Data Structures Lab

Keeping Everything Sorted
Notes on Assignment 1

- Assignment 1 grades have been sent
  ○ Check your CS email
  ○ Good job overall

- A few late/missing submissions
  ○ Talk to me if you're having trouble
  ○ Come to office hours

- One pair submission
  ○ I'd love to see more
Notes on Assignment 1

- Two cases for insert
  - Inserting first node
  - Inserting any other node

- Also two (main) cases for remove
  - Removing generic node
  - Removing last node

- What about removing from an empty list?
  - Easy for a void remove method
  - Trickier if you're expecting a string
  - NULL is not a string type!
Notes on Assignment 1

- Any lingering linked list questions?
Pointers

- What exactly does the * operator do?
  - Depends on context

- Declares a pointer type
  - Node* np = new Node();

- Dereferences a pointer
  - Node n = *np;
Pointers

- We can use the * operator to dereference a pointer
  - Get the object at a specific location in memory

- What if we want to do the opposite?
  - Get the memory location of a specific object
Pointers

● We can use the * operator to dereference a pointer
  ○ Get the object at a specific location in memory

● What if we want to do the opposite?
  ○ Get the memory location of a specific object

● Use the & operator
  ○ Node* np = &n;
Pointers

● We can use the * operator to dereference a pointer
  ○ Get the object at a specific location in memory

● What if we want to do the opposite?
  ○ Get the memory location of a specific object

● Use the & operator
  ○ Node* np = &n;

● What gets printed in the following code?
  ○ int x = 5;
  ○ int * y = & x;
  ○ cout << x << endl << y << endl << * y << endl;
Pointers

- Be careful when using * and &
  - Just because your compiler errors go away, doesn't mean your code is right.

- Consider the following code:
  - int x = 5;
  - int * ptr;
  - ptr = x; | * ptr = x; | ptr = & x;

- Which lines compile?
- Which lines work as intended?
Pointers - Advantages

● Why use pointers instead of objects?

● When you copy an object, you make a new copy of it in memory
  ○ Node n = Node();
  ○ Node m = n;

● When you copy a pointer, you only copy a link to an object

● Similarly for function calls
  ○ When you pass an object into a function, you make a new copy of the object
Pointers - Disadvantages

● Advantages of objects
  ○ Don't need to use "->" to access data members
  ○ Don't need to dereference all the time
  ○ Syntactically easier to deal with

● Is there some middle ground?
  ○ A pointer we could treat like an object...
References

- The & operator has two meanings too
  - Gets the address of an object
  - Declares a reference type

- References act like constantly dereferenced pointers
  - `int x = 5;`
  - `int & z = x;`
  - `cout << x << endl;`
  - `cout << z << endl;`

- If we modify a reference, the original object is also modified
  - `++z;`
  - `cout << x << endl;`
Pop Quiz (don't worry, it's not graded)

- Consider three functions:
  - void f1(int x) { ++x; }
  - void f2(int * x) { ++x; }
  - void f3(int & x) { ++x; }

- What happens when we call them?
  - int x = 5;
  - int * y = & x;
  - int & z = x;
  - f1(x);
  - f2(y);
  - f3(z);
Pop Quiz (don't worry, it's not graded)

- Consider three functions:
  - void f1(int x)    { ++x; }
  - void f2(int * x)  { ++x; }
  - void f3(int & x) { ++x; }

- How could we change f2 to act like f3?
Pop Quiz (don't worry, it's not graded)

- Consider three functions:
  - void f1(int x) { ++x; }
  - void f2(int * x) { ++x; }
  - void f3(int & x) { ++x; }

- How could we change f2 to act like f3?
  - void f2(int * x) { ++(*x); }
Review

● The * operator
  ○ Declares a pointer type
  ○ Dereferences a pointer (returns an object)

● The & operator
  ○ Declares a reference type
  ○ Gets the address of an object (returns a pointer)

● References act like constantly dereferenced pointers

● Modifying a reference modifies the object it's a reference to
Assignment 2

- Due Friday, February 10th
  - One week

- Binary Search Trees
  - Have your cake and sort it too

Heigh Ho...
Assignment 2

- The dwarves have returned home with a cart of diamonds
  - (all of unique integer weights)

- Unfortunately, the wicked witch has imposed a diamond tax
  - Each tax specifies a weight $w$
  - It must be paid with a diamond weighing at least $w$

- If the dwarves can't pay, she'll repossess their house
Homework 2

- Solutions are not trivial
- How can we sort through sets of arbitrary numbers
  - Need some data structure that imposes order
- What do we need to do?
  - Insert
  - Next
  - Remove
Binary Search Trees

- Each node holds a value
- Each node contains two children
  - Value of left child < value of node
  - Value of right child > value of node
- BST Operations:
  - Insert
  - Next
  - Remove
Binary Search Trees - Inserting

- How do we insert a new element into a BST?
Binary Search Trees - Inserting

- Compare it to the root
  - If it's smaller, insert into the left subtree
  - If it's larger, insert into the right subtree
  - Stop when you reach an empty subtree
Binary Search Trees - Inserting
Binary Search Trees - Inserting
Binary Search Trees - Inserting
Binary Search Trees - Inserting

Diagram:
- Root: 10
- Left of 10: 5
  - Left of 5: 3
  - Right of 5: 7
- Right of 10: 13
  - Right of 13: 12
Binary Search Trees - Find

