CPSC 535 – Advance Algorithms

Project 1 – Savvy Traveller

Report

<u>Team Members</u>

Shivangi Shakya

Nicholas Bidler

Saoni Mustafi

Prof. Doina Bein

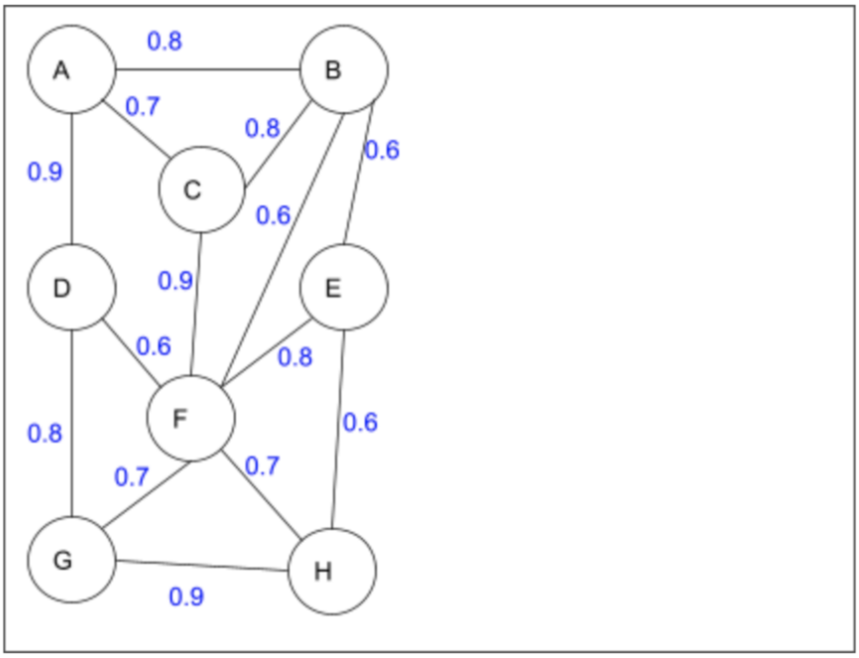
California State University, Fullerton

800 N State College Blvd, Fullerton, CA 92831

Summary

This report contains details about the work done to solve the Savvy Traveller – a weighted directed graph problem as a part of our project. We were given a graph, denoting various cities a traveller could fly to from every other city with a certain probability of on-time arrival. We had to compute the path with the highest probability of on-time arrival and the most reliable travel destination.

We were provided with the below graph as one of the examples –



We were given to compute the following:

(i) what route will maximize the probability to arrive on time between any two cities, for example, city A and city F.

(ii) what city among {A, B, C, D, E, F, G, H} is the most reliable travel destination.

To compute **(i)**, we considered all the routes that could lead to city F from city A. Below are the probable routes and their corresponding-

Path from city A to city F	Probability of on-time Arrival
A->B->C->F	0.576
A->B->E->F	0.384
A->B->E->G->D->F	0.0
A->B->E->G->F	0.0
A->B->E->G->H->F	0.0
A->B->E->H->F	0.2016
A->B->E->H->G->D->F	0.124416
A->B->E->H->G->F	0.18144

A->B->F	0.48
A->C->B->E->F	0.2688
A->C->B->E->G->D->F	0.0
A->C->B->E->G->F	0.0
A->C->B->E->G->H->F	0.0
A->C->B->E->H->F	0.14112
A->C->B->E->H->G->D->F	0.0870912
A->C->B->E->H->G->F	0.127008
A->C->B->F	0.336
A->C->F	0.63
A->D->F	0.54
A->D->G->F	0.504
A->D->G->H->E->B->C->F	0.1679616
A->D->G->H->E->B->F	0.139968
A->D->G->H->E->F	0.31104
A->D->G->H->F	0.4536

We see that the highest is that of A->C->F (highlighted) with a probability of 0.63. Hence the answer to the first question is A->C->F.

We firstly created a *graph1.txt* file which contains the on-time arrival probabilities of cities for their adjacent ones in a dictionary format. We provide this file name and then enter the source and destination vertex we want to find the best route for.

To implement this task, we created a function named – *printBestRoute()* which takes source and destination cities as its parameters. This function prints the best route based on the maximum probability value computed by the function *BestRoute()* which takes the same source and destination parameters. The *BestRoute()* keeps a track of the visited cities and appends that to a list called *'path'*. It then checks if the source has reached its destination, if yes, it takes the probabilities of the cities that have been added to the *'path'* list and finds the product *'prod'*.

prod = prod * float(graph[self.path[i-1]][self.path[i]])

Once, this calculation is complete for one route, it pops the last element and finds another possible route to the destination from the same source. These probabilities of all the routes and the routes themselves are stored in a dictionary named *'list1'*.

