

**Submission instruction:** Students will submit the assignment using an online system. The student will receive the submission link via email. Student should submit the homework in a single document, in the same format as the assignment. There will be enough space for students to answer the questions in the assignment document. Make sure the page numbers are in proper order. Failure to follow instructions will result in delay and/or errors in grading.

Please do not use additional pages.

**Due Date:** February 10th

- **Late homework:** you lose 20% of the homework's grade per 24-hour period that you are late. Beware, the penalty grows very fast:  $\text{grade} = \text{points} * (1 - n * 0.2)$  where  $n$  is the number of days late ( $n=0$  if submitted on time,  $n=1$  is submitted between 1 second and 24h late, etc).
- **Grade review/adjustment:** Requests will be considered up to 2 weeks after the grade is released. After that, it will be too late and requests for grading review will be denied.
- Homework assignments are to be solved **individually**.
- You are welcome to discuss class material in review groups, but do not discuss how to solve the homework.

## Question-1 [20 Points]:

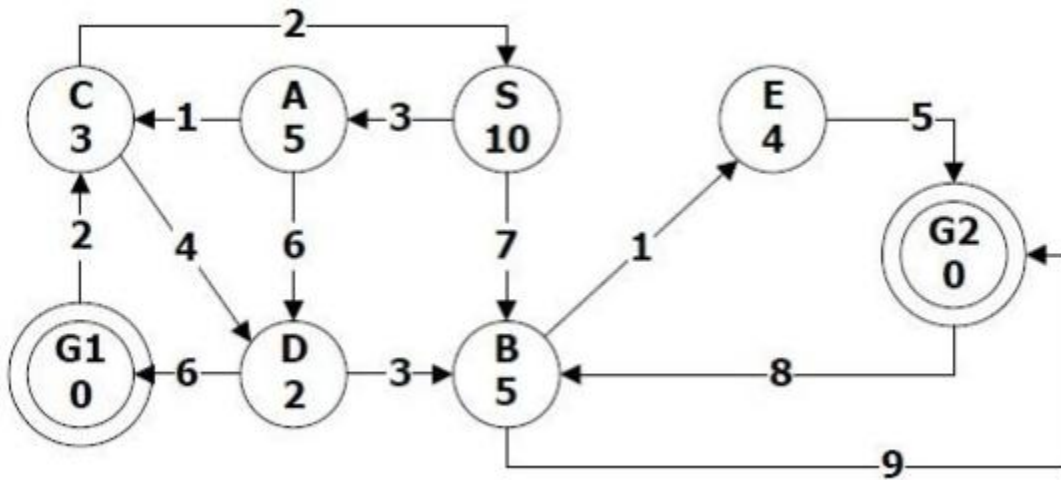
For each of the following activities, give a PEAS description of the task environment and characterize it in terms of the properties listed in Section 2.3.2. State your assumptions clearly if any.

- [5 Points] Playing soccer
- [5 Points] Bidding on an item at an auction
- [5 Points] Performing a high jump
- [5 Points] Knitting a sweater



## Question-2 [25 Points]:

Assume that you have the following search graph, where  $S$  is the start node and  $G1$  and  $G2$  are goal nodes. Edges are labeled with the cost of traversing them and the estimated cost to a goal is reported inside nodes.



For each of the search strategies listed below,

- Indicate which goal state is reached if any
- List, in order, the states expanded (Break the ties in alphabetical order)
- State the number of visited (i.e. expanded) nodes (including goal node)

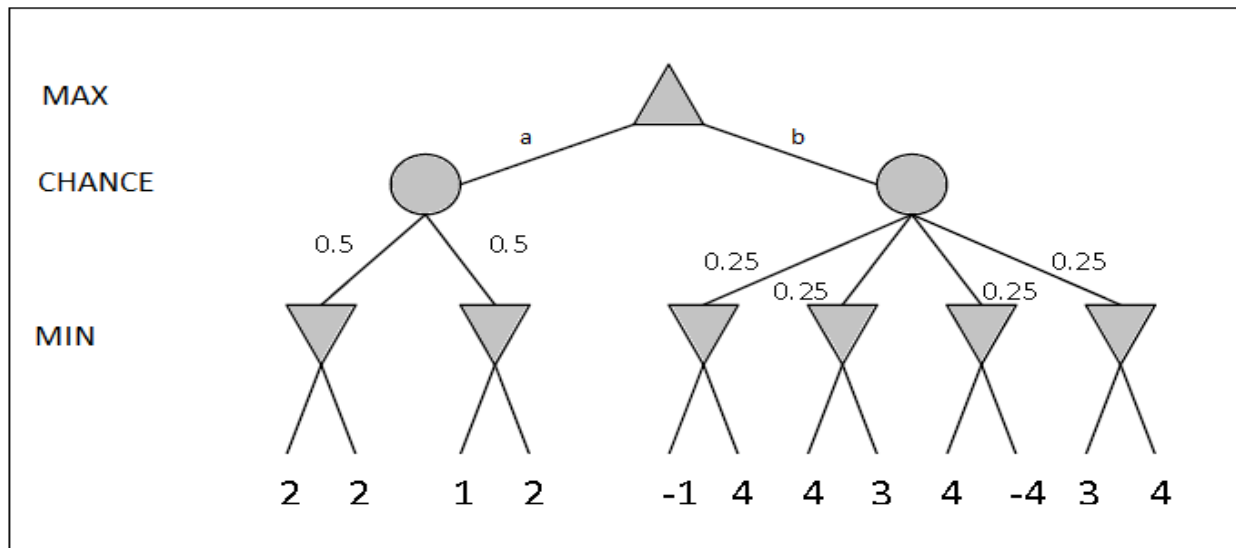
- [5 Points] *Breadth-first*
- [5 Points] *Depth-first*
- [5 Points] *Bidirectional* (Start from  $S$ , and both  $G1$  and  $G2$ )
- [5 Points] *Best-first* (using  $f=h$ )
- [5 Points] *A\** (using  $f=g+h$ )





