CSCI 561: Foundations of Artificial Intelligence

Instructor: Prof. Laurent Itti

Homework #1: Uninformed Search

Due on September 24 at 11:59pm, 2014

Andy is a graduate student at USC. He is shy and has no idea how to connect to his dream girl, ***Alice***. He saw her at the school orientation, but he couldn't find the opportunity to approach her. One day ***Andy*** reads an article on the Internet: "Six degrees of separation is the assumption that everyone and everything is six or fewer steps away, by way of introduction, from any other person in the world, so that a chain of a-friend-of-a-friend statements can be made to connect any two people in a maximum of six steps." This assumption strongly encourages ***Andy***. He thinks if he were introduced to the friend of his friend, he eventually would meet his true love, Alice. The question is how to build the link from Andy to Alice.

Now Andy needs your help. You know the social network around Andy as Fig. 1, and your job is planning the path for Andy to meet Alice. In Fig. 1, each node is a person, each link is the friendship, and the number of the link means the distance of their relationship. For example, ***Andy*** feels closer to ***Calvin(2)*** than ***Beck(10)***.

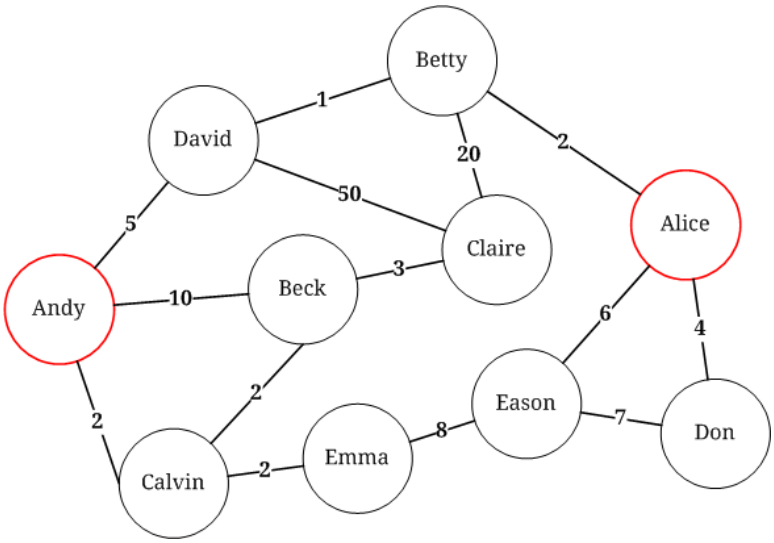

Figure 1: the social network around Andy

There are some constraints when you plan the link:

1. ***Andy*** is so shy that he can only be introduced to a person through a mutual friend. For example, if you plan ***Andy*** to meet ***Eason***, ***Andy*** must be first introduced to ***Emma*** via ***Calvin***, and then let ***Emma*** introduce ***Andy*** to ***Eason***. This creates a link such as ***Andy-Calvin-Emma-Eason***.
2. ***Andy*** wants to know ***Alice*** ASAP. If a mutual close friend introduces ***Andy*** to someone, he would know this person faster. For example, to reach ***Claire***, the time cost of ***Andy-Beck-Claire*** is **13** but the one of ***Andy-David-Claire*** is **55**, so **Andy** prefers to know ***Claire*** by ***Beck***.

Programming Task

You will need to write a program to implement the following search algorithms, to help ***Andy*** find optimal traversal link(s) to reach ***Alice***.

(1) Breadth-first search (30% of test cases)
(2) Depth-first search (30% of test cases)
(3) Uniform-cost search using the distance of their relationship as cost (40% of test cases)

Note: When the costs of two or more nodes are equal, you need to make sure these nodes are ***popped off the search queue in alphabetical order.*** This will resolve ambiguity and ensure that there is only one correct solution for each problem.

Note 2: Your algorithm should perform ***loop detection***. As studied in lectures 2-4, do not enqueue a child that has a state already visited, unless the child has a better cost than when we previously visited that state (see slides about "A clean robust algorithm" in session02-04.pptx lecture slides).

Note 3: You can assume that all step costs are positive or zero.