When all probabilities of all the routes have been calculated, *printBestRoute()* provides the routes with the highest probability, in our case, it is A->C->F with a probability of 0.63.

To compute **(ii)**, we find out the maximum probabilities of all cities from every other city and the city with the highest probability among all is considered to be the most reliable city. We do the same calculation as we did above for every city and then return the result. Below is the calculation for the given cities-

City A		
Source City	Max Probability	Max Prob Path
From B	0.8	B->A
From C	0.7	C->A
From D	0.9	D->A
From E	0.504	E->F->C->A
From F	0.63	F->C->A
From G	0.72	G->D->A
From H	0.648	H->G->D->A
Sum	**4.902**	

City B		
Source City	Max Probability	Max Prob Path
From A	0.8	A->B
From C	0.8	C->B
From D	0.72	D->A->B
From E	0.6	E->B
From F	0.72	F->C->B
From G	0.576	G->D->A->B
From H	0.51	H->G->D->A->B
Sum	**4.726**	

City C		
Source City	Max Probability	Max Prob Path
From A	0.7	A->C
From B	0.8	B->C
From D	0.63	D->A->C
From E	0.72	E->F->C
From F	0.9	F->C
From G	0.63	G->F->C
From H	0.63	H->F->C
Sum	**5.01**	

City D		
Source City	Max Probability	Max Prob Path
From A	0.9	A->D
From B	0.72	B->A->D
From C	0.63	C->A->D
From E	0.48	E->F->D
From F	0.6	F->D
From G	0.8	G->D
From H	0.72	H->G->D
Sum	**4.85**	

City E		
Source City	Max Probability	Max Prob Path
From A	0.504	A->C->F->E
From B	0.6	B->E
From C	0.72	C->F->E
From D	0.48	D->F->E
From F	0.8	F->E
From G	0.56	G->F->E
From H	0.6	H->E
Sum	**4.264**	

City F		
Source City	Max Probability	Max Prob Path
From A	0.63	A->C->F
From B	0.72	B->C->F
From C	0.9	C->F
From D	0.6	D->F
From E	0.8	E->F
From G	0.7	G->F
From H	0.7	H->F
Sum	**5.05**	

City G		
Source City	Max Probability	Max Prob Path
From A	0.72	A->D->G
From B	0.576	B->A->D->G
From C	0.63	C->F->G
From D	0.8	D->G
From E	0.56	E->F->G
From F	0.7	F->G
From H	0.9	H->G
Sum	**4.886**	

City H		
Source City	Max Probability	Max Prob Path
From A	0.648	A->D->G->H
From B	0.5184	B->A->D->G->H
From C	0.63	C->F->H
From D	0.72	D->G->H
From E	0.6	E->H
From F	0.7	F->H
From G	0.9	G->H
Sum	**4.7164**	

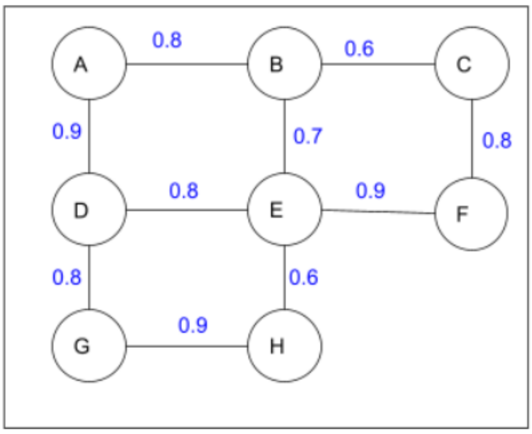
From these calculations, we see that city F is the most reliable destination to travel to from any other city.

To implement this, we used a function named – *ReliableDest()* which does the computation shown above for every city. This function declares a dictionary *'dict'* and a list *'vertices'*. The *'vertices'* stores all the cities present in the graph. A nested *'for loop'* is then run over all the cities in the *'vertices'*. The inner *'for loop'* runs the *BestRoute()* function to compute the maximum probability between any two cities. Once, the entire loop runs for a particular source and destination, it computes the maximum of all probabilities computed for the different routes between a specific source and destination and stores it in a variable called *'sum'*. As already mentioned, the probabilities of all the routes and the routes themselves are stored in a dictionary named *'list1'*. Below is the computation of *'sum'* we are doing -

sum = sum + max(self.list1.values())

This maximum sum of probabilities for all possible routes from a specific source is then stored in the *'dict'* dictionary. In the end, we print the maximum probabilities with which all other cities can be reached from a particular source city.

