

## CS 540-3: Introduction to Artificial Intelligence Homework Assignment # 1

**Assigned: Wednesday, January 28**  
**Due: Monday, February 9 by 11:59 p.m.**

### Hand in your homework:

This homework includes two written problems and a programming problem. Turn in your answers to each of the written problems in separate pdf files called **P1.pdf** for Problem 1 and **P2.pdf** for Problem 2. The first page of each pdf file must include a header with: **your name, Wisc username, class section, HW#, date, and, if handed in late, how many days late it is.** (If you write out your answers by hand, you'll have to scan your answers and convert the files to pdfs.) The programming problem must be written in Java and must be done **individually**. All files needed to run your program, including any support files, should be submitted. Put all of these files in a folder named **<wisc username>\_HW1** where **<wisc username>** is replaced by your wisc.edu user name. Also put in this same folder the two pdf files with your answers to the written problems. Then compress this folder to create a file called **<wisc username>\_HW1.zip**. This final zip file should then be uploaded to the "HW1 Hand-In" dropbox on the course Moodle website.

### Late Policy:

All assignments are due at 11:59 p.m. on the due date. One (1) day late, defined as a 24-hour period from the deadline (weekday or weekend), will result in 10% of the total points for the assignment deducted. So, for example, if a 100-point assignment is due on a Wednesday at 11:59 p.m., and it is handed in between Wednesday 11:59 p.m. and Thursday 11:59 p.m., 10 points will be deducted. Two (2) days late, 25% off; three (3) days late, 50% off. No homework can be turned in more than three (3) days after the due date. Written problems and programming problems have the same deadline. A total of three (3) free late days may be used throughout the semester without penalty. Free late days will be used automatically on a first-needed basis. **Note that no submission will be accepted after three late days even if you have free late days left.**

Assignment grading questions must be raised with the instructor within one week after the assignment is returned.




### Collaboration Policy:

**You are to complete this assignment individually.** However, you are encouraged to discuss the general algorithms and ideas with classmates, TAs, and instructor in order to help you formulate your answers to the problems. You are also welcome to give each other examples that are not on the assignment in order to demonstrate how to solve problems. But we require you to:

- not explicitly tell each other the answers
- not to copy answers or code fragments from anyone or anywhere
- not to allow your answers to be copied
- not to get any code on the Web

## Problem 1: Search Algorithms[30]

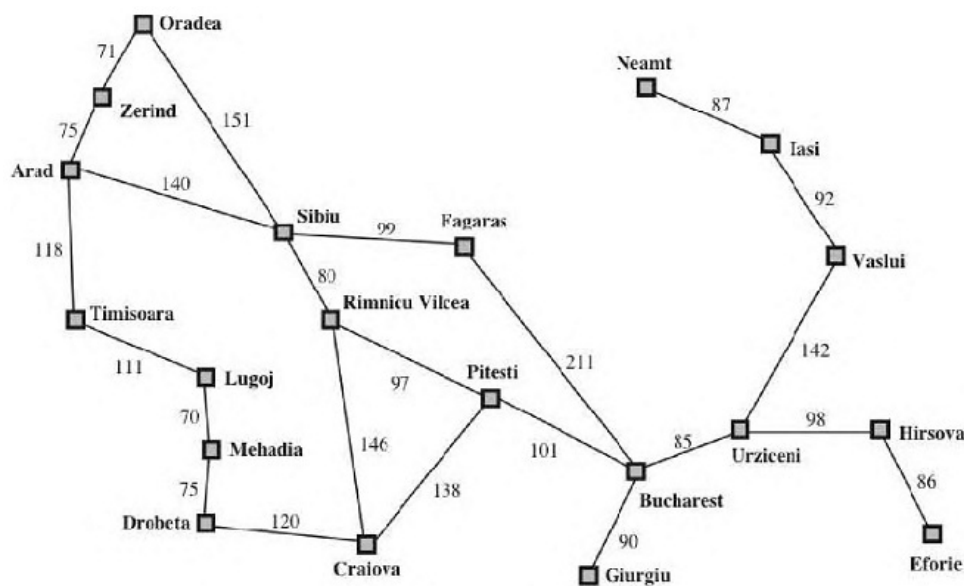
In the following figure is a modified chess board on a  $5 \times 5$  grid. The task is to capture the black king (fixed in square R) by moving the white knight (starting in square D) using only "knight moves" as defined in Chess. Assume the successor function **generates** legal moves in a clockwise order: (1 right, 2 up); (2 right, 1 up); (2 right, 1 down); (1 right, 2 down); (1 left, 2 down); (2 left, 1 down); (2 left, 1 up); (1 left, 2 up). Note that not all of these moves may be legal from a given square. It is not legal for a knight to move to a square with a wall on it (squares G and O).

