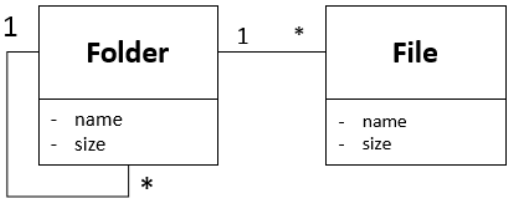
Project 2

- Choose either project 3A or 3B. **Deadline: Nov 30, 2017.** All the submissions should be made electronically on Blackboard by the end of the day.
- Please stick to the teams that I assigned you with.

Project 2A

We would like to implement a system that keeps track of folders and the files they contain. A folder has a name, a size (in bytes), and a collection of files. A folder may contain multiple folders and multiple files. A file has a name and a size. Whenever a file is added to or deleted, the size of the parent folder changes. Similarly, whenever a folder (containing files) is deleted, the size of the parent folder changes. However, adding a new (empty) folder doesn't change the size of its parent folder.



- **Requirements:** Use *mainly* ***self-balancing binary search trees*** to keep track of the data in the project. **You are welcome to use these classes:** `AVL_Tree, red_black_tree`

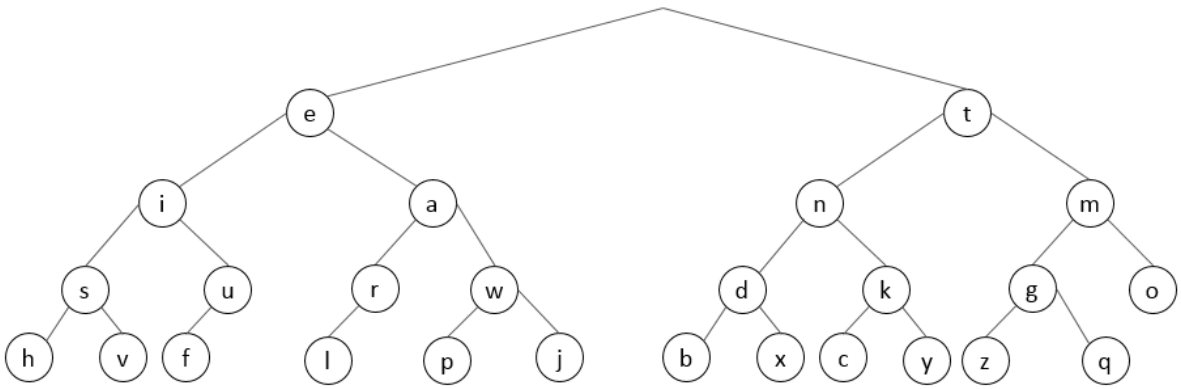
- You don't have to write a menu-based program. You can simply test your functions manually in the main function.
- **Implement functions that support the following:**
    - `void add_folder(string path, string folder_name)`
      The function adds a folder. The folder is added inside a parent folder. The function searches for the parent folder using the given path. The path is a hierarchy of folders where the last folder in the hierarchy is the parent of the to-be-added folder. Here is an example of a path:
      documents/programming/data_structures. The root folder is *documents*, and it contains folder *programming*, which contains the *data_structures* folder.
      The parent of the root folder is NULL.
    - `void delete_folder(string path, string folder_name)`
      This function deletes a folder with a given name. It searches for parent folder of the given folder using the given path.
      Notice: deleting a folder that contains files changes the size of the parent folder.
    - `void add_file(string path, string file_name, int size)`
      This function adds a file with a given name. It searches for parent folder of the file using the given path.
      Notice: adding a file changes the size of the parent folder.
    - `File get_file(string path, string file_name)`
      This function finds a file with a given name. It searches for the file using the given path.
    - `list<File> get_files(string path, string file_name)`
      This function finds all files which have names that start with the given parameter (file_name). It searches for the files using the given path.

    - `void delete_file(string path, string file_name)`
      The function deletes the file with the given name. The location of the file is specified in the path. Deleting the file changes the size of its parent folder.

Project 2B

- Morse code (see the table below) is a common code that is used to encode messages consisting of letters and digits. Each letter consists of a series of dots and dashes, for example, the code for the letter a is •- and the code for the letter b is -•••. Store each letter of the alphabet in a node of a binary tree of depth 4. The root node is at depth 0 and stores no letter. The left node at depth 1 stores the letter e (code •) and the right node stores the letter t (code is -). The four nodes at depth 2 stores the letters with codes (••,•-, -•, --). To build the tree (See the figure below), read a file in which each line consists of a letter followed by its code. The letters should be ordered by tree depth. To find the position for a letter in the tree, scan the code and branch left for a dot and branch right for a dash. Encode a message by replacing each letter by its code symbol. Then decode the message using Morse code tree. Make sure you use a delimiter symbol between coded letters.

a •-	b -•••	c -•-•
d -••	e •	f ••-•
g --•	h ••••	i ••
j •---	k -•-	l •-••
m --	n -•	o ---
p •--•	q --•-	r •-•
s •••	t -	u ••-
v •••-	w •--	x -••-
y -•--	z --••	



Technical Requirements

- (Weight: 40%) Write a function that builds the morse tree shown in the figure above. The information of the tree (the letters and the codes) is stored in a file. You can find the file here: https://www.dropbox.com/s/3cj8yb8gcdsrefg/morse.txt?dl=0 (Notice that the file hasn't been laid out in convenient order for building the tree).
- (Weight: 30%) Your system should be able to decode a message using the morse tree that you built. For example, decoding -●● --● results in "dg". The problem text briefly explains how you can do that. Notice that between the symbols (dots and dashes) is a space. The space is a delimiter that separates the codes for letters.
- (Weight: 30%) Your system should also encode a message. For example, encoding "ac" results in ●- -●-●.
- You may use a binary search tree or a map to store the codes for letters.

Facts and Assumptions

- You may assume that the character delimiters are simply spaces.
- You may assume the string has one word only.
- You are welcome to use the source code on Blackboard (e.g. `Binary_Tree`, `Binary_Search_Tree`).
- You may just call the decode and encode functions in the main function. There is no need for getting input from the user or a menu-based system.