Considering **example 2** given below -

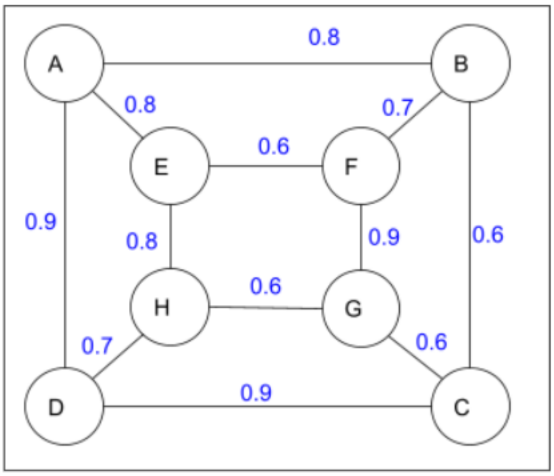


(i) what route will maximize the probability to arrive on time between C and A, and
(ii) what city among {A, B, C, D, E, F, G, H} is the most reliable travel destination.

With the same implementation as discussed for example 1, we get the answer to **(i)** as - path C->F->E->D->A with a probability of probability 0.5184 and city D as the highest reliable city among all for **(ii)**

For **example 3** below, we compute the following -

(i) what route will maximize the probability to arrive on time between E and C, and
(ii) what city among {A, B, C, D, E, F, G, H} is the most reliable travel destination.



With the same implementation, the answer to **(i)** is path E-A-D-C with a probability of 0.648 and for **(ii)** city A is the most reliable city.

Pseudocode -

```
create a Graph class
    initialize the class with graph and empty list1{}, path[], visited{}
    set the visited list = False for all other vertices

    define a function printGraph(self):
        print graph

    define function BestRoute(self, src, dest):
        set the visited[i] for the src = True
        append this src in path[]
        set prod = 1

        if the dest is found:
            for i in the range 0 to length(path):
                prod = prod * probability of the path between the current and
                previous city
            store the prod and the path in list1{}
            set prod = 1
        else:
            for city in graph[src]:
                if visited[src][city] = False and graph[src][city] is not = 0:
                    BestRoute(city,dest)
        pop the last element from the path[]
        set visited[src] = False

    define printBestRoute(src,dest):
        call BestRoute(src,dest)
        print list1 containing the path and probability
        retrieve the maximum probability and its corresponding path in max_path and
        max_value
        print max_path and max_value

    define function ReliableDest(self):
        create empty dictionary dict{} and list vertices[]
        store the cities from graph into vertices[]to access them

        for i in the range 0 to length(vertices): # Access destination in vertices[]
            set sum = 0
            for j in the range 0 to length(vertices): # Access source in vertices[]
                if destination is present in dict of source and i is not = j:
                    call BestRoute(vertices[j], vertices[i])
                    store the maximum probability for the paths obtained in
                    list1{} in sum
                    set the visited[] list for source cities traversed as False
                    empty list1{} and path[]
```

store the *sum* in *dict[dest]*

print *dict*

print the most reliable city *dict.get* with the maximum *dict.values()*

inside the *main()* function

take the file name as input from the user

read the content of the input file

close the file and print the file content

take the source city *src* as input from the user for the graph

take the destination city *dest* as input from the user for the graph

create an object of the graph and call the functions *printBestRoute(src, dest)* and *ReliableDest()*

Instructions to run the Code:

The code has been implemented in Python. The name of the file is *'SavvyTraveller.py'*. The three examples have been incorporated in three separate files – *graph1.txt* (for example1), *graph2.txt* (for example 2), and *graph3.txt* (for example 3).

1. Save the SavvyTraveller.py, graph1.txt, graph2.txt, and graph3.txt in your desired location.

2. In the terminal, change the directory to the folder where you have saved these files using the command *cd <pathname>*.