### Question-3 [20 points]:

In this question we tackle the problem of stochastic games. We will be using the  $\text{EXPECTIMINIMAX}(s)$  function described in section 5.5 in the book to evaluate the value for the MAX node. The figure below shows the complete game tree. Assume that the evaluation order of the leaf nodes are from left-to-right order, and before the leaf node is evaluated, we do not know the value of that leaf node or any other node to right side of that leaf node. You may assume that the possible range of values is -4 to 4.



- a) [4 Points] Copy the figure, mark the value of all the internal nodes (including the chance node), and indicate the best move to the root with an arrow.

- b) [4 Points] Given the values of the first eight leaves (counting from left to right), do we need to evaluate the ninth and tenth leaves? Explain why.
- c) [4 Points] After the first two leaves are evaluated, what is the value range for the left-hand chance node?



- d) [4 Points] Circle all the leaves that need not be evaluated assuming the evaluation order of left-to-right. Explain why.

## Question-4 [15 Points]:

Given a set of locations and distances between them, the goal of the Traveling Salesman Problem (TSP) is to find a shortest tour that visits each location exactly once. We would like to solve the TSP problem using a greedy hill-climbing algorithm. Each state corresponds to a permutation of all the locations (called a *tour*). The operator *neighbors(s)* generates all neighboring states of state *s* by swapping two locations. For example, if  $s = \langle A-B-C \rangle$  is a tour, then  $\langle B-A-C \rangle$ ,  $\langle C-B-A \rangle$  and  $\langle A-C-B \rangle$  are the three neighbors generated by *neighbors(s)*. We can set the evaluation function for a state to be the total distance of the tour where each pair wise distance is looked up from a distance matrix. Assume that ties in the evaluation function are broken randomly. (Note: We don't consider a tour as returning to the start city.)

a) [3 Points] If you have  $n$  locations, how many neighboring states does the *neighbors(s)* function produce?

b) [3 Points] What is the total size of the search space, i.e. how many possible states are there in total? Assume again that there are  $n$  locations.

c) [9 Points] Imagine that a student wants to hand out fliers about an upcoming programming contest. The student wants to visit the Leavey Library (L), Salvatori Computer Science Center (S), Olin Hall (O), and Kaprielian Hall (K) to deliver the fliers. The goal is to find a tour as short as possible. The distance matrix between these locations is given as follows:

	<b>L</b>	<b>S</b>	<b>O</b>	<b>K</b>
<b>L</b>	0	0.6	0.9	0.7
<b>S</b>	0.6	0	0.3	0.2
<b>O</b>	0.9	0.3	0	0.4
<b>K</b>	0.7	0.2	0.4	0

The student starts applying hill-climbing algorithm from the initial state: <L-O-S-K>. What is the next state reached by hill-climbing, or explain why there is no neighboring state. When will we know if we should stop or continue the search? Will we know if the state is a global optimal solution when we stop? Briefly explain your answers.



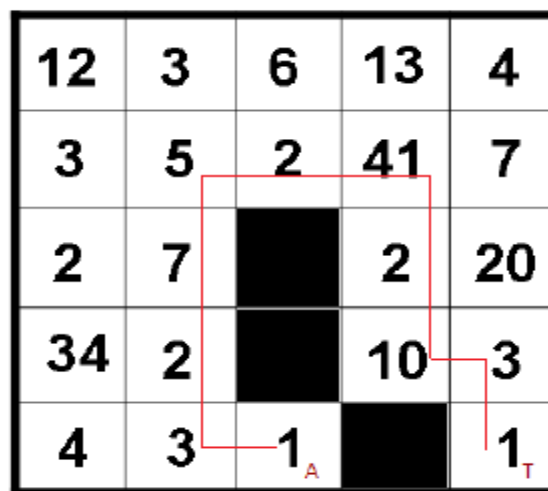
## Question-5 [20 Points]:

In this programming problem, you are required to get familiar with *python* programming language and debug an algorithm implemented in python. We will be solving a navigation problem of an agent in a grid world.

