CSCI 561: Foundations of Artificial Intelligence
Summer 2018

Homework 1
Due on Monday, June 18 at 11:59 PM PST

Problem Description:

Uber and Lyft drivers must decide which regions of a city to drive around in each day. Consider the following artificial scenarios involving how two ride-sharing drivers (**R1** and **R2**), who are roommates, decide which region of the city to drive around in each day. Both of the roommates are interested in maximizing their profit. For this scenario we assume that the roommates will make more money if they choose not to pick up passengers in the same regions of the city.

Assume that each morning the two roommates take turns picking a region of the map until the regions are split up between them. The roommates have a deal that once a region of the map is picked; the person who picked it is the only one who has the right to pick up new passengers in that region. For their first pick the roommates can pick any region. After they choose a region any subsequent regions they pick must be an adjacent regions (regions are adjacent if their borders touch) to a region that they have already picked. If at some point a roommate can no longer pick a region of the map that is adjacent to a region they already picked then that player may not pick any other regions and the other roommate is free to pick the rest of the regions that are adjacent to regions that the other roommate has already picked. We signify the choice of the roommate who can no longer pick regions with the symbol ***PASS.*** If neither roommate can make a choice the activity is terminated with the rest of the regions left un-chosen.

We assume that each day at midnight profitability numbers are released by the ride-sharing companies and posted online in the ***Region Profitability List (RPL)***. This list assigns numbers to each region that represents how profitable the region will be for the day for drivers who have the rights to pick up passengers there. We assume that all profitability factors are taken into account in these numbers and the number is the best criterion to use for estimating how much money you will make if you have rights to pick up passengers in that region. Therefore, each of the roommates want to maximize the sum of the numbers corresponding to the regions they choose in the picking activity, i.e.- their **total score**.

For this problem; we represent the RPL in an ordered list of tuples (Region_Identifier, Profit_Number). So for a map with 4 regions: A,B,C,D. The Region Profitability List might be (A,10),(B,8),(C,10),(D,7) where A and C have a 10 profitability number followed by B with 8 and D with 7.

Evaluation Function:

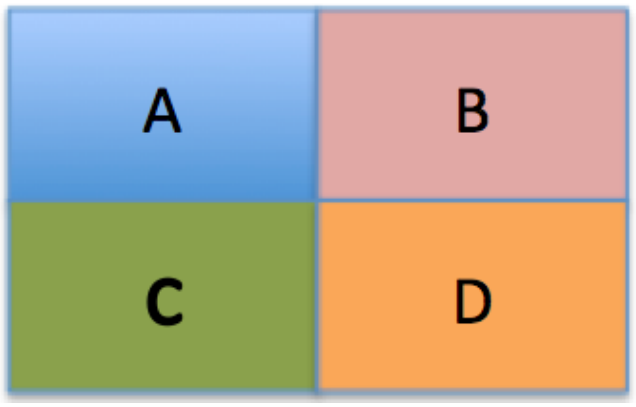
Sometimes the Region Profitability list is not available due to the ride-sharing company not having enough resources to generate the list that day. In this case the roommates pick the next region using the following heuristic evaluation function:

$$\frac{\left[\sum_{Region\ \in Map}\left(\frac{1}{\#\ of\ Regions}\right) * (Region\ Profitability\ \#\ From\ Yesterday)\right] + Next\ Region\ to\ Pick\ Profitability\ \#\ From\ Yesterday}{2}$$

In this problem we will represent a map with an adjacency matrix. The adjacency matrix is an abstract representation of a map with regions that captures the necessary information about the adjacency relationships of all the regions on the map. Each row,column value in the matrix represents whether the region associated with the column and the region associated with the row are adjacent (i.e. – whether or not their borders touch). A value of 1 indicates regions are adjacent and a value of zero indicates the regions are not adjacent. So for example consider a map with 4 regions A,B,C,D and the following adjacency matrix:

	A	B	C	D
A	1	1	1	0
B	1	1	0	1
C	1	0	1	1
D	0	1	1	1

A map with this matrix could look like:



Task:

In this assignment, you will be responsible for reporting utility values for a specific roommate for a specific point in the game. The task is to generate the tree from a given point that lets you pick an optimal choice no matter what choices have been made already. If the RPL provided is current you should output the depth-bounded node utility values of the activity search tree. If the RPL provided is stale (from yesterday) you should output the utility values yielded by the heuristic value function for the next level of nodes in the activity search tree. Your program will take the input from the file "input.txt" and print out its output to the file "output.txt". We provide file definitions and example files in the last section of this document.

