CSCI 561 – Spring 2013

Assignment 1: Search Algorithms (100 points)

Due date: **February 27, 2013**

1. Objective

The objective of this assignment is to learn about various search algorithms in a path planning application. You will implement breadth-first search (BFS), depth-first search (DFS), Beam search and A* search for solving a maze. You will be provided with several input and output examples for use in development.

2. Project Description

A robot agent wants to reach the goal point from a starting point in a maze domain. The maze is modeled as a grid. Each cell in the grid is marked as one of the following:

- Start cell (S)
- Goal cell (G)
- Mud cells (M)
- Wall cells (*)
- Road cells (white space)

Figure 1 shows an example maze.

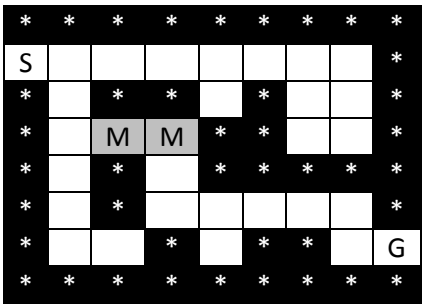
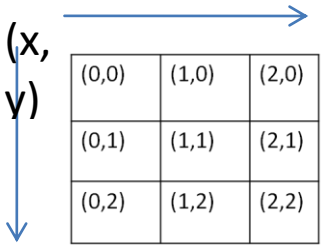


Figure 1 – Example maze

3. Problem Setup

1) The robot agent has two attributes: 1) Speed and 2) Position. Time serves as the cost function.

2) The top left corner is the coordinate (x,y) = (0,0). As shown in the picture below, the x index increases from left to right and the y index increases from top to bottom.



3) The robot can move only to the 4 adjacent locations (up, down, left, right). Diagonal moves are not allowed.

4) The table below shows the possible properties of each location.

Symbol	Meaning	Properties
"S"	A starting point	Your agent will start here
"G"	A goal point	Your goal is to move to this location
" "	An empty space	Your agent can pass through this location without any special effect
"*"	A wall	Your agent cannot move to this location
"M"	A mud swamp	Your agent can pass through this location but 0.1 will be subtracted from its speed from that location on

5) Your agent always starts at location "S" with **initial time = 0.**

6) It takes **1/s** time for an agent to move from cell X to its neighbor Y, where **s** is the speed of the agent at state X.

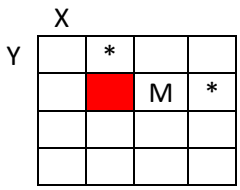
7) If the current speed is ≤ 0, no movement is possible.

8) Time serves as the cost function. The optimal solution is the one that takes the minimum time to reach the goal state from the starting point.

9) All algorithms need to be implemented as graph search. That is, you should not expand states that have been expanded before.

10) The maximum size of the maze you will have to deal with is 100x100.

4. Example



Let assume that the current position of the robot is (1,1). The current time and speed are 11.0 and 1.5 respectively. Figure 2 shows two levels of successor states.

Figure 2 – Successor states

Positions indicated with a wall (*) in the maze do not represent valid moves and so are marked by a red cross. When the robot moves into a mud swamp, its speed decreases permanently. As you can see in Figure 2, all child states of position (2,1) have speed 1.4. Note that the state with position (1,1) at the first level and the state with position (1,1) at the third level are two different states because their speed is different. It takes 0.66 units of time to go to one of the child states from the state at the top level, because the current speed in that state is 1.5 and the increase in time is computed as 1/1.5 = 0.66.

5. Project Questions

For question 1,2,4 and 5, your code will be tested on input files you haven't seen. For question 6, you need to run your own experiments with the problems in input1.txt, input2.txt and input3.txt and complete the table.

Question 1 (15 Points)

Implement Breath First Search (BFS) to generate a path from S to G.

Question 2 (15 Points)

Implement Depth First Search (DFS) to generate a path from S to G.

Question 3 (10 Points)

Invent at least two consistent heuristic functions (h1, h2) for the mentioned maze domain and explain why they are consistent. Does either heuristic dominate the other, and why?

Question 4 (25 Points)

Implement the A* search algorithm and use one of your heuristic functions in question 3 to find a path that takes the agent from the starting point to the goal in the minimum amount of time.

