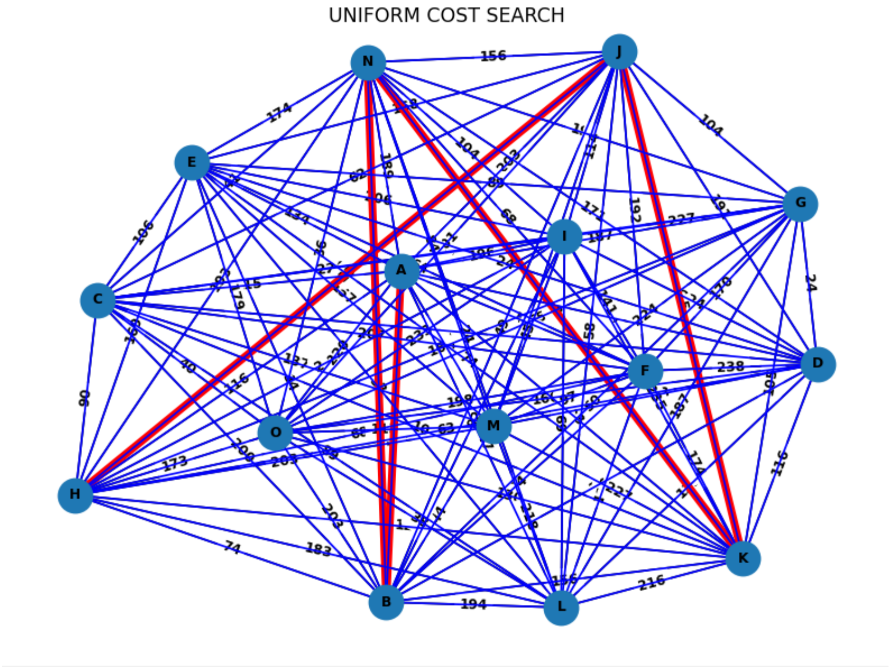
input

```
A   B   C   D   E   F   G   H   I   J   K   L   M   N   O
  0  53 115  76 134  51 187 116 196 203 133 147 104 189 220
 53   0 200 192  34 164  50  74  92  49 156 194  44  30 203
115 200   0  31 106 207  47  90 224  62  10  89 137  41  40
 76 192  31   0  24 238  24  63 224 191 116 131 223 171 160
134  34 106  24   0  64  89 169 206 168  74  53 157 174 179
 51 164 207 238  64   0 170  68 141 192 174 113  97  68 198
187  50  47  24  89 170   0  16 227 104 105 187 224 196  65
116  74  90  63 169  68  16   0   2  13 125 183 203 193 173
196  92 224 224 206 141 227   2   0 114 235  69  45 104 233
203  49  62 191 168 192 104  13 114   0   5  58  66 156 101
133 156  10 116  74 174 105 125 235   5   0 216 227   8 130
147 194  89 131  53 113 187 183  69  58 216   0 218  24  38
104  44 137 223 157  97 224 203  45  66 227 218   0  84 113
189  30  41 171 174  68 196 193 104 156   8  24  84   0  36
220 203  40 160 179 198  65 173 233 101 130  38 113  36   0
```

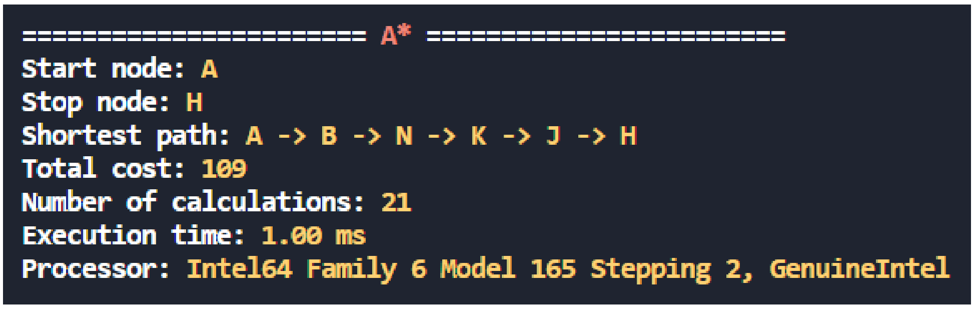
output UCS



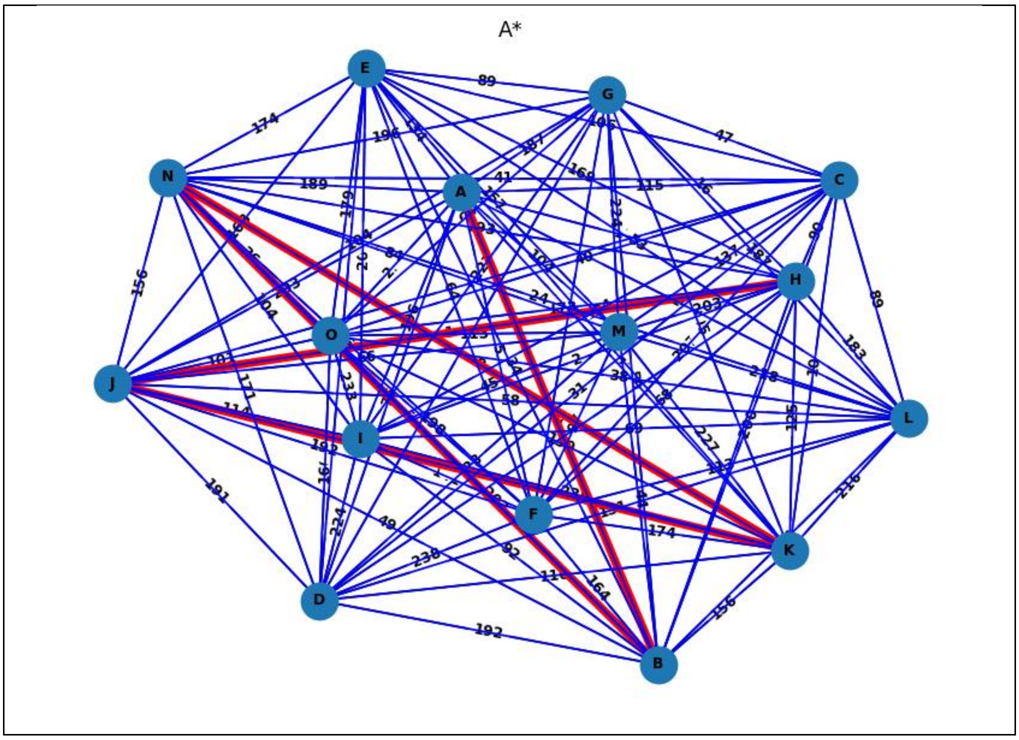
```
=============== UNIFORM COST SEARCH ===============
Start node: A
Stop node: H
Shortest path: A -> B -> N -> K -> J -> H
Total cost: 109
Number of calculations: 17
Execution time: 1.00 ms
Processor: Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
```



output Astar



```
======================= A* =======================
Start node: A
Stop node: H
Shortest path: A -> B -> N -> K -> J -> H
Total cost: 109
Number of calculations: 21
Execution time: 1.00 ms
Processor: Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
```

