CS 4320/5314
Programming Assignment 1:
Search and Pathfinding

Part 1 DUE: Tues, Oct 3 at 11:59 PM
Part 2 DUE: Mon, Oct 9 at 11:59 PM

Groups: You will work in groups of 2 or 3 on this assignment. You may select your own partner, but if you have difficulty finding one let me know and I will try to match you up with someone.

Turn in only one copy per group, clearly listing all team members. In your writeup you should including a brief discussion of what the contributions of each individual team member were towards the final submission. It is expected that each team member will contribute significantly towards the design, coding, testing, and documentation aspects of the project. Paired coding and similar approaches are encouraged. This is a project of significant complexity, and you should plan for it to take multiple programming sessions over a couple of weeks to get it completed.

Objective: This assignment will allow you to practice some of the basic search methods used for problem solving in artificial intelligence, and to see the benefits of heuristic search. You will experiment with a basic formulation of the general pathfinding problem.

Pathfinding
Pathfinding is a common problem that artificial agents must solve, including mapping services, artificial vehicles, AIs in real-time strategy games, and robots that must navigate in the physical world. For this assignment we will focus on simplified pathfinding problem. Your program will search for the shortest path from a starting location to a goal location on a square grid. However, this grid will have some impassable locations and varying terrain costs that must be accounted for. The agent is allowed to move in any of the four primary directions (up, down, left, right) but not diagonally or outside the bounds of the map.

Your program should read in a map file that specifies the map in the following format. The file name should be able to be specified on the command line. The first line has the dimensions of the map, the second line has the coordinates of the starting location (row column), the third line has the coordinates of the goal location. After that is the specification of the map, which consists of digits between 0 and 5, separated by spaces. These numbers represent the movement cost for moving to a given space on the grid. **The number 0 is a special case and is considered impassable terrain.** The numbers 1-5 are the number of turns required to move to the given square, with 1 being the lowest cost and 5 being the

highest. There is no cost for moving to the starting location. The following is an example of the map format.

```
5 7
1 2
4 3
2 4 2 1 4 5 2
0 1 2 3 5 3 1
2 0 4 4 1 2 4
2 5 5 3 2 0 1
4 3 3 2 1 0 1
```

Search Algorithms

You will implement and compare three of the search algorithms discussed in class and in the book:

1) Breadth-first search
2) Iterative deepening search
3) A* search

For all three algorithms you should implement repeat-state checking, so that you do not revisit states you have already expanded.

For A* search you should use the Manhattan distance as your heuristic, but implement the algorithm in a general way so that a new heuristic could be used.

Your program should run each search algorithm with a 3 minute time cutoff (i.e., you should stop the search with no result after a maximum of 3 minutes of runtime). For each of the three algorithms, print out the following information to the console:

1) The cost of the path found
2) The number of nodes expanded
3) The maximum number of nodes held in memory
4) The runtime of the algorithm in milliseconds
5) The path as a sequence of coordinates (row, col), (row col), … , (row, col)

If the algorithm terminates without finding a result, print -1 for the path cost and NULL for the path sequence. You should still print the number of nodes and the runtime.

What to Turn In

You must implement your program in either Java or Python (or ask me if you prefer another language) and turn in both the source code and an executable file that can be run on the command line with the name of the map file and the solution

algorithm (BFS, IDS, AS) specified on the command line. For example, I should be able to run your program using something similar to (for Java):

java –jar myProgram.jar map1.txt IDS

The code should be clean and well documented. You should also include a brief readme file with instructions on how to run your code, including the command line format if it is different from the above.

Next, turn in a minimum of 4 test cases you plan to use to evaluate your code with increasing sizes (e.g., 5x5, 10x10, 15x15, and 20x20). These four test cases should illustrate the differences in scalability between the different algorithms. You may turn in the test cases even if you are not able to complete the code, but if you can run the code you should also document the performance of your implementation on your test cases. **Your code will also be tested on some (unknown) test cases, so be sure to test your code carefully, beyond just the cases you submit.**

PART 1 Submission:

- **General environment and tree search code**
- **Initial implementation of Breadth-First search**
- **Test cases**

PART 2 Submission:

- **Code from part 1 (updated if needed)**
- **Implementations of IDS and A***
- **Brief report comparing algorithm results for test cases**

<u>Some General Tips on How to Proceed</u>

Write a method to read in/print out map file to a 2d array; store source/goal nodes

Write a node class (stores coordinates and all necessary helper data; use the same node class for all three searches)

Write a method to generate successor nodes (this should be the same for all three searches!)

Write BFS; follow the pseudocode as closely as possible.
* Add a "closed" list to check for repeated states
* Test this first!

Write a depth-limited search method
* Base methods should look similar to BFS
* Write the IDS wrapper that gradually increases search depth

Write A* search
 * write method to compute Manhattan distance heuristic
 * write a simple comparator class, or implement a compareTo method in your Node class to allow you to use a PriorityQueue (builtin java class) to maintain a sorted list of nodes
 * Implement the IDS method (follow psuedocode)

Add any necessary logic to run different algorithms based on command line parameters, record the necessary information about runtimes, nodes created, etc.

Add in logic for cutoff; this can be as simple as passing each search method a deadline and including an if statement to terminate the search within the main loop of the algorithm if the time exceeds the deadline

Create and run several test cases on different maps; document results and write up your findings

Generate an executable file, and test it to make sure it works!

Make sure your submission includes ALL of the following:
 * working executable file
 * source code
 * test cases (i.e., map files you used for testing)
 * a writeup/discussion of your results