Each input file contains an abstract representation of a particular picking activity (region profitability list, and the map adjacency matrix) and a particular state in that activity (whose move it is (roommate 1 (R1) or roommate 2 (R2), and the choices (region picks) up to that point in the activity). The input file will also specify a search cut-off depth D and the day the region profitability list was posted.

In the case of a current RPL, your program should output the utility values for the nodes at depth D. That is, the leaf nodes of the corresponding activity tree should be either an activity position after exactly D picks (alternating between R1 and R2) or an end-activity position after no more than D picks. In the case of a stale RPL, your program should output the utility values according to the heuristic function for the nodes of the activity tree that correspond to the next pick of the specified room-mate.

File Formats:

Input.txt

<DAY>

Contains "*today" or "yesterday" indicating which day the RPL was posted*

<PLAYER>

Contains either *"R1" or "R2"* indicating which roommate has next turn

< REGION PROFITABILITY LIST>

Ordered list of tuples (Region_Identifier, Profit_Number).

<**ADJACENCY MATRIX ROWS**>

The rows of the adjacency matrix representing the map.

<**REGIONS PICKED SO FAR>**

Comma separated list of regions picked so far. Will contain "*" if no activity yet.

<**MAX DEPTH**>

# that determines the maximum depth of your search tree

Output.txt

<NEXT REGION TO PICK>

The next region the <PLAYER> should pick.

<UTILITY VALUE LIST>

In the case of a current RPL: A comma separated list of the utility of the <PLAYER> at the terminal nodes of the search tree (considering the maximum depth of <MAX DEPTH>) for the activity under the assumption that both players select moves maximizing their utilities.

In the case of a stale RPL:

A comma separated list of the utility of the <PLAYER> at the next pick nodes under the assumption that both players select moves maximizing their utilities according to the heuristic function.

**FILE SHOULD HAVE NO BLANK LINE*

Expansion order and Tie breaking:

If the specified player has more than one region to choose from you should sort the regions into alphabetical order based on Region_Identifiers extracted from the **< REGION PROFITABILITY LIST>** in the input file. The tree should ***always*** be expanded based on ***alphabetical*** order.

If more than one region results in the same utility for the player; your code should ***always*** choose the region with a Region_Identifier closest to the beginning of the alphabet.

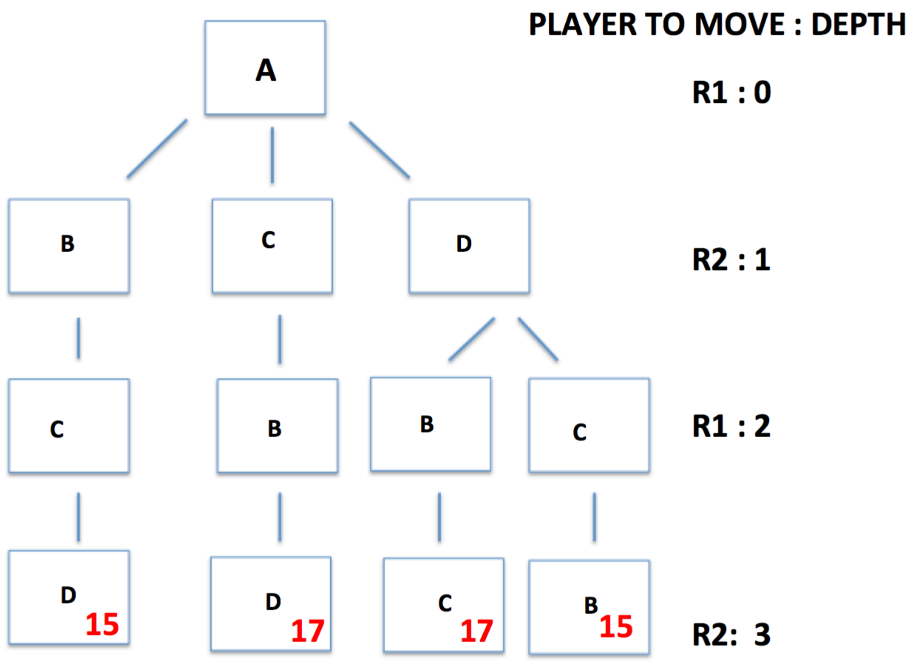
Explanatory Examples:

Input File Example 1:
Today
R2
(A,10),(B,8),(C,10)(D,7)
[1,1,1,0]
[1,1,0,1]
[1,0,1,1]
[0,1,1,1]
A
3

Output File Example 1:
C
15,17,17,15

Map for Example 2: *see above map in hw description.*

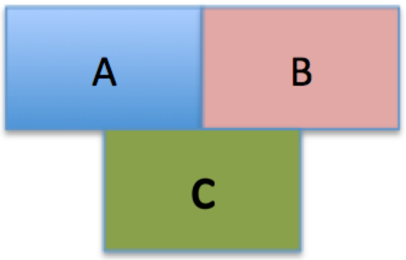
Explanatory Search Tree Example 1:
We note the terminal nodes in this tree all have the ***total score*** *(utility value)* in red associated with the regions R2 has picked as shown on the associated path of the tree. We also note the R2 optimal next move is C according to the assumption that R1 will pick optimally next turn. We note that the terminal nodes in this tree are at depth 3 as specified by *input.txt* even though these are not the terminal nodes of the activity tree.



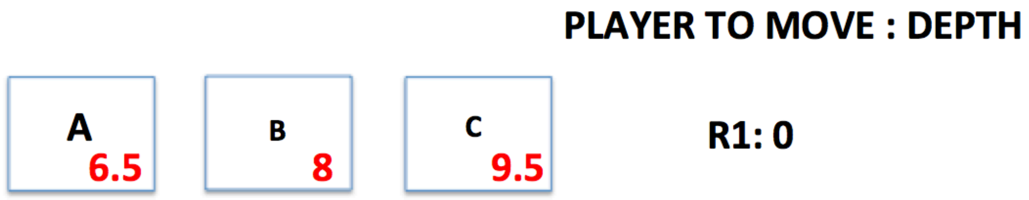
Input File Example 2:
Yesterday
R1
(A,5),(B,8),(C,11)
[1,1,1]
[1,1,1]
[1,1,1]
*
0

Output File Example 2:
C
6.5,8,9.5

Map for Example 2:



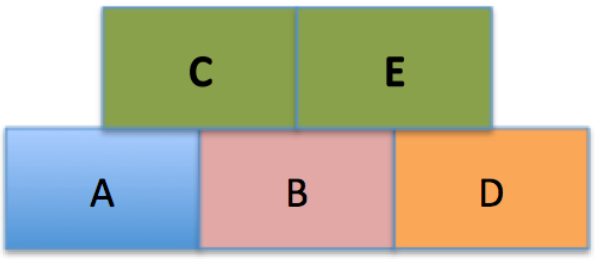
Explanatory Tree Example 2:
Since the input file specified no activity yet we must consider activity trees associated with each initial pick. Since the RPL is stale we output the utility values generated from using the heuristic function (as opposed to total score) on all of the possible next moves for roommate R1 which in this case are the root nodes of the possible activity trees.