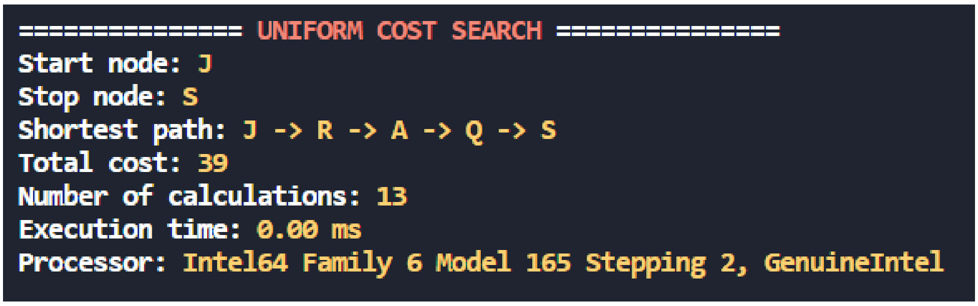


c. graph3.txt

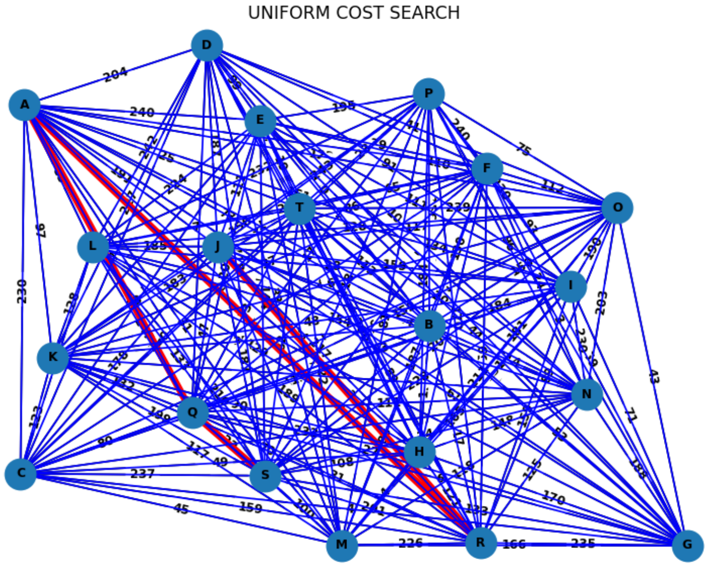
input

```
A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q   R   S   T
 35  73 230 204 240 201   2  27 161 191  97  85  31  32 123   0   4  10  84 225
 73  85  93 130 222 130  53  11 184  82  35  79  88 214 150 156  45 217  35 113
230  93  35  37 127   2  45  49 113 178 123 233  45  46  93  27  80 159 237 190
204 130  37  70  99 134  76  17  91 187 227 242  33 108  41   0  12   8 118 212
240 222 127  99  35 239  44 238 111 111 119 224  55 158 110 195 202  78 144  19
201 130   2 134 239 214 159 169  93  86  37 220 187  24 112 240  37 159 107  65
  2  53  45  76  44 159 243 170  71  61  27  29 166 188  43   3   6 235 133  60
 27  11  49  17 238 169 170 206 131 217 190 129 214 148 181 184 233 121 108 200
161 184 113  91 111  93  71 131 126 159  48  28 165 230 190 230  53 159 228 134
191  82 178 187 111  86  61 217 159 168  18 185 235 124 117 243  47   3 187  91
 97  35 123 227 119  37  27 190  48  18 176 128 117 154  63 156 132  70 199 183
 85  79 233 242 224 220  29 129  28 185 128 105 210 154 128 232 247 189 133   7
 31  88  45  33  55 187 166 214 165 235 117 210 103 118 211  81 192 226 100  22
 32 214  46 108 158  24 188 148 230 124 154 154 118  10 203  96 111 125  24 205
123 150  93  41 110 112  43 181 190 117  63 128 211 203 128  75 159  89 202 239
  0 156  27   0 195 240   3 184 230 243 156 232  81  96  75  68  17 170 218  23
  4  45  80  12 202  37   6 233  53  47 132 247 192 111 159  17  18  31  22  73
 10 217 159   8  78 159 235 121 159   3  70 189 226 125  89 170  31  78 201  89
 84  35 237 118 144 107 133 108 228 187 199 133 100  24 202 218  22 201  11  82
225 113 190 212  19  65  60 200 134  91 183   7  22 205 239  23  73  89  82 158
```

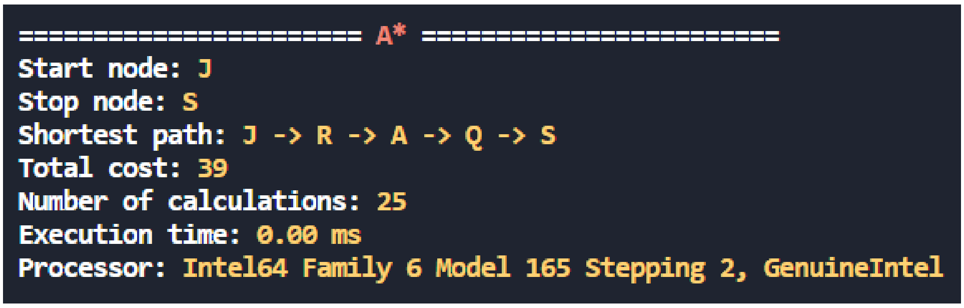
output UCS



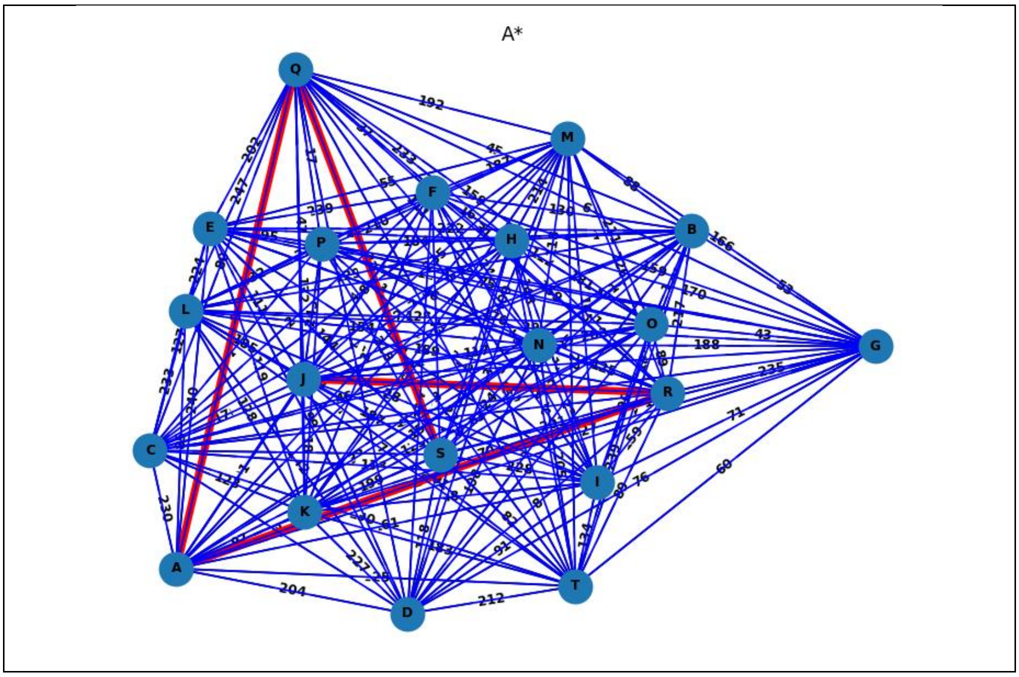
```
=============== UNIFORM COST SEARCH ================
Start node: J
Stop node: S
Shortest path: J -> R -> A -> Q -> S
Total cost: 39
Number of calculations: 13
Execution time: 0.00 ms
Processor: Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
```



output Astar



```
======================= A* ========================
Start node: J
Stop node: S
Shortest path: J -> R -> A -> Q -> S
Total cost: 39
Number of calculations: 25
Execution time: 0.00 ms
Processor: Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
```

