CS174: Final Project (S2019)

Dr. Alvin Grissom II

April 23, 2019

If you would like to design your own final project, it must be cleared with the professor beforehand. You will store your projects on GitHub.

1 Compact Search Tree (Java or C++)

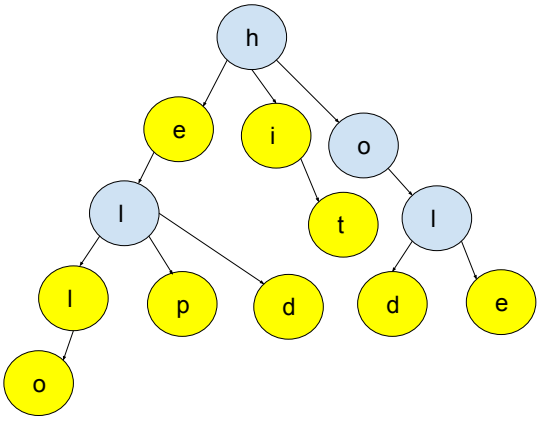
Suppose that you want to store a dictionary of strings. One way of doing this is to store each string in an array or array list. But this leads to slow lookups, since you must search through the array. If you're storing a lot of strings, this could take a long time. Another option is to use a hash table. This, however, uses a lot of RAM.

One way of accomplishing this is to use a special kind of tree, where each node in the tree stores a character in a string. Each descendant of a given node will contain the next character in a string. In this way, the tree only needs to store enough information for single characters. Internally, there should be no strings or tables inside of the tree. At most, your nodes may contain an `Character` objects and a boolean to determine whether or not a given element is contained within the tree. If you use Java, your tree should implement the Java `Set` interface and be called `SearchTreeSet`. Your implementation may also have an `ArrayList` of all of the elements in the tree.

You should be able to instantiate your tree in the following way:

```
Set<String> s = new SearchTreeSet<>();
```

Figure 1: Search tree. Blue nodes are non-terminal nodes; yellow nodes are terminal nodes. This tree contains the following words: he, hello, help, held, hit, hold, hold.



2 Maze Solver (C++)

Your second option for your final project is to create a maze game and a computer program to solve it. Your program must:

- Generate a solvable random maze given certain dimensions from the command prompt.
- Solve said maze.
- Print the solution graphically and as a list of moves (e.g., LLRUD for left, left, right, up down.).

  You can represent a maze as a matrix. The simplest way to accomplish this is to first generate a path through the maze and then fill in the rest of the matrix with walls.

  To solve the maze, your maze solver should keep track of what it has tried and how far it has gotten; otherwise, you will try the same failed paths repeatedly. Generating solutions randomly will not work. I recommend generating only simple mazes at first and solving those before generating things like dead ends.

3 General Tips for Grading

- If you use Java, do not use static methods for everything. I will take off points for this.
- Follow the Ursinus style guide.
- Compile often and save your code often.
- **Back up your code.** If you delete your code and have nothing to turn in, you will get a 0. I recommend that all of you create a free student GitHub account to backup your code and use it.
- If you prefer, you may use an IDE, such as NetBeans, for this project, but what you turn in must be runnable without it.
- Do not cheat. (Obviously).
- Ask for help if you get stuck. Do not wait until the last minute.
- You may work with 1-2 other students and turn in a joint project. This is recommended. You must describe what each group member did when you turn in your project.
- Come to exam day to present your project. If you do not, you will lose half of your grade. Discuss what you did, challenges you had, and how you overcame them (if you did).