Assignment 4 – Data Structures and Algorithms
Deadline: Monday April 15 by 11:59 pm
Type: Individual Assignment
Weight: 7%

(70 points)

Q1 (5)

3.3.8 Show all possible ways that one might represent a 4-node with three 2-nodes bound together with red links (not necessarily left-leaning).

Q2 (6)

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.

Q3 (8)

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?

Q4 (7)

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?

Q5 (7)

To reduce the maximum number of collisions in the hash table described in Question(4) 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.

Q6 (8)

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*.

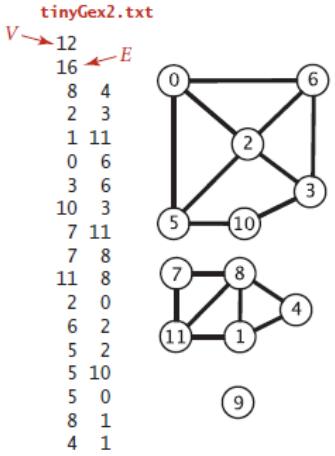
(a) What is the size of the longest cluster caused by the above insertions?
(b) What is the number of occurred collisions as a result of the above operations?

Q7 (7)

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*.

Q8 (8)

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.



Q9 (8)

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[]*

Q10 (6)

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).

Programming Question (30 points)

Assume that you want to distribute a group of employees into different brackets of salary. There is an input file (Employees.txt) that contains the information (Employee ID, Salary) of 500 employees (you can find Employees.txt file on the folder of Assignment 4). The minimum and maximum salaries (among all the 500 employees) have been written on the first two line of Employees.txt file.

Here is the explanation of the functionality of your implemented code:

1- The program asks the user to enter an arbitrary value (say k) that shows the required number of salary brackets. By having the value of k, the program will be able to calculate the range of salaries in each bracket.

2. Then the program reads the information of the employees from Employees.txt file (each entry[line] is a pair (employee ID, Salary)) and insert them to the proper brackets. (assume employee IDs are unique).

3. If the user enters the information of a new employee, the program should find the proper bracket for that employee and then insert it to that bracket if it does not already exist. If it already exists it print out a proper message to show the duplication.

Important Notes:

- Make sure you choose a data structure for your implementation that is efficient for the insert, delete and search operations. Although this program processes a small file (500 employees), it should work efficiently for the large size files with thousands or millions of employees too.

- Provide enough test cases to show the functionality of your program.