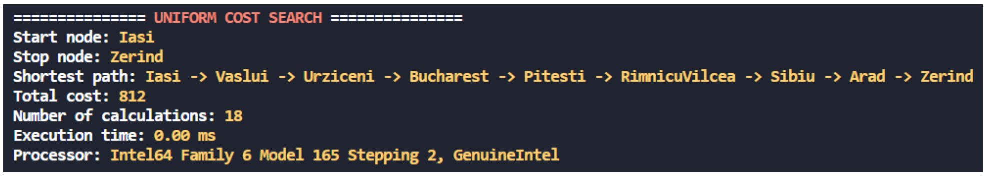


d. graph4.txt

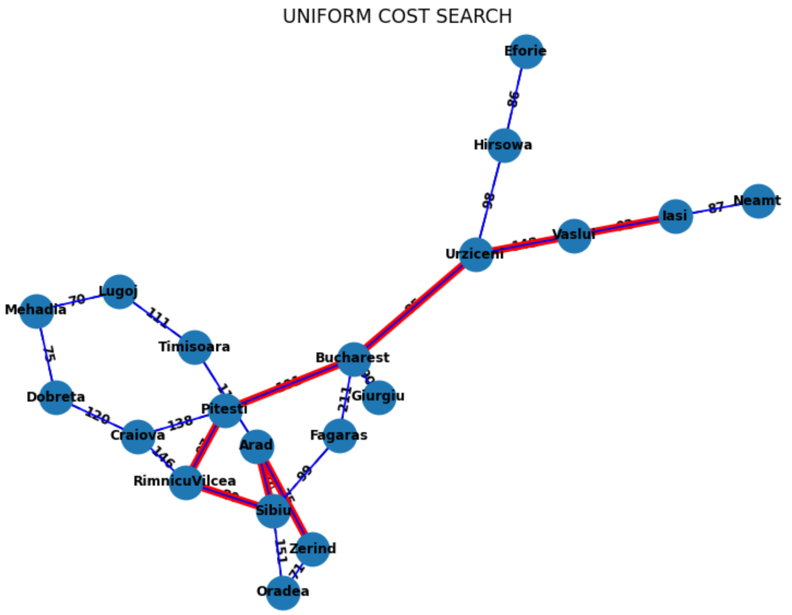
input

```
Arad   Bucharest   Craiova   Dobreta   Eforie   Fagaras   Giurgiu   Hirsowa
Iasi   Lugoj   Mehadia   Neamt   Oradea   Pitesti   RimnicuVilcea   Sibiu
Timisoara   Urziceni   Vaslui   Zerind
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 140 118   0   0  75
  0   0   0   0   0 211  90   0   0   0   0   0   0 101   0   0   0  85   0   0
  0   0   0 120   0   0   0   0   0   0   0   0   0 138 146   0   0   0   0   0
  0   0 120   0   0   0   0   0   0   0  75   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0  86   0   0   0   0   0   0   0   0   0   0   0   0
  0 211   0   0   0   0   0   0   0   0   0   0   0   0   0  99   0   0   0   0
  0  90   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0  86   0   0   0   0   0   0   0   0   0   0   0   0  98   0   0
  0   0   0   0   0   0   0   0   0   0   0  87   0   0   0   0   0   0  92   0
  0   0   0   0   0   0   0   0   0   0  70   0   0   0   0   0 111   0   0   0
  0   0   0  75   0   0   0   0   0  70   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0  87   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 151   0   0   0  71
  0 101 138   0   0   0   0   0   0   0   0   0   0   0  97   0   0   0   0   0
  0   0 146   0   0   0   0   0   0   0   0   0   0  97   0  80   0   0   0   0
140   0   0   0   0  99   0   0   0   0   0   0 151   0  80   0   0   0   0   0
118   0   0   0   0   0   0   0   0 111   0   0   0   0   0   0   0   0   0   0
  0  85   0   0   0   0   0  98   0   0   0   0   0   0   0   0   0   0 142   0
  0   0   0   0   0   0   0   0  92   0   0   0   0   0   0   0   0 142   0   0
 75   0   0   0   0   0   0   0   0   0   0   0  71   0   0   0   0   0   0   0
```

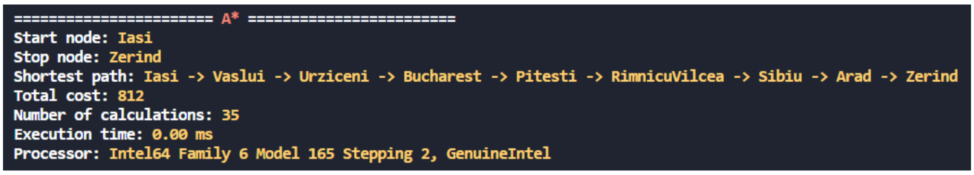
output UCS



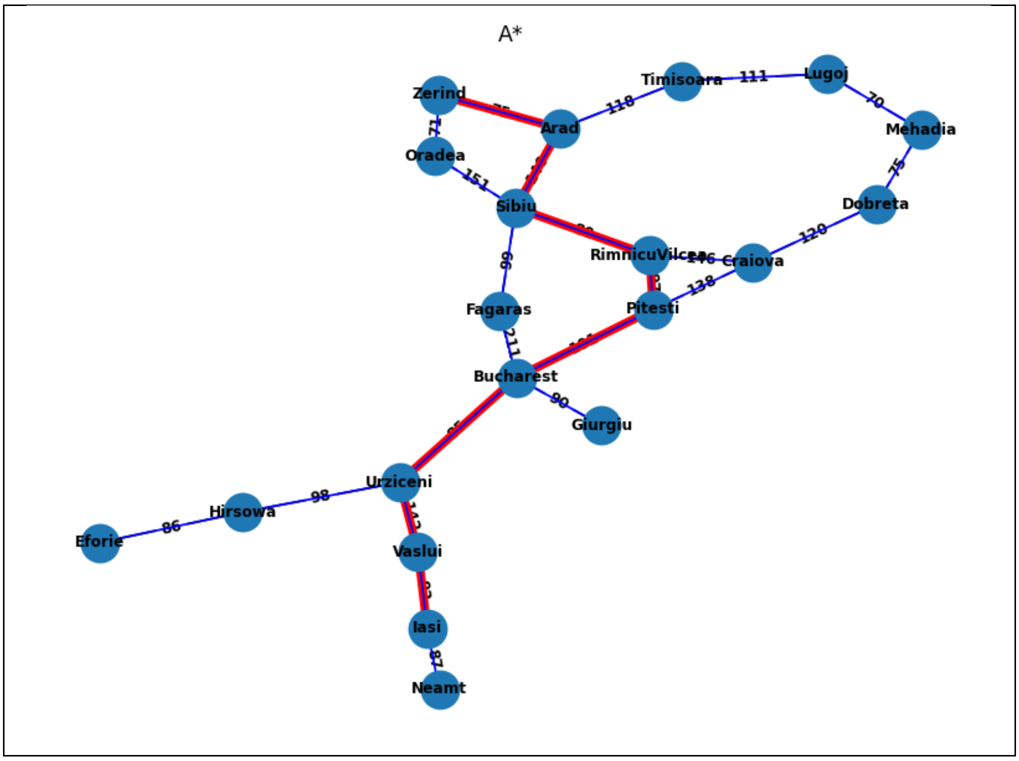
```
=============== UNIFORM COST SEARCH ===============
Start node: Iasi
Stop node: Zerind
Shortest path: Iasi -> Vaslui -> Urziceni -> Bucharest -> Pitesti -> RimnicuVilcea -> Sibiu -> Arad -> Zerind
Total cost: 812
Number of calculations: 18
Execution time: 0.00 ms
Processor: Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
```



output Astar



```
====================== A* =======================
Start node: Iasi
Stop node: Zerind
Shortest path: Iasi -> Vaslui -> Urziceni -> Bucharest -> Pitesti -> RimnicuVilcea -> Sibiu -> Arad -> Zerind
Total cost: 812
Number of calculations: 35
Execution time: 0.00 ms
Processor: Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
```

