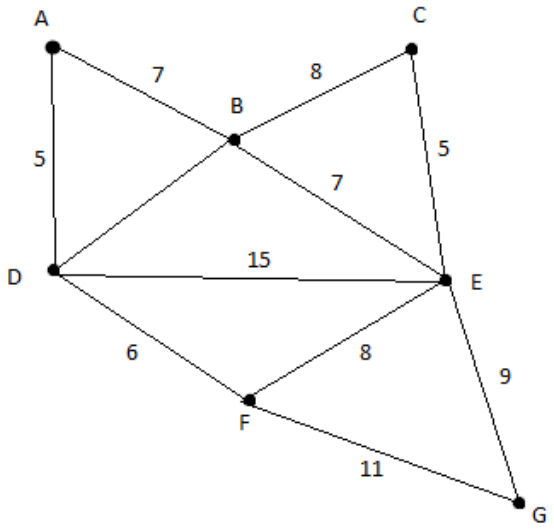
CS61B Fall 2014 - Guerrilla Section 3

1. Revisiting some Graph Algorithms

Help Jene review some graph algorithms! Given the graph below. Run the following algorithms, starting at vertex A:
(Break ties in alphabetic order)
a) DFS
b) BFS
c) Dijkstra's
d) Prim's
e) Kruskal's

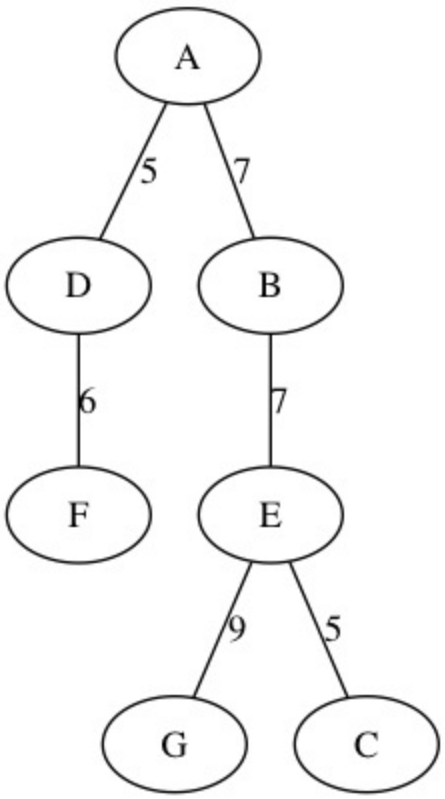


Edit: edge DB has weight 9.

a) DFS : ABCEDFG
b) BFS : ABDCEFG
c) Dijkstra's:

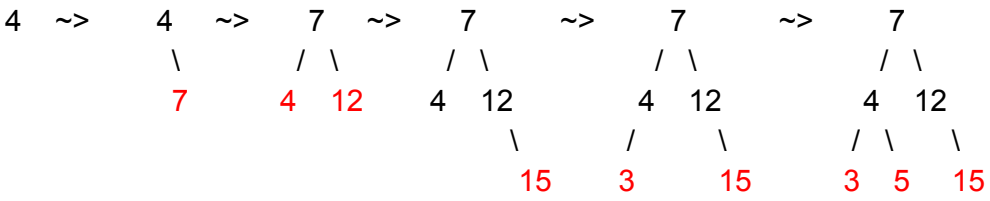
Node	Parent Pointer	Distance from Source
A	null	0
B	A	7
C	B	15
D	A	5
E	B	14
F	D	11
G	F	22

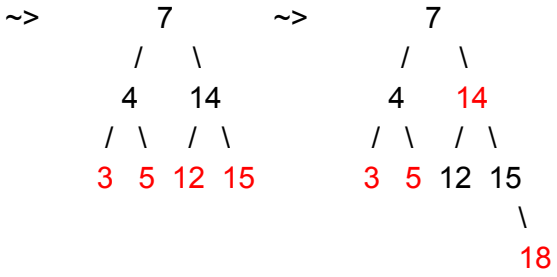
d) e)
MST:



2. Balanced Search Trees

Q. Starting with an empty red-black tree, draw the result of inserting each of the following values, in this order [4, 7, 12, 15, 3, 5, 14, 18] in the space below (old 61b worksheet).





b) Given a (2, 4) tree containing N keys, how would you obtain the keys in sorted order in worst case O(N) time? We don't need actual code—pseudo code or an unambiguous description will do (Final Fall '13).

