CS 2c03

Assignment #1. Due March 8 (Sunday), 2020, 23:59 via the course' svn depository. Do not hesitate to discuss with TA or instructor all the problems as soon as you discover them.
The assignment is labour consuming. Start early!

Submission instructions:

Please submit your entire assignment as a single **RAR** file. You must upload one RAR file only, including all your Java files and a single **PDF file** with the corresponding instructions on how to compile it and run some test cases. Also, put the answer of all non-programming questions in the same PDF file. Moreover, good programming style will also be marked.
Please include your MacID and student number in the PDF file.
Please use "asg#_Macid.rar" for your file name where # should be replaced by the assignment number (1,2 or 3) and Macid should be replaced by your MacID.
You must upload your solutions in the course' svn repository. Any problem with svn repository please discuss with Morteza Alipourlangouri <alipoum@mcmaster.ca>, a TA for this course.
Please check your RAR file can be extracted with no issue before submitting.

1.[5] Draw the Binary Search Tree (BST) that results when you insert the keys E A S Y Q U e s T I O N, in that order (associating the value i with the ith key, as per the convention in the text) into an initially empty tree. **Show all steps!** How many compares are needed to build the tree?

2.[8] Add to BST a method height() that computes the height of the tree. Develop two implementations: a recursive method (which takes linear time and space proportional to the height), and a method like size() that adds a field to each node in the tree (and takes linear space and constant time per query).

3.[8] Give nonrecursive implementations of get() and put() for BST.

Partial solution : Here is an implementation of get():

```
public Value get(Key key)
{
Node x = root;
while (x != null)
{
int cmp = key.compareTo(x.key);
if (cmp == 0) return x.val;
else if (cmp < 0) x = x.left;
else if (cmp > 0) x = x.right;
}
return null;
}
```

The implementation of put() is more complicated because of the need to save a pointer to the parent node to link in the new node at the bottom. Also, you need a separate pass to check whether the key is already in the table because of the need to update the counts. Since there are many more searches than inserts in performance-critical implementations, using this code for get() is justified; the corresponding change for put() might not be noticed.

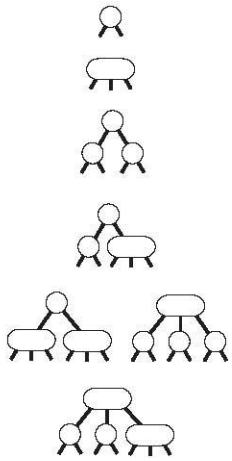
4.[5] Prove that if a node in a BST has two children, its successor has no left child and its predecessor has no right child.

5.[7] Draw the sequence of BSTs that results when you delete the keys from the tree of Question 1, one by one, in the order they were inserted.

6.[5] Draw the 2-3 tree that results when you insert the keys E A S Y Q U e s T I O N in that order into an initially empty tree.

7.[5] Find an insertion order for the keys S E A R C H X M that leads to a 2-3 tree of height 1.

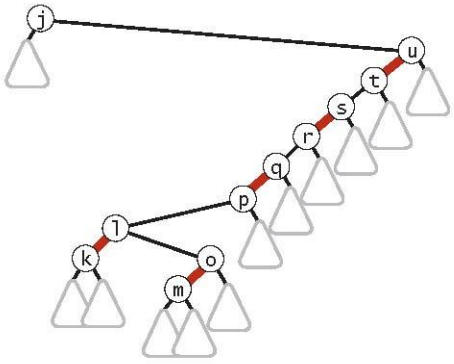
8.[7] The figure below shows all the *structurally different* 2-3 trees with *N* keys, for *N* from 1 up to 6 (ignore the order of the subtrees). Draw all the structurally different trees for *N* = 7, 8, 9, and 10.



9.[5] Prove that the height of a 2-3 tree with N keys is between ~$\lfloor \log_3 N \rfloor$ (~0.63 $\log_2 N$ (for a tree that is all 3-nodes) and ~$\lfloor \log_2 N \rfloor$ (for a tree that is all 2-nodes).