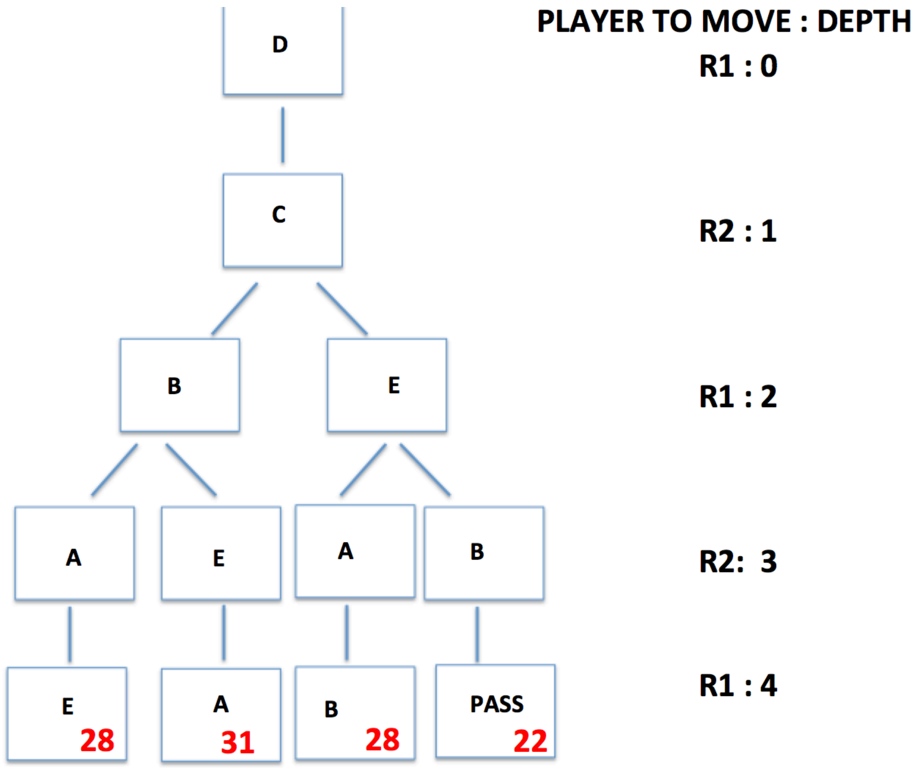
Input File Example 3:
Today
R1
(A,12),(B,6),(C,10),(D,13),(E,9)
[1,1,1,0,0]
[1,1,1,1,1]
[1,1,1,0,1]
[0,1,0,1,1]
[0,1,1,1,1]
D,C
4

Output File Example 3:
B
28,31,28,22

Map for Example 3:



Explanatory Tree Example 3:
Here is the activity tree associated with the input file for example 3. Roommate 1 picked D first followed by roommate 2 who picked C. The rest of the tree is generated to depth 4 as specified although the activity tree is not fully generated to terminal leaves yet. R1's next best pick according to maximizing utility is Region B as specified in the output file when examining the final total score values at depth 4. We note that R1 must Pass in one circumstance as there are no available adjacent regions to D and E.



Grading Notice:

Please follow the instructions carefully. Any deviations from the instructions will lead your grade to be zero for the assignment. If you have any doubts, please use the discussion board on Piazza. Please do ***not*** post code snippets on Piazza. Do not assume anything that is not explicitly stated.

•You must use PYTHON (Python 2.7) to implement your code. You must not use any other Python libraries besides the default libraries provided by the Python 2.7 environment (Python Standard Library). You must implement any other functions or modules by yourself.

•You need to create a file named "hw1cs561s18.py". The command to run your program will be as follows: (When you submit the homework on vocarium, the following command will be executed automatically by the grading script. python hw1cs561s18.py –i <inputFile>where <inputFile>is the filename of the input file that your program needs to read.

•The input and output files use UNIX line endings ("\n").

•The generated output file needs to be named as "output.txt".

•Submit your code through Vocareum. Please refer to http://help.vocareum.com/article/30-getting-started-students for help with the system. Please only upload your code to the "/work" directory. Don't create any subfolders or upload any other files.

•If we are unable to execute your code successfully, you will not receive any credits.

•You will get partial credit based on the percentage of test cases that your program gets right for each task.

•Per test case, the output file is considered correct only if it exactly matches the solution (string-based matching)

.•String comparisons (alphabetical or alphanumerical) must follow the ASCII ordering of the characters.

•Your program should handle each test case within 3 minutes. The complexity of test cases is similar to, but not necessarily the same as, the ones provided in the assignment description. The deadline for this assignment is Monday, June 18 11:59pm. No late homework will be accepted. Any late submissions will not be graded. Any emails for late submission will be ignored.