EECS268:Lab8

Navigation

Home

Information

Syllabus
Schedule
Lecture Archive

Classwork

Labs
Submitting Work

Contents

Due Date

This lab is due one week from the start of your lab.

Overview

The next two labs will use Binary Search Trees. For this lab we will only implement adding, searching, and traversing a BST. The next lab is when we'll implement other methods like removal and copy. We'll cover more BST topics in lecture this week.

Phase 1:

- Adding to a BST
- Searching a BST
- Visiting a BST in pre, in, and post orders

Phase 2:

- Coming in lab 09...

Pokemon File

You can download a sample file here (https://github.com/jwgibbo/public_html/blob/master/eecs268/2022spring/labs/lab08/pokemon.txt). Each line is organized in the following manner (white-space delimited):

```
<american pokemon name> <pokedex number> <japanese pokemon name>
```

American Name	What the pokemon is called in the U.S.A.
pokedex number	A number associated with a particular pokemon. Think of it as a pokemon's SSN
Japanese pokemon name	What the pokemon is called in Japan

Example entries:

```
Abra    63      Casey
Aerodactyl      142     Ptera
Alakazam        65      Foodin
Arbok   24      Arbok
Arcanine        59      Windie
Articuno        144     Freezer
Beedrill        15      Spear
Bellsprout      69      Madatsubomi
Blastoise       9       Kamex
Bulbasaur       1       Fushigidane
```

Requirements

This lab and the next lab will involve implementing and using a **node-based implementation of a Binary Search Tree** (you cannot use built-in dicts like in 168). The table lists the functionality that you can accomplish by loading the Pokedex entries into a single Binary Search Tree. **Only do the requirements for phase 1 even if you know how to accomplish other tasks (e.g. removal)**

Notes:

- The user will provide the name of an input file formatted in the way described above
- Using that file, create and load a BST full of Pokemon
- Until the user wants to quit, let them use your pokedex in the ways listed in the table below:
  - The table list the menus labels and desired outcomes, but you will need to use the BinarySearchTree method that accomplishes the task
  - Example: If the user wants to **Search** you will need to call the **search(key)** method from the BinarySearchTree and see whether it produces a result or raises an exception to verify if the Pokemon is in the BST

Binary Search Tree Overview

- BSTs are binary trees (so you can use your binary nodes we made in class)
- They have rules for adding:
  - If a subtree is empty, add new value
  - If it's non-empty compare new value to value in current node
    - If new value is less than current node's value, add to left subtree
    - If new value is greater than current node's value, add to right subtree
  - No duplicates allowed (raise exceptions should this occur, but keep your program crashing!)
- Searching should take advantage of the rules for adding
  - Example if I'm looking for 10 and I'm on a node with 20, I should search 20's left subtree

Recall our three traversal orders:

Pre order	In order	Post order
1)Visit 2)Traverse LST 3)Traverse RST	1)Traverse LST 2)Visit 3)Traverse RST	1)Traverse LST 2)Traverse RST 3)Visit

Phase 1

Search	Given pokedex number (id) print all information (US name, Japanese name, pokedex number) to the user
Add	Prompt the user for a new Pokemon name (US), then new Japanese name and Pokedex number and add the entry to the tree. **Duplicates should not be allowed.** Make your add method throw an exception (std::runtime_error) if a duplicate is attempted to be added.
Print	Prompt the user for the following: - Traversal order; The user can choose for the pokedex to be written in in, pre, or post order.
Quit	Exits the program

Implementation Details

- We ask that you BST distinguish between the type of object is in the tree and the type used to search; an ItemType and a KeyType
  - The ItemType would be the type returned from a search
  - The KeyType would be the type that you can search on
  - For example, if I search for the Pokemon with ID 25 I should get back the whole entry for that word when I search.
- The BST should only use the default comparison operators, and **not be coupled** to the methods of any specific class
  - Overload the needed comparison operators for your class
  - We will order them numerically based on their pokedex number
  - A helpful function your Pokemon's operator overloads could use is isinstance (https://docs.python.org/3/library/functions.html#isinstance) (e.g. isinstance(5, int) returns True)
- You'll notice that the traversal functions take a function as a parameter. Essentially, you're passing a function that will be called on each entry in the BST.
  - The function will belong to a class/module other than the BST

Example of passing functions as parameters and using them.

```
def everything_is_awesome(something):
    print(something, "is awesome!")

def call_func_on_something(func, something):
    func(something)

def main():
    user_input = input("Hey user, what's on your mind?: ")
    call_func_on_something(everything_is_awesome, user_input) #note we just say the name of the function, no

main()
```

Sample run:

```
Hey user, what's on your mind?:
pudding
pudding is awesome!
```

Rubric

- 65% Pokedex Interaction
  - 20% Searching
  - 20% Adding entry
  - 25% Correct traversal orders
- 10% Terminal output
  - 10% well formatted output
- 15% Program stability
- 5% Logical user interface
- 5% Needed Operators overloaded