CSE 6400
Lab Assignment #1
Due: Thursday, October 10, 11:59 PM

You are allowed to work either individually or in a group of two (if working in a group of two, each person must still make his/her own submission and you must indicate the contribution of each person). For this lab assignment, you are given a starting location and goal location (or destination), and must find a path to the goal location. In this problem, the optimal path is the one which has the shortest travel time. You will be implementing the following four search algorithms: **Breadth-First Search (BFS), Depth-First Search (DFS), Uniform Cost Search (UCS), and A* Search.** **Please follow the algorithm definitions/pseudocode which we have covered during lecture (please see the lecture slides on Canvas).**

Your program will read an input.txt file. The input.txt file will specify one of the four searching algorithms, as well as starting and goal locations. In addition, the input.txt file will contain **live traffic** information, which is a list of current traveling times (in minutes) between different locations. For example, suppose the starting location is your home and the destination is Staples Center. An example of live traffic data is as follows:

```
Home CA73/NewportCoastDr 5
CA73/NewportCoastDr I405/CA73 10
I405/CA73 I110/I405 25
I110/I405 I10/I110 20
I10/I110 I110/I405 30
I10/I110 I10/I405 9
I105/I110 I10/I110 7
I10/I110 StaplesCenter 3
```

Here, each line indicates the current traveling time between two locations. For example, line 3 indicates that it takes 25 minutes to go from I405/CA73 to I110/I405. Note that the location can be an intersection (such as I405/CA73).

Traveling time may not be the same for both directions. For example,

```
I110/I405 I10/I110 20
```

indicates that it takes 20 minutes to travel from `I110/I405` to `I10/I110`, but

```
I10/I110 I110/I405 30
```

indicates that it takes 30 minutes in the other direction.

Beside the live traffic information, you also have an idea of how long it takes on a traffic-free Sunday from each location to your goal location. Hence, the input.txt file will also contain the **Sunday traffic** estimate of traveling time from each location listed in the file to your destination. For the example above, the Sunday traffic data may look like this:

```
Home 55
CA73/NewportCoastDr 50
I405/CA73 40
I110/I405 28
I10/I110 8
I10/I405 39
I105/I110 23
StaplesCenter 0
```

Your program should write, in an output.txt file, the list of locations traveled over in your solution path (including the starting and goal locations) and the **accumulated** time from start to each location, in order of travel. The following is an example of output.txt:

```
Home 0
CA73/NewportCoastDr 5
I405/CA73 15
I110/I405 40
I10/I110 60
StaplesCenter 63
```

The full specification of input.txt and output.txt is shown on the next page.

Full specification for input.txt:

```
<ALGO>
<START STATE>
<GOAL STATE>
<NUMBER OF LIVE TRAFFIC LINES>
<... LIVE TRAFFIC LINES ...>
<NUMBER OF SUNDAY TRAFFIC LINES>
<... SUNDAY TRAFFIC LINES ...>
```

where:

<ALGO> is the algorithm to use and will be one of the following: "BFS", "DFS", "UCS", "A*"

<START STATE> is a string with the name of the start location (e.g., "Home")

<GOAL STATE> is a string with the name of the goal location (e.g., "StaplesCenter")

<NUMBER OF LIVE TRAFFIC LINES> is the number of lines of live traffic information that follow.

<... LIVE TRAFFIC LINES ...> are lines of live traffic information in the format previously described, i.e., <STATE1> <STATE2> <TRAVEL TIME FROM STATE1 TO STATE2>

<NUMBER OF SUNDAY TRAFFIC LINES> is the number of lines of Sunday traffic estimates that follow.

<... SUNDAY TRAFFIC LINES ...> are lines of Sunday traffic information in the format previously described, i.e., <STATE> <ESTIMATED TIME FROM STATE TO GOAL>

Full specification for output.txt:

Any number of lines with the following format for each line:

```
<STATE> <ACCUMULATED TRAVEL TIME FROM START TO STATE>
```

******Important guidelines on next two pages******

Guidelines:

Your program should not require any command-line argument. It should read a text file called "input.txt" in the current directory that contains a problem definition. It should write a file called "output.txt" (in the current directory) with your solution. Format for input.txt and output.txt is specified in the previous page.