- How do you find an element in a BST?
  - We'll get to "Next" soon
Binary Search Trees - Find

- Just like insert
  - Check the root node
  - If it's lower, search the left subtree
  - If it's greater, search the right subtree
  - Stop if you find the element, or reach an empty subtree
Binary Search Trees - Find

```
    7?
   /   
  10   13
   /     
  5    11
 /     /  
3 7 12
```
Binary Search Trees - Find

```
    7?
   / 
  5   10
 / \
3  7  13
 / \ \
7   12
```
Binary Search Trees - Find

Diagram:

```
  10
   /  
  5   13
  /  \/  /
 3   7? 12
```

- The diagram represents a binary search tree.
- The root node is 10.
- The left child of 10 is 5, and the right child is 13.
- The left child of 5 is 3, and the right child is 7.
- The value 7? is highlighted, indicating the search for the value 7.
- The search in this binary search tree would go left from 10 to 5, then right from 5 to 7, confirming that 7 is present in the tree.
Binary Search Trees - Next

- Find returns a boolean
  - Either the node is present or it isn't

- What if you want to return the next largest number instead?
Binary Search Trees - Next

- Answer is either in current node or somewhere in child tree
  - Recurse on appropriate child
  - Check result
Binary Search Trees - Next

- Answer is either in current node or somewhere in child tree
  - Recurse on appropriate child
  - Check result
Binary Search Trees - Next

- Answer is either in current node or somewhere in child tree
  - Recurse on appropriate child
  - Check result
Binary Search Trees - Next

- Answer is either in current node or somewhere in child tree
  - Recurse on appropriate child
  - Check result
Binary Search Trees - Next

- Answer is either in current node or somewhere in child tree
  - Recurse on appropriate child
  - Check result
Binary Search Trees - Next

- Answer is either in current node or somewhere in child tree
  - Recurse on appropriate child
  - Check result
How do we remove a node from a BST?
Binary Search Trees - Remove

- How do we remove a node from a BST?
- Depends on the node:
  - Nodes with no children
  - Nodes with one child
  - Nodes with two children
- Nodes with no children are easy
  - Just delete them
Binary Search Trees - Remove

- Nodes with no children are easy
  - Just delete them
Binary Search Trees - Remove

- Nodes with one child are a little trickier...
• Nodes with one child are a little trickier...
  ○ Connect the parent to the child
  ○ And then delete it
Binary Search Trees - Remove

- Nodes with one child are a little trickier...
  - Connect the parent to the child
• Nodes with one child are a little trickier…
  ○ Connect the parent to the child
What about nodes with two children?
- Can't delete it
- Can't replace it with a child
Binary Search Trees - Remove

- Don't delete it at all!
  - Find its in-order predecessor or successor node
  - Swap values
  - Delete the other node

- Why does this work?
Binary Search Trees - Remove

- **Issues**
  - What happens if we always choose the predecessor?
  - What happens if we always choose the successor?
Binary Search Trees - Remove

• Issues
  ○ We don't know which node to delete until we get there
  ○ Why is that a problem?
Binary Search Trees - Remove

● Issues
  ○ We don't know which node to delete until we get there
  ○ Why is that a problem?

● Need to modify parent's child pointers
  ○ Any ideas?
Binary Search Trees - Remove

- Parent pointers
  - Good. But each parent has two children
  - if (curr->parent->left == curr) ...

- Have remove method take pointer pointer argument
  - Don't need to check anything
    - remove(Node** childpp)
    - *childpp = NULL

- Pointer pointers are ugly
  - There must be a better way...
References to the rescue

● References act like constantly dereferenced pointers
  ○ int x = 5;
  ○ int & z = x;
  ○ cout << x << endl;
  ○ cout << z << endl;

● If we modify a reference, the original is also modified
  ○ ++z;
  ○ cout << x << endl;

● What if we pass a Node reference into our delete method?
Binary Search Trees vs Linked Lists

- Why don't we just use a linked list instead?
  - It's much easier to implement
  - Besides, you've already made one

- We could modify our linked list from last class to find and delete elements
  - Save a lot of time and effort

- Are Binary Search Trees really that much more efficient?
Binary Search Trees vs Linked Lists

• Insert
Binary Search Trees vs Linked Lists

- Insert
  - Binary Search Tree - $O(\log n)$
  - Linked List - $O(1)$

- Find
Binary Search Trees vs Linked Lists

- **Insert**
  - Binary Search Tree - $O(\log n)$
  - Linked List - $O(1)$

- **Find**
  - Binary Search Tree - $O(\log n)$
  - Linked List - $O(n)$

- **Remove**
Binary Search Trees vs Linked Lists

- **Insert**
  - Binary Search Tree - $O(\log n)$
  - Linked List - $O(1)$

- **Find**
  - Binary Search Tree - $O(\log n)$
  - Linked List - $O(n)$

- **Remove**
  - Binary Search Tree - $O(\log n)$
  - Linked List - $O(n)$
Binary Search Trees vs Linked Lists

- So what's the big deal?
  - $n$ insert operations
  - $n$ find operations
  - $n$ delete operations
Binary Search Trees vs Linked Lists

- So what's the big deal?
  - n insert operations
  - n find operations
  - n delete operations

- Binary Search Tree
  - $O(3 \times n \log n) = O(n \log n)$

- Linked List
  - $O(n + 2 \times n^2) = O(n^2)$

- That's a big difference
Code Review