Simply generalize an inorder traversal: traverse the left (first) child of the node, emit the first key, traverse the second child of the node, emit the second key, etc

c) If a (2,4) tree has depth h (that is, the (empty) leaves are at distance h from the root), what is the maximum number of comparisons done in the corresponding red-black tree to find whether a certain key is present in the tree? (Final Spring '06)

2h comparisons.

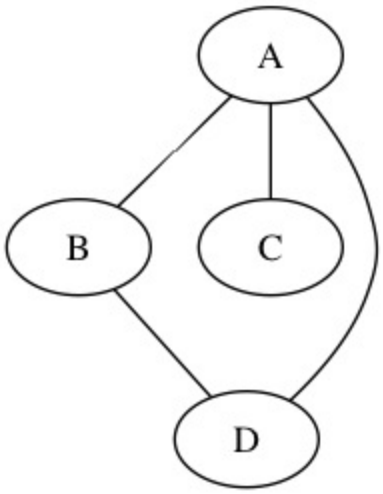
3. Edgehands' Adventures (Summer '14 Review):

a) Master brogrammer, Edwin Edgehands decides to try his hand at implementing the Depth First traversal algorithm. Here is Edgehands' pseudocode:

```
Create a new Stack of Vertices
Push the start vertex and mark it
While the fringe is not empty:
      pop a vertex off the fringe and visit it
      for each neighbor of the vertex:
            if neighbor not marked:
                  push neighbor on to the fringe
                  mark neighbor
```

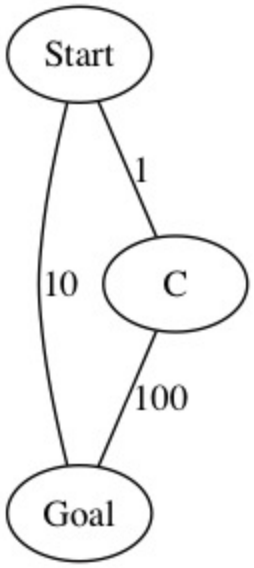
Your 61B TA, Shu claims that the traversal algorithm above is not quite DFS. Give an example of a graph which it may not traverse in valid DFS order.

Start at A for the graph below.



b) Edgehand's favorite algorithm is Dijkstra's shortest path algorithm. He likes it so much that he applies it to all of his life's problems. One day he decides to apply it to find the longest path. He modifies the algorithm to use a max priority queue instead of a min priority queue. Is Edgehand's plan a good one? Explain why, if so. Provide a case where it fails, if not.

Edgehands' plan fails on the following graph (chooses start -> goal instead of start -> C -> goal).



4. Paulittho's Adventures in HashLand (Summer '14 Review):

Paulittho Fingerman has come up with an ingenious scheme to make a Person class where each instance has a unique hashCode. His code is shown below.

```
public class Person {
   private int id;
   private String name;
   private static int count = 0;

   public Person(String name) {
     this.name = name;
      count++;
      id = count;
   }
   public boolean equals(Object obj) {
      if (obj == null || !(obj instanceof Person)) return false;
      return name.equals(((Person) obj).name);
   }
   public int hashCode() {
       return id;
   }
   public String getName() {
       return name;
   }
}
```

Q1: Are there any compile time errors?
No
Q2: say we add the following main method to the Person class:

```
public static void main(String[] args) {
   Person Jimmy = new Person("Ryan");
   Person Leo = new Person("Leo");
   System.out.println(Leo.hashCode());
}
```

What does the program print?
2

Q3: Is this hashCode method produce a valid hashCode?
Why or why not?
Invalid because Persons that are considered equal may not have the same hashCode.

5. Fun with Numbers (Fall 2013 Final)

Patrick Mathemagician is obsessed with number theory, he decides to quiz 61b students on their numeric skills. Provide short answers to the following questions that Patrick poses. In each case, include an explanation of the answer.

Q. What does n & (n - 1) == 0 test?

Tests whether n (treated as an unsigned-nonegative-number is a power of 2 or 0.