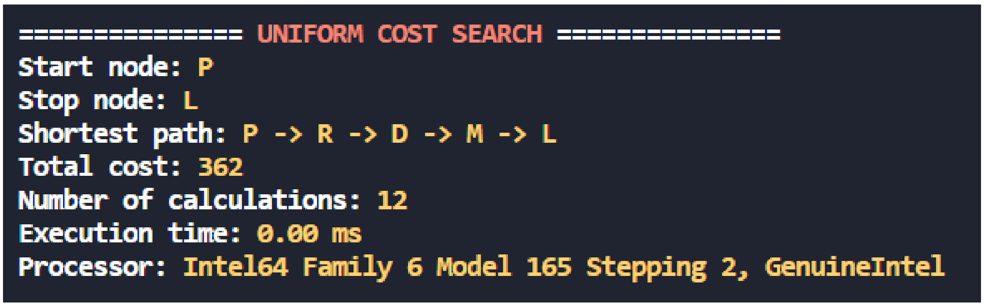


e. graph5.txt

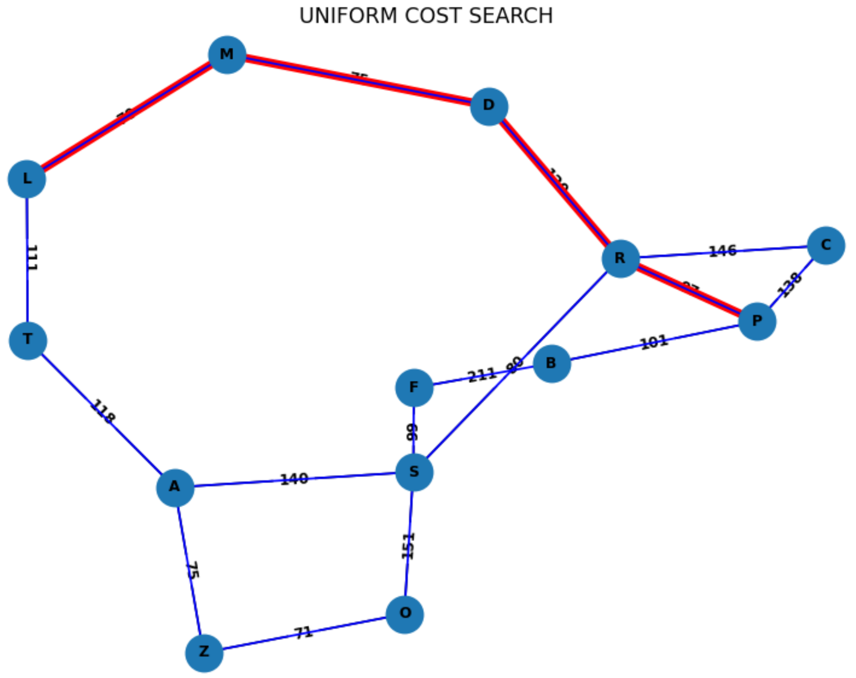
input

```
A    Z    T    O    L    M    S    D    R    F    C    P    B
   0   75  118    0    0    0  140    0    0    0    0    0    0
  75    0    0   71    0    0    0    0    0    0    0    0    0
 118    0    0    0  111    0    0    0    0    0    0    0    0
   0   71    0    0    0    0  151    0    0    0    0    0    0
   0    0  111    0    0   70    0    0    0    0    0    0    0
   0    0    0    0   70    0    0   75    0    0    0    0    0
 140    0    0  151    0    0    0    0   80   99    0    0    0
   0    0    0    0    0   75    0    0  120    0    0    0    0
   0    0    0    0    0    0   80  120    0    0  146   97    0
   0    0    0    0    0    0   99    0    0    0    0    0  211
   0    0    0    0    0    0    0    0  146    0    0  138    0
   0    0    0    0    0    0    0    0   97    0  138    0  101
   0    0    0    0    0    0    0    0    0  211    0  101    0
```

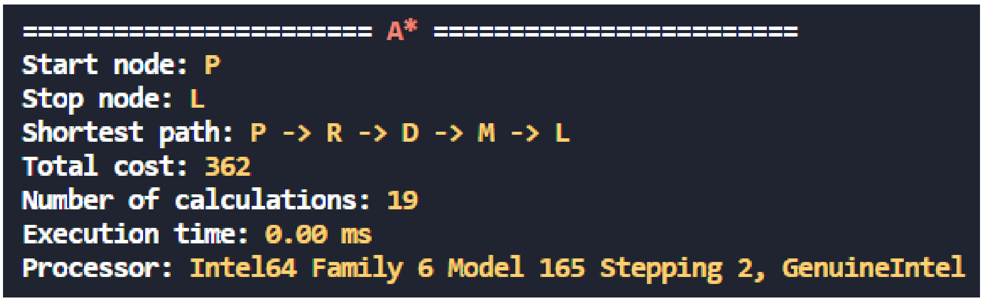
output UCS



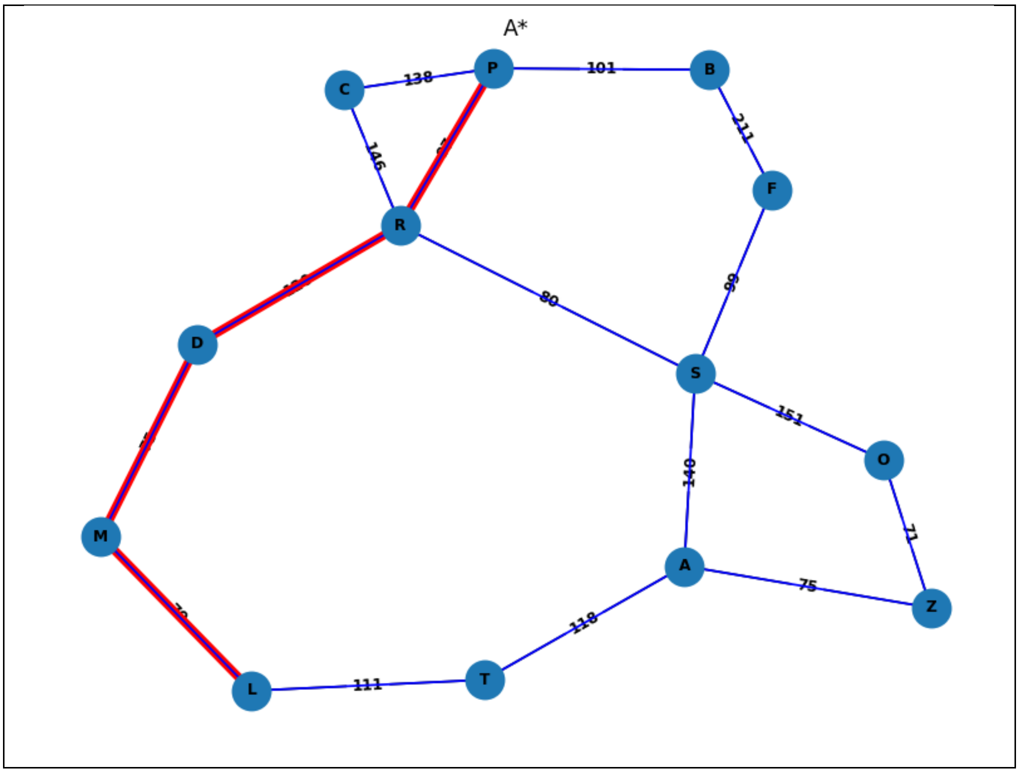
```
=============== UNIFORM COST SEARCH ===============
Start node: P
Stop node: L
Shortest path: P -> R -> D -> M -> L
Total cost: 362
Number of calculations: 12
Execution time: 0.00 ms
Processor: Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
```



