A Lab Project Report submitted

for Artificial Intelligence (UCS-411)

by

Dhruv Bansal	**102103753**
Shivam Khurana	**102103754**
Rohan Thakur	**102103762**
Prabhmeet Kaur	**102103785**

2COE27

Submitted to:

Dr Anu Bajaj

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY (A DEEMED TO BE UNIVERSITY), PATIALA, PUNJAB

INDIA

Jan-May 2023

CODE: https://github.com/rohanthakur336/MC_FPA-AI-.git

PROBLEM STATEMENT: AI-based project to find the quickest route.

INTRODUCTION: This AI-based project will use the below-mentioned algorithms to find the fastest route between two locations. This type of system could be helpful for drivers, delivery companies, and other entities that need to determine the fastest way to get from one place to another.

The way it works is: It will be taking input from the user about the source and destination and will be using different algorithms (while keeping in mind the traffic condition and speed limit of each path) to predict the path that will be quickest and most efficient for the user to reach the destination.

LIBRARIES IMPORTED:

Copy: The copy() method creates a shallow copy of an object, which means it creates a new object with the same content, but the content itself is not copied. The deepcopy() method creates a deep copy of an object. This means it creates a new object with the same content and copies all the content within it.

```python
import copy
```

Random: In Python, the "random" library is a built-in module that provides functions for generating pseudorandom numbers. The randint() Python function returns a randomly generated integer from the specified range with both inclusive parameters.

```python
import random
```

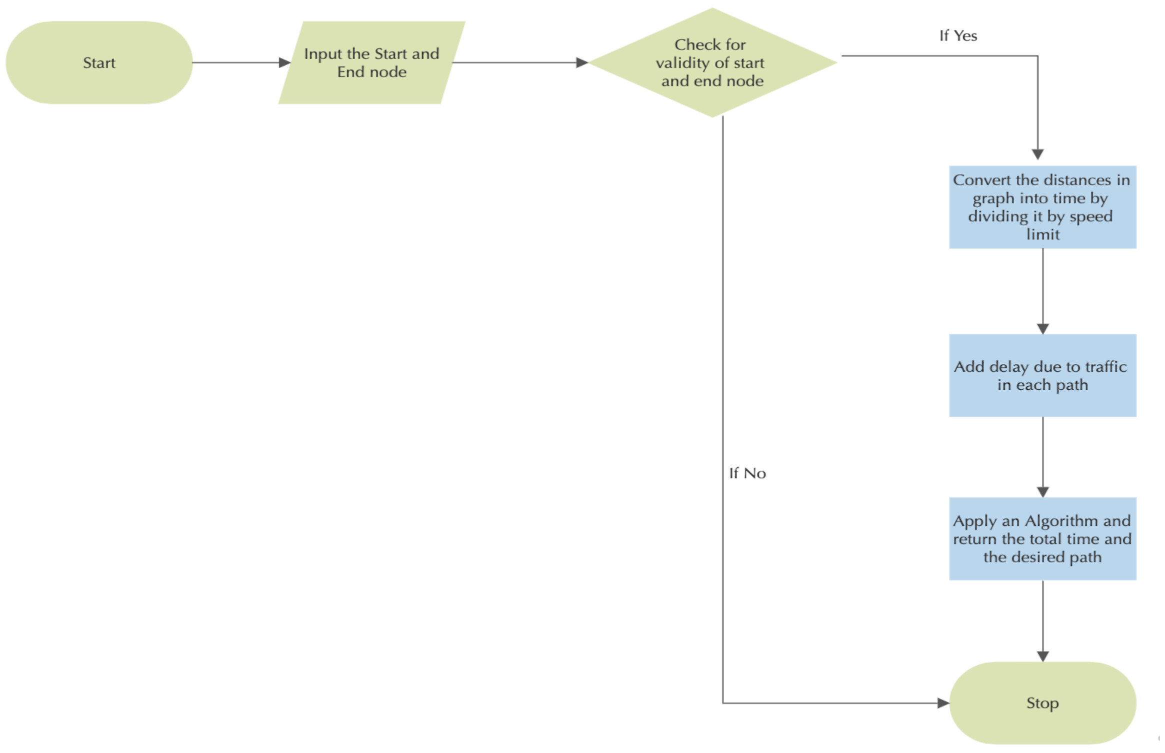
Math: Math is a built-in module in the Python 3 standard library that provides standard mathematical constants and functions.

```python
import math
```

Queue: In Python, the queue module provides a queue data structure that allows items to be added to the end of the queue and removed from the front of the queue in a first-in, first-out (FIFO) order. Priority Queue is a queue where items are assigned a priority and removed from the queue in order of priority.

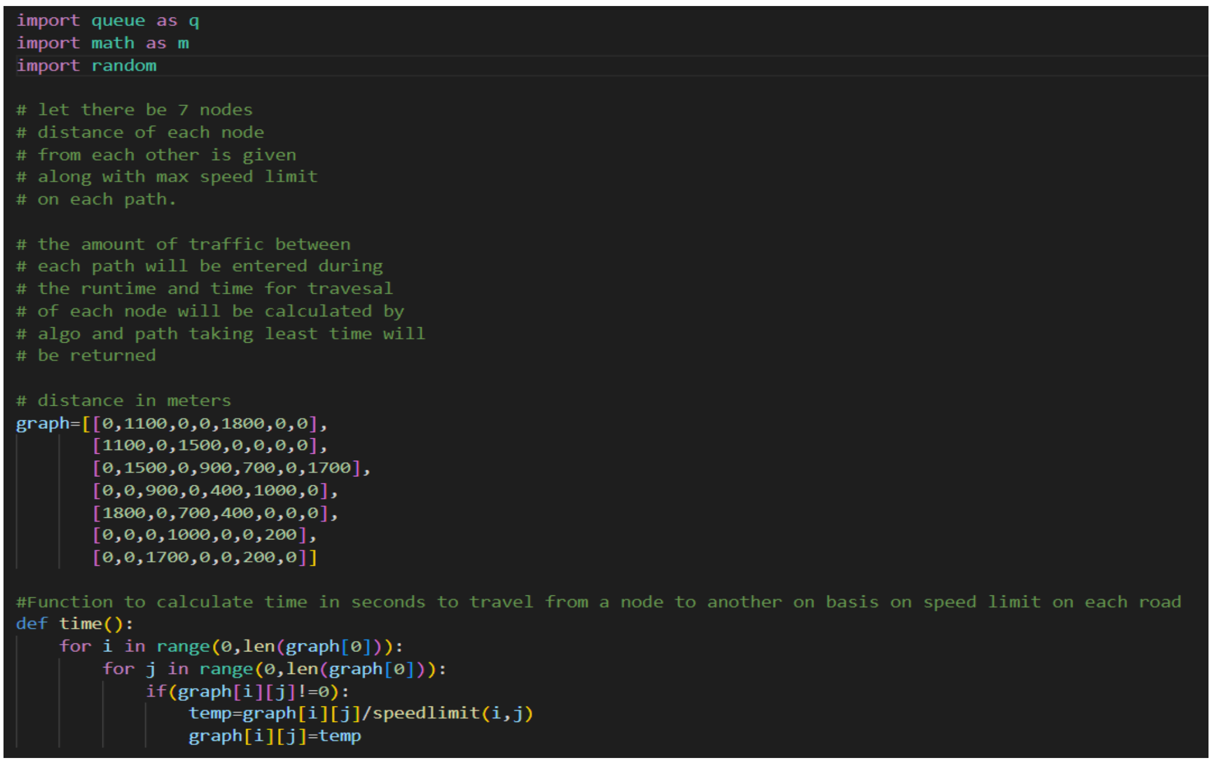
```python
import queue as q
```

FLOW CHART:


Start
Input the Start and End node
Check for validity of start and end node
If Yes
If No
Convert the distances in graph into time by dividing it by speed limit
Add delay due to traffic in each path
Apply an Algorithm and return the total time and the desired path
Stop

ALGORITHMS:

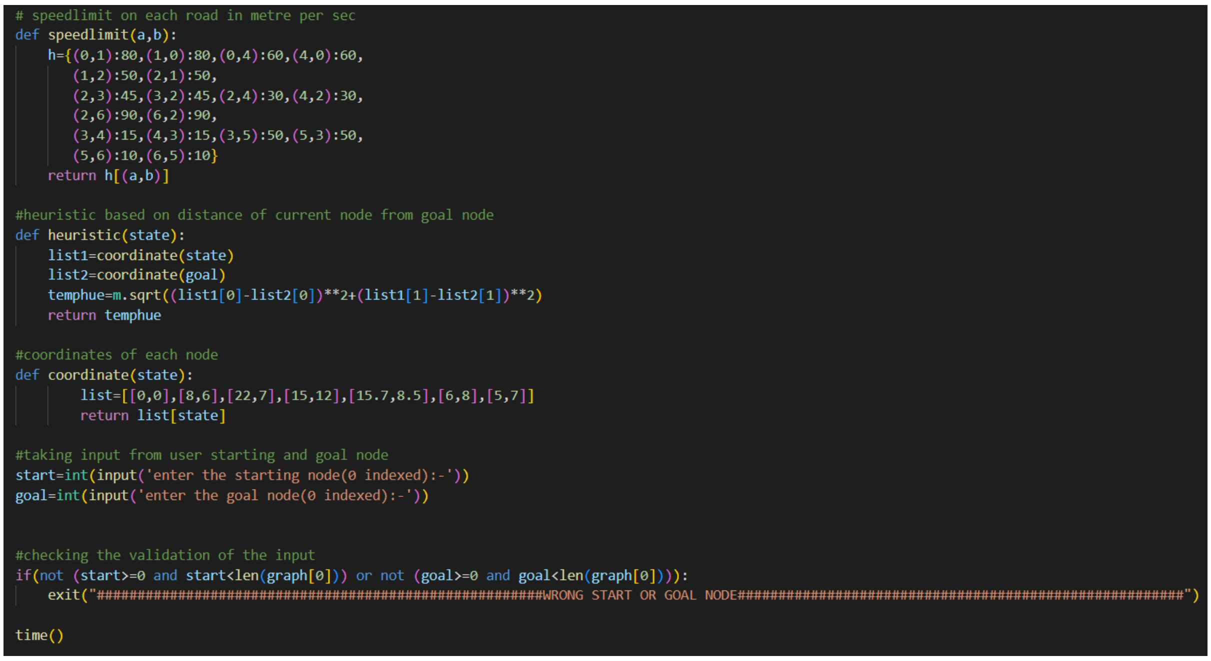
- **BEST FIRST SEARCH(BFS):**



```
import queue as q
import math as m
import random

# let there be 7 nodes
# distance of each node
# from each other is given
# along with max speed limit
# on each path.

# the amount of traffic between
# each path will be entered during
# the runtime and time for travesal
# of each node will be calculated by
# algo and path taking least time will
# be returned

# distance in meters
graph=[[0,1100,0,0,1800,0,0],
       [1100,0,1500,0,0,0,0],
       [0,1500,0,900,700,0,1700],
       [0,0,900,0,400,1000,0],
       [1800,0,700,400,0,0,0],
       [0,0,0,1000,0,0,200],
       [0,0,1700,0,0,200,0]]

#Function to calculate time in seconds to travel from a node to another on basis on speed limit on each road
def time():
    for i in range(0,len(graph[0])):
        for j in range(0,len(graph[0])):
            if(graph[i][j]!=0):
                temp=graph[i][j]/speedlimit(i,j)
                graph[i][j]=temp
```

First of all, we imported three libraries required for the functioning of code that are *queue*, *math and random*. Then we initialized the graph, which contains distances between each node.

The *time()* function defined here alters the value in the graph by dividing each with its corresponding speed limit.