A	B	C	D	E
				
F		G	H	I
K	L	M	N	
P	Q	R	S	T
U	V	W	X	Y

- [8] Using **Depth-First Search**, list the squares in the order they are expanded (including the goal node if it is found). Assume that square positions are pushed onto the stack in the clockwise order given above so that when they are removed from Frontier (by popping the stack) the children are visited in counter-clockwise order. State D is expanded first (hint: State M will be examined next). Assume cycle checking is done so that a node is not inserted in the search tree if the grid square position associated with the node occurs somewhere on the path from this node back to the root node. Write down the list of states you expanded in the order they are expanded. Write down the solution path found (if any), or explain why no solution is found.
- [8] Using **Iterative Deepening Search**, draw the trees built at each depth until a solution is reached. Use the same ordering for pushing the successors of a node onto the stack as used in (a) and also use the same method for cycle checking as in (a).
- [4] Let each move of the knight have cost 1. Consider the heuristic function  $h(n) = |u - p| + |v - q|$ , where the grid square associated with node  $n$  is at coordinates  $(u, v)$  on the board, and the goal node G is at coordinates  $(p, q)$ . That is,  $h(n)$  is the "City-Block" distance between  $n$  and G. Is  $h(n)$  admissible? Why or why not?
- [10] Regardless of your answer to (c), perform **A\* Search** using the heuristic function  $h(n)$  defined in (c). In the case of ties, expand states in alphabetical order. Use repeated state checking by keeping track of both the Frontier and Expanded sets. If a newly generated node,  $n$ , does not have the same state as any node already in Frontier or Expanded, then add  $n$  to Frontier. If  $n$ 's state does already exist in either Frontier or Expanded, then check the  $g$  values of both  $n$  and the existing node,  $m$ . If  $g(n) \geq g(m)$ , then just throw away the new node,  $n$ . If  $g(n) < g(m)$ , then remove  $m$  from either Frontier or Expanded, and insert  $n$  into Frontier. List each cell in the order they are added to Frontier, and mark it with  $f(n) = g(n) + h(n)$  (show  $f$ ,  $g$  and  $h$  separately). When expanded (including node R), label a state with a number indicating when it was expanded (position D should be marked "1"). Highlight the solution path found (if any), or explain why no solution is found.

## Problem 2: Meet Your Friend[15]

Suppose two friends live in different cities on the following map. On every turn, we can simultaneously move each friend to a neighboring city on the map. The amount of time needed to move from city  $i$  to neighbor city  $j$  is equal to the road distance  $d(i, j)$  between the cities, but on each turn the friend that arrives first must wait until the other one arrives (and calls the first on his/her cell phone) before the next turn can begin. We want the two friends to meet as quickly as possible.



- (a) [6] Write a detailed formulation for this search problem. (Use formal notation.) An example formulation is shown for the Water Jugs Problem given in the lecture slides.
- (b) [3] Let  $D(i, j)$  be the straight-line distance between cities  $i$  and  $j$ . Which of the following heuristic functions are admissible?
  - (i) 1
  - (ii)  $2 \times D(i, j)$
  - (iii)  $D(i, j)/2$
- (c) [3] Are there completely connected maps for which no solution exists? Briefly explain why or why not.
- (d) [3] Are there maps in which all solutions require one friend to visit the same city twice? Briefly explain why or why not.