Question 5 (25 Points)

Implement Beam search with one of your heuristic functions from question 3. The implementation should be similar to A*, however, you will keep only the best k states in the frontier list.

Question 6 (10 Points)

Use a table like the following one to compare the results of your search algorithms. Record your results assuming the initial speed is 2. Briefly, compare the search algorithms in terms of the time complexity, space complexity, completeness, and optimality. What are the effects of the different heuristics in A* and the different values of k in Beam search?

		Runtime	Number of iteration	Path length	Solution cost (time)
Input1.txt	BFS				
	DFS				
	A* (heuristic 1)				
	A* (heuristic 2)				
	Beam (k=5)				
	Beam (k=10)				
	Beam (k=50)				
Input2.txt	…				
Input3.txt	…				

6. Implementation Details

Input

Sample textual representations of 3 different mazes are provided to you in input1.txt, input2.txt, and input3.txt. The textual representation of an example map is shown in Figure 3. The size of the input map is defined by “WIDTH” and “HEIGHT”, which correspond to the number of cells in each row and column.

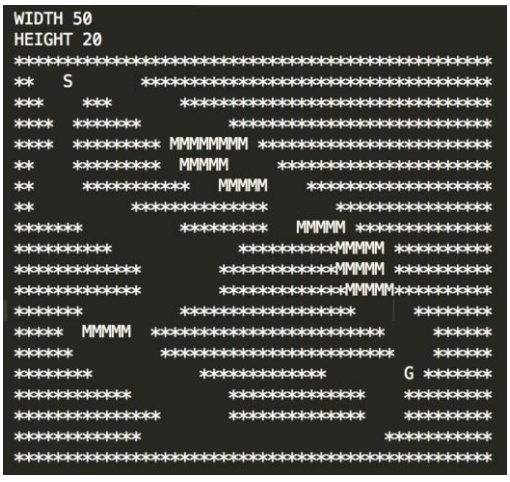


Figure 3 – Input file example

Output

As exemplified in Figure 4, the output should be divided into two parts for each problem solved:

1) the solution as a sequence of moves and states

2) the search log

In the search log, include at least the following information for the first **100** iteration of the search:

Iteration = <iteration number>

Current Node : <Current state>

Child List: <Expanded states>

Frontier List: <states in frontier list>

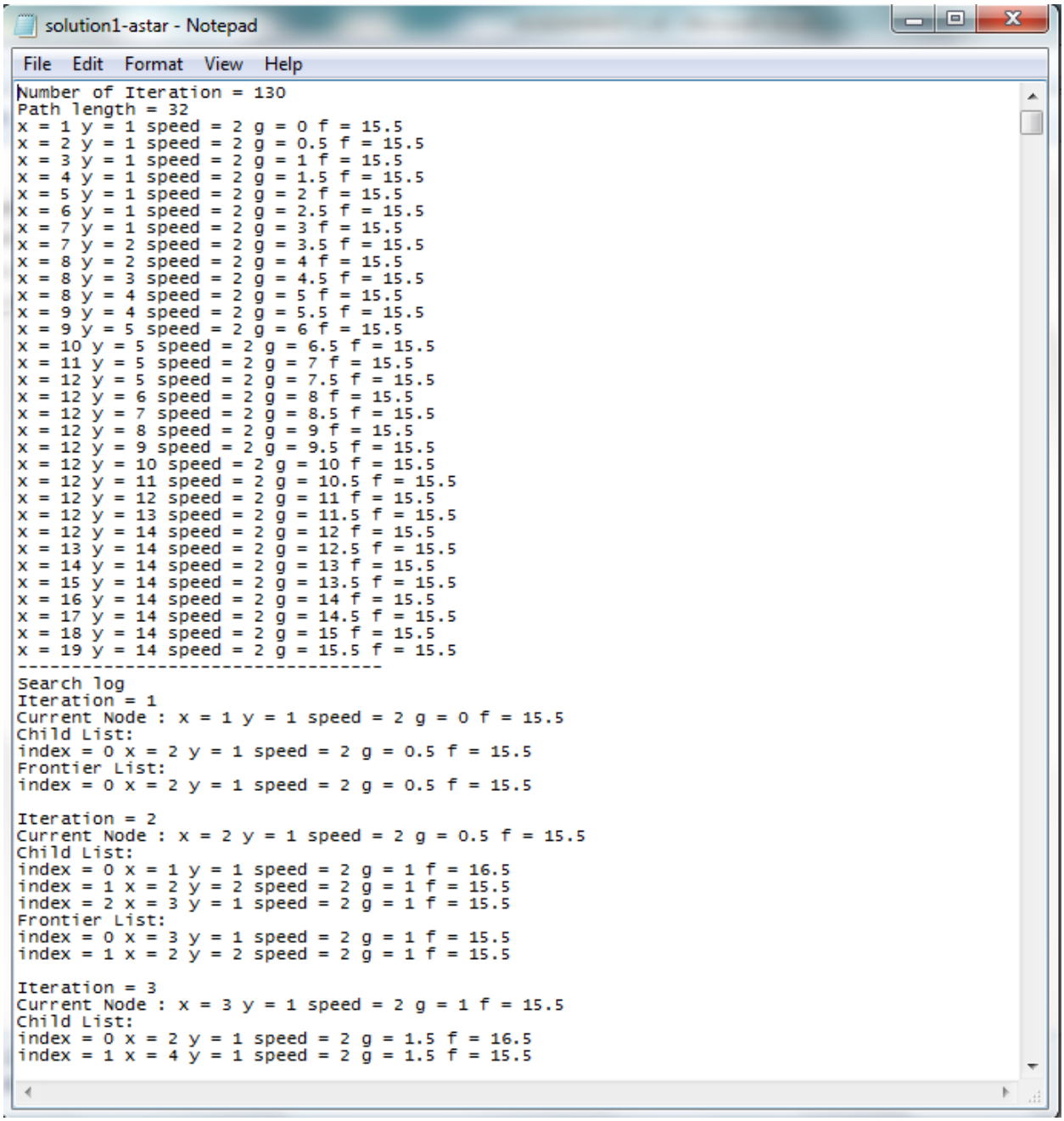


Figure 4 – `solution1-astar.txt` example

Note that in Figure 4, *index* represents the position of the state in the queue. You can find the examples in the output example folder in **assignment1.zip** file.

Execution Details

Make sure that your programs can be executed with the following flags:

```
search {BFS|DFS|AStar|Beam [-k value]} -s initialspeed -i inputfilename -o outputfilename
```

or

```
java search {BFS|DFS|AStar|Beam [-k value]} -s initialspeed -i inputfilename -o outputfilename
```

inputfilename is the map input file, **outputfilename** is the output text file which will contain the results. For example,

```
search Astar -s 5 input1.txt -o solution1-astar.txt
```

performs an A* search with initial speed of 5 and

```
search Beam -k 20 -s 10 input1.txt -o solution1-beam-10.txt
```

performs a Beam search with k = 20 and initial speed = 10. The default k value for Beam search is 10 and the grader can run the command without declaring a k value. Note that in addition to the three input files provided to you, we will also test your program with two more input files. Your runtime must be under 5 minutes for each search.

7. Submission

Report

Your report must be submitted in PDF format. Please include your full name, USC login name and USC ID on the front page of the report. Name the report "*Firstname _Lastname_project1_report.pdf*". Your report should include answers for questions 3 and 6.

Code

Make sure that your source code (C/C++/Java) can be compiled and executed on aludra.usc.edu. Provide a README file with instructions on how to run your program and a Makefile that compiles it.

Submitting your project

Use the following steps to submit your project:

- Place all required files (including the project report) into a single directory.
- Use tar and gzip to compress this folder into an archive called *"Firstname_Lastname_project1.tar.gz"* (for C/C++ coders) or *"Firstname_Lastname_project1_java.tar.gz"* (for java coders).
- Submit the archive via Blackboard.

8. Ground Rules

No collaboration will be allowed on this project. It must reflect just the work of the individual student, with no outside help (except for questions asked of the instructor or TA). Use of any code not produced by the individual student – whether from a friend, the Internet, or anywhere else – without explicit

permission from the instructor or TA is explicitly forbidden. ***The standard penalty for violating this policy is an F in the course.***

You are allowed a total of two late days that can be used on the programming projects. This means that, without a penalty, project one can be two days late OR project two can be two days late OR each project can be one day late. Once you have used up your two late days, one additional day late will result in a 25% reduction in the total score, two additional days late will yield a 50% reduction, and no credit will be given for three or more additional days late. Late days are in units of days, not hours, so using up part of a day uses up the whole day.