Q. Write a single integer expression that is equal to x*273 (273 in binary is 100010001), but that does not use '*', '/', or '%'. The expression must use fewer than 30 characters. Ignore overflow.

(x << 8) + (x << 4) + x

c. Assuming that 0 ≤ x ≤ 15, write a single integer expression that is equal to x*273, as for part (b), but that not use any of the standard arithmetic operators '*', '/', '%', '+', or '-'. Again, the expression must use fewer than 30 characters.

(x << 8) | (x << 4) | x

6. Which Data Structure? (Spring 2006 Final)

Jerry has decided to spend winter break working on 6 different projects. For each project he has come up with a task that's integral to the project's functionality. Jerry decides to ask his 61b students for help. For each task below, list one or more of the following search-related data structures that might reasonably be used as part of an efficient solution of the problem for large inputs. In each case, say concisely what the data structure would be used for (that is, what part it would play in the program).

1. a trie
2. hash table
3. priority queue (or heap)
4. stack (array implementation)
5. queue (circular buffer implementation)
6. sorted array
7. red-black tree (or B-tree).

a) Implementing the Lisp (or Scheme) reader. This module takes identifiers (strings) and maps them to "symbols" (some kind of object) so that the same string always stands for the same symbol object and differing strings stand for different symbol objects. It operates throughout the execution of a Lisp program, creating new symbols as new identifiers are encountered during the reading of Lisp expressions from input.

Hash Table

b) Computing how best to intercept a fleeing car, knowing a map of the city and the presumed route of the car. We are interested here in the data structure used in doing this computation, assuming the data are already present.

Priority Queue

c) Storing information about a set of points along a line that can be modified (that is, new points may be added or deleted from the set), and one can find points given an approximate position

Red-Black Tree

d) Managing a discrete-event simulation. In this kind of simulation, the program simulates the occurrence and interaction of events that occur at particular (simulated) times in the order in which they would occur in “real life.” Any event, when simulated, may cause new events that will happen later (e.g., the event of a machine tool starting will cause the later event of a finished piece of metal leaving that work station for the next). We say the times are simulated because the program processes events as fast as possible, essentially skipping over periods when nothing happens.

Priority Queue

e) Keeping track of a set of tasks to be completed in the order they are received.

Queue

f) Storing a sequence of records that is often accessed in sorted order and often added to, but in which almost all additions sort before all previous entries.

Sorted array.

7. Generics (Su 14 Review)

1. Inverse Mapper

An InverseFunction is a function that reverse the effect of its corresponding function. Fill in the

```
public interface Function<Input, Output> {
	public Output apply(Input item);
}
public interface InverseFunction<Input, Output> extends Function<Input,
Output> {
	public Input invert(Output item);
}
public class AddsTen implements Function<Integer, Integer> {
	public Integer apply (Integer num) {
			return num + 10;
	}
}
public class InverseAddsTen extends AddsTen implements
InvertibleFunction<Integer, Integer> {
	// YOUR CODE HERE
	public Integer invert(Integer item) {
		return num - 10;
	}




}
```

2. Filter

Note: There are three parts to this question.

```
public interface Function<Input, Output> {
	public Output apply(Input item);
}


public interface BooleanFunction<T> extends Function<T, Boolean> {
}
```

```
/* A BooleanFunction that returns ture if the input is longer than 5
* characters.*/
public class LongerThanFiveChars implements BooleanFunction<String> {
	// PART A: YOUR CODE HERE
	public Boolean apply(String input) {
		return input.length() > 5;
	}









}

public class Filter<T> {

	private BooleanFunction<T> func;

	public Filter(BooleanFunction<T> function) {
			this.func = function;
	}

	/* This should be a non-destructive function. It should not change
	*  the contents of the input ArrayList. */
	public ArrayList<T> filter(ArrayList<T> in) {
			// PART B: YOUR CODE HERE
			ArrayList<T> out = new ArrayList<T>();
			for (T item : in) {
				if (func.apply(item)) {
					out.add(item);
				}
			}
			return out;


	}
```

```
    public static void main(String[] args) {
            List<String> names = Arrays.asList("green", "black",
            "chai", "oolong", "assam", "english breakfast", "earl
            grey", "milk", "genmai", "hibiscus", "white", "thai
            iced", "iced", "Quoc's famous blend");
            ArrayList<String> teas = new ArrayList<String>(names);
            // PART C: Joshua Shrug, being the tea connoisseur he
            // is, likes teas that are greater than 5 characters in
            // length. Help Joshua by using the code above to
            // print the teas that have length greater than 5.
            Filter<String> longNames = new Filter<String>(new
                                        LongerThanFiveChars());
            ArrayList<String> teas = new ArrayList<String>(names);
            ArrayList<String> longTeas = longNames.filter(teas);
            System.out.println(longTeas);
    }
}
```