3. Type *'python3 SavvyTravller.py'*.

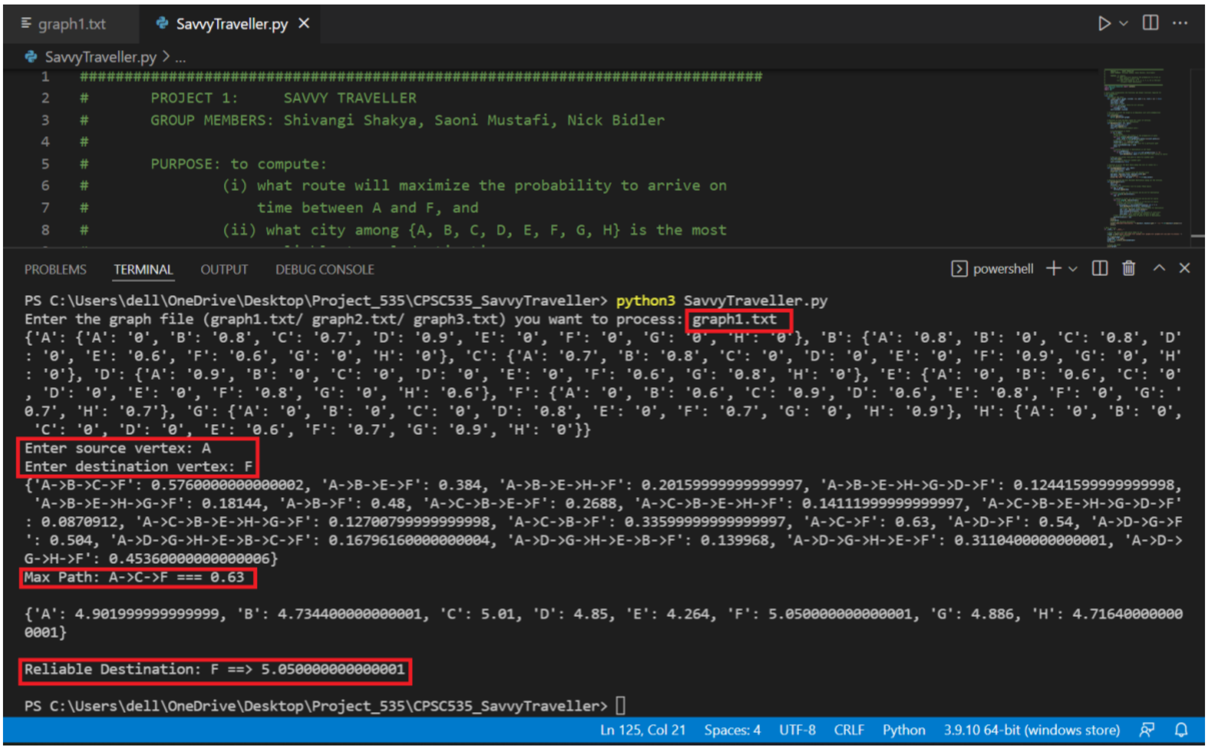
4. Enter the file name you want to run the program for – *'graph1.txt'* to execute for the graph provided in example 1, *'graph2.txt'* for example 2 graph and *'graph3.txt'* for graph in example 3.