output Astar

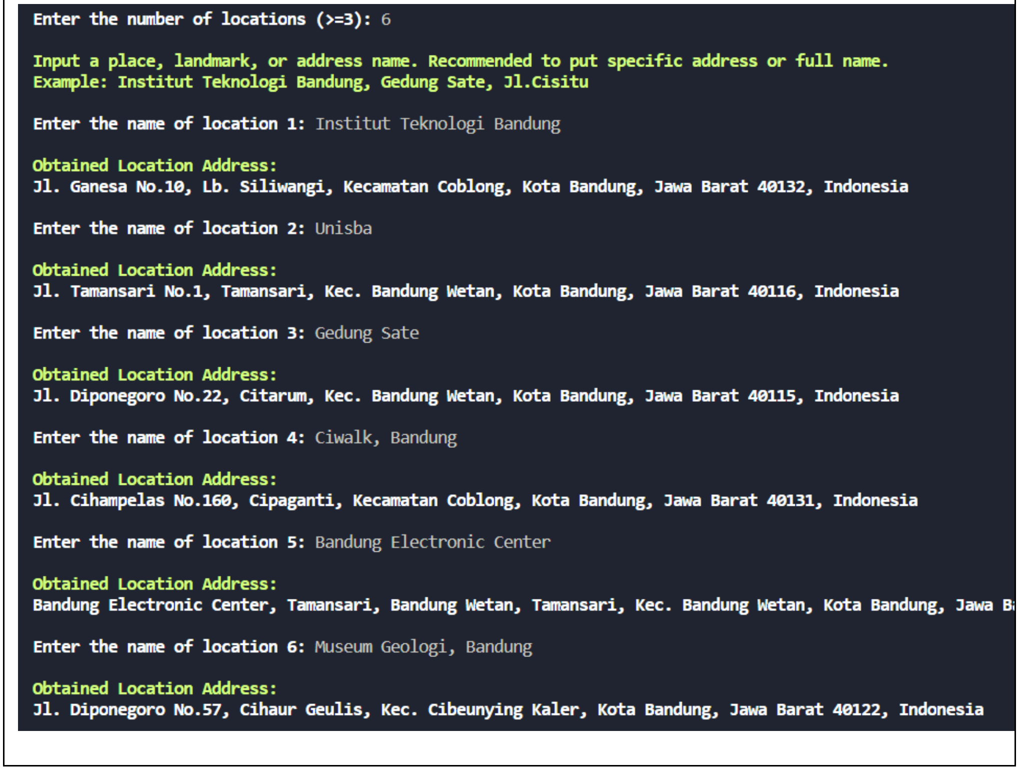


```
======================= A* ========================
Start node: P
Stop node: L
Shortest path: P -> R -> D -> M -> L
Total cost: 362
Number of calculations: 19
Execution time: 0.00 ms
Processor: Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
```

