Assignment 3 – Data Structures and Algorithms
Deadline: Wednesday April 15 by 11:59 pm
Type: Individual Assignment
Weight: 5%

(100 points)

Q1 (12)

3.3.10 Draw the red-black BST that results when you insert items with the keys E A S Y Q U T I O N in that order into an initially empty tree.

Q2 (12)

A team of biologists keeps information about DNA structures in an **left leaning red-black tree** using as key the specific weight (an integer) of a structure. The biologists routinely ask questions of the type:
"Are there any structures in the tree with specific weights between *a* and *b* (both inclusive)", and they hope to get an answer as soon as possible. Design an efficient algorithm that given integers a and b returns true if there is a key x in the tree such that a _ x _ b, and returns false if no such key exits.

a) Describe your algorithm in **pseudo-code**.
b) What (and why) is the time complexity of the algorithm?

Q3 (12)

Assume a hash table utilizes an array of 13 elements and that collisions are handled by **separate chaining**. Considering the hash function is defined as: *h(k)=k mod 13*.

(a) Draw the contents of the table after inserting elements with the following keys:
{32, 147, 265, 195, 207, 180, 21, 16, 189, 202, 91, 94, 162, 75, 37, 77, 81, 48}

(b) What is the maximum number of collisions caused by the above insertions?

Q4 (12)

To reduce the maximum number of collisions in the hash table described in Question(3) above, someone proposed the use of a larger array of 15 elements (that is roughly 15% bigger) and of course modifying the hash function to: *h(k)=k mod 15*. The idea was to reduce the *load factor* and hence the number of collisions. Does this proposal hold any validity to it?

• If yes, indicate why such modifications would actually reduce the number of collisions.
• If no, indicate clearly the reasons you believe/think that such proposal is senseless.

Q5 (12)

Draw the 13-entry hash table that results from using the hash function *h(k) = (7k + 3) mod 13* to hash the keys 31, 45, 14, 89, 24, 95, 12, 38, 27, 16, and 25, assuming that collisions are handled by *linear probing*.

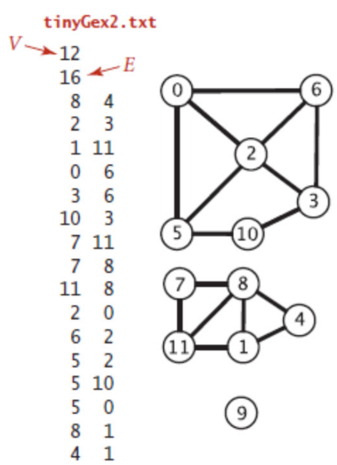
(a) What is the size of the longest cluster caused by the above insertions?
(b) What is the number of occurred collisions as a result of the above operations?

Q6 (10)

3.4.4 Write an algorithm (pseudo code) to find values of a and M, with M as small as possible, such that the hash function (a * k) % M for transforming the *k*th 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*.

Q7 (10)

4.1.2 Draw, in the style of the figure showed in the slides, the adjacency lists built by graph's input stream constructor for the file tinyGex2.txt depicted below.



Q8 (12)

4.1.9 For the graph shown in the previous question show the detailed trace of call *DFS(0)*. Also draw the tree represented by *edgeto[]*

Q9 (8)

4.1.18 The girth of a graph is the length of its shortest cycle. If a graph is acyclic, then its girth is infinite. Add a method girth() to GraphProperties that returns the girth of the graph. **Hint:** Run *BFS* from each vertex. The shortest cycle containing *s* is an edge between *s* and some vertex *v* concatenated with a shortest path between *s* and *v* (that does not use the edge s-v).