Samples of input and output are shown on the last few pages of this assignment description. The goal of the samples is to check that you can correctly parse the problem definitions, and generate a correctly formatted output. Your output should match the specified format exactly. It should not be assumed that if your program works on the samples it will work on all test cases. We will run your program on additional test cases and it is your task to make sure that your program will work correctly on any valid input. You are encouraged to come up with additional test cases and test your program on them.

Your code should be written in either C++, Java, or Python. Please submit your source code to the Lab #1 dropbox on Canvas. Please also submit a README file that indicates how to compile/run the code and indicates the Python or C++ version number. In addition, if working in a group of two, each person must still make his/her own submission, and in the README file, please indicate the contribution of each person. We will be using our own test cases, so please do not include files like input.txt or output.txt in your submission.

We will not be able to grade your assignment if your code does not compile, or fails to load and parse input.txt, or writes an incorrectly formatted output.txt. So, please make sure to test your program using the provided samples.

Please make sure that you read START STATE and GOAL STATE from input.txt and do not assume they will be the same as in the samples.

Times are non-negative integers (32-bit or larger).

The input filename should be "input.txt" and the output filename should be "output.txt" (note that these filenames are case-sensitive).

Note that the text in the input.txt and output.txt files (such as state names or other names like "BFS") is case-sensitive.

Guidelines (Continued)

State names (such as “StaplesCenter”) will not have any white spaces in them and will consist of any number of the following characters: a-z, A-Z, 0-9, /

There will not be any duplicate entries in the live traffic data.

There will be exactly one entry in the Sunday traffic data for each state mentioned in the live traffic data.

In output.txt, the first line should always be “<START STATE> 0” and the last line’s state should always be <GOAL STATE> (and the total accumulated travel time should follow). Hence, if the start state is the same as the goal state, you should output a single line “<START STATE> 0”.

The goal state will always be reachable from the start state, so you do not need to worry about cases where no path can be found from start to goal.

There are multiple definitions around the web and in various textbooks for the algorithms that you will implement. Different implementations may sometimes give different results. **Please follow the algorithm definitions/pseudocode which we have covered during lecture (please see the lecture slides on Canvas).**

Please use the following tie-breaking strategy in your implementation. When expanding a node, there is a tie if there are multiple children (in the case of BFS and DFS), multiple children of equal path cost (in the case of UCS), or multiple children of equal evaluation cost (in the case of A*). In this case, the children should be enqueued/inserted in the order in which they are listed in the live traffic data. In addition, suppose you have a node already in the queue with path cost (or evaluation cost) c, and a newly generated node with path cost (or evaluation cost) c. In this case, the new node should be enqueued after the older one that was already in the queue.

******Please see input/output samples on next two pages*****

Example 1: Consider this input.txt:

```
BFS
A
D
4
A B 5
A C 3
B D 1
C D 2
4
A 4
B 1
C 1
D 0
```

Should yield the following output.txt:

```
A 0
B 5
D 6
```

Example 2: Consider this input.txt:

```
UCS
A
E
5
A B 80
A D 90
B C 10
D E 20
C E 20
5
A 50
B 40
C 30
D 20
E 0
```

Should yield the following output.txt:

```
A 0
D 90
E 110
```

<u>Example 3:</u> Consider this input.txt:

```
DFS
A
E
6
A C 1
A B 1
B C 1
B D 1
C E 1
D E 1
5
A 2
B 1
C 1
D 1
E 0
```

Should yield the following output.txt:

```
A 0
B 1
D 2
E 3
```

<u>Example 4:</u> Consider this input.txt:

```
A*
A
D
4
A B 3
A C 3
B D 2
C D 1
4
A 4
B 2
C 1
D 0
```

Should yield the following output.txt:

```
A 0
C 3
D 4
```