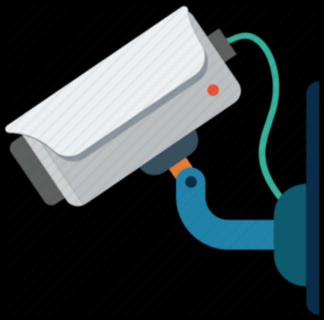
CS302

Assignment 5: Spy Cameras



Description

Facebook, Instagram, and Snapface (or whatever it's called), had an outage which set the whole world in panic...for some reason, I actually just carried on living my life, anyway, it seems like someone was able to break into the social media head quarters and cause this issue. Thus from now on we need to make sure all rooms are visible. Each room has two doors that go to two new rooms and the doors are always open (which might have been the issue from the very beginning), so we can install cameras into any room, and this camera can see the room itself and any adjacent room (every room has at most two new rooms it can go to and the room it came from). We want to set up as few cameras as possible and be able to see every room.

So you will implement a binary tree, where each node is a room, each node has up to 2 children and a parent. When a node contains a camera, that camera can see the node that it's in, it's two children and its parent. For example if you have the following binary tree

```
       1
      /
     2
    / \
   3   4
```

Each room can have a camera and all the rooms would be seen, however, if we place a camera into room 2, then all rooms can be seen, thus only 1 camera is needed

In main you will need to define the following struct type, this type will be used to declare the binary tree object

```
struct visibilityType
{
  int id; //room id number, this only is useful for debugging
  bool camera; //true/false to denote if the node contains a camera
  bool seen; //true/flase to denote if the node can be seen

  //This is used in buildTree function, you need to compare if the id is
  //equal to some integer (most likely x used will be -1), but you
  //can't access the id field directly in the binary tree class since
  //it's a templated class (well actually it just wouldn't be a good idea)
  bool operator==(int x)
  {
    return id == x;
  }
};
```

This is the binary tree class, should all be in a file: binTree.hpp

```
template <typename type>
class binTree
{
  struct binTreeNode
  {
    type data;
    binTreeNode * left;
    binTreeNode * right;
  };

public:
  class binTreeIterator
  {
  public:
    friend class binTree;
    binTreeIterator();
    binTreeIterator(binTreeNode*);
    binTreeIterator leftChild() const;
    binTreeIterator rightChild() const;
    type& operator*() const;
    bool operator==(const binTreeIterator&) const;
    bool operator!=(const binTreeIterator&) const;
private:
  binTreeNode * binTreeNodePointer;
};

  binTree();
  binTree(const binTree<type>&);
  const binTree& operator=(const binTree<type>&);
  ~binTree();

  void buildTree(std::vector<type>);
  binTreeIterator rootIterator() const;

private:
  void destroyTree(binTreeNode*);
  void copyTree(binTreeNode*, binTreeNode*);
  void buildTree(std::vector<type>, binTreeNode*, int);

  binTreeNode * root;
};
```

Each member of the `binTreeIterator` class contains/performs the following

- `binTreeNode * binTreeNodePointer` - a pointer that represents a node in a binary tree
- `binTree<type>::binTreeIterator::binTreeIterator()` - constructor that sets `binTreeNodePointer` to `NULL`
- `binTree<type>::binTreeIterator::binTreeIterator(binTreeNode* ptr)` - constructor that sets `binTreeNodePointer` to `ptr`

- `typename binTree<type>::binTreeIterator binTree<type>::binTreeIterator::leftChild()` - returns the pointer that points to the left of `binTreeNodePointer`
- `typename binTree<type>::binTreeIterator binTree<type>::binTreeIterator::rightChild()` - returns the pointer that points to the right of `binTreeNodePointer`
- `type& binTree<type>::binTreeIterator::operator*() const` - returns the data field of the node that `binTreeNodePointer` points to
- `bool binTree<type>::binTreeIterator::operator==(const binTree<type>::binTreeIterator& rhs) const` - returns `true` if `binTreeNodePointer` and `rhs.binTreeNodePointer` point to the same node, returns `false` otherwise
- `bool binTree<type>::binTreeIterator::operator!=(const binTree<type>::binTreeIterator& rhs) const` - returns `true` if `binTreeNodePointer` and `rhs.binTreeNodePointer` point to a different node, returns `false` otherwise