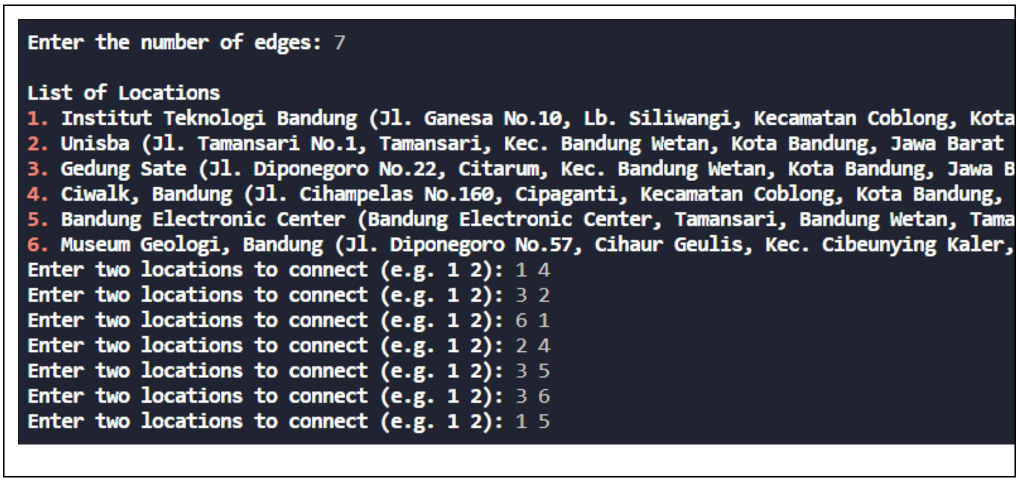


f. Google Map

input

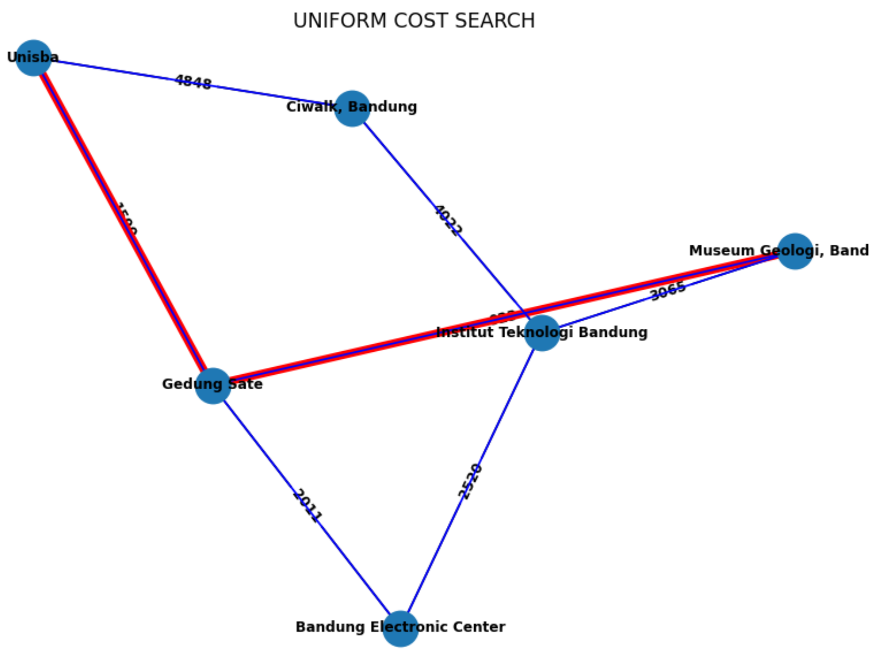


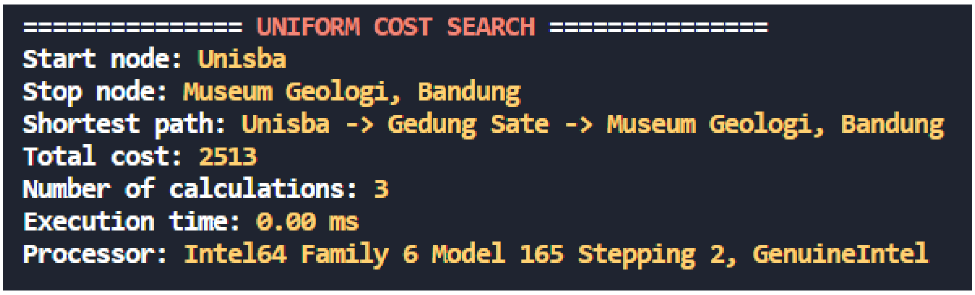
```
Enter the number of locations (>=3): 6

Input a place, landmark, or address name. Recommended to put specific address or full name.
Example: Institut Teknologi Bandung, Gedung Sate, Jl.Cisitu

Enter the name of location 1: Institut Teknologi Bandung

Obtained Location Address:
Jl. Ganesa No.10, Lb. Siliwangi, Kecamatan Coblong, Kota Bandung, Jawa Barat 40132, Indonesia

Enter the name of location 2: Unisba

Obtained Location Address:
Jl. Tamansari No.1, Tamansari, Kec. Bandung Wetan, Kota Bandung, Jawa Barat 40116, Indonesia

Enter the name of location 3: Gedung Sate

Obtained Location Address:
Jl. Diponegoro No.22, Citarum, Kec. Bandung Wetan, Kota Bandung, Jawa Barat 40115, Indonesia

Enter the name of location 4: Ciwalk, Bandung

Obtained Location Address:
Jl. Cihampelas No.160, Cipaganti, Kecamatan Coblong, Kota Bandung, Jawa Barat 40131, Indonesia

Enter the name of location 5: Bandung Electronic Center

Obtained Location Address:
Bandung Electronic Center, Tamansari, Bandung Wetan, Tamansari, Kec. Bandung Wetan, Kota Bandung, Jawa Ba

Enter the name of location 6: Museum Geologi, Bandung

Obtained Location Address:
Jl. Diponegoro No.57, Cihaur Geulis, Kec. Cibeunying Kaler, Kota Bandung, Jawa Barat 40122, Indonesia
```



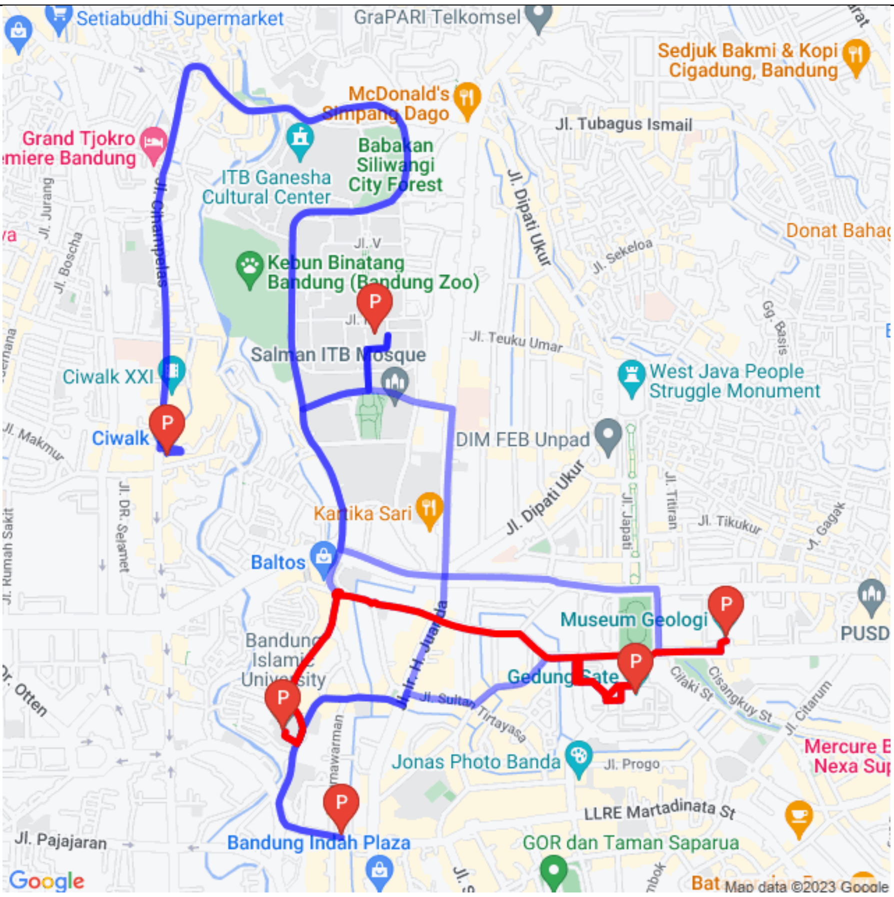
```
Enter the number of edges: 7

List of Locations
1. Institut Teknologi Bandung (Jl. Ganesa No.10, Lb. Siliwangi, Kecamatan Coblong, Kota
2. Unisba (Jl. Tamansari No.1, Tamansari, Kec. Bandung Wetan, Kota Bandung, Jawa Barat
3. Gedung Sate (Jl. Diponegoro No.22, Citarum, Kec. Bandung Wetan, Kota Bandung, Jawa B
4. Ciwalk, Bandung (Jl. Cihampelas No.160, Cipaganti, Kecamatan Coblong, Kota Bandung,
5. Bandung Electronic Center (Bandung Electronic Center, Tamansari, Bandung Wetan, Tama
6. Museum Geologi, Bandung (Jl. Diponegoro No.57, Cihaur Geulis, Kec. Cibeunying Kaler,
Enter two locations to connect (e.g. 1 2): 1 4
Enter two locations to connect (e.g. 1 2): 3 2
Enter two locations to connect (e.g. 1 2): 6 1
Enter two locations to connect (e.g. 1 2): 2 4
Enter two locations to connect (e.g. 1 2): 3 5
Enter two locations to connect (e.g. 1 2): 3 6
Enter two locations to connect (e.g. 1 2): 1 5
```