## Problem 3: Smurf Navigation[55]

In this problem, you are going to help Hefty Smurf get back to the Smurf colony. Hefty Smurf went to the forest to collect wood. But he was chased by wolves and has now lost his way. Your task is to write a program to guide him back home. But since there are wolves in the forest, Hefty must avoid the forests. He can only move through open locations. Also, he can only move vertically or horizontally (not diagonally).

### Game Description

The provided program reads a file with a world state containing Hefty's initial position, the location of the village, and the positions of the forests. From this input you will have to write code to find a path (if one exists) from Hefty's initial location to the Smurf colony. A world state will be given in a text file as a matrix in which the start position is indicated by "H", the goal position is indicated by "C", forests are indicated by "%", and empty positions are possible locations where Hefty can move through. Hefty is allowed to move only horizontally or vertically (not diagonally). We have provided you with skeleton code. **You are not responsible for any input or output.**

You have to implement both DFS search and A\* search to find a solution path. For DFS, first push move-Left, second push move-Down, third push move-Right, and lastly push move-Up onto the stack that implements the Frontier for this search method. In this way the Up move will be popped and visited first. **Implement cycle checking so that a newly generated node is not inserted onto the stack (i.e., Frontier) if the node's state is the same as one on the path back to the root (follow back-pointers to find these nodes).** Do NOT do more general repeated state checking.

For A\* search use as the heuristic function,  $h$ , the City-Block distance from the current position to the goal position. That is, if the current position is  $(u, v)$  and the goal position is  $(p, q)$ , the City-Block distance is  $|u - p| + |v - q|$ . In case of ties, use the priority order: U, R, D, L. **Implement repeated state checking by maintaining both Frontier and Expanded sets.** If a newly generated node,  $n$ , does not have the same state as any node already in Frontier or Expanded, then add  $n$  to Frontier. If a newly generated node,  $n$ , has the same state as another node,  $m$ , that is already in either Frontier or Expanded, you must compare the  $g$  values of  $n$  and  $m$ . If  $g(n) \geq g(m)$ , then throw node  $n$  away (i.e., do not put it on Frontier). If  $g(n) < g(m)$ , then remove  $m$  from either Frontier or Expanded, and insert  $n$  into Frontier. Assume all moves have cost 1.

### Output

Test both of your two search algorithms on the input file `input1.txt`.

After a solution is found, print out:

1. the maze with a "." in each square that is part of the solution path
2. the length of the solution path (i.e., the solution cost)
3. the number of nodes expanded

If the goal position is not reachable from the start position, the standard output should contain the line "No Solution" and nothing else. An example solution to `input1.txt` can be found in `output1(DFS).txt` and `output1(ASTAR).txt`.

### Code

You must use the code skeleton provided. You are to complete the code by implementing `search()` methods in the `AStarSearcher` and `DepthFirstSearcher` classes and the `getSuccessors()` method of the `State` class. You are permitted to add or modify the classes, but we highly recommend that you use

the IO class as is in order to ease grading. The HW1 class contains the main function. You should also look at the classes **Square**, **Maze** and **Searcher** to get an idea about how the program works.

## Program Testing

We will test your program on multiple test cases where we will vary the search type (**dfs** or **astar**), and the format of testing commands will be like this:

- The program will take two command line arguments. The first one is the input file name and the second one is the searchType.
- When searchType=**dfs**, you will use DFS for searching. When it is **astar** you will use A\* search. So, an example command for testing the code could be:

```
java HW1 input1.txt dfs
```

As part of our testing process, we will unzip the file you return to us, remove any class files, call `javac *.java` to compile your code, and call the main method of HW1 with certain parameters. Make sure that your code runs on one of the computers in the department because we will conduct our test on such computers.

## Deliverables

1. Hand in all .java files that you wrote or modified.

## Extra Credit

Add other bells and whistles to your program such as implementing other heuristics or allowing different costs for the four move types (U, D, L, R), adding a better visualization of the maze and solution path, or allowing multiple goal nodes and have the task be to find a path through all of them. Show results that illustrate the features you added. Document what you did and how you did it. Up to 10% extra credit points will be awarded. Put your documentation of what you did for extra credit in a file called **extracredit.pdf**