Each member of the `binTree` class contains/performs the following

- `binTreeNode * root` - a pointer that points to the root node
- `binTree<type>::binTree()` - default constructor that sets the root to `NULL`
- `binTree<type>::binTree(const binTree<type>& copy)` - copy constructor, performs a deep copy of the object passed in, allocates a node to root, then copies `copy.root->data` into `root->data`, then calls copyTree, passes in root and rhs.root
- `const binTree<type>& binTree<type>::operator=(const binTree<type>& rhs)` - assignment operator, performs a deep copy of the `rhs` object (calls `destroyTree` and then does the same procedure as the copy constructor)
- `binTree<type>::~binTree()` - destructor, deallocates the binary tree (calls `copyTree` function)
- `void binTree<type>::buildTree(std::vector<type> treeValues)` - builds a binary tree based on the array passed in, first you would assign a `new binTreeNode` to `root`, then set `root->data` to `treeValues[0]`, set the left and right of the root to `NULL`, then you need to build the rest of the tree by calling

  ```
  buildTree(treeValues, root, 0);
  ```

  This calls the buildTree function described below
- `void binTree<type>::buildTree(std::vector<type> treeValues, binTreeNode * r, int index)` - function that builds the left and right subtree of `r`, the steps you can follow would be
  1. `leftIndex = index * 2 + 1`
  2. `rightIndex = (index + 1) * 2`
  3. If `treeValues[leftIndex]` does not contain -1 (uses `visibilityTree` operator) for its id field, assign a `new binTreeNode` to `r->left` and assign the data field with `treeValies[leftIndex]`, otherwise set `r->left` with `NULL`
  4. If `r->left != NULL`, the call the function recursively, and pass in the vector, `r->left`, and `leftIndex` to build the left side of `r`
  5. If `treeValues[rightIndex]` does not contain -1 (uses `visibilityTree` operator) for it's value field, assign a `new binTreeNode` to `r->left` and assign the data field with `treeValies[rightIndex]`, otherwise set `r->right` with `NULL`
  6. If `r->right != NULL`, the call the function recursively, and pass in the vector, `r->right`, and `rightIndex` to build the left side of `r`
  7. Always make sure to set any new binTreeNode's left and right pointers to `NULL`

- `typename binTree<type>::binTreeIterator binTree<type>::rootIterator() const` - returns an iterator object whose `binTreeNodePointer` points to the root node
- `void binTree<type>::destroyTree(binTreeNode * r)` - deallocates the entire tree
- `void binTree<type>::copyTree(binTreeNode * i, binTreeNode * j)` - allocates a new node to `i->left` (if `j->left != NULL`), and copies the data from `j->left` into `i->left`, then calls the function recursively on the left, then does the same work on the right

Input/Contents of main

The input for the program which will be read in via command line argument, the input file will contain an integer T (denotes the test cases), then there will be T lines with a set of integers separated by space and terminated by end of line (each integer represents an id, a -1 means NULL, so the parent of this element has a NULL child). You construct a vector of `visibilityType`, set each element's id with the number on the line and camera/seen would be set to `false`

After reading the line and create the `vector` of `visibiltyType`, you allocate an object of `binTree<visibilityType>`, and call its buildTree function (by passing in the vector), then you would need to traverse the tree and determine which nodes will contain cameras and then add up all the nodes that contain cameras for the test input. Then you perform the same steps on the next line of input.

Example Output

```
$ g++ Assignment05.cpp
$ ./a.out i1.txt
Case 1:  1 cameras needed
Case 2:  1 cameras needed
Case 3:  1 cameras needed
Case 4:  2 cameras needed
Case 5:  2 cameras needed
```

Specifications

- Comment your code and your functions
- Make sure your code is memory leak free
- Make sure you have 3 files uploaded to code grade: Assignment05.cpp binTree.hpp and DeepCopy.cpp (Given to you)

Submission

Upload your source code and write up to the class site by the deadline

References

- The image used in this write can be found at https://www.downloadclipart.net/browse/83473/security-camera-png-file-clipart