CSE 100: BST
OPERATIONS
Announcements

• Gradesource and clickers:
  • We’ll be making one more pass for unregistered clickers tonight, but after that you’ll be on your own…
How is Assignment 1 going?

A. I haven’t looked at it.
B. I’ve read it, but I haven’t done anything
C. I’ve gotten the code and possibly started looking at it/playing around with it.
D. I’ve implemented some of the required functions, but I’m not done.
E. I’m done!
Q1: Which of the following is NOT a type of depth-first traversal of a BST?

A. In–order \[LVR\]
B. Pre-order \[VLR\]
C. Post-order \[LRV\]
D. Parallel-order
Q2: Which abstract data type is naturally used to support a depth-first traversal of a BST?

A. Stack  
B. Queue  
C. Dictionary  
D. Graph

1. Recursion (using runtime stack)
2. Stack
3. PA1: Successor & iterators
Q3: True or False: An element inserted into a BST is always inserted as a leaf. (Recall that a leaf is simply a node with no children, which includes the root).

A. True  
B. False
Q4: When is 'Deletion by Merging' used while deleting a node from a BST?

A. When the node being deleted is a leaf node

B. When the node being deleted has one child

C. When the node being deleted has two children

D. When the node being deleted has one subtree
Deleting a node from a BST

Delete by merging.

\[ \text{key}(T_r) > \text{key}(T_l) > \max(\text{key } T_l) \]
Which of the following results from an in-order traversal of a BST?

A. Nodes are visited in the order in which they were inserted into the BST

B. Nodes are visited in order of the number of children that they have

C. Nodes are visited in sorted order of their keys

D. None of the above
The successor method performs an in-order tree traversal, one step at a time.

A. True  
B. False

but the successor method should start with the node that has the smallest key.
The successor method performs an in-order tree traversal, one step at a time.

Find several different cases to consider in the tree below; describe how to find the successor in each case.

**Case 1:** Node has a right subtree. The successor is the node with min key in the right subtree.

**Case 2:** Node has no right subtree. Find a parent (a node higher up in the tree) that contains the node in its left subtree.

**Proof:** If a node has a right subtree, its successor is contained in the right subtree.
Prove: If a node has a right subtree, its successor resides in its right subtree.

Case 1. Consider a node that is in the left subtree of its parent (P).

By the property of BSTs:

key \( (N) < \min \{ \text{key}(T_R) \} \quad \text{--- (1)}

Since \( T_R \) is in the left subtree of \( P \)

\( \min \{ \text{key}(T_R) \} < \text{key}(P) \)

From (1), (2)

\( \text{key}(N) < \min \{ \text{key}(T_R) \} < \text{key}(P) \)
Case 2: The node $N$ is in the right subtree of its parent $P$.

$T_R$: Right subtree of $N$

By the property of BSTs:

$\text{key}(P) < \text{key}(N) < \text{key}(T_R)$

In this case again the successor of $N$ should be in $T_R$, specifically it is the min \{key($T_R$)\} i.e. the node with the min key in $T_R$. 

Case 2
In Java:

```java
public class BSTNode {
    public BSTNode left;
    public BSTNode right;
    public BSTNode parent;
    public int data;

    public BSTNode(int d) {
        data = d;
    }
}
```

C++, attempt 2:

```cpp
class BSTNode {
    public:
        BSTNode left;
        BSTNode right;
        BSTNode parent;
        int const data;

    BSTNode(const int & d) {
        data = d;
    }

};
```

Which of the following is a problem with the C++ implementation above?

A. Because data is a constant variable, the constructor will cause an error.
B. You cannot pass an integer by reference into a function. Integers must be passed by value.
C. Since d is passed by reference, you cannot assign its value to data, which is an int. You need to dereference it first.
D. The constructor needs a semi-colon at the end of its definition.