### Problem Description:

Consider grid-worlds with square cells. Cells are either blocked or unblocked. The start cell of the agent is unblocked. The agent can only move from its current cell to one of the four adjacent cells within the proximity of the grid at each step (no diagonal moves allowed), the cost of move to blocked cells is infinity (which means the agent cannot move across the blocked cell), for unblocked cell, the cost is as given in the input grid. Its objective is to move from its start cell to a given goal cell minimizing the cost. The path cost is calculated as the sum of the cost of traversing all the cells in the path. The path cost for the path shown (red line) in the figure below is 76.

The following figure shows an example of 5\*5 grid terrain with black cells referring to the blocked cells, white cells referring to the unblocked cell. The number inside each cell indicates the cost to reach the cell. Here, Start cell = Agent (A). Goal cell = Target (T).



For the problem above, you are given a python program that implements *Breath First Search (BFS)* and *Uniform Cost Search (UCS)* search strategies. You are required to analyze BFS in order to get familiar with programming in python. Then you need to debug the UCS function implemented in the program in order for it to function correctly.

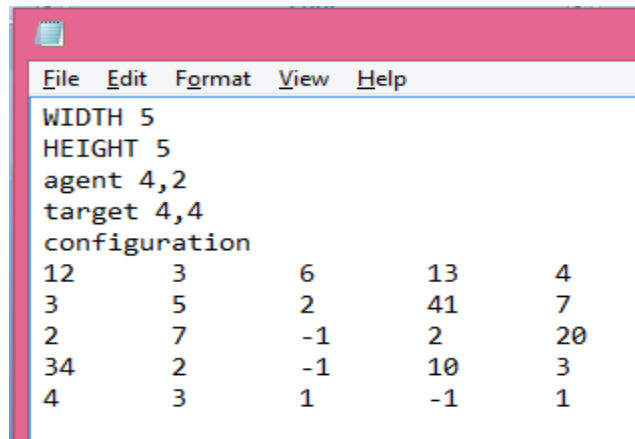
**Files:**

*searchPath.py*: python program to solve the navigation problem

*inputfilex.txt*: input files to the python program describing the grid environment. 'x' indicates the input file number

**Input:**

6 input files are provided, which are named from “inputfile0.txt” to “inputfile5.txt”. The format of a sample input file is given below.



```
File Edit Format View Help
WIDTH 5
HEIGHT 5
agent 4,2
target 4,4
configuration
12      3      6      13      4
3       5      2      41      7
2       7      -1     2      20
34      2      -1     10     3
4       3      1      -1     1
```

The size of the input grid is defined by “WIDTH” and “HEIGHT” which corresponds to the number of cells in each row and the number of cells in each column respectively. Cell(0,0) is top-left cell of the input grid. The (x,y) coordinates of the agent and target are given in the third and fourth line respectively, with x presents the vertical position, y presents the horizontal position. The blocked cells are shown as “-1”. The unblocked cells are shown as positive numbers which also indicate the costs to traverse them. The numbers in the same line are separated by tab.

**Output:**

The program shows the path between the agent cell and the target cell. Blocked cells are displayed as “X”, the Agent cell as “A”, the target cell as “T” and the cells on the path are displayed by “.”. Other cells are displayed as single white space. In the end the program displays the cost of traversing from agent cell to target cell through the path determined by the search strategy selected. The following figure shows an example of the output:

```

01234
0
1  ...
2  .X.
3  .X..
4  .AXT
Path cost: 76

```

**Running the program:**

To run the program, use the following command:

```
python searchPath.py -i inputfile_name -s search_strategy
```

where,

*inputfile\_name* is the name of the input file.

*search\_strategy* is the selected search strategy (1=BFS and 2=UCS)

The following command shows an example where it selects “inputfile0.txt” as the input file and BFS as the search strategy:

```
python searchPath.py -i inputfile0.txt -s 1
```

**NOTES:**

- 1) You can download the python program and input files from the course website (<https://courses.uscdcn.net/d2l/home/7593>). Under the “Content” tab find “Homework”. Under that you will find all the files in “HW1” folder.
- 2) We use Python 2.7 for this implementation. (<https://www.python.org/download/releases/2.7/>)
- 3) The program assumes that the input files are in the same folder as the program.

Answer the following questions:

a) [10 Points] Debug the UCS function in the program to make it function as described in Figure 3.14 in the book. Indicate the modification/addition to the program in terms of python code. Use the line number in the program as the reference to where you would modify/add code. Explain why each modification/addition was required.



b) [6 Points] For each input file, show the path from Agent cell to target cell and report the path cost for the debugged UCS function.

A\* can be thought as an enhanced version of UCS that makes use of additional knowledge for estimating the cost of a path heuristics. If we were to convert the UCS program to do A\* search, we would need to find heuristic for estimating the cost in this grid environment. With that in mind:

c) [2 Points] Give a simple argument that shows that the heuristic function directly using Manhattan Distance is consistent.

d) [2 Points] Design one admissible heuristic function other than using Manhattan Distance.