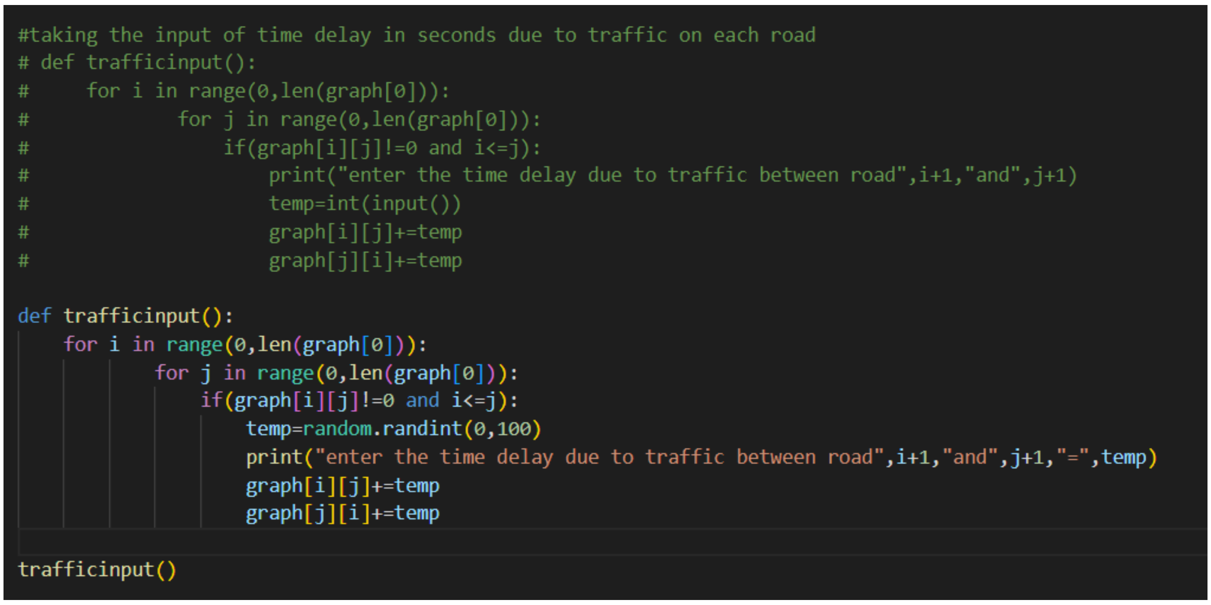
```python
# speedlimit on each road in metre per sec
def speedlimit(a,b):
    h={(0,1):80,(1,0):80,(0,4):60,(4,0):60,
       (1,2):50,(2,1):50,
       (2,3):45,(3,2):45,(2,4):30,(4,2):30,
       (2,6):90,(6,2):90,
       (3,4):15,(4,3):15,(3,5):50,(5,3):50,
       (5,6):10,(6,5):10}
    return h[(a,b)]

#heuristic based on distance of current node from goal node
def heuristic(state):
    list1=coordinate(state)
    list2=coordinate(goal)
    temphue=m.sqrt((list1[0]-list2[0])**2+(list1[1]-list2[1])**2)
    return temphue

#coordinates of each node
def coordinate(state):
        list=[[0,0],[8,6],[22,7],[15,12],[15.7,8.5],[6,8],[5,7]]
        return list[state]

#taking input from user starting and goal node
start=int(input('enter the starting node(0 indexed):-'))
goal=int(input('enter the goal node(0 indexed):-'))


#checking the validation of the input
if(not (start>=0 and start<len(graph[0])) or not (goal>=0 and goal<len(graph[0]))):
    exit("#######################################################WRONG START OR GOAL NODE#######################################################")

time()
```

The *speedlimit(a,b)* function returns the speed limit of the path between two nodes passed as argument to the function.

The *heuristic(state)* function returns the displacement between the current node and the goal node, which we use as heuristic value for our algorithm.

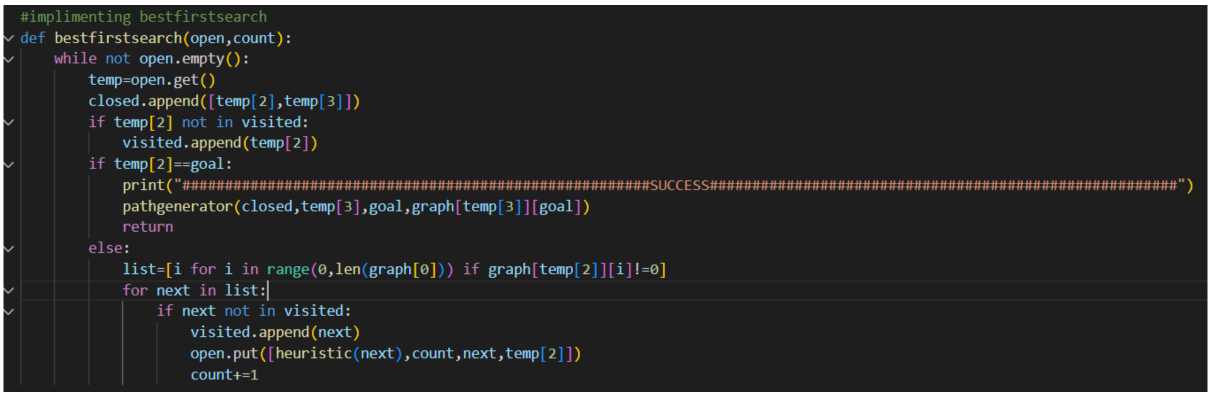
The *coordinate(state)* function returns the coordinate of the nodes passed to it as an argument.

The *start* variable takes the starting node as an input. And the *goal* variable takes the destination node as an input.



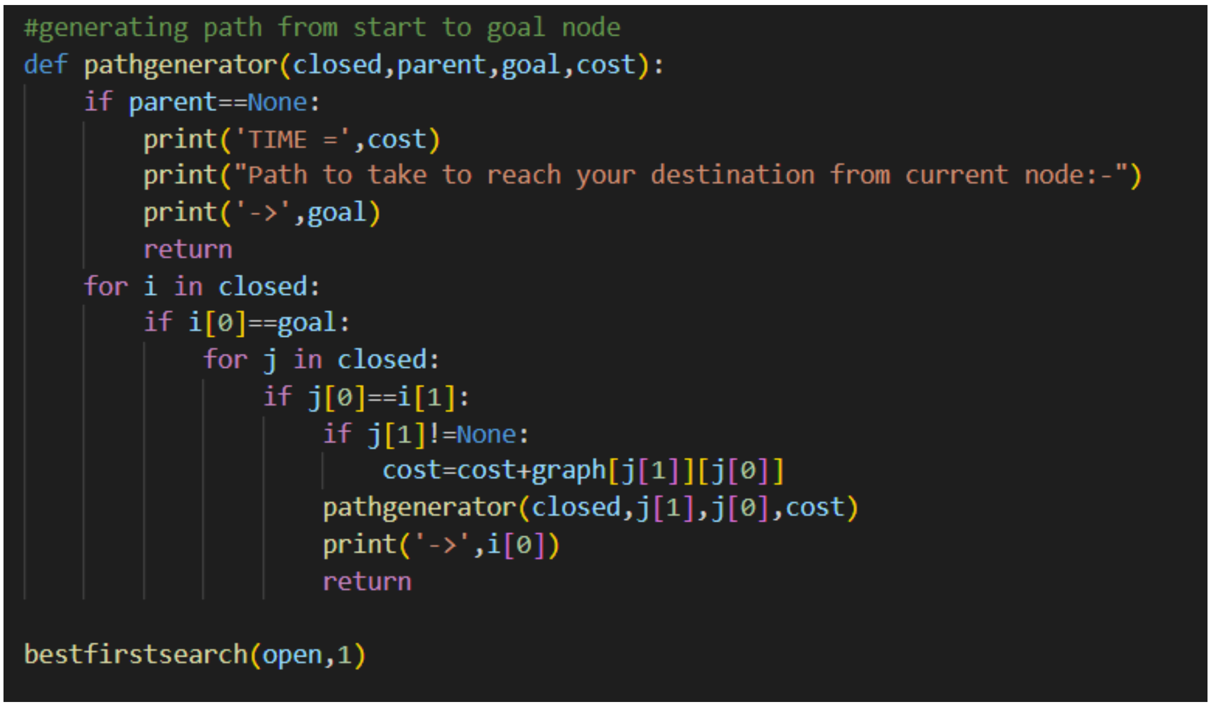
```python
#taking the input of time delay in seconds due to traffic on each road
# def trafficinput():
#     for i in range(0,len(graph[0])):
#         for j in range(0,len(graph[0])):
#             if(graph[i][j]!=0 and i<=j):
#                 print("enter the time delay due to traffic between road",i+1,"and",j+1)
#                 temp=int(input())
#                 graph[i][j]+=temp
#                 graph[j][i]+=temp

def trafficinput():
    for i in range(0,len(graph[0])):
        for j in range(0,len(graph[0])):
            if(graph[i][j]!=0 and i<=j):
                temp=random.randint(0,100)
                print("enter the time delay due to traffic between road",i+1,"and",j+1,"=",temp)
                graph[i][j]+=temp
                graph[j][i]+=temp

trafficinput()
```

The *trafficinput()* function generates random time delay values between each node. The time delay here is between the range of 0 to 100.



```python
#implimenting bestfirstsearch
def bestfirstsearch(open,count):
    while not open.empty():
        temp=open.get()
        closed.append([temp[2],temp[3]])
        if temp[2] not in visited:
            visited.append(temp[2])
        if temp[2]==goal:
            print("###########################################################SUCCESS##############################################################")
            pathgenerator(closed,temp[3],goal,graph[temp[3]][goal])
            return
        else:
            list=[i for i in range(0,len(graph[0])) if graph[temp[2]][i]!=0]
            for next in list:
                if next not in visited:
                    visited.append(next)
                    open.put([heuristic(next),count,next,temp[2]])
                    count+=1
```

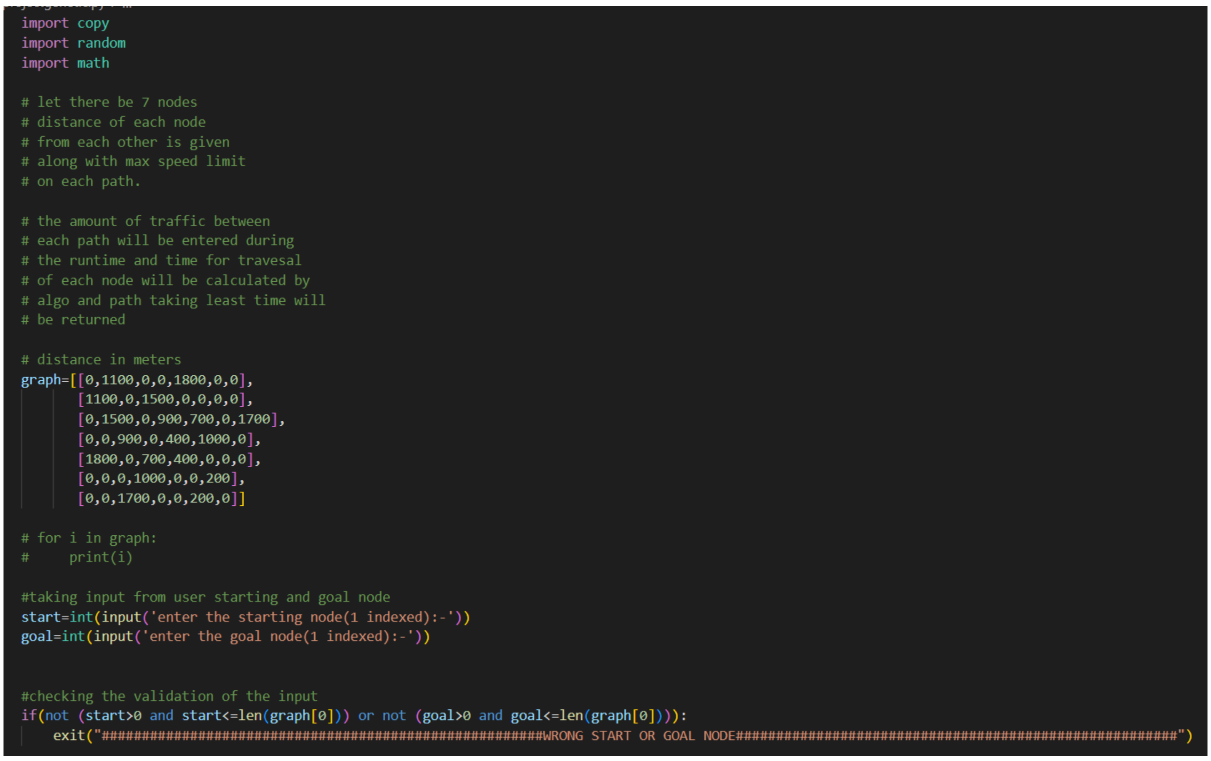
Best-first search is an informed search algorithm used in artificial intelligence and computer science to find the optimal path from a starting state to a goal state in a search space. The basic idea behind best-first search is to maintain a priority queue of nodes that have been visited but not yet expanded, with the most promising node at the front of the queue. The heuristic function estimates the distance from each node to the goal state, and the priority queue is sorted based on these estimates. And here, we are implementing it using *bestfirstsearch(open, count)* function.



```python
#generating path from start to goal node
def pathgenerator(closed,parent,goal,cost):
    if parent==None:
        print('TIME =',cost)
        print("Path to take to reach your destination from current node:-")
        print('->',goal)
        return
    for i in closed:
        if i[0]==goal:
            for j in closed:
                if j[0]==i[1]:
                    if j[1]!=None:
                        cost=cost+graph[j[1]][j[0]]
                    pathgenerator(closed,j[1],j[0],cost)
                    print('->',i[0])
                    return

bestfirstsearch(open,1)
```

Pathgenerator(closed,parent,goal,cost) function generates the path from source to destination node.