Compilability

You should take full responsibility to make your code executable. You are required to write Makefile to compile (if necessary) and run your code. In your Makefile, you are required to have 2 targets: **agent** and **run**. The **agent** target is to compile, if needed, otherwise it may do nothing. If "make **agent"** fails, your grade is **0**. The **run** target is to execute your program. If "make run" fails sometimes, you will get a partial score out of 100. We will run N test cases in total. if "**make run**" passes and the output is correct on all N cases, then your score is **100**. Otherwise, if only G tests pass out of N, then the score is 50 - 50*(N-G)/N (i.e., 50*G/N). Please refer to the example Makefile files below:

Example for a C++ agent:

```
agent: agent.cpp
	g++ agent.cpp -o agent

run: agent
	./agent
```

Example for a Java agent:

```
agent: agent.class

agent.class: agent.java
	javac agent.java

run: agent.class
	java agent
```

For python or other non-compiled languages:
agent: run: ./agent.py

Input:

You are provided with a file **input.txt** that describes the network as exemplified in Fig. 1.

<task#> bfs = 1, dfs = 2, ucs = 3
<source> the name of the first node, such as Andy in Fig. 1
<destination> the name of the target node, such as Alice in Fig. 1
<#nodes> the total number of nodes
<nodes> See the explanation below
<graph> See the explanation below

The few lines in **<nodes>** will look something like this:
Andy
Bill
Alice
This is a list of names representing the nodes of the graph.
The lines in **<graph>** of the file contains the matrix representation of the graph edges. It will look something like this:
0 5 10
5 0 2
10 2 0
The rows and columns correspond to the nodes in the same order. For example, the first row tells us that ***Andy*** is not connected to himself (0), connected to ***Bill*** with a value 5, and connected to ***Alice*** with a value 10. In this assignment, we will assume that all relationships are mutual, so the matrix is symmetric. A weight of 0 indicates that there is no edge between such two people, and any other value describes the distance of their relationship.

Output:

The program should output in the format:
<Expansion>
<Output>
<PathCost>

For example,
Stacy-Emma-Helen-Frank-Jennifer-John-Jenny-Gerald-Claire-Patrick
Stacy-Emma-Frank-Gerald-Patrick

130

The nodes (separated by "**-**") are in the order that show the link of the friendship. If your program cannot find any solution path in the graph, please output **NoPathAvailable.** The output filename is fixed to **output.txt**. The grader will examine the file **output.txt** for grading.

Examples:

Please make sure your program follows the example here.

Nodes: Andy Bob Claire David Alice

Edges: 5

Source: Andy

Destination: Alice

	Andy	Bob	Claire	David	Alice
Andy	0	1	1	0	0
Bob	1	0	5	4	0
Claire	1	5	0	0	1
David	0	4	0	0	0
Alice	0	0	1	0	0

BFS:

Log: Andy-Bob-Claire-David-Alice

Path: Andy-Claire-Alice

DFS:

Log: Andy-Bob-David-Claire-Alice

Path: Andy-Claire-Alice

Uniform-cost Search:

Log: Andy-Bob-Claire-Alice

Path: Andy-Claire-Alice

Grading Notice:

1. **If the grader is unable to compile or execute your code successfully on aludra, you will not receive any credits. Your Makefile must contain the "agent" part to compile (or do nothing if no compilation is needed) and the "run" part to execute your program. You are allowed**

to use standard libraries only, such C++ STL. You have to implement any other function or method by yourself.

2. **In the Hw1.zip, you can find two examples and the grading script. Please modify your program to follow the file IO as the given two examples and your Makefile to work with the grading script.**

Deliverables:

You are required to hand in all the code that you wrote to complete this assignment. You are free to implement in any language of your choice. However, if you code in C++ or Java, the TA will be better able to assist you. Also, we require that your code be able to run from the command line on the USC aludra.usc.edu Linux server. For information on accessing aludra, please see (https://itservices.usc.edu/web/hosting/students)

The deadline for this assignment is September 24, at 11:59pm Los Angeles time. Please turn in all materials as a .zip file via Blackboard, with the title format [firstname]_[lastname]_HW1.zip (e.g., Tommy_Trojan_HW1.zip).