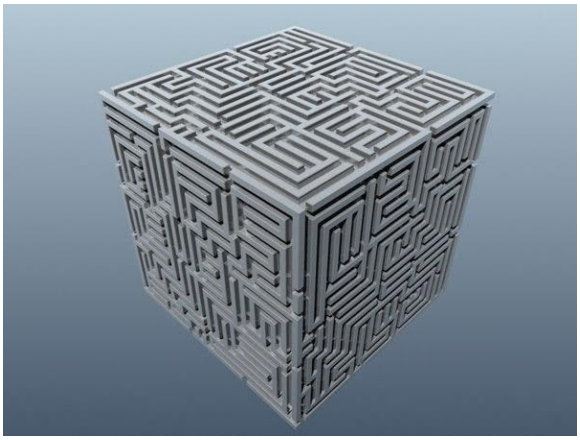
CSCI-561 – 2020 Fall - Foundations of Artificial Intelligence Homework 1

Due September 22, 2020, 23:59:59



1. Overview

This is a programming assignment in which you will apply AI search techniques to solve some sophisticated 3D Mazes. As shown in Figure 1, each 3D maze is a grid of points (not cells) with (x, y, z) locations in which your agent may use one of the 18 elementary actions (see their definitions below), named X+, X-, Y+, Y-, Z+, Z-; X+Y+, X-Y+, X+Y-, X-Y-, X+Z+, X+Z-, X-Z+, X-Z-, Y+Z+, Y+Z-, Y-Z+, Y-Z-; to move to one of the 18 neighboring grid point locations. At each grid point, your agent is given a list of actions that are available for the current point your agent is at. Your agent can select and execute one of these available actions to move inside the 3D maze. For example, in Figure 1, there is a "path" from (0,0,0) to (10,0,0) and to travel this path starting from (0,0,0), your agent would make nine actions: X+, X+, X+, X+, X+, X+, X+, X+, X+, and visit the following list of grid points: (0,0,0), (1,0,0), (2,0,0), (3,0,0), (4,0,0), (5,0,0), (6,0,0), (7,0,0), (8,0,0), (9,0,0), (10,0,0). At each grid point, your agent is given a list of available actions to select and execute. For example, in Figure 1, at the grid point (60,45,30), there are two actions for your agent: Z+ for going up, and y- for going backwards. At the grid point (60,103,97), the available actions are X+ and Y-. At (60,45,97), the three available actions are Y+, Z-, and X-Y+. If a grid point has no actions available, then that means such a point has nowhere to go. For example, the point (24,86,31) (not shown in Figure 1) has nowhere to go and is not accessible.

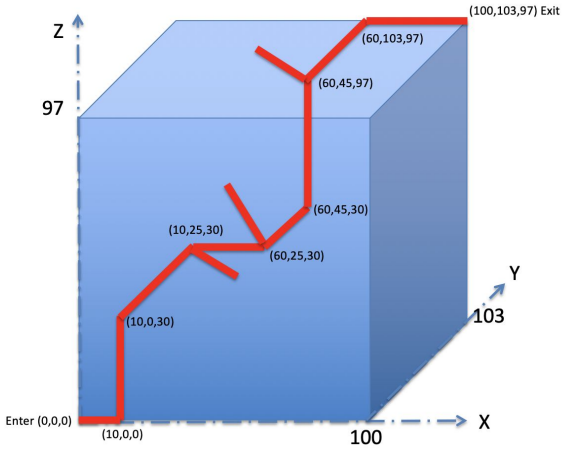


Figure 1: 3D Maze Configuration: grids world that contains travelable actions

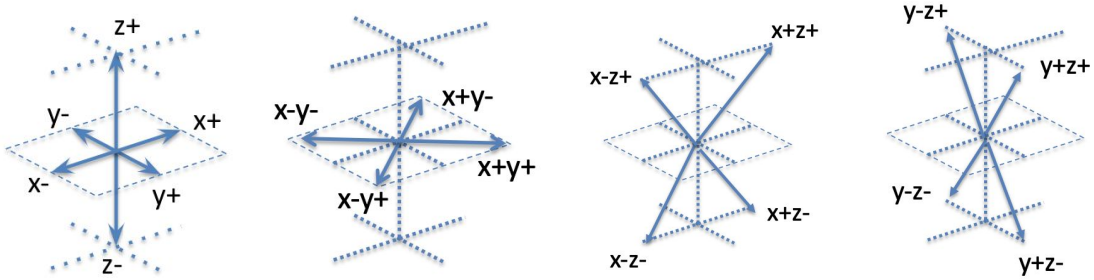


Figure 2: Definitions of Actions in the Maze

The 18 actions are defined as follows. They are roughly divided as "straight-move" and "diagonal-move" actions. As shown in Figure 2, the six straight-move actions are X+, X-, Y+, Y-, Z+, Z-, and they allow your agent to move in a straight-line to the next grid point. The diagonal-move actions are further defined on xy, xz, and yz planes, respectively. For example, the actions X+Y+, X+Y-, X-Y+, and X-Y-, are those moves diagonally on the xy plane. Similarly, the actions X+Z+, X+Z-, X-Z+, and X-Z-, are those moves diagonally on the xz plane. Finally, the actions Y+Z+, Y+Z-, Y-Z+, and Y-Z-, are those moves diagonally on the yz plane. Notice that not all actions may be available for a given grid location, and not all grid locations may have actions. For clear format purpose, we name or encode these actions as follows:

Act	X+	X-	Y+	Y-	Z+	Z-	X+Y+	X+Y-	X-Y+	X-Y-	X+Z+	X+Z-	X-Z+	X-Z-	Y+Z+	Y+Z-	Y-Z+	Y-Z-
Code	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Your programming task is as follows. Given as inputs: (1) a list of grid points with their available actions, (2) an entrance grid location, e.g., (0,0,0) in Figure 1, and (3) an exit grid location, e.g., (100,103,97) , your program must search in the maze configuration and find the ***optimal*** shortest path from the entrance to the exit, using a list of actions that are available along the way.

Conceptually, the specification of a grid location and its associated actions is given as a grid location with a list of actions. For example (Note: The exact input format will be given in section 5 and 6 below),

INPUT LINE: (60 45 97), Y+, Z-, X-Y+
INPUT LINE: (60 46 97), Y-, Y+
INPUT LINE: (60 45 96), Z+, Z-,
INPUT LINE: (59 46 97), X+Y-, X-Y+

is a specification, for Figure 1, that at the grid location (60,45,97), the available actions are Y+, Z-, and X-Y+. At the grid (60,46,97), the available actions are Y- and Y+, and at the grid (60,45,96), the available actions are Z+ and Z-, and at the grid (59,46,97), the available actions are X+Y- and X-Y+ for moving diagonally on the xy plane.

Once your agent finds an optimal path from the entrance to the exit, your agent should output a list of points that have been visited along the path. For example, if the entrance and exit would be changed at (60,103,97) and (64,103,97) respectively, then the correct output path would be:

OUTPUT: (60,103,97), (61,103,97), (62,103,97), (63,103,97), (64,103,97).

To assist your programming, you will be provided some sample inputs and outputs (see below). Please understand that the goal of these samples is to check that you can correctly parse the problem definitions and generate a correctly formatted output. The samples are very simple, and it should not be assumed that if your program works on the samples it would definitely work on all test cases for grading. There will be more complex test cases and it is your task to make sure that your program will work correctly on any valid input. You are encouraged to design and try your own test cases to check how your program would behave in some complex special cases that you might think of. Since **each homework is checked via an automated A.I. script**, your output should match the specified format ***exactly***. Failure to do so will most certainly cost some points. The output format is simple, and examples are provided. You should upload and test your code on vocareum.com at their terminal window which is a Linux-like environment. Please make sure you test your program at the terminal window at vocareum.com before you click the submit button there. You can submit as many times as you like, and the last submission before the due time will be used to grade your results. You may use any of the following programming languages: **C++, Java, Python**, but Python may be the preferred language to use in today's large-scale AI program applications.

2. Grading

Your code will be tested and graded as follows: Your program should not require any command-line argument. It should read a text file called "input.txt" in the current directory that contains a problem definition. It should create and write a file "output.txt" with your solution in the same current directory. Format for input.txt and output.txt are specified as in section 5 below and will be supplemented with some details in section 6. **End-of-line character is LF** (since vocareum is a Unix system and follows the Unix convention).

The grading A.I. script will test your program for 40 test cases for grading as follows:

- Create an input.txt file and delete any old output.txt file.
- Run your code to create your output.txt file.
- Check the correctness of your program's output.txt file.
- If your outputs for all 40 test cases are correct, you get 100 points.
- The 40 test cases are divided into four classes: Class1 (easy), 10 cases; Class2 (medium), 10 cases; Class3 (hard), 10 cases; and Class4 (complex), 10 cases.
- For Class1, each test is worthy of 1 point. For Class 2, each test is 2 points. For Class 3, each test is 3 points, and for Class 4, each test is 4 points.
- The total points for each class are: Class1: 10x1=10 points; Class2: 10x2=20; Class3: 10x3=30; and Class4: 10x4=40. The total points for the whole homework is: Class1+Class2+Class3+Class4 = 10+20+30+40 = 100 points.

Note that if your code does not compile, or somehow fails to load and parse input.txt, or writes an incorrectly formatted output.txt, or no output.txt at all, or OuTpUt.TxT, **you will get zero points**. Anything you write to stdout or stderr will be ignored and is ok to leave in the code you submit (but it will likely slow you down). Please test your program on Vocareum's terminal window with the provided sample files to avoid any problems.

3. Academic Honesty and Integrity

All homework material is checked vigorously for dishonesty using several methods. All detected violations of academic honesty are forwarded to the Office of Student Judicial Affairs. To be safe you are urged to err on the side of caution. Do not copy work from another student or off the web. Keep in mind that sanctions for dishonesty are reflected in *your permanent record* and can negatively impact your future success. As a general guide:

Do not copy code or written material from another student. Even single lines of code should not be copied.

Do not collaborate on this assignment. The assignment is to be solved individually.

Do not copy code off the web. This is easier to detect than you may think.

Do not share any custom test cases you may create to check your program's behavior in more complex scenarios than the simplistic ones considered below.

Do not copy code from past students. We keep copies of past work to check for this. Even though this problem differs from those of previous years, do not try to copy from homeworks of previous years.

Do not ask on piazza how to implement some function for this homework, or how to calculate something needed for this homework.

Do not post code on piazza asking whether or not it is correct. This is a violation of academic integrity because it biases other students who may read your post.

Do not post test cases on piazza asking for what the correct solution should be.

Do ask the professor or TAs if you are unsure about whether certain actions constitute dishonesty. It is better to be safe than sorry.

4. Project Description

In this project, we twist the problem of path planning a little bit just to give you the opportunity to deepen your understanding of search algorithms by modifying search techniques to fit the criteria of a more realistic application. To give you a realistic context for expanding your ideas about search algorithms, we invite you to take part in a maze-solving mission in an unknown world. The goal of this mission is to send your sophisticated and intelligent agent from a specific entrance-location and travel as quickly as possible to an exit location. You are invited to develop three algorithms to find the optimal path and navigate through a complex 3D maze configuration based on a particular objective.

The input of your program includes three elements: the maze configuration, an entrance location, and an exit location, plus perhaps some other quantities that control the quality of the solution. Each possible 3D maze world can be imagined as a 3D grid of points in a 3-dimensional space. At each point inside the maze, your agent can perform one of the 18 actions, and move to one of the **18 possible neighbor grid points (See Fig. 2)**. To simplify things, your agent's actions are assumed to be deterministic and error-free. If your agent's action is legal, then your agent will always end up at the intended neighbor grid point. If your agent tries to move into a wall or outside a maze world, the result will be nil and your agent will remain in its current point location.

5. Search for the Optimal Paths

You will write a program that will take an input file that describes the maze configuration, the initial entrance grid location, the exit grid location, and characteristics of the agent. You should find the optimal path **from the initial entrance grid location to that exit grid location**. A path is composed of a sequence of legal moves. Each legal move consists of moving the agent from a point to one of its 18 neighbor points, using one of the elementary actions that are available at the current location.

Your agent must search through possible paths of movements and find the optimal path to travel from the entrance to the exit, and then output the results.

To find the solution you will use the following algorithms:

- Breadth-first search (BFS)
- Uniform-cost search (UCS)
- A* search (A*).

Your algorithm should return an **optimal path**, that is, with shortest possible operational path length. Operational path length is further described below and may not always be equal to geometric path length as per the specifications given ahead. If an optimal path cannot be found, your algorithm should return "FAIL" as further described below.

To help us distinguish between your three algorithm implementations, you must follow the

following conventions for computing operational path length:

Breadth-first search (BFS)
In BFS, each move from one location to any of its neighbors counts for a unit path cost of 1. You do not need to worry about the fact that moving diagonally actually is a bit longer than moving along the North/South, East/West, and Up/Down directions. So, any allowed move from one location to an adjacent location costs 1.

Uniform-cost search (UCS)
When running UCS, you should compute unit path costs in any of the 2D plane XY, XZ, YZ, on which you are moving. Let us assume that a grid location's center coordinates projected to a 2D plane are spaced by a 2D distance of 10 units on X and Z plane respectively. That is, on the XZ plane, move from a grid location to one of its 4-connected straight neighbors incurs a unit path cost of 10, while a diagonal move to a neighbor incurs a unit path cost of **14 as an approximation to $10\sqrt{2}$ when running UCS.**

A* search (A*).
When running A*, you should compute an approximate integer unit path cost of each move as in the UCS case (unit cost of 10 when moving straight on a plane, and unit cost of 14 when moving diagonally). Notice for A*, **you need to design an admissible heuristic for A* for this problem.**

Input: The file input.txt in the current directory of your program will be formatted as follows:

- First line: Instruction of which algorithm to use, as a string: BFS, UCS or A*
- Second line: Three strictly positive 32-bit integers separated by one space character, for the size of X, Y, and Z dimensions, respectively.
- Third line: Three non-negative 32-bit integers for the entrance grid location.
- Fourth line: Three non-negative 32-bit integers for the exit grid location.
- Fifth line: A strictly positive 32-bit integer N, indicating the number of grids in the maze where there are actions available.
- Next N lines: Three non-negative 32-bit integers separated by one space character, for the location of the grid, followed by a list of actions that are available at this grid. The grid location is guaranteed to be legal and within the maze.

For example:

```
A*
100 200 100
0 0 0
3 3 0
4
0 0 0 7
1 1 0 7 10
```

2 2 0 7 10
3 3 0 10

In this example, the 3D maze is of size 100 x 200 x 100 (**specifically, points range from (0,0,0) to (99,199,99)**), the entrance grid location is at (0,0,0), and the exit grid location is at (3,3,0). In this maze, there are 4 grid locations that have actions and they are specified in the next four lines. Namely, the grid (0,0,0) has one action X+Y+ (encoded as 7); the grid (1,1,0) has two actions X+Y+ and X-Y- (encoded as 7 and 10); the grid (2,2,0) has two actions X+Y+ and X-Y- (encoded as 7 and 10); and the grid (3,3,0) has one action X-Y- (encoded as 10).
.

Output: The file output.txt that your program creates in the current directory should be formatted as follows:

- First line: A single integer C, indicating the total cost of your found solution. If no solution was found (the exit grid location is unreachable from the given entrance, then write the word "FAIL" (all capital) without any other lines following.
- Second line: A single integer N, indicating the total number of steps in your solution including the starting position.
- N lines: Report the steps in your solution travelling from the entrance grid location to the exit grid location as were given in the input.txt file.
  - Write out one line per step with cost. Each line should contain a tuple of four integers: X, Y, Z, Cost, separated by a space character, specifying the grid location with the single step cost to visit that grid location by your agent from its last grid during its traveling from the entrance to the exit.

For example, the following is a sample output of the corresponding input above:

42
4
0 0 0 0
1 1 0 14
2 2 0 14
3 3 0 14

Notes and hints:

- Please name your program "**homework.xxx**" where 'xxx' is the extension for the programming language you choose ("py" for python, "cpp" for C++, and "java" for Java). If you are using C++11, then the name of your file should be "homework11.cpp" and if you are using python3 then the name of your file should be "homework3.py". Please use the programming languages mentioned above for this

homework.

- Most likely (but no guarantee) we will create 13 BFS, 13 UCS, and 14 A* grading cases for grading your agents. These cases are called “Grading Cases” because they are reserved for the grading purposes only. During your development, there would be some “test cases” for you to use to test, debug, and improve your agents. But in general, you are responsible for testing your agents thoroughly using any test cases you would like.
- Your program will be killed after some time if it appears stuck on a given test case, to allow us to grade the whole class in a reasonable amount of time. We will make sure that the time limit for a given test or grading case (or class) is sufficient and long enough for solving the case for a correct algorithm implementation. These time limits are typically determined by a standard algorithm implementation that has been tested thoroughly in solving the given 3D mazes, and we ensure that these time limits are typically very generous.
- The time limit is the total combined CPU time as measured by the Unix **`time`** command. This command measures pure computation time used by your program, and discards time taken by the operating system, disk I/O, program loading, etc. Beware that it cumulates time spent in any threads spawned by your agent (so if you run 4 threads and use 400% CPU for 10 seconds, this will count as using 40 seconds of allocated time).
- There is no limit on input size, number of grid points that have actions, etc., other than specified above (32-bit integers, etc). However, we will seriously consider the complexity of each test case. You should take care of the data structure used in your algorithms so that the program returns in a bounded time. Additional information may be posted on Piazza if necessary. Please keep your eyes on it.
- If several optimal solutions exist, then any of them will count as correct.
- Please submit your homework code through Vocareum (https://labs.vocareum.com/) under the assignment HW1. Your username is your email address (DEN username). Click “forgot password” for the first-time login. You should have been enrolled in this course on Vocareum. If not, please post a **private** question with your email address and USC ID on Piazza (under “hw1” folder) so that we will invite you again.
- You can submit your homework code (by clicking the “submit” button on Vocareum) as many times as you want. Only the latest submission will be considered for grading. Late penalty will be applied as 20% per day if your latest submission is later than the due time.
- Every time you click the “submit” button on Vocareum, your submitted agent will be run and tested by our AI script on a number of test cases and the results will be reported to you in the submission-report which you can read and exam. You may use these reports for debugging and improving your agent. Notice that these are “test cases”, and they are not the “grading cases”. The grading cases are reserved for grading purposes and may not be available to your agent before the grading process begins.

6. Sample Inputs and Outputs

Example 1 (BFS):

=======input.txt=============

```
BFS
10 10 10
1 3 1
5 3 4
12
1 3 1 5
1 3 2 1 11
2 3 3 16
2 4 2 11
2 3 2 5 11
3 3 3 17 18
3 2 2 3
3 3 2 5
3 2 4 7
4 3 4 1 11
5 3 4 2
5 3 5 6
```

============================

=======output.txt============

```
6
7
1 3 1 0
1 3 2 1
2 3 2 1
3 3 3 1
3 2 4 1
4 3 4 1
5 3 4 1
```

============================

Example 2 (UCS):

=======input.txt=============

```
UCS
10 10 10
7 0 1
5 2 3
9
5 2 3 1
5 1 3 3
5 2 2 5 17
```

6 1 2 9 3
6 1 1 5 15
7 1 1 9
6 2 1 4
8 0 1 9
7 0 1 9 1
==========

=======output.txt==========
48
5
7 0 1 0
6 1 1 14
6 1 2 10
5 2 2 14
5 2 3 10
==========

Example 3 (A*):
=======input.txt==========
A*
5 5 5
1 0 4
3 1 2
8
1 0 1 2
1 0 4 3 16
1 1 4 6 12
2 1 3 2
1 1 3 3 12 16
1 2 3 9
2 1 2 1 12
3 1 1 5
==========

=======output.txt==========
38
4
1 0 4 0
1 1 3 14
2 1 2 14
3 1 2 10
==========

Example 4 (BFS with no solution):

=======input.txt=============

```
BFS
4 4 4
0 0 0
2 3 1
6
0 0 0 7 11
1 1 0 7
1 0 1 7
2 1 1 15
2 2 0 5
2 2 2 3
```

==========

=======output.txt=============

```
FAIL
```

==========