8. Compressing with Linked Lists? (Spring 2012)

After attending the 61B Data Compression lecture, Sarah decides to compress all her homework for easy distribution on cheatcode.berkeley.edu (yes, you can find solutions to every hw/project there). She decides to use Run Length Encoding to help her do this.

From Wikepedia:
***"Run-length encoding** (**RLE**) is a very simple form of data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run"*

Help Sarah with her totally ethical project!

Write a method called rle in the SListNode class below that takes as input an array of Strings (the input parameter strings) and returns the head of a singly-linked list representing a run-length encoding of that array. Two adjacent Strings should be considered the same and compressed into a single run if they represent the same sequence of characters, even if they're different objects.

There is no SList class; just an SListNode class. Each SListNode has three fields: an item reference that points to a String, an int named count that records the number of occurrences of that String, and a reference to the next SListNode in the list.

Feel free to manipulate references directly. Do not assume that any methods are available unless you write them, except the constructor provided below. Assume that strings references an array whose length is at least one, so the run-length encoding will have at least one node and rle never returns null. The run-length encoding is allowed to reference the same String objects as the array, but you can copy the Strings if you prefer. (Hint: the code is simpler if you start at the end of the array and encode backward.)

Write your solution on the following page.

```
public class SListNode {
	public String item;
	public int count;
	public SListNode next;
	public SListNode(String i, int c, SListNode n) {
		item = i; count = c; next = n;
	}

	public SListNode rle(String[] strings) {
		int i = strings.length - 1;
		SListNode head = new SListNode(strings[i], 1, null);
		for (i = i - 1; i >= 0; i -= 1) {
			if (strings[i].equals(head.item)) {
				head.count += 1;
			} else {
				head = new SListNode(strings[i], 1, head);
			}
		}
		return head;
	}









}
```

9. Tree in a Tree

Write the isSubtree method which checks if a given Binary tree rooted at T1 contains another binary tree rooted at T2.

```
class TreeNode {
      int value;
      TreeNode left;
      TreeNode right;
}
```

```
public static boolean isSubTree(TreeNode t1, TreeNode t2) {
    if (t2 == null) {
            return true;
    }
    if (t1 == null) {
       return false;
    }
    return equal(t1, t2) || isSubTree(t1.left, t2) || isSubTree(t1.right,
                t2);
}

private static boolean equal(t1, t2) {
    if (t1 == null && t2 == null) {
       return true;
    }
    if (t1 == null || t2 == null) {
       return false
    }
    return t1.value == t2.value && equal(t1.left, t2.left) &&
           equal(t1.right, t2.right);
}
```

10. Dynamic Programming - Longest Common Subsequence

Given two Strings s1 and s2, write a method longestCommonSubsequence, that efficiently computes the length of the longest subsequence contained in **both** s1 and s2. Remember, subsequences are not the same as substrings.

Example.

Input:
`longestCommonSubsequence("1234", "1224533324")`
`longestCommonSubsequence("melanie", "aleks")`

`Output:`
`4`
`2`
Bonus for Bosses: Make the function return the actual lcs, that is, longestCommonSubsequence("melanie", "aleks") should return "le".

```
public static int longestCommonSubsequence(String s1, String s2) {
	int [][] memo = new int[s1.length() + 1][s2.length() + 1];
	for (int i = 0; i < s1.length(); i += 1) {
		for (int j = 0; j < s2.length(); j += 1) {
			if (s1.charAt(i) == s2.charAt(j)) {
				memo[i + 1][j + 1] = memo[i][j] + 1;
			} else {
				memo[i + 1][j + 1] = Math.max(memo[i + 1][j],
					memo[i][j + 1]);
			}
		}
	}
	return memo[s1.length()][s2.length()];
}
```