10.[5] Draw the red-black BST that results when you insert items with the keys E A S Y Q U e s T I O N in that order into an initially empty tree. **Show all steps!**

11.[5] Show the result of inserting n into the red-black BST drawn below (only the search path is shown, and you need to include only these nodes in your answer). **Show all steps!**



12.[8] Generate two random 16-node red-black BSTs. Draw them and then compare them with the (unbalanced) BSTs built with the same keys.

13.[5] Consider the B-Tree from page 20 of Lecture Notes 8. Insert first D and then L. Show all steps.

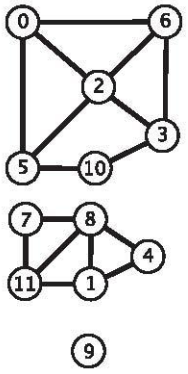
14.[5] Demonstrate the insertion of the keys 5, 28, 19, 15, 20, 33, 12, 17, 10 into a hash table with collisions resolved by chaining. Let the table have 9 slots, and let the hash function be $h(k) = k \bmod 9$.

15.[5] Professor Marley hypothesizes that substantial performance gains can be obtained if we modify the chaining scheme so that each list is kept in sorted order. How does the professor's modification affect the running time for successful searches, unsuccessful searches, insertions, and deletions?

16.[12] Consider inserting the keys 10, 22, 31, 4, 15, 28, 17, 88, 59 into a hash table of length $m = 11$ using open addressing with auxiliary has function $h'(k) = k \bmod m$. Illustrate the result of inserting these keys using *linear probing*, *quadratic probing* with $c_1 = 1$ and $c_2 = 3$, and using *double hashing* with $h_2(k) = 1 + (k \bmod (m-1))$.

17.[10] Write a program to find values of a and M, with M as small as possible, such that the hash function $h(k) = (a\ k) \bmod M$ for transforming the kth letter of the alphabet into a table index produces distinct values (no collisions) for the keys S E A R C H X M P L.
The result is known as a *perfect hash function*.

18.[12] Consider the graph G presented below:

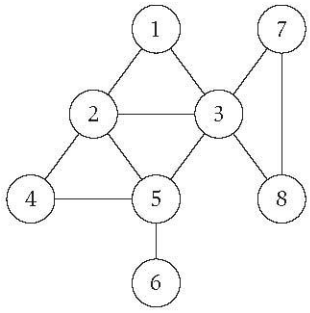


Provide its representation using:
a. Adjacency matrix
b. Array/list of edges
c. Adjacency lists

19.[10] Prove Theorem from page 6 of Lecture Notes 10.

20.[7] Add a method hasEdge() to Graph (page 526 of the textbook) which takes two int arguments v and w and returns true if the graph has an edge v-w, false otherwise.

21.[5] Consider the graph *G* below:



Find a Depth-First Search tree *T* for the above graph starting with the vertex 1. Show all the vertices as they are discovered in sequence starting from 1 to the last vertex included in *T*, as shown on page 17 of Lecture Notes 10.

22.[10] Prove that every connected graph has a vertex whose removal (including all adjacent edges) will not disconnect the graph, and write a DFS method that finds such a vertex.
Hint: Consider a vertex whose adjacent vertices are all marked.

23.[10] We can implement Depth First Search without using recursion, by implementing stack explicitly. The below pseudo-code describes such implementation:

```
DFS(s):
  Initialize S to be a stack with one element s
  While S is not empty
    Take a node u from S
    If Explored[u] = false then
       Set Explored[u] = true
       For each edge (u, v) incident to u
         Add v to the stack S
       Endfor
    Endif
  Endwhile
```

Implement (and test) the above algorithm in Java. Consider two cases:
a. A graph is represented by adjacency list.
b. A graph is represented by adjacency matrix.

24.[10] Suppose that the sequence P R I O * R * * I * T * Y * * * Q U E * * * U * E (where a letter means *insert* and an asterisk means *remove the maximum*) is applied to an initially empty priority queue represented a max-oriented heap. Give the sequence of heaps produced by operations described by this sequence.

25[10] Sort the sequence 7, 6, 9, 5, 4, 3 using (min-oriented) heap sort.

26.[10] Suppose that your application will have a huge number of *insert* operations, but only a few *remove the maximum* operations. Which priority-queue implementation do you think would be most effective: heap, unordered array, or ordered array? Justify your answer.