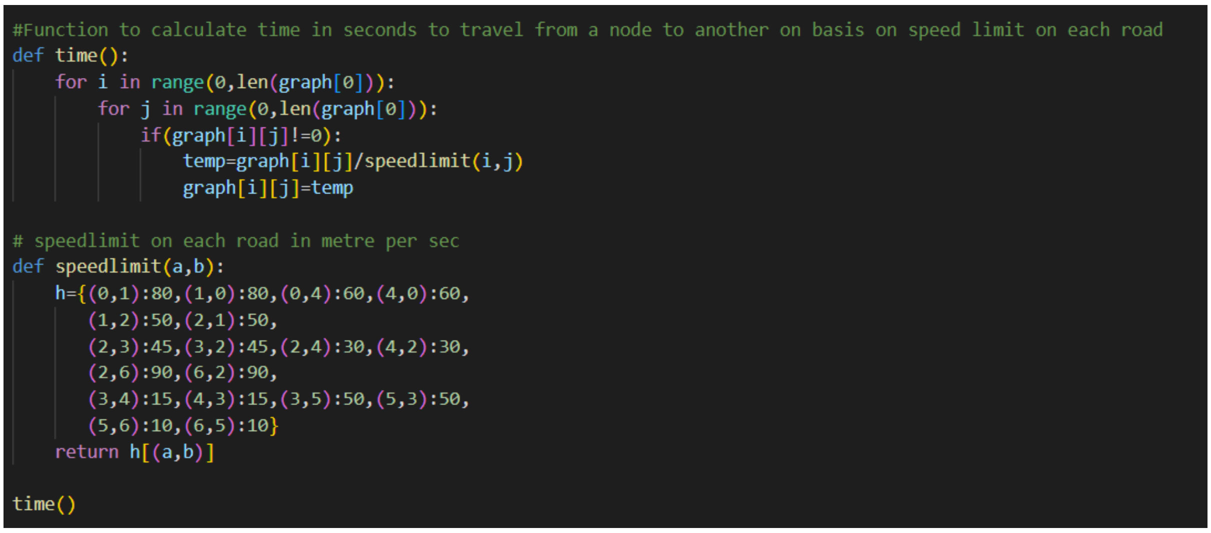
- **GENETIC ALGORITHM:**



```python
import copy
import random
import math

# let there be 7 nodes
# distance of each node
# from each other is given
# along with max speed limit
# on each path.

# the amount of traffic between
# each path will be entered during
# the runtime and time for travesal
# of each node will be calculated by
# algo and path taking least time will
# be returned

# distance in meters
graph=[[0,1100,0,0,1800,0,0],
       [1100,0,1500,0,0,0,0],
       [0,1500,0,900,700,0,1700],
       [0,0,900,0,400,1000,0],
       [1800,0,700,400,0,0,0],
       [0,0,0,1000,0,0,200],
       [0,0,1700,0,0,200,0]]

# for i in graph:
#     print(i)

#taking input from user starting and goal node
start=int(input('enter the starting node(1 indexed):-'))
goal=int(input('enter the goal node(1 indexed):-'))


#checking the validation of the input
if(not (start>0 and start<=len(graph[0])) or not (goal>0 and goal<=len(graph[0]))):
    exit("#######################################################WRONG START OR GOAL NODE#######################################################")
```

First of all, we imported three libraries required for the functioning of code: *copy*, *math and random*. Then we initialized the graph, which contains distances between each node.

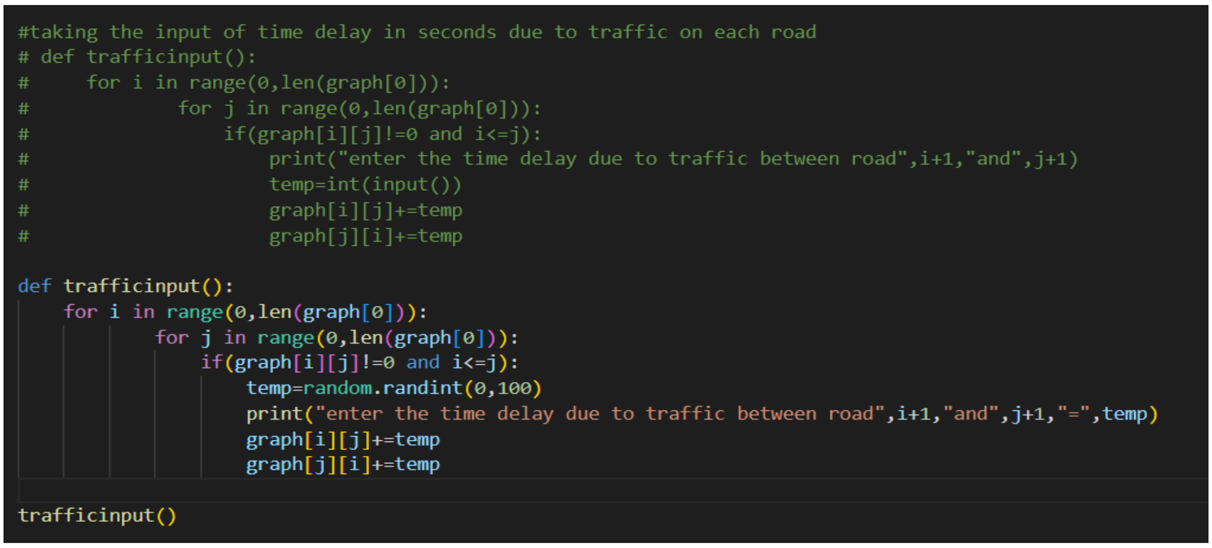
The *start* variable takes the starting node as an input. And the *goal* variable takes the destination node as an input.



```
#Function to calculate time in seconds to travel from a node to another on basis on speed limit on each road
def time():
    for i in range(0,len(graph[0])):
        for j in range(0,len(graph[0])):
            if(graph[i][j]!=0):
                temp=graph[i][j]/speedlimit(i,j)
                graph[i][j]=temp

# speedlimit on each road in metre per sec
def speedlimit(a,b):
    h={(0,1):80,(1,0):80,(0,4):60,(4,0):60,
       (1,2):50,(2,1):50,
       (2,3):45,(3,2):45,(2,4):30,(4,2):30,
       (2,6):90,(6,2):90,
       (3,4):15,(4,3):15,(3,5):50,(5,3):50,
       (5,6):10,(6,5):10}
    return h[(a,b)]

time()
```

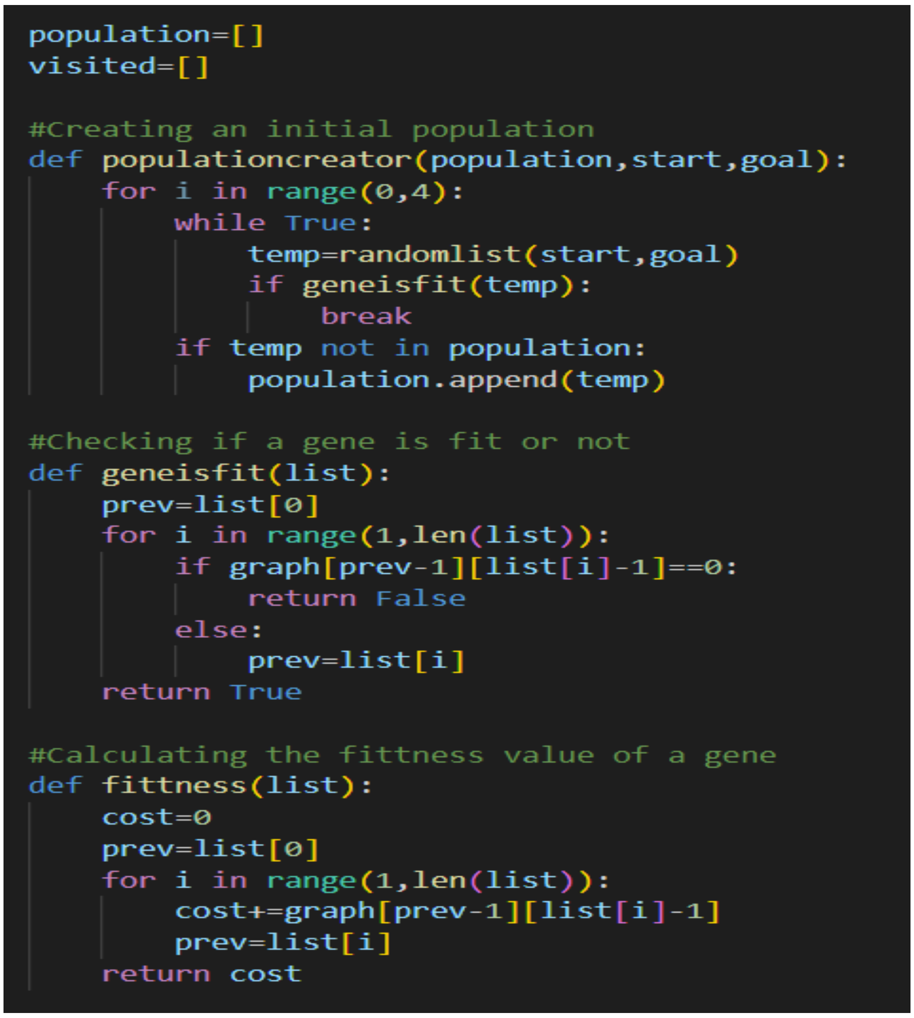
The *time()* function defined here alters the value in the graph by dividing each with its corresponding speed limit.

The *speedlimit(a,b)* function returns the speed limit of the path between two nodes passed as argument to the function.



```
#taking the input of time delay in seconds due to traffic on each road
# def trafficinput():
#     for i in range(0,len(graph[0])):
#             for j in range(0,len(graph[0])):
#                 if(graph[i][j]!=0 and i<=j):
#                     print("enter the time delay due to traffic between road",i+1,"and",j+1)
#                     temp=int(input())
#                     graph[i][j]+=temp
#                     graph[j][i]+=temp

def trafficinput():
    for i in range(0,len(graph[0])):
            for j in range(0,len(graph[0])):
                if(graph[i][j]!=0 and i<=j):
                    temp=random.randint(0,100)
                    print("enter the time delay due to traffic between road",i+1,"and",j+1,"=",temp)
                    graph[i][j]+=temp
                    graph[j][i]+=temp

trafficinput()
```

The *trafficinput()* function generates random time delay values between each node. The time delay here is between the range of 0 to 100.

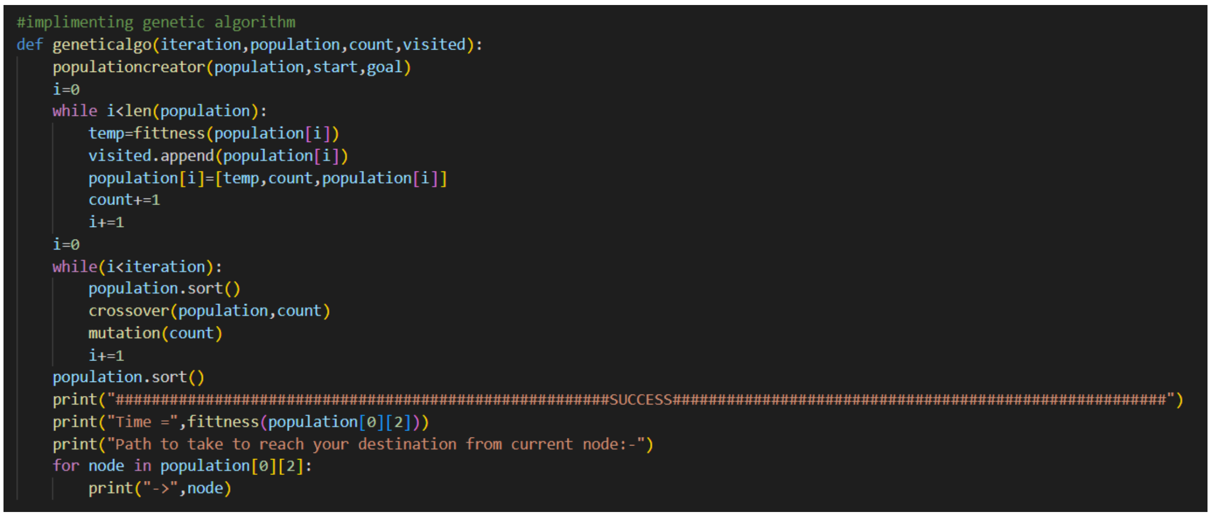


```python
population=[]
visited=[]

#Creating an initial population
def populationcreator(population,start,goal):
    for i in range(0,4):
        while True:
            temp=randomlist(start,goal)
            if geneisfit(temp):
                break
        if temp not in population:
            population.append(temp)

#Checking if a gene is fit or not
def geneisfit(list):
    prev=list[0]
    for i in range(1,len(list)):
        if graph[prev-1][list[i]-1]==0:
            return False
        else:
            prev=list[i]
    return True

#Calculating the fittness value of a gene
def fittness(list):
    cost=0
    prev=list[0]
    for i in range(1,len(list)):
        cost+=graph[prev-1][list[i]-1]
        prev=list[i]
    return cost
```

The *populationcreator(population,start,goal)* creates an initial population using *randomlist()* function.

The *geneisfit(list)* function returns whether the gene passed to it is fit or not.