5. Enter source city and destination city for the graph you just provided as input.

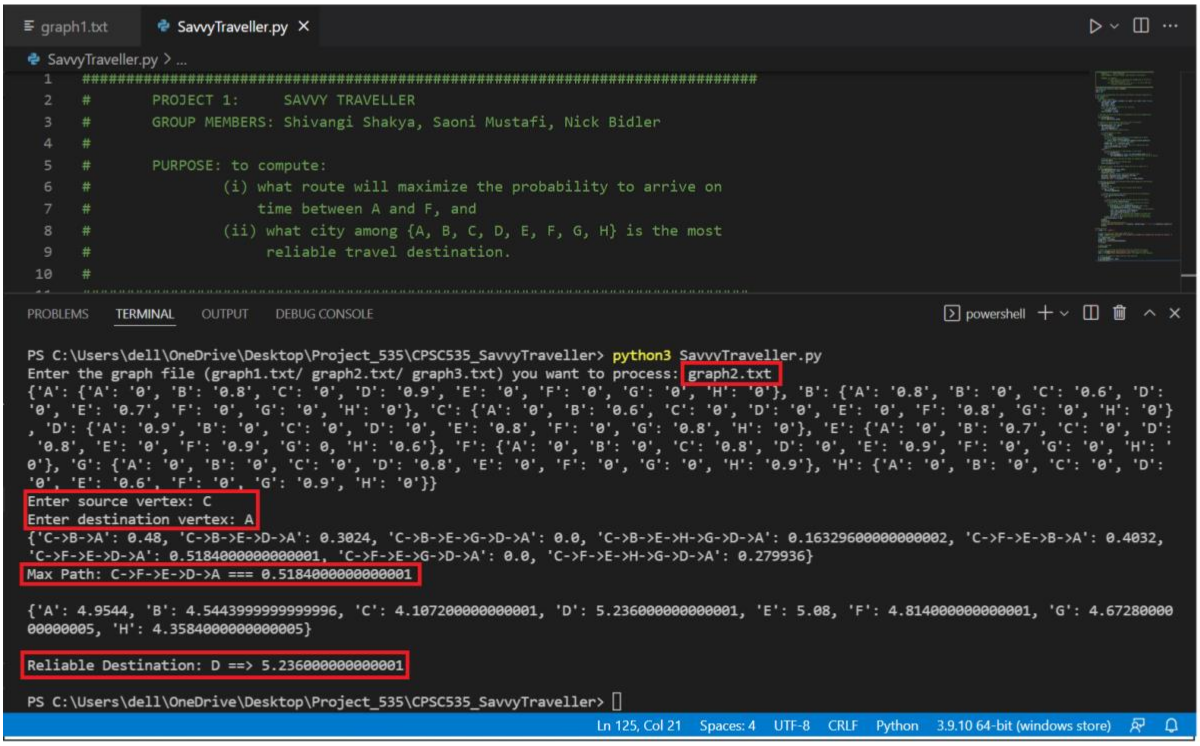
Screenshots -

Below are the screenshots of the three examples' code execution output and the team.

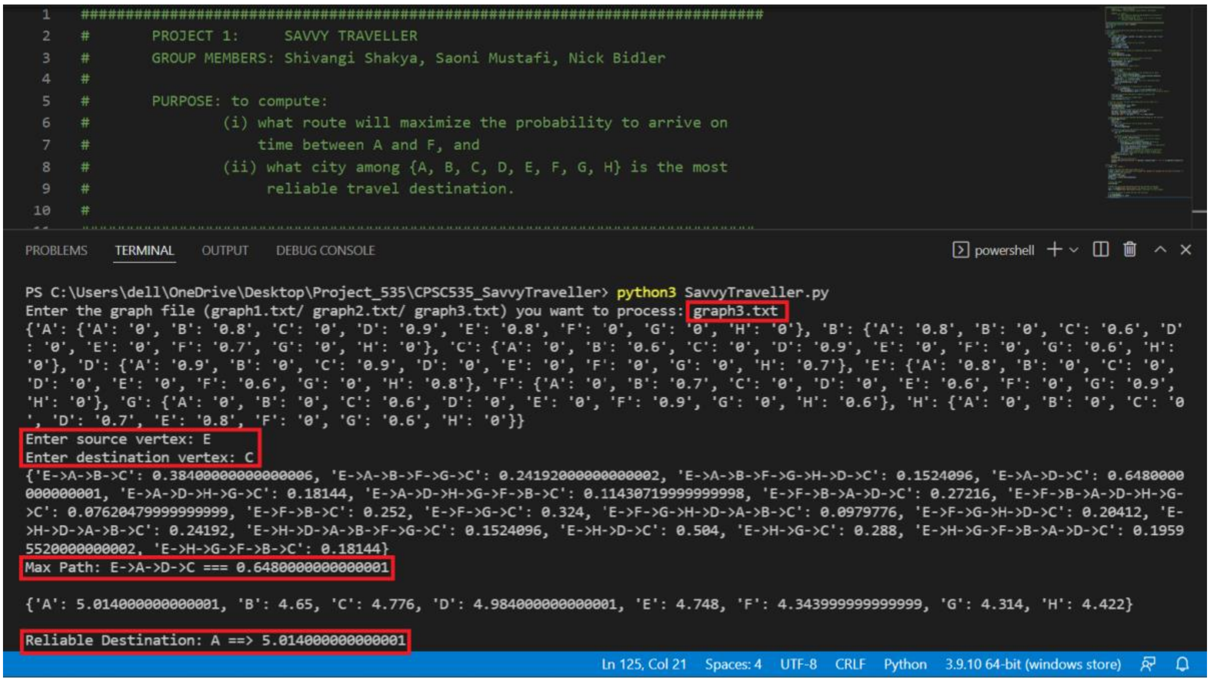
Screenshot of example 1:



Screenshot of example 2:



Screenshot of example 3:



Screenshot of the Group: