CECS 428: Programming Assignment 1

Darin Goldstein

1 Deadline

2/5/2021 at 5:00 PM

2 Mazes

You have encountered a race of time-traveling aliens who are desperate for amusement.

For this assignment, your goal is to write a program for these time-traveling aliens that creates a maze in 4 dimensions, given the maximum size of any given dimension N *using the algorithm that arises from the disjoint data structure presented in lecture.* A point in 4 dimensions can be represented by (t, z, y, x) where x, y, z are the normal 3-dimensional coordinates and the fourth is the time coordinate. Assume that if an alien is situated at any point in the 4D maze, then it can, in one move, change any one of its coordinates up or down by 1 as long as it does not "go off the board" (i.e. while progressing through the maze, all coordinates must remain between 0 and $N - 1$; you are required to keep the walls on the outside of the 4D cube intact) or hit a wall. In other words, your maze *must* remain closed to the outside. Outer walls must all stay up.

You are required to use every cell in $\{0, 1, \ldots, N - 1\}^4$; the maze cannot contain any cells in the grid that are unreachable from any others. Because of this, the starting and ending positions of the alien in the maze are irrelevant. It is also required that you do not create any loops within the maze; the underlying graph of the maze *must* be a tree.

Finally. and most importantly, your maze must truly be *random* in the sense that any two runs of your program should yield different results for any given slice of the maze. In other words, if you hold any 3 coordinates constant, then two runs of your program should yield two different results for the array formed by the fourth coordinate.

The input to the program will be the value of N which will be passed as an integer. Your output should be a maze as represented by an array of bytes. The header of your function in the `StudentSolver` class should look as follows:

```
public static byte[] solve(int n)
```

The idea is to think of the 1 bits as walls.

Your output should consist of an array of bytes, one for each cell in the grid starting with $(0,0,0,0), (0,0,0,1), (0,0,0,2), \ldots, (0,0,0,N-1), (0,0,1,0), \ldots$, etc., and ending with $(N-1, N-1, N-1, N-1)$. Each byte will contain 4 2-bit codes: the 2 lowest order bits will represent the x-direction, the next 2 bits will represent the y-direction, and so on, until the 2 highest order bits that represent the t-direction. The 4 possible codes are as follows:

- 00: In this cell, along this coordinate axis, the alien may move ± 1 units. In other words, there are no walls barring movement along this axis.
- 01: In this cell, along this coordinate axis, the alien may move in the -1 direction only. There is a wall blocking the positive direction along this axis.
- 10: In this cell, along this coordinate axis, the alien may move in the $+1$ direction only. There is a wall blocking the negative direction along this axis.
- 11: In this cell, along this coordinate axis, the alien may not move at all. There are walls blocking both the positive and negative direction along this axis.

For example, a byte with ASCII code (0 from 255 possibilities) equal to 75 with corresponding bits 01001011 would indicate that there are walls preventing any movement in the x-direction, movement is possible in the positive y-direction and the negative t-direction, and any movement is possible in the z-direction.