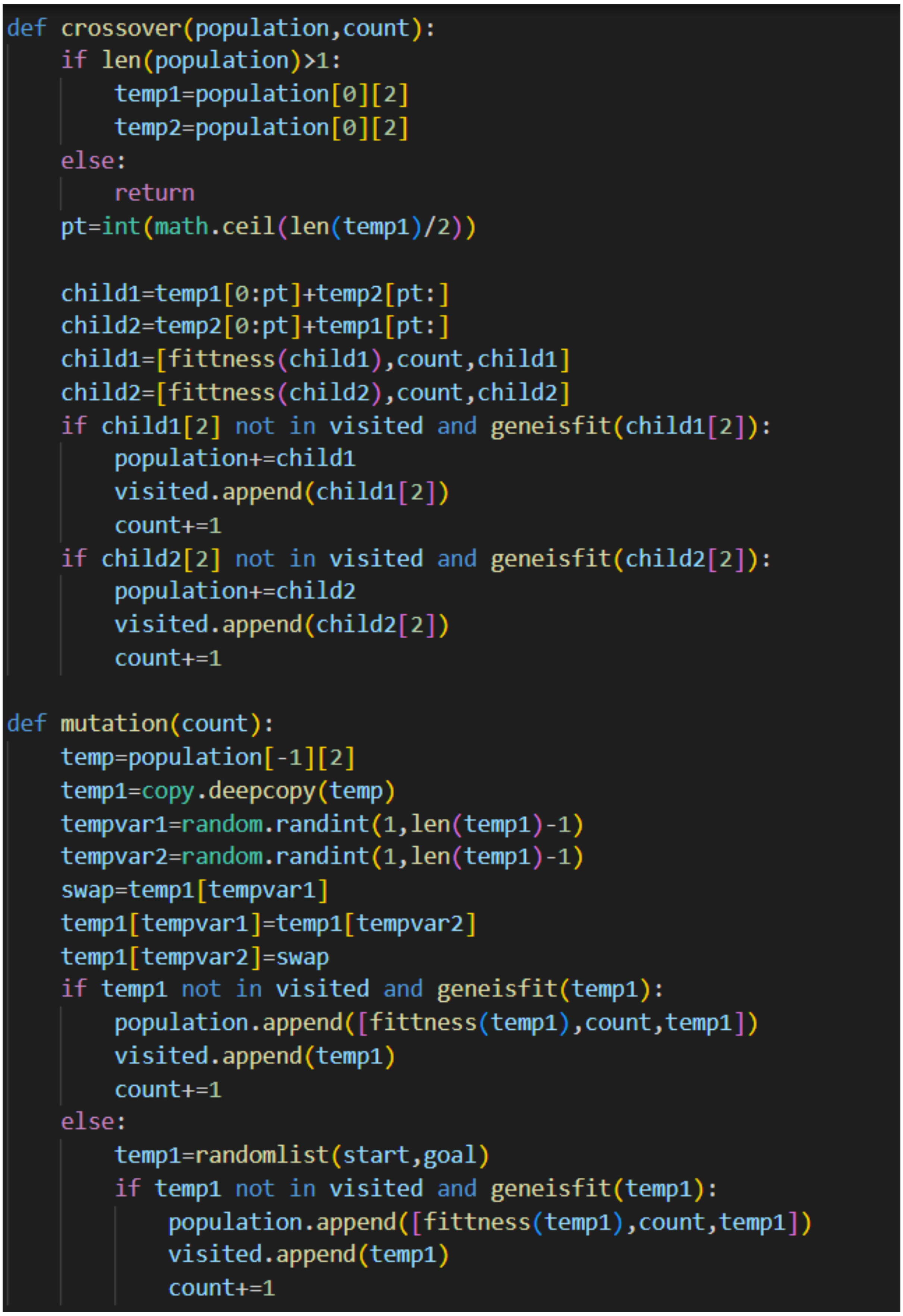
The *fitness(list)* function takes the input as a (list) gene and returns the cost of traversing the list of nodes in the given order.



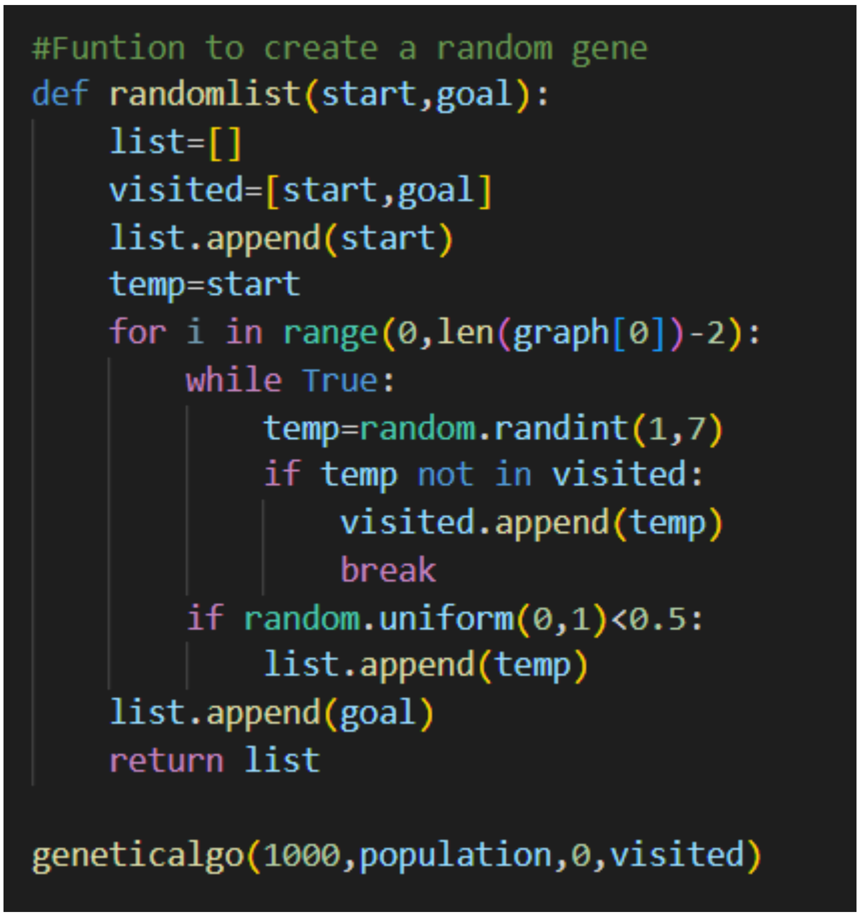
```python
#implimenting genetic algorithm
def geneticalgo(iteration,population,count,visited):
    populationcreator(population,start,goal)
    i=0
    while i<len(population):
        temp=fittness(population[i])
        visited.append(population[i])
        population[i]=[temp,count,population[i]]
        count+=1
        i+=1
    i=0
    while(i<iteration):
        population.sort()
        crossover(population,count)
        mutation(count)
        i+=1
    population.sort()
    print("#######################################################SUCCESS#######################################################")
    print("Time =",fittness(population[0][2]))
    print("Path to take to reach your destination from current node:-")
    for node in population[0][2]:
        print("->",node)
```

Genetic algorithm is a popular optimization algorithm in artificial intelligence (AI) inspired by natural selection and evolution principles. The algorithm is used to find the optimal solution to a problem by iteratively searching for a set of candidate solutions and selecting the best ones based on a fitness function.

The algorithm then applies three main operators to the population: selection, crossover, and mutation. Selection involves selecting the best solutions from the population based on their fitness scores. Crossover involves creating new candidate solutions by combining elements of two or more selected solutions. Mutation involves randomly changing some digits in a candidate solution to introduce new variations.And here, we are implementing it using *geneticalgo(iteration,population,count,visited)* function.



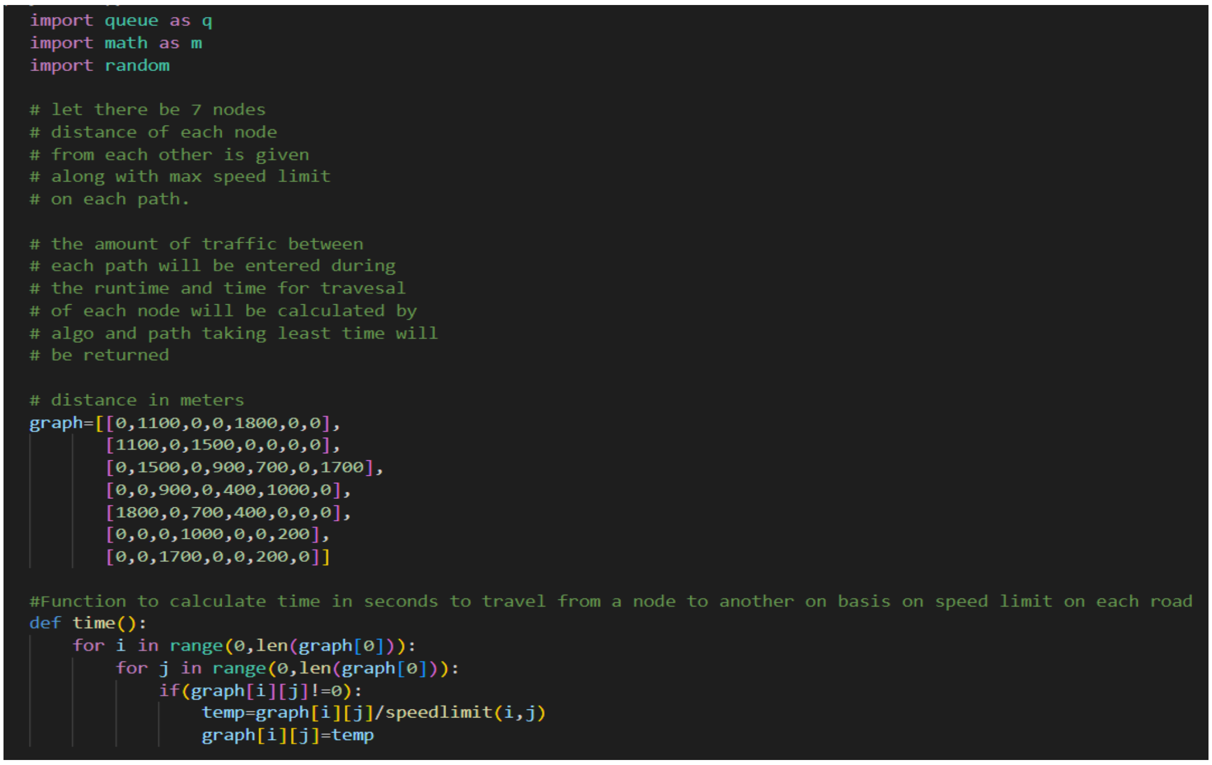
```python
def crossover(population,count):
    if len(population)>1:
        temp1=population[0][2]
        temp2=population[0][2]
    else:
        return
    pt=int(math.ceil(len(temp1)/2))

    child1=temp1[0:pt]+temp2[pt:]
    child2=temp2[0:pt]+temp1[pt:]
    child1=[fittness(child1),count,child1]
    child2=[fittness(child2),count,child2]
    if child1[2] not in visited and geneisfit(child1[2]):
        population+=child1
        visited.append(child1[2])
        count+=1
    if child2[2] not in visited and geneisfit(child2[2]):
        population+=child2
        visited.append(child2[2])
        count+=1

def mutation(count):
    temp=population[-1][2]
    temp1=copy.deepcopy(temp)
    tempvar1=random.randint(1,len(temp1)-1)
    tempvar2=random.randint(1,len(temp1)-1)
    swap=temp1[tempvar1]
    temp1[tempvar1]=temp1[tempvar2]
    temp1[tempvar2]=swap
    if temp1 not in visited and geneisfit(temp1):
        population.append([fittness(temp1),count,temp1])
        visited.append(temp1)
        count+=1
    else:
        temp1=randomlist(start,goal)
        if temp1 not in visited and geneisfit(temp1):
            population.append([fittness(temp1),count,temp1])
            visited.append(temp1)
            count+=1
```



```python
#Funtion to create a random gene
def randomlist(start,goal):
    list=[]
    visited=[start,goal]
    list.append(start)
    temp=start
    for i in range(0,len(graph[0])-2):
        while True:
            temp=random.randint(1,7)
            if temp not in visited:
                visited.append(temp)
                break
        if random.uniform(0,1)<0.5:
            list.append(temp)
    list.append(goal)
    return list

geneticalgo(1000,population,0,visited)
```

The *randomlist(start, goal)* returns a random (list) gene with the starting node as its first element and the goal node as its last.