output UCS

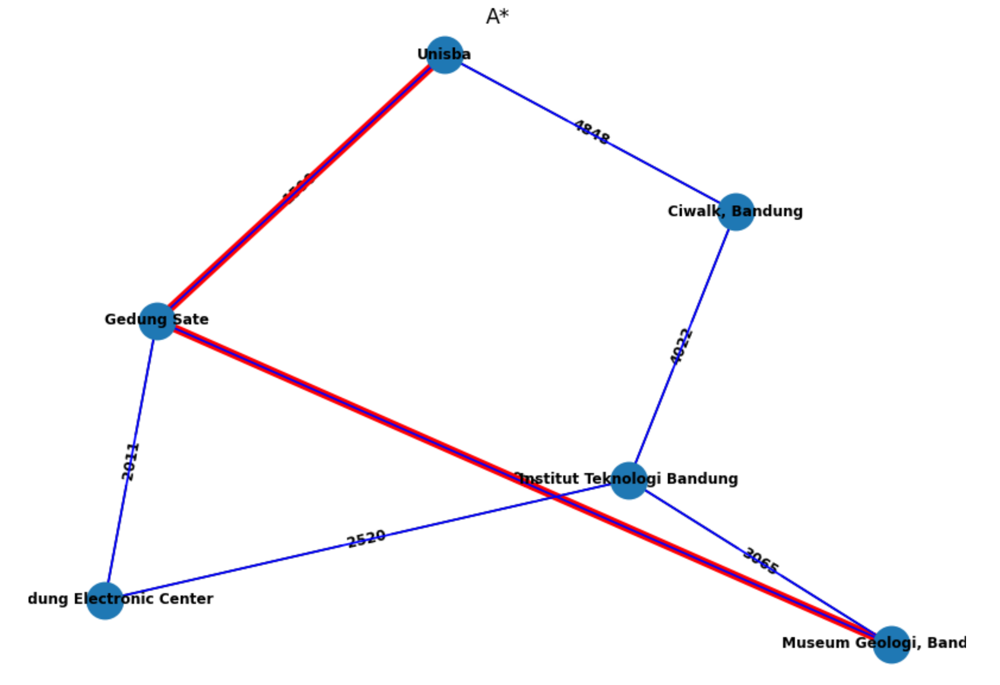


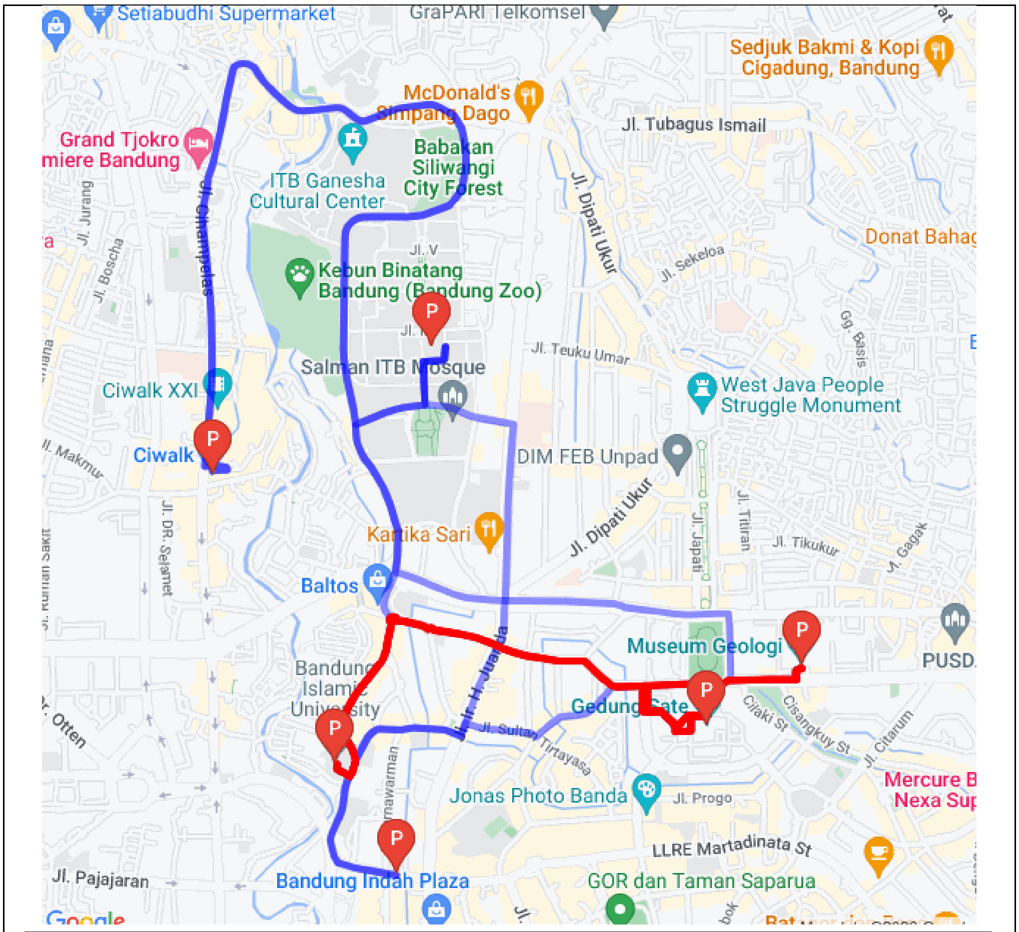


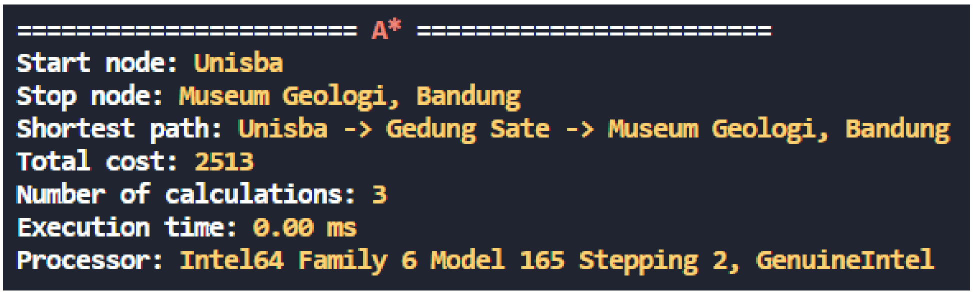
```
=============== UNIFORM COST SEARCH ===============
Start node: Unisba
Stop node: Museum Geologi, Bandung
Shortest path: Unisba -> Gedung Sate -> Museum Geologi, Bandung
Total cost: 2513
Number of calculations: 3
Execution time: 0.00 ms
Processor: Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
```



output Astar

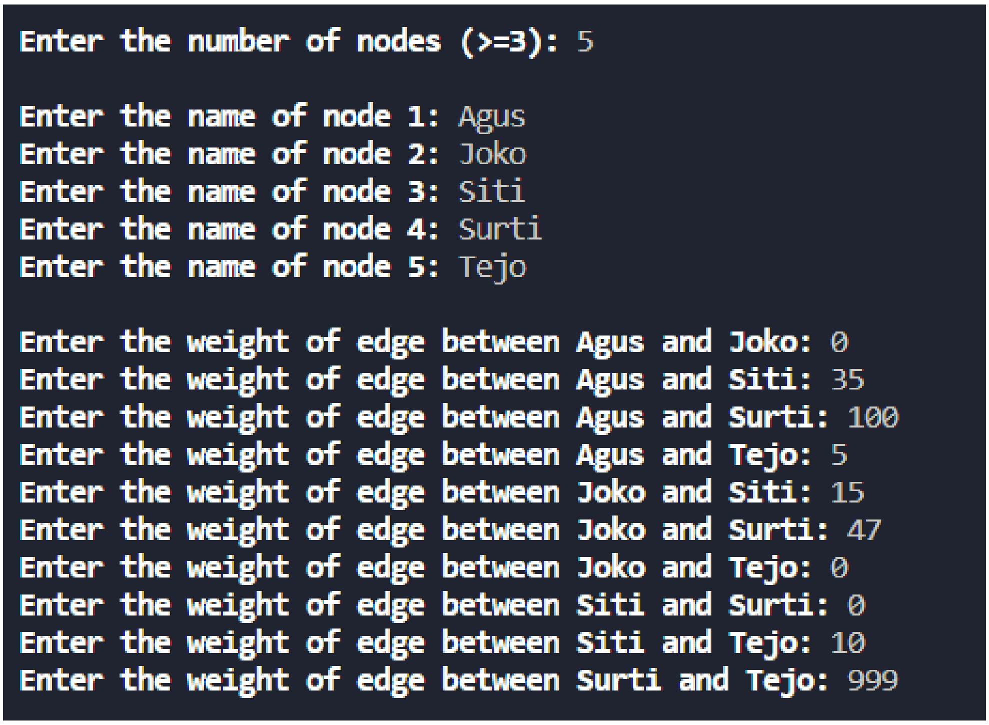





```
======================= A* ========================
Start node: Unisba
Stop node: Museum Geologi, Bandung
Shortest path: Unisba -> Gedung Sate -> Museum Geologi, Bandung
Total cost: 2513
Number of calculations: 3
Execution time: 0.00 ms
Processor: Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
```

