Data Structures Lab

Keeping Everything Sorted
Notes on Assignment 1

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

- Grades ranged all over the place
  - Don't worry if you got fewer points than you expected
  - I can be a stingy grader
  - This is a pass/fail course

- Four pair submissions
  - Good number
  - But I'd love to see more
Notes on Assignment 1

- Lots of submissions with runtime issues
- The following operations should take constant time:
  - Inserting into a linked list
  - Removing the first element of a linked list
  - Cycling the head of your linked list
- Cycling through your entire list does not take constant time
- I provide test cases for a reason
  - Use them
Notes on Assignment 1

● Always end lines with a newline
  ○ endl
  ○ "\n"

● Delete objects when they are no longer used

● Check your edge cases
  ○ I will always test your code on a very large input
  ○ I will always test your code on a trivially small input
Notes on Assignment 1

- Any lingering linked list questions?
Code Review
Tips for Future Assignments

● GCC gives really poor error messages by default
  ○ But you can ask it for warnings
  ○ g++ -Wall myFile.cpp

● Your data structures are going to get more complicated
  ○ Harder to keep track of what's in them
  ○ A print method will do wonders for you

● So far I haven't specified filenames.
  ○ Starting on the next assignment, I will
  ○ Make sure to check the submission section
Pointers

- What exactly does the * operator do?
Pointers

- What exactly does the * operator do?
- Depends on context:
  - Declares a pointer type
  - Dereferences a pointer
Pointers - Declaring

● Where do pointers come from?

● Creating an object on the heap with "new"
  ○ Node * n = new Node();

● Using the "address of" operator on an existing variable
  ○ int x = 5;
  ○ int * y = & x;

● In both of these cases, the * denotes a pointer type
Pointers - Dereferencing

● What if we want to get values out of pointers?

● The * operator dereferences a pointer
  ○ Node * n = new Node();
  ○ Node m = * n;

● A dereferenced pointer can be treated like an object
  ○ n->next
  ○ (*n).next

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

● 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?
Assignment 2

- Due Friday, October 28th
  - 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
Assignment 2

Diamonds

10
5
12
3

Taxes

5
8
11
Assignment 2

Diamonds: 10, 5, 12, 3
Taxes: 5, 8, 11

There is a connection from 10 in Diamonds to 5 in Taxes.
Assignment 2

Diamonds       Taxes
10             5
 12             8
  3             11
Assignment 2

Diamonds

10
5
12
3

Taxes

6
8
11
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
  - Delete
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
  - Delete
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
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

5

3 7 12

13
Binary Search Trees - Find
Binary Search Trees - Find
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

- Similar to find
  - Search through the tree
  - If you find the number, return it
  - But keep track of the next best number, just in case
Binary Search Trees - Next

nextBest = 10
nextBest = 10

Binary Search Trees - Next
Binary Search Trees - Next

nextBest = 7
Binary Search Trees - Next

- Complications:
  - What should nextBest be initially?
  - What if no number satisfies next?
    - How will we know?
How do we delete a node from a BST?
How do we delete a node from a BST?

- Depends on the node:
  - Nodes with no children
  - Nodes with one child
  - Nodes with two children
Binary Search Trees - Delete

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

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

- Nodes with one child are a little trickier...
Binary Search Trees - Delete

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

- Nodes with one child are a little trickier...
  - Connect the parent to the child
Binary Search Trees - Delete

- Nodes with one child are a little trickier...
  - Connect the parent to the child
Binary Search Trees - Delete

- What about nodes with two children?
  - Can't delete it
  - Can't replace it with a child
Binary Search Trees - Delete

- 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 - Delete

- Issues
  - What happens if we always choose the predecessor?
  - What happens if we always choose the successor?
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