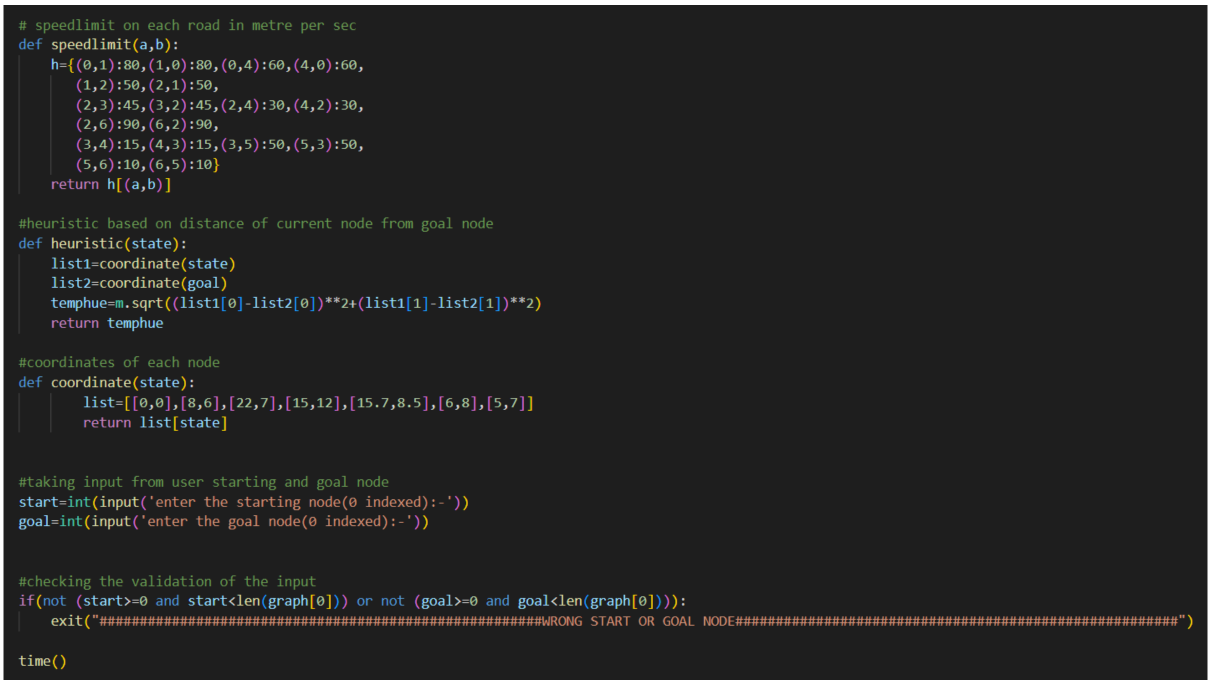
- **A* ALGORITHM:**



```python
import queue as q
import math as m
import random

# let there be 7 nodes
# distance of each node
# from each other is given
# along with max speed limit
# on each path.

# the amount of traffic between
# each path will be entered during
# the runtime and time for travesal
# of each node will be calculated by
# algo and path taking least time will
# be returned

# distance in meters
graph=[[0,1100,0,0,1800,0,0],
       [1100,0,1500,0,0,0,0],
       [0,1500,0,900,700,0,1700],
       [0,0,900,0,400,1000,0],
       [1800,0,700,400,0,0,0],
       [0,0,0,1000,0,0,200],
       [0,0,1700,0,0,200,0]]

#Function to calculate time in seconds to travel from a node to another on basis on speed limit on each road
def time():
    for i in range(0,len(graph[0])):
        for j in range(0,len(graph[0])):
            if(graph[i][j]!=0):
                temp=graph[i][j]/speedlimit(i,j)
                graph[i][j]=temp
```

First of all, we imported three libraries required for the functioning of code that are *queue*, *math and random*. Then we initialized the graph, which contains distances between each node.

The *time()* function defined here alters the value in the graph by dividing each with its corresponding speed limit.



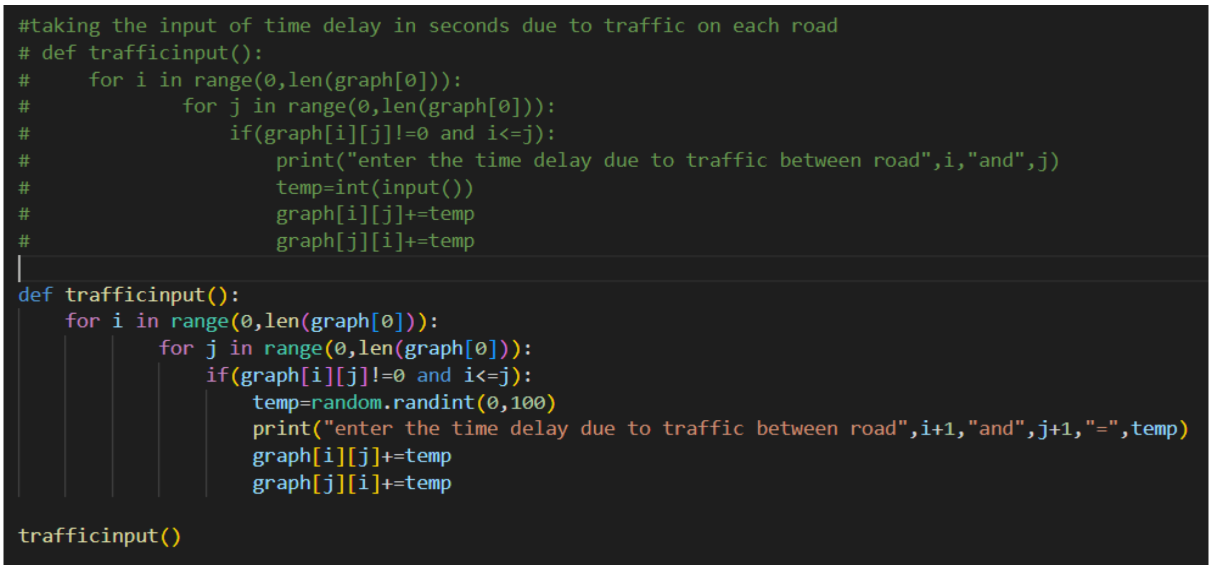
```python
# speedlimit on each road in metre per sec
def speedlimit(a,b):
    h={(0,1):80,(1,0):80,(0,4):60,(4,0):60,
       (1,2):50,(2,1):50,
       (2,3):45,(3,2):45,(2,4):30,(4,2):30,
       (2,6):90,(6,2):90,
       (3,4):15,(4,3):15,(3,5):50,(5,3):50,
       (5,6):10,(6,5):10}
    return h[(a,b)]

#heuristic based on distance of current node from goal node
def heuristic(state):
    list1=coordinate(state)
    list2=coordinate(goal)
    temphue=m.sqrt((list1[0]-list2[0])**2+(list1[1]-list2[1])**2)
    return temphue

#coordinates of each node
def coordinate(state):
        list=[[0,0],[8,6],[22,7],[15,12],[15.7,8.5],[6,8],[5,7]]
        return list[state]


#taking input from user starting and goal node
start=int(input('enter the starting node(0 indexed):-'))
goal=int(input('enter the goal node(0 indexed):-'))


#checking the validation of the input
if(not (start>=0 and start<len(graph[0])) or not (goal>=0 and goal<len(graph[0]))):
    exit("#######################################################WRONG START OR GOAL NODE#######################################################")

time()
```

The *speedlimit(a,b)* function returns the speed limit of the path between two nodes passed as argument to the function.

The *heuristic(state)* function returns the displacement between the current node and the goal node, which we use as heuristic value for our algorithm.

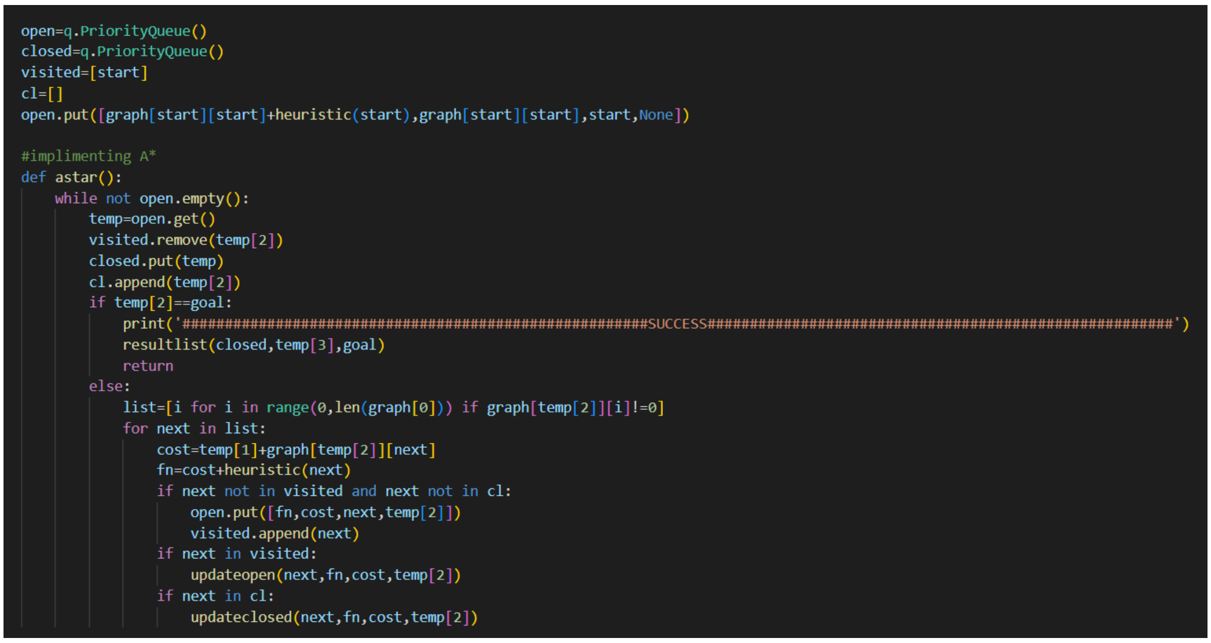
The *coordinate(state)* function returns the coordinate of the nodes passed to it as an argument.

The *start* variable takes the starting node as an input. And the *goal* variable takes the destination node as an input.

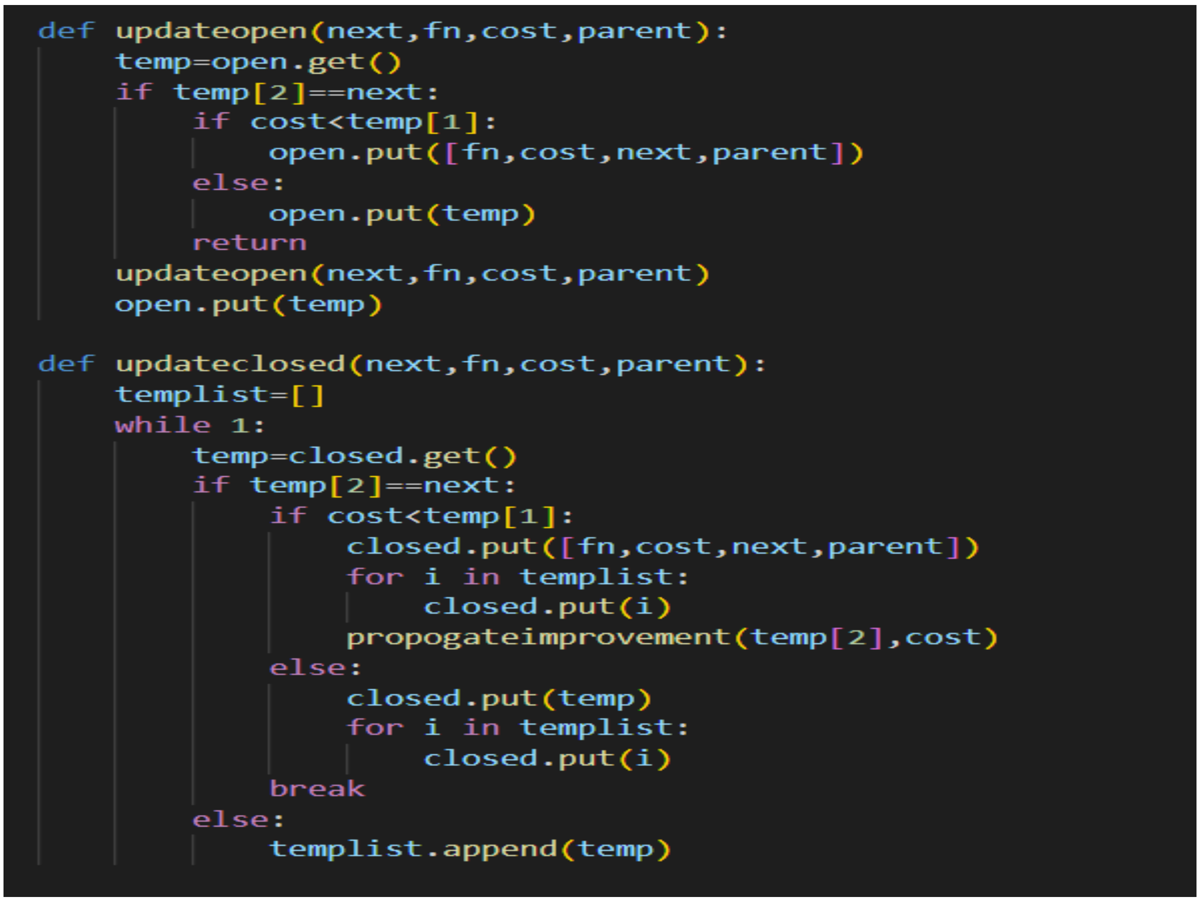


```
#taking the input of time delay in seconds due to traffic on each road
# def trafficinput():
#     for i in range(0,len(graph[0])):
#             for j in range(0,len(graph[0])):
#                 if(graph[i][j]!=0 and i<=j):
#                     print("enter the time delay due to traffic between road",i,"and",j)
#                     temp=int(input())
#                     graph[i][j]+=temp
#                     graph[j][i]+=temp

def trafficinput():
    for i in range(0,len(graph[0])):
            for j in range(0,len(graph[0])):
                if(graph[i][j]!=0 and i<=j):
                    temp=random.randint(0,100)
                    print("enter the time delay due to traffic between road",i+1,"and",j+1,"=",temp)
                    graph[i][j]+=temp
                    graph[j][i]+=temp

trafficinput()
```