**Answer:** C.
In Java:

```java
public class BSTNode {
    public BSTNode left;
    public BSTNode right;
    public BSTNode parent;
    public int data;

    public BSTNode( int d ) {
        data = d;
    }
}
```

C++, attempt 2:

```cpp
class BSTNode {
public:
    BSTNode left;
    BSTNode right;
    BSTNode parent;
    int const data;

    BSTNode( const int & d ) {
        data = d;
    }
};
```

Which of the following is a problem with the C++ implementation above?
A. Because data is a constant variable, the constructor will cause an error.
In Java:

```java
public class BSTNode {
    public BSTNode left;
    public BSTNode right;
    public BSTNode parent;
    public int data;

    public BSTNode( int d ) {
        data = d;
    }
}
```

C++, attempt 3:

```cpp
class BSTNode {
public:
    BSTNode left;
    BSTNode right;
    BSTNode parent;
    int const data;

    BSTNode( const int & d ) : data(d) {  }
};
```

What is the problem with how we have declared left, right and parent above?

A. They should be `BSTNode*` (pointers to `BSTNodes`) and not `BSTNode` type.
B. They should be declared to be `const`
C. They should be declared as `BSTNode&` (reference variables).
In Java:

```java
public class BSTNode {
    public BSTNode left;
    public BSTNode right;
    public BSTNode parent;
    public int data;

    public BSTNode(int d) {
        data = d;
    }
}
```

C++, attempt 4:

```cpp
class BSTNode {
public:
    BSTNode* left;
    BSTNode* right;
    BSTNode* parent;
    int const data;

    BSTNode(const int & d) :
        data(d) {
    }
};
```

And now, a little practice with pointers…
Which of the following statements is true about this code?

```c
int a = 5;
int b = a;
int* pt1 = a;
```

A. Both ptr and b can be used to change the value of a.
B. Only pts can be used to change the value of a.
C. This code causes a compile error.

```
int* pt1 = &a; // Need to assign address of a to pt1
```
int a = 5;
int b = a;
int* pt1 = &a;

<table>
<thead>
<tr>
<th>address</th>
<th>memory cell</th>
<th>identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>512000</td>
<td>5</td>
<td>a</td>
</tr>
<tr>
<td>512004</td>
<td>5</td>
<td>b</td>
</tr>
<tr>
<td>512008</td>
<td>512000</td>
<td>pt1</td>
</tr>
</tbody>
</table>
int a = 5;
int b = a;
int* pt1 = &a;
How does the diagram change if we change the code as follows. Which is the correct picture now?

```c
int a = 5;
int & b = a;
int* pt1 = &a;
```

A.  
```
  a: 5
  b:  
  pt1:  
```

B.  
```
  a: 5
  b:  
  pt1:  
```

C.  
```
  a: 5
  b:  
  pt1:  
```
In Java:

```java
public class BSTNode {
    public BSTNode left;
    public BSTNode right;
    public BSTNode parent;
    public int data;

    public BSTNode( int d ) {
        data = d;
    }
}
```

C++, attempt 4:

```cpp
class BSTNode {
public:
    BSTNode* left;
    BSTNode* right;
    BSTNode* parent;
    int const data;

    BSTNode( const int & d ) :
        data(d) {} 
};
```

Are there any remaining problems with this C++ implementation?
A. Yes
B. No
In Java:

```java
public class BSTNode {
    public BSTNode left;
    public BSTNode right;
    public BSTNode parent;
    public int data;

    public BSTNode( int d ) {
        data = d;
    }
}
```

C++, attempt 5:

```cpp
class BSTNode {
    public:
        BSTNode* left;
        BSTNode* right;
        BSTNode* parent;
        int const data;

    BSTNode( const int & d ) :
        data(d) {
            left = right = parent = 0;
        }
}
```

**ALWAYS initialize in C++. C++ won’t do it for you. Why not?**

**What if we don’t want to be stuck with ints?**
BST, with templates:

template<
typename Data>

class BSTNode {
public:
    BSTNode<Data>* left;
    BSTNode<Data>* right;
    BSTNode<Data>* parent;
    Data const data;

    BSTNode( const Data & d ) :
    data(d) {
        left = right = parent = 0;
    }
};