CECS 428 Programming Assignments

Darin Goldstein

Every problem will be sent as an email attachment called input.zip. Unzip this file in the standard way to retrieve the files that you are supposed to have for each assignment. When you submit your file(s), you will zip them into a file called output.zip. In the first assignment, for example, the output file is gray.txt, but when you submit, you will submit the file output.zip (which will be gray.txt, zipped up and renamed if necessary).

If your response files contain extra white space or added characters, they will be counted as incorrect.

For each assignment, an example input and output will be emailed to everyone via Beachboard so that you can get an idea of how the formatting should look.

1 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 in the file parameter.txt. Your output file should be called maze.txt and be submitted in the following format.

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

Your output should consist of a series 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.

These *bytes* should be written to maze.txt. Please note that this file will *not* be a text file.

2 Cleanup

You are a summer intern for a scientist of an advanced race of aliens living in m-dimensional space. He/She/It has been performing experiments and has left sensors and equipment lying all over the the universe. He/She/It has successfully completed the experiment and is now looking to clean up the sensors that have been strewn all over the universe. As he/she/it does not want to waste too much of his/her/its valuable time cleaning up, he/she/it has assigned you to go do it. You will need to map out the shortest path through m-space to gather up all of the sensors. You may assume that your supervisor has the ability to teleport you to the first point and will also be able to teleport you back from that same point once you've finished the job.

Each sensor should be thought of as residing at a point in m-dimensional space. You will be given n data points in m-dimensional Euclidean space in the file points.txt. There will be a single point with m comma-separated coordinates on each line. Your goal will be to identify the shortest path you can find from point to point that touches each exactly once and returns to the starting point. When traveling from one point to another, it is assumed that you will always move in a perfectly straight line.

You may choose to start at any point you want. The output file path.txt will consist of a permutation of the points/lines in the input file. You should assume that the final step in your path is from the last point you list back to the first (though you may assume that you teleport into whatever point in the plane you choose to start at).

For this assignment, you will receive only one instance. Your grade will be based on the quality of your answer to that single instance. You may submit solutions until you are happy with your grade or until the deadline. Please be careful: Your solution will be marked incorrect if you don't exactly copy each point's coordinates and write a newline after each.

3 The Museum

You are the owner of a security guard business that hires out security guards to buildings. Your business model is to station your guards at the intersection of two or more hallways so that he can see down every hallway within his range of vision. Any hallway within range of vision of a guard is considered covered. A museum curator approaches you with a potential job. He claims that his museum was original built such that the layout was in the form of a tree.

Immediately, you turn him down because, given your extensive computer science background, you know that covering trees usually takes many more guards than covering other, more general types of graphs. However, the curator tells you that the museum is very old and many hallways have been added on since the original floorplan. In fact, there are roughly twice as many added-on hallways as there are original ones.

Every hallway of the museum must have a guard stationed on one of the ends of the hallway. (Guards have infinite range of vision so the length of the hallways is irrelevant.) Your goal is to hire the minimum number of guards that can fully cover the museum.

The input to your program, graph.txt, will be a graph in edge list form that represent the floorplan of the museum. The vertices will be labelled by nonnegative integers. So the edges connected to vertex i will come after i followed by a colon. An x will indicate the end of an edge list. A triangle might then be described by

$$0 : 1, 2x1 : 0, 2x2 : 0, 1x$$

The output of your program, cover.txt, will be a file with a list of vertex names followed by x to indicate on which intersections you plan to station your guards. So, for example, your program might return

$$0x1x$$

for the triangle above.

For this assignment, your grade will be based on the number of guards that you need (assuming that you cover all the hallways of the museum). You may resubmit as many times as you like before the deadline.

4 Placing gas stations

Assume that you are in charge of setting up a series of gas stations across the country in the 1950s. The goal is to maximize efficiency while still serving the public: Gas stations are always required to be at intersections, and, from any given road intersection, you are *required* to have a gas station somewhere within k miles. Given these requirements, your goal is to place the minimum number of gas stations that you can throughout the network.

The input to your program, network.txt, will be a file containing description of the network and the value of the number k. The value of k will be the first line of the input file, and the graph/map of the roadways will consist of the lines underneath.

The graph will be given to you in an edge-list representation. The first number on the edge will represent the vertex (all non-negative integers), and the second will represent the weight. Thus, a triangle with 2 edges of weight 10 and one edge with weight 2 might look like the following.

0:[1,10][2,10]
1:[0,10][2,2]
2:[0,10][1,2]

The first line indicates that vertex 0 is attached to vertex 1 with weight 10 and vertex 2 with weight 10. The second line indicates that vertex 1 is attached to vertex 0 with weight 10 and vertex 2 with weight 2. The third line indicates that vertex 2 is attached to vertex 0 with weight 10 and vertex 1 with weight 2.

In the file stations.txt, You will output the vertex numbers on which the gas stations are to be placed, separated by x's. For example if $k = 3$, then the results for the triangle above might be

$$0x1x$$

Assuming that your file does in fact describe a valid covering by gas stations, your grade will be based on the number of gas stations you require to service the road network.

5 Maximum satisfiability

You are a quality control officer at a major chipmaker. You have a VLSI chip that is supposed to return a false response no matter what inputs are given to it. (Obviously, nobody is ever going to design a chip that does this, but you can think of the situation as being on in which several other inputs have been set in some particular way and the resulting function circuit should therefore evaluate uniformly to false.)

A crackerjack intern has just informed your boss that the chip does not uniformly evaluate to false, but he is unable to back up his assertion to the boss. You have been ordered to determine whether the intern is correct.

You will be given a file, instance.txt, that contains a large instance of the maximum satisfiability problem. In theory, no matter what boolean values are assigned to the variables, at least one clause should evaluate to false... but you suspect that is not true. An integer will represent a variable with that name and a "-" in front of an integer will indicate negation. Literals will be separated by commas, and clauses will be separated by newlines.

The goal of this assignment is to return a file for each assignment to the variables that makes every single clause satisfied. Your files should be called solution1.txt, solution2.txt, and so on. (Any other files sent will be ignored.) Each solution file will consist of a string of zeroes and ones that correspond respectively to false and true assignments for each variable in order.

For example, a file might look like the following.

1,2,-2,-4
2,-3,5
-2

This would represent the following trivial instance of maximum satisfiability.

$x_1, x_2, \neg x_2, \neg x_4$
$x_2, \neg x_3, x_5$
$\neg x_2$

Your output should be a file that only contains 0's and 1's. A 0 (resp. 1) value in byte i will indicate a FALSE (resp. TRUE) value for x_i (where we start counting at 1 as opposed to 0). An example output might be 10110 (x_1 is TRUE, x_2 is FALSE, etc.).