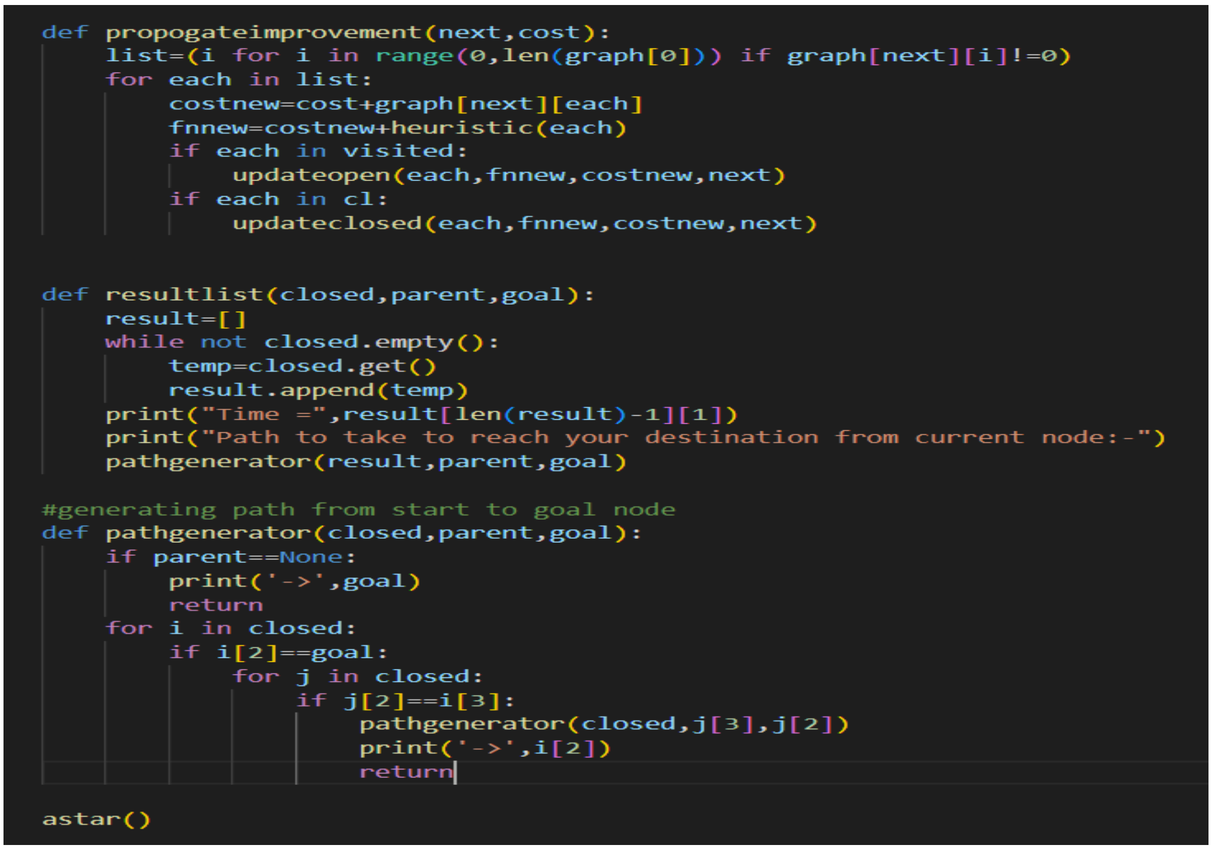
The *trafficinput()* function generates random time delay values between each node. Time delay here is between the range of 0 to 100.



```
open=q.PriorityQueue()
closed=q.PriorityQueue()
visited=[start]
cl=[]
open.put([graph[start][start]+heuristic(start),graph[start][start],start,None])

#implimenting A*
def astar():
    while not open.empty():
        temp=open.get()
        visited.remove(temp[2])
        closed.put(temp)
        cl.append(temp[2])
        if temp[2]==goal:
            print('###############################################################SUCCESS###############################################################')
            resultlist(closed,temp[3],goal)
            return
        else:
            list=[i for i in range(0,len(graph[0])) if graph[temp[2]][i]!=0]
            for next in list:
                cost=temp[1]+graph[temp[2]][next]
                fn=cost+heuristic(next)
                if next not in visited and next not in cl:
                    open.put([fn,cost,next,temp[2]])
                    visited.append(next)
                if next in visited:
                    updateopen(next,fn,cost,temp[2])
                if next in cl:
                    updateclosed(next,fn,cost,temp[2])
```



```python
def updateopen(next,fn,cost,parent):
    temp=open.get()
    if temp[2]==next:
        if cost<temp[1]:
            open.put([fn,cost,next,parent])
        else:
            open.put(temp)
        return
    updateopen(next,fn,cost,parent)
    open.put(temp)

def updateclosed(next,fn,cost,parent):
    templist=[]
    while 1:
        temp=closed.get()
        if temp[2]==next:
            if cost<temp[1]:
                closed.put([fn,cost,next,parent])
                for i in templist:
                    closed.put(i)
                propogateimprovement(temp[2],cost)
            else:
                closed.put(temp)
                for i in templist:
                    closed.put(i)
            break
        else:
            templist.append(temp)
```



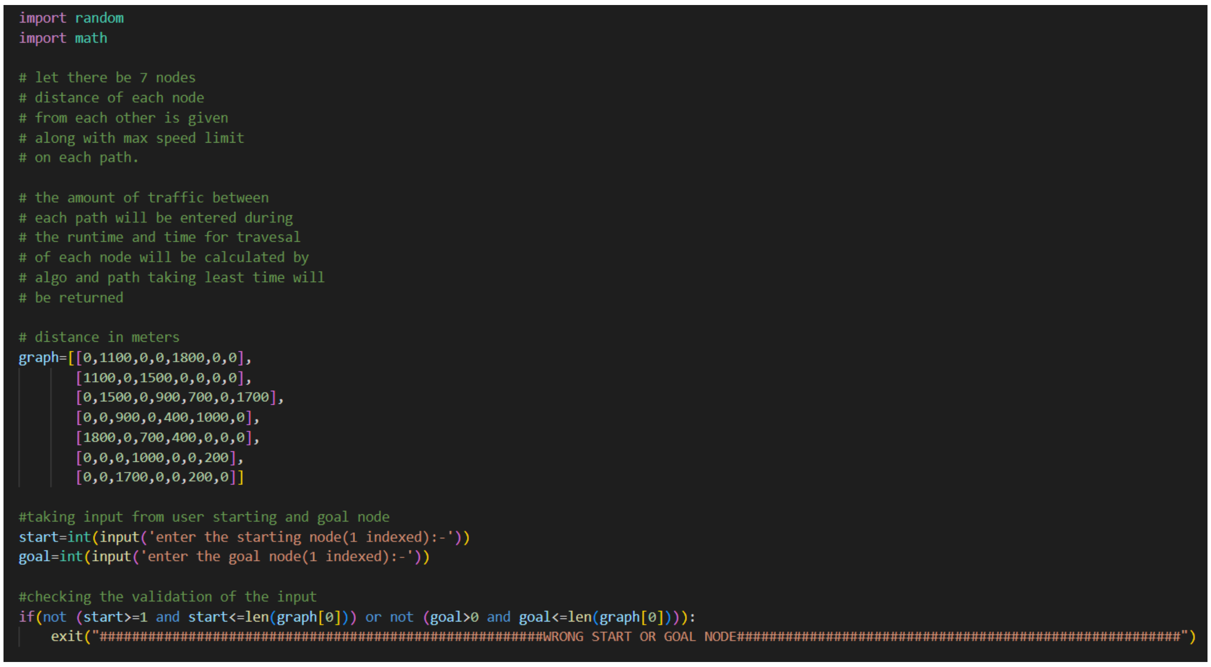
```python
def propogateimprovement(next,cost):
    list=(i for i in range(0,len(graph[0])) if graph[next][i]!=0)
    for each in list:
        costnew=cost+graph[next][each]
        fnnew=costnew+heuristic(each)
        if each in visited:
            updateopen(each,fnnew,costnew,next)
        if each in cl:
            updateclosed(each,fnnew,costnew,next)


def resultlist(closed,parent,goal):
    result=[]
    while not closed.empty():
        temp=closed.get()
        result.append(temp)
    print("Time =",result[len(result)-1][1])
    print("Path to take to reach your destination from current node:-")
    pathgenerator(result,parent,goal)

#generating path from start to goal node
def pathgenerator(closed,parent,goal):
    if parent==None:
        print('->',goal)
        return
    for i in closed:
        if i[2]==goal:
            for j in closed:
                if j[2]==i[3]:
                    pathgenerator(closed,j[3],j[2])
                    print('->',i[2])
                    return

astar()
```

A* (pronounced "A-star") is a widely used informed search algorithm in artificial intelligence and computer science. A* search starts with an initial state and expands the node with the lowest estimated total cost f(n) = g(n) + h(n), where g(n) is the actual cost of reaching node n from the starting state, and h(n) is the heuristic estimate of the cost of reaching the goal state from node n. The heuristic function h(n) is admissible if it never overestimates the actual cost of reaching the goal state from node n.

The algorithm maintains a priority queue of open nodes and a set of closed nodes that have already been expanded. At each step, it selects the node with the lowest f value from the priority queue, expands it, and adds its successors to the priority queue if they have not been visited before or have a lower f value than their previous visit. And here, we are implementing it using *astar()* function.

Pathgenerator(closed, parent, goal, cost) function generates the path from source to destination node.

- **SIMULATED ANNEALING:**

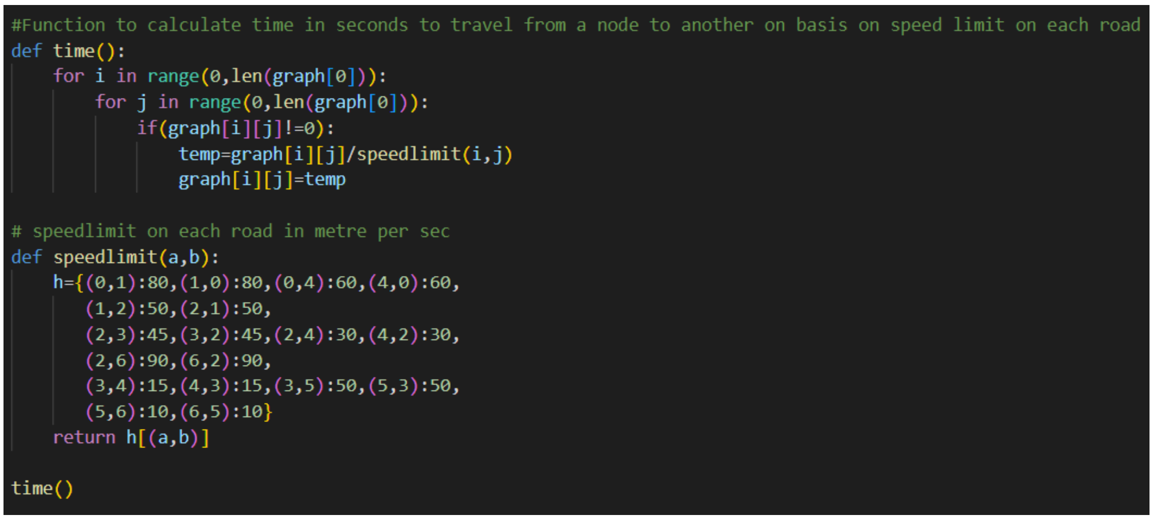


```python
import random
import math

# let there be 7 nodes
# distance of each node
# from each other is given
# along with max speed limit
# on each path.

# the amount of traffic between
# each path will be entered during
# the runtime and time for travesal
# of each node will be calculated by
# algo and path taking least time will
# be returned

# distance in meters
graph=[[0,1100,0,0,1800,0,0],
       [1100,0,1500,0,0,0,0],
       [0,1500,0,900,700,0,1700],
       [0,0,900,0,400,1000,0],
       [1800,0,700,400,0,0,0],
       [0,0,0,1000,0,0,200],
       [0,0,1700,0,0,200,0]]

#taking input from user starting and goal node
start=int(input('enter the starting node(1 indexed):-'))
goal=int(input('enter the goal node(1 indexed):-'))

#checking the validation of the input
if(not (start>=1 and start<=len(graph[0])) or not (goal>0 and goal<=len(graph[0]))):
    exit("#######################################################WRONG START OR GOAL NODE#######################################################")
```

First, we imported three libraries required for the functioning of code, that is, *math and random*. Then we initialized the graph, which contains distances between each node.

