COVINS-20
Programming Project 4 (major = minor + minor)
due at the final exam

I had written a story that involved our intrepid "hero" gallivanting through the forest, hacking and clawing his way to victory as he reached the realms of Mordor. But then the Coronavirus infected Sauron IN HIS EYE and the whole saga was over and everyone is sheltering in place. Luckily for you, you're still pretty dynamic and can work on this dynamic and happening 2-3 tree, since you have 2 or 3 weeks left to do this project. Most of the memes work without having to stretch even a little bit, so you can tell that it was obviously pre-ordained.

Job Details. Your task is to write a C++ program that implements a sequential 2-3 tree. There is no parallelism in this project. I recommend using two classes, a `Node` and a `Tree`. I've written some suggested class definitions below. You will have to fill in the details of the functions (and you may need helper functions) to implement some of these:

```
class Node {

Node * parent;
int value[6];
Node * child[6];


public:
Node(int val); // constructor
bool numChildren();
void absorb(Node * newChild);
void discard(Node * removeChild);
}

class Tree {

Node * root;
void print(Node * start);

public:
Tree(); // constructor
Node * search(int valToFind);
bool insert(int valToAdd);
bool delete(int valToKill);
void print();

}
```

Here are the steps I recommend doing the project in:

1. Create the node class and write some code that reads "nodes" from a text file or otherwise manually create nodes that are "hooked together". In other words, manually build a tree. You'll want to have a clear idea of how this tree looks for testing purposes. (You might choose to implement the tree example from Uzi.) Let's assume that all the values for the nodes will range from 1 to 999.

2. The tree class is easy to create, since all you have to do is set its root, which you would have set up earlier as you built the tree.

3. First things first, write code that prints out the tree. You will need to utilize some sort of tree traveral from a previous prerequisite class. If you don't have this, you won't be able to test anything. Make sure it works. TEST THOROUGHLY.

4. Next, write `search`. You'll get a sense of how recursive calls work, and you'll return the successor. In the event that you're searching for something that is larger than the largest thing in the current tree, return 2000. (We are going to *tailor* our search range and not allow any search greater than 1989, because that could *swiftly* get out of control.)

5. Next, you'll write the `insert` function, which will require a call to `search`. (Same rules apply for the range of insertion.) Make sure that you first implement the simple cases for `absorb`, where no "bubbling" up is required. Once you've got that down, then work on the recursive cases. Don't forget that if you're inserting something that's larger than anything in the tree, you will have to handle a special base case at the root to account for it. (We discussed this in class.) TEST THOROUGHLY, and make sure that your code works for each case on both the left and right side of the tree.

6. Then write the `delete` function. Remember once again to handle the simple cases before you do the more complicated things. Take each case one at a time, and make sure you depend on the specific examples you're facing to get it to work. TEST THOROUGHLY, and make sure that your code works for each case on both the left and right side of the tree.

We have written pseudo-code for these functions in class, so all you have to do is work out all of the details and throw it together. Totally easy, right? Your project, in total, is worth 60 project points, divided by insert (30) and delete (30). You may work in groups of 2. I can help you in times of great sadness.