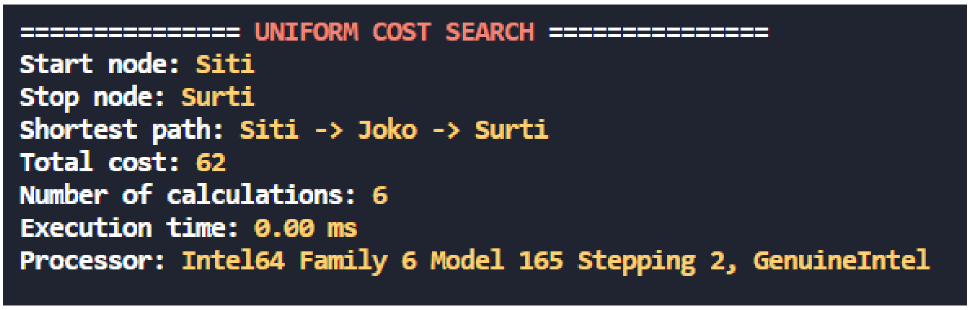
g. Manual Input

input



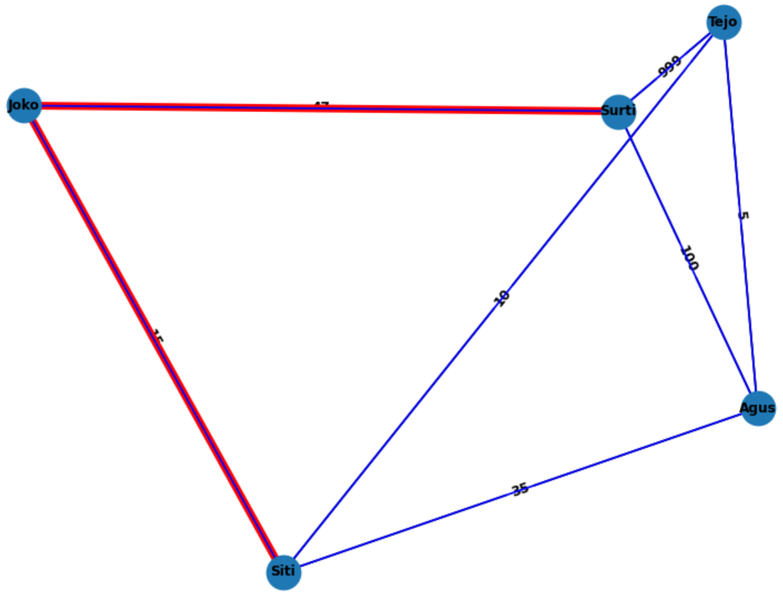
```
Enter the number of nodes (>=3): 5

Enter the name of node 1: Agus
Enter the name of node 2: Joko
Enter the name of node 3: Siti
Enter the name of node 4: Surti
Enter the name of node 5: Tejo

Enter the weight of edge between Agus and Joko: 0
Enter the weight of edge between Agus and Siti: 35
Enter the weight of edge between Agus and Surti: 100
Enter the weight of edge between Agus and Tejo: 5
Enter the weight of edge between Joko and Siti: 15
Enter the weight of edge between Joko and Surti: 47
Enter the weight of edge between Joko and Tejo: 0
Enter the weight of edge between Siti and Surti: 0
Enter the weight of edge between Siti and Tejo: 10
Enter the weight of edge between Surti and Tejo: 999
```

output UCS

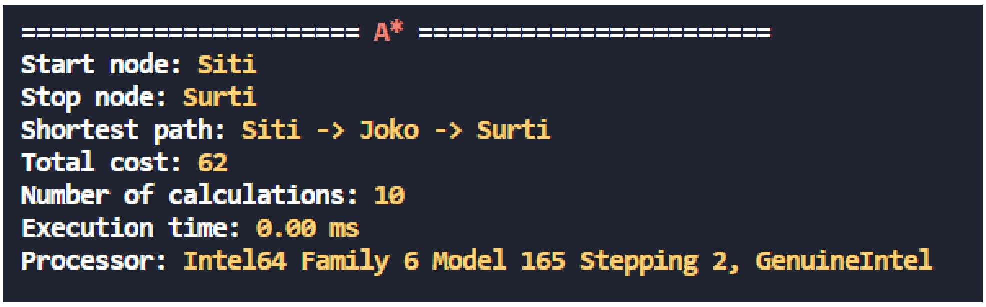


```
=============== UNIFORM COST SEARCH ===============
Start node: Siti
Stop node: Surti
Shortest path: Siti -> Joko -> Surti
Total cost: 62
Number of calculations: 6
Execution time: 0.00 ms
Processor: Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
```

UNIFORM COST SEARCH

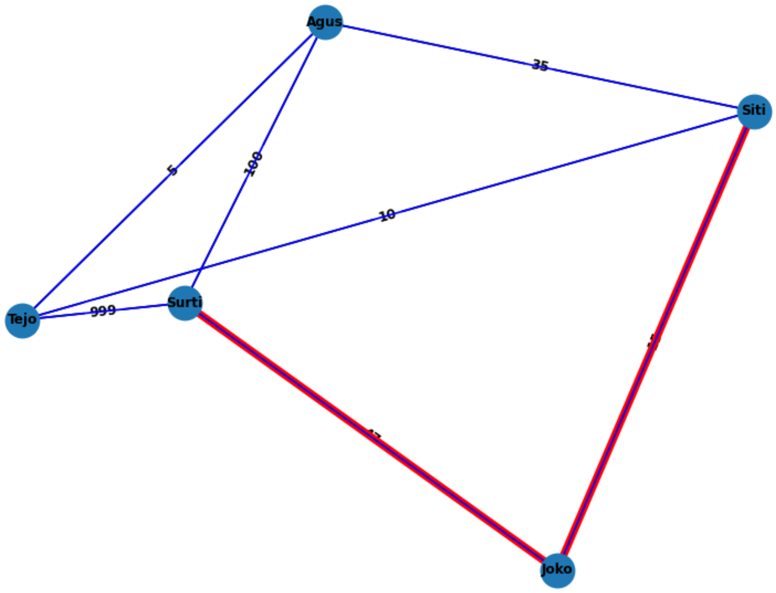


output Astar



```
======================== A* =========================
Start node: Siti
Stop node: Surti
Shortest path: Siti -> Joko -> Surti
Total cost: 62
Number of calculations: 10
Execution time: 0.00 ms
Processor: Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
```

A*



G. Kesimpulan

Pada tugas berikut, implementasi algoritma UCS dan A* secara langsung menyebabkan pemahaman akan algoritma tersebut menjadi lebih dalam. Seperti tugas Strategi Algoritma lainnya, tugas berikut menantang dan mengasah kemampuan logika dan pemecahan masalah. Namun pada tugas ini terdapat kebutuhan untuk eksplorasi salah satu penyedia layanan digital terbesar, Google. Bonus dalam tugas ini menghadirkan tuntutan untuk mengakses dan menggunakan API Google Maps yang pada prosesnya menambah pengalaman dan pengetahuan akan API komersil. Pada akhirnya pengerjaan tugas mengembangkan kemampuan dan pengetahuan kami sebagai programmer dan diharapkan hasilnya dapat memenuhi ekspektasi spesifikasi yang diberikan.

H. *Link to Repository*

https://github.com/ghaziakmalf/Tucil3_13521050_13521058

I. *Check List Table*

Poin	Ya	Tidak
1. Program dapat menerima input graf	✓	
2. Program dapat menghitung lintasan terpendek dengan UCS	✓	
3. Program dapat menghitung lintasan terpendek dengan A*	✓	
4. Program dapat menampilkan lintasan terpendek serta jaraknya	✓	
5. Bonus: Program dapat menerima input peta dengan Google Map API dan menampilkan peta serta lintasan terpendek pada peta	✓	