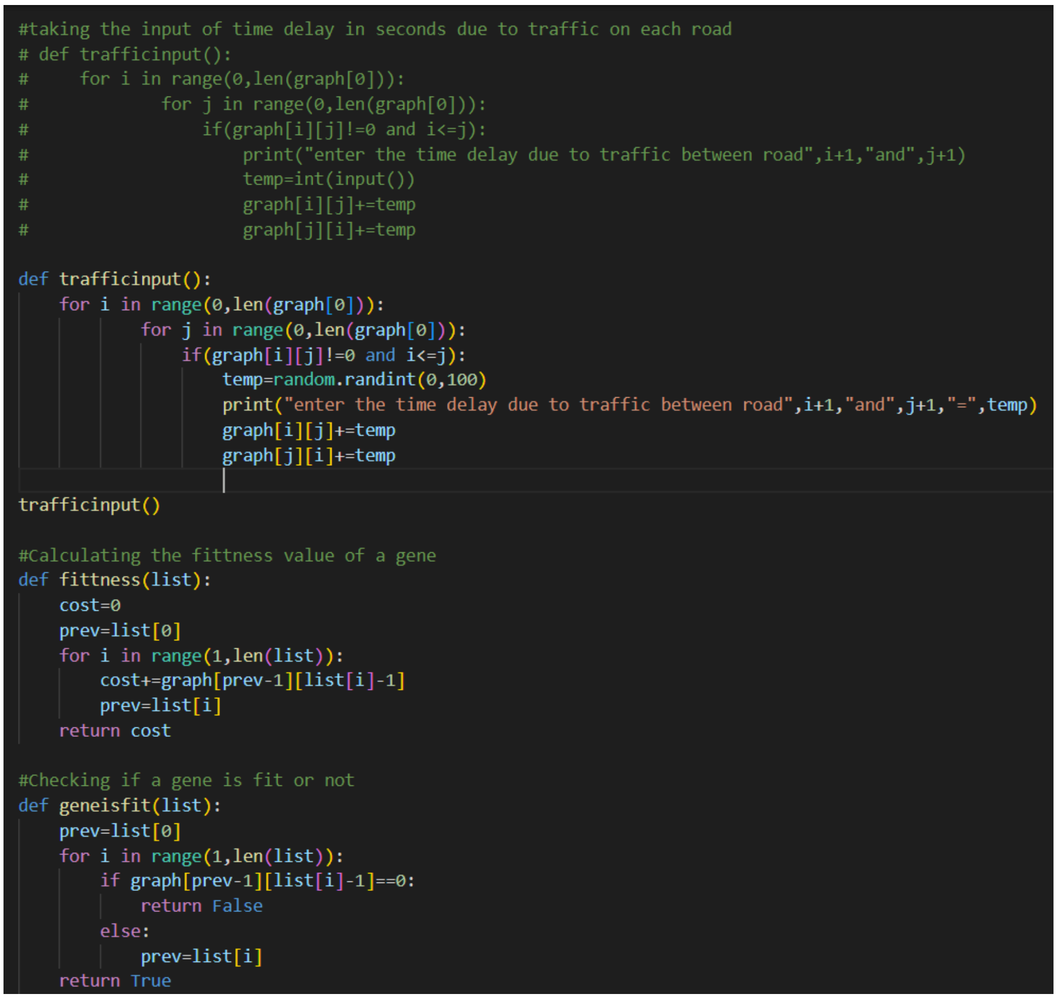
The *time()* function defined here alters the value in the graph by dividing each with its corresponding speed limit.

The *start* variable takes the starting node as an input. And the *goal* variable takes the destination node as an input.



```
#Function to calculate time in seconds to travel from a node to another on basis on speed limit on each road
def time():
    for i in range(0,len(graph[0])):
        for j in range(0,len(graph[0])):
            if(graph[i][j]!=0):
                temp=graph[i][j]/speedlimit(i,j)
                graph[i][j]=temp

# speedlimit on each road in metre per sec
def speedlimit(a,b):
    h={(0,1):80,(1,0):80,(0,4):60,(4,0):60,
       (1,2):50,(2,1):50,
       (2,3):45,(3,2):45,(2,4):30,(4,2):30,
       (2,6):90,(6,2):90,
       (3,4):15,(4,3):15,(3,5):50,(5,3):50,
       (5,6):10,(6,5):10}
    return h[(a,b)]

time()
```

The *speedlimit(a,b)* function returns the speed limit of the path between two nodes passed as argument to the function.

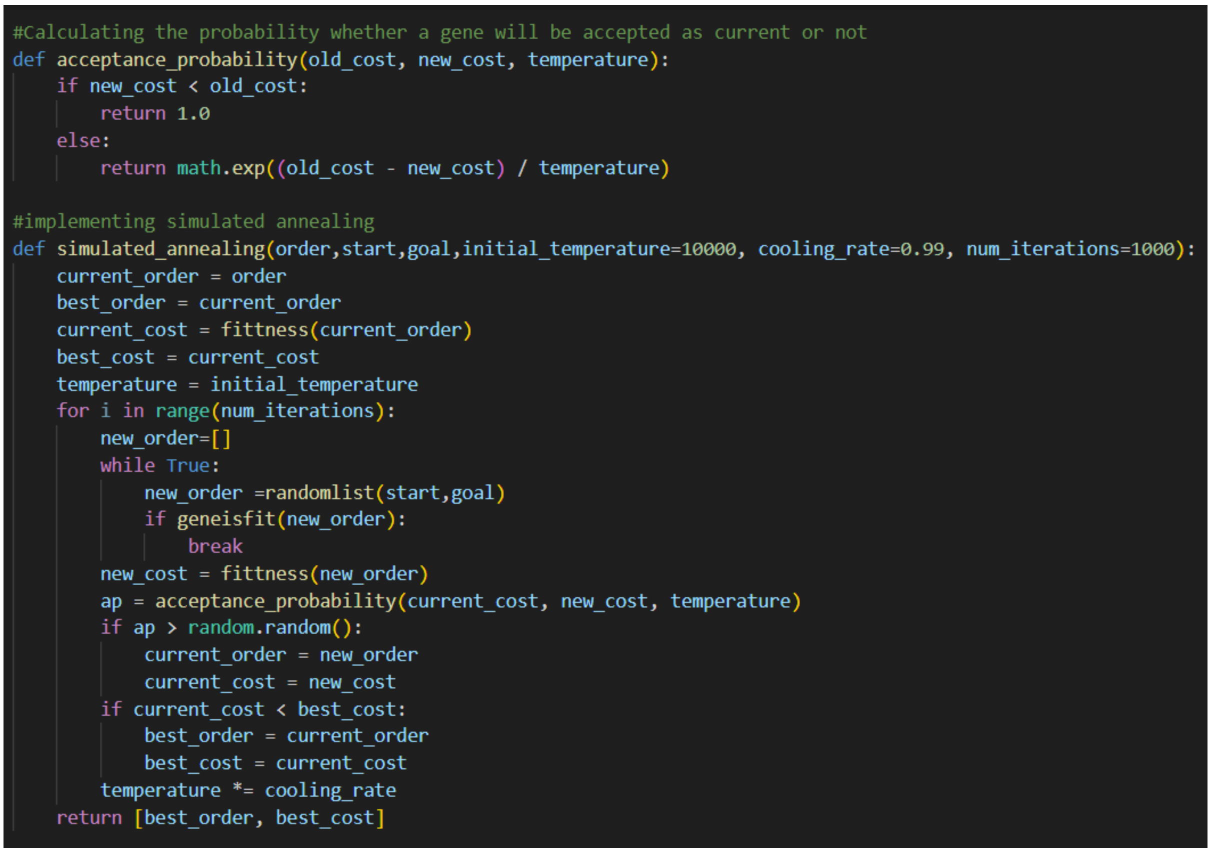


```
#taking the input of time delay in seconds due to traffic on each road
# def trafficinput():
#     for i in range(0,len(graph[0])):
#             for j in range(0,len(graph[0])):
#                 if(graph[i][j]!=0 and i<=j):
#                     print("enter the time delay due to traffic between road",i+1,"and",j+1)
#                     temp=int(input())
#                     graph[i][j]+=temp
#                     graph[j][i]+=temp

def trafficinput():
    for i in range(0,len(graph[0])):
            for j in range(0,len(graph[0])):
                if(graph[i][j]!=0 and i<=j):
                    temp=random.randint(0,100)
                    print("enter the time delay due to traffic between road",i+1,"and",j+1,"=",temp)
                    graph[i][j]+=temp
                    graph[j][i]+=temp

trafficinput()

#Calculating the fittness value of a gene
def fittness(list):
    cost=0
    prev=list[0]
    for i in range(1,len(list)):
        cost+=graph[prev-1][list[i]-1]
        prev=list[i]
    return cost

#Checking if a gene is fit or not
def geneisfit(list):
    prev=list[0]
    for i in range(1,len(list)):
        if graph[prev-1][list[i]-1]==0:
            return False
        else:
            prev=list[i]
    return True
```

The *trafficinput()* function generates random time delay values between each node. The time delay here is between the range of 0 to 100.

The *fitness(list)* function takes the input as a (list) gene and returns the cost of traversing the list of nodes in the given order.

The *geneisfit(list)* function returns whether the gene passed to it is fit or not.

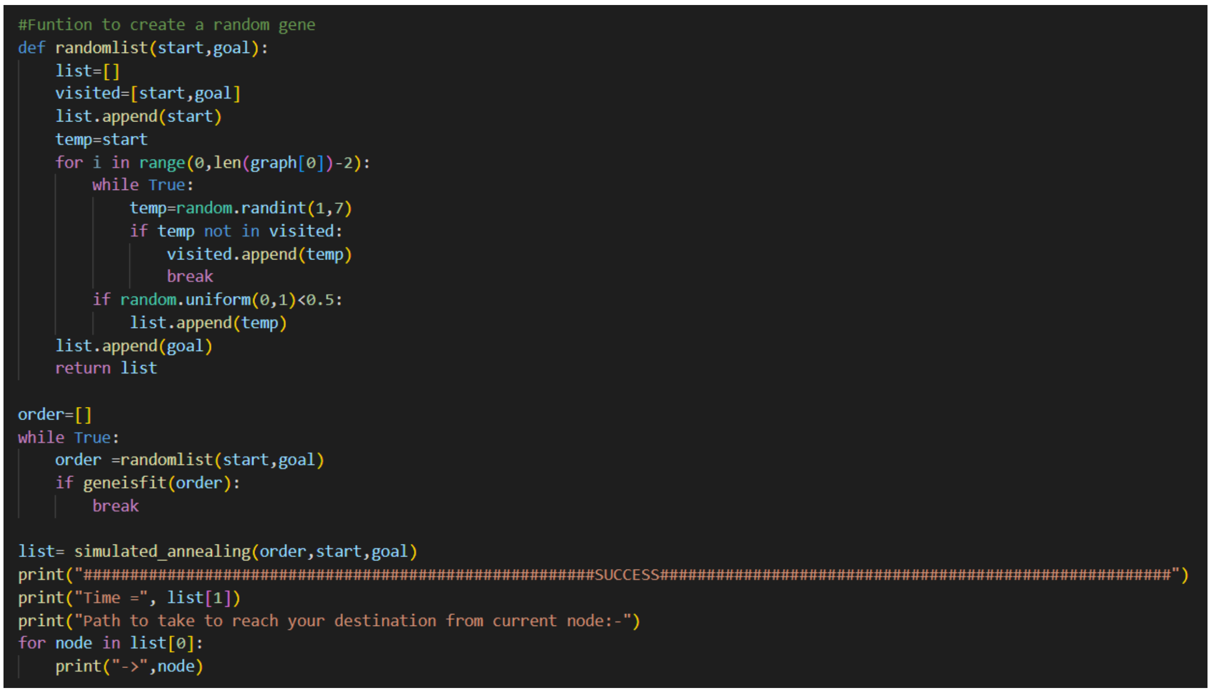


```python
#Calculating the probability whether a gene will be accepted as current or not
def acceptance_probability(old_cost, new_cost, temperature):
    if new_cost < old_cost:
        return 1.0
    else:
        return math.exp((old_cost - new_cost) / temperature)

#implementing simulated annealing
def simulated_annealing(order,start,goal,initial_temperature=10000, cooling_rate=0.99, num_iterations=1000):
    current_order = order
    best_order = current_order
    current_cost = fittness(current_order)
    best_cost = current_cost
    temperature = initial_temperature
    for i in range(num_iterations):
        new_order=[]
        while True:
            new_order =randomlist(start,goal)
            if geneisfit(new_order):
                break
        new_cost = fittness(new_order)
        ap = acceptance_probability(current_cost, new_cost, temperature)
        if ap > random.random():
            current_order = new_order
            current_cost = new_cost
        if current_cost < best_cost:
            best_order = current_order
            best_cost = current_cost
        temperature *= cooling_rate
    return [best_order, best_cost]
```

The *acceptance_probability(old_cost,new_cost, temperature)* returns the probability of accepting the newly generated gene.

Simulated annealing is a probabilistic metaheuristic algorithm used in artificial intelligence and optimisation problems to find a given objective function's global optimum or near-optimum solution.

It can be used to search for the fastest path by iteratively exploring different paths and adjusting the path based on the objective function that measures the time it takes to travel from the starting point to the destination. The algorithm starts with an initial path and then randomly perturbs the path by adding or removing edges or nodes. The new path is then evaluated based on the objective function, and it is accepted or rejected based on a probability that depends on the "temperature" of the algorithm.

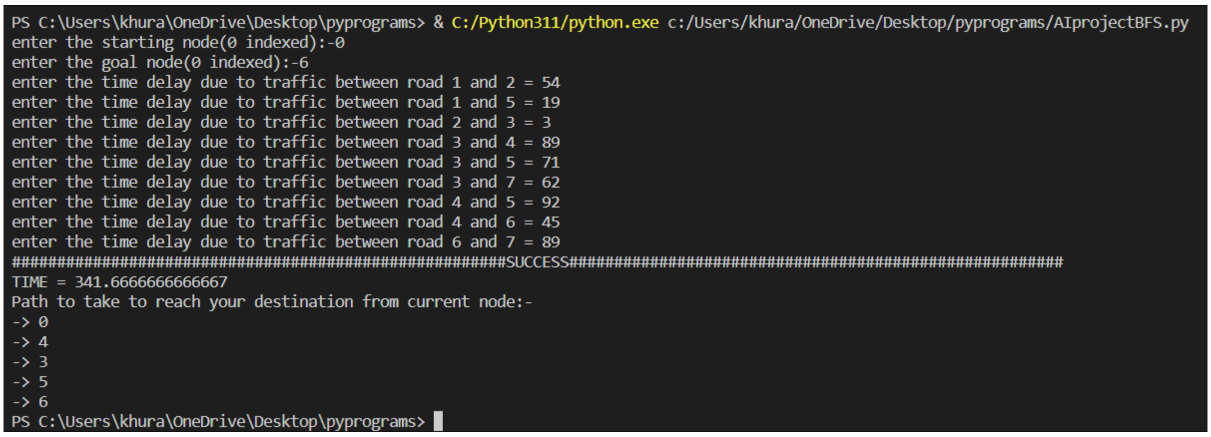


```python
#Funtion to create a random gene
def randomlist(start,goal):
    list=[]
    visited=[start,goal]
    list.append(start)
    temp=start
    for i in range(0,len(graph[0])-2):
        while True:
            temp=random.randint(1,7)
            if temp not in visited:
                visited.append(temp)
                break
        if random.uniform(0,1)<0.5:
            list.append(temp)
    list.append(goal)
    return list

order=[]
while True:
    order =randomlist(start,goal)
    if geneisfit(order):
        break

list= simulated_annealing(order,start,goal)
print("##############################################SUCCESS##############################################")
print("Time =", list[1])
print("Path to take to reach your destination from current node:-")
for node in list[0]:
    print("->",node)
```

The *randomlist(start, goal)* returns a random (list) gene with the starting node as its first element and the goal node as its last.

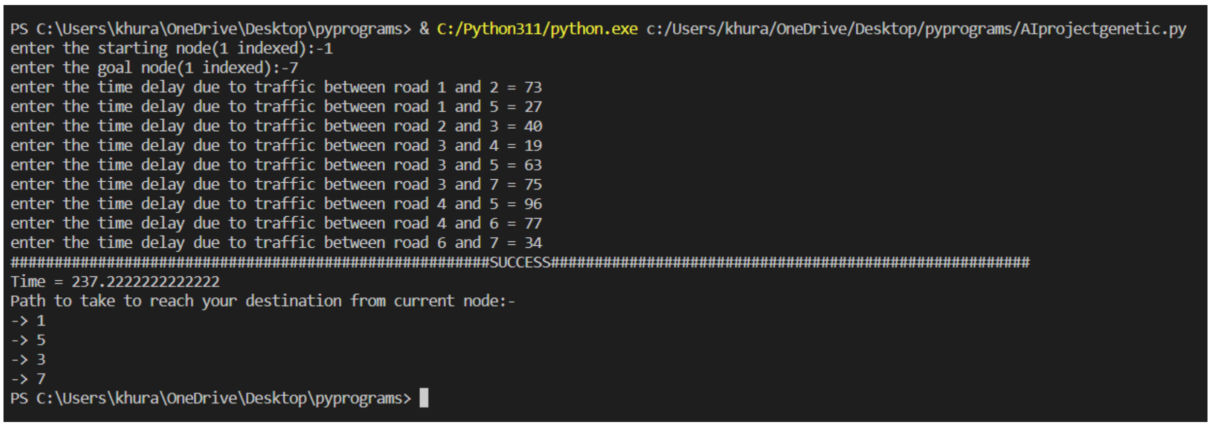
FINAL PATH GENERATION:

1. BEST FIRST SEARCH:



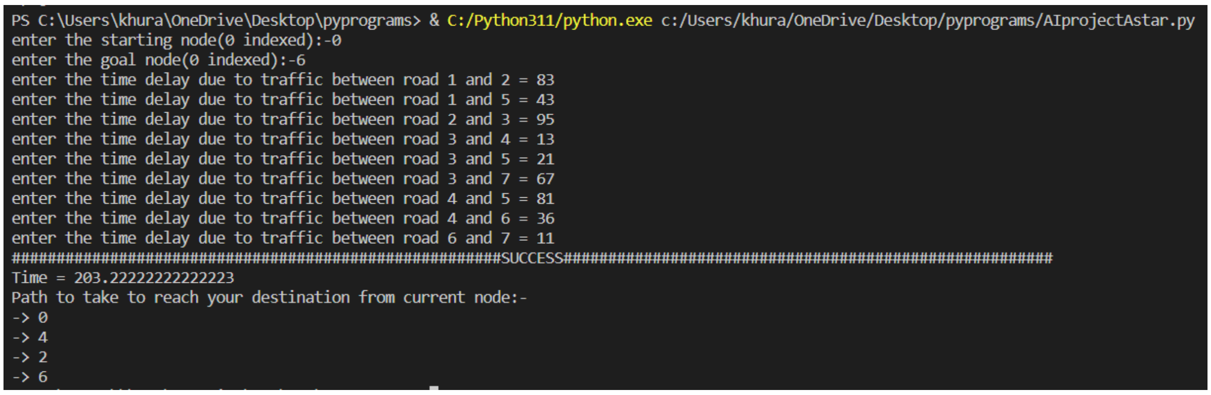
```
PS C:\Users\khura\OneDrive\Desktop\pyprograms> & C:/Python311/python.exe c:/Users/khura/OneDrive/Desktop/pyprograms/AIprojectBFS.py
enter the starting node(0 indexed):-0
enter the goal node(0 indexed):-6
enter the time delay due to traffic between road 1 and 2 = 54
enter the time delay due to traffic between road 1 and 5 = 19
enter the time delay due to traffic between road 2 and 3 = 3
enter the time delay due to traffic between road 3 and 4 = 89
enter the time delay due to traffic between road 3 and 5 = 71
enter the time delay due to traffic between road 3 and 7 = 62
enter the time delay due to traffic between road 4 and 5 = 92
enter the time delay due to traffic between road 4 and 6 = 45
enter the time delay due to traffic between road 6 and 7 = 89
########################################################SUCCESS########################################################
TIME = 341.6666666666667
Path to take to reach your destination from current node:-
-> 0
-> 4
-> 3
-> 5
-> 6
PS C:\Users\khura\OneDrive\Desktop\pyprograms>
```

2. GENETIC ALGORITHM:



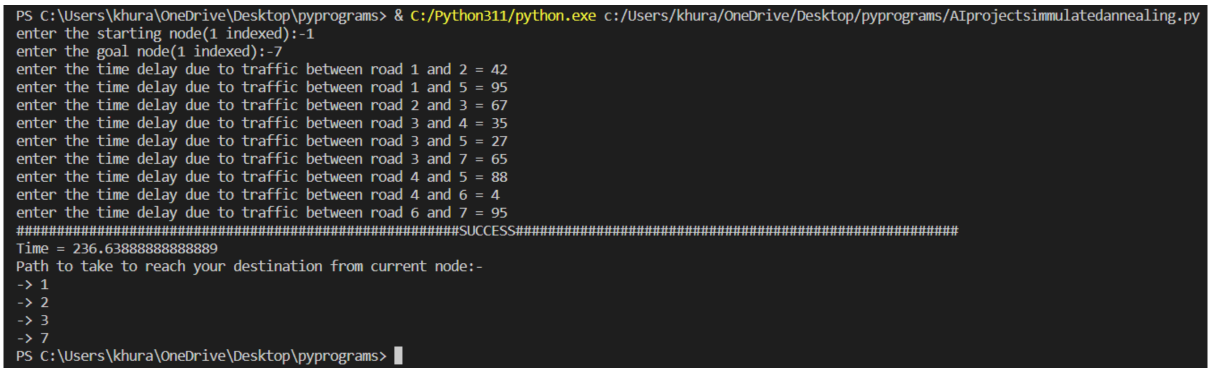
```
PS C:\Users\khura\OneDrive\Desktop\pyprograms> & C:/Python311/python.exe c:/Users/khura/OneDrive/Desktop/pyprograms/AIprojectgenetic.py
enter the starting node(1 indexed):-1
enter the goal node(1 indexed):-7
enter the time delay due to traffic between road 1 and 2 = 73
enter the time delay due to traffic between road 1 and 5 = 27
enter the time delay due to traffic between road 2 and 3 = 40
enter the time delay due to traffic between road 3 and 4 = 19
enter the time delay due to traffic between road 3 and 5 = 63
enter the time delay due to traffic between road 3 and 7 = 75
enter the time delay due to traffic between road 4 and 5 = 96
enter the time delay due to traffic between road 4 and 6 = 77
enter the time delay due to traffic between road 6 and 7 = 34
######################################################SUCCESS######################################################
Time = 237.2222222222222
Path to take to reach your destination from current node:-
-> 1
-> 5
-> 3
-> 7
PS C:\Users\khura\OneDrive\Desktop\pyprograms>
```

3. A* ALGORITHM:



```
PS C:\Users\khura\OneDrive\Desktop\pyprograms> & C:/Python311/python.exe c:/Users/khura/OneDrive/Desktop/pyprograms/AIprojectAstar.py
enter the starting node(0 indexed):-0
enter the goal node(0 indexed):-6
enter the time delay due to traffic between road 1 and 2 = 83
enter the time delay due to traffic between road 1 and 5 = 43
enter the time delay due to traffic between road 2 and 3 = 95
enter the time delay due to traffic between road 3 and 4 = 13
enter the time delay due to traffic between road 3 and 5 = 21
enter the time delay due to traffic between road 3 and 7 = 67
enter the time delay due to traffic between road 4 and 5 = 81
enter the time delay due to traffic between road 4 and 6 = 36
enter the time delay due to traffic between road 6 and 7 = 11
######################################################SUCCESS######################################################
Time = 203.22222222222223
Path to take to reach your destination from current node:-
-> 0
-> 4
-> 2
-> 6
```

4. SIMULATED ANNEALING:



```
PS C:\Users\khura\OneDrive\Desktop\pyprograms> & C:/Python311/python.exe c:/Users/khura/OneDrive/Desktop/pyprograms/AIprojectsimmulatedannealing.py
enter the starting node(1 indexed):-1
enter the goal node(1 indexed):-7
enter the time delay due to traffic between road 1 and 2 = 42
enter the time delay due to traffic between road 1 and 5 = 95
enter the time delay due to traffic between road 2 and 3 = 67
enter the time delay due to traffic between road 3 and 4 = 35
enter the time delay due to traffic between road 3 and 5 = 27
enter the time delay due to traffic between road 3 and 7 = 65
enter the time delay due to traffic between road 4 and 5 = 88
enter the time delay due to traffic between road 4 and 6 = 4
enter the time delay due to traffic between road 6 and 7 = 95
######################################################SUCCESS######################################################
Time = 236.63888888888889
Path to take to reach your destination from current node:-
-> 1
-> 2
-> 3
-> 7
PS C:\Users\khura\OneDrive\Desktop\pyprograms>
```

SIGNIFICANCE AND APPLICATIONS:

- To develop an algorithm that can find the fastest path between two nodes based on multiple criteria such as distance, traffic, and speed limit.
- To test and validate the algorithm on real-world data and demonstrate its effectiveness in solving the fastest path problem.
- To provide a useful tool for various applications such as transportation, logistics, and routing.
- To evaluate the algorithm's performance using appropriate metrics and compare it with other existing algorithms.

SUMMARY:

Best First Search algorithm is used to find the shortest path between two nodes in a graph. The graph is represented as an adjacency matrix, where the value of each edge is the distance between the nodes. The program calculates the time taken to travel between nodes based on the speed limit of each path, and takes input from the user for the amount of traffic on each road. The program uses a priority queue to keep track of the nodes to be visited and implements the heuristic based on the distance of each node from the goal node. The program also generates the path to be taken to reach the goal node from the starting node along with the time taken to travel.

Genetic algorithm is used to find the shortest path between two nodes in a given graph. The program takes input of the graph, the starting and ending nodes, the speed limit on each road, and the time delay due to traffic on each road. The program then creates an initial population, checks the fitness of each gene, and calculates the fitness value of each gene. The genetic algorithm is then implemented with a crossover and mutation function to evolve the population towards the optimal solution. The program outputs the shortest path and the time it takes to travel from the starting node to the ending node.

A* algorithm is used to find the shortest path from a start node to a goal node. The program takes as input the distance between nodes and the speed limit on each path. It calculates the time taken to travel between nodes based on the speed limit, and then prompts the user to enter the traffic delay on each path, which is added to the travel time. The heuristic used is based on the distance of the current node from the goal node. The program then implements the A* algorithm to find the shortest path between the start and goal nodes. The output is the path with the lowest travel time, including traffic delays.

Simulated annealing algorithm is used for finding the shortest path between two nodes in a graph. The graph represents distances between different nodes with speed limit and traffic conditions as input. The algorithm calculates the fitness value of the gene, checks if it's fit, and calculates the probability of accepting a new gene as the current one. Finally, it generates a random gene and applies simulated annealing to find the best route with the shortest distance.

RESULT ANALYSIS:

All the algorithms are designed to provide with a fast solution to the user but Genetic Algorithm turns out to be the best with respect to the user. Since it provides a good solution and that too in a timely manner we concluded that Genetic Algorithm is practically the best for Multi Criteria Fastest Path Algorithm.