Assignment problem

1. What would be a good hash code for a vehicle identification that is a string of numbers and letters of the form "9X9XX99X9XX999999", where a "9" represents a digit and an "X" represents a letter?

Either Summing components, Polynomial hash codes, or Cyclic Shift hash codes would be appropriate (see Section 8.2.3, pages 362-364). In each case, we need to break the key into pieces. The key consists of six letters and 11 digits, so we could break it into two groups of 3 letters, two groups of four digits and one group of three digits. Since 263=17576, this gives us five numbers *x0, x1, x2, x3, x4* - whose maximum values are 17575, 17575, 9999, 9999 and 999, respectively. For the size of our hash table, we can choose a prime number near 20000 (for example, *N=19997*).

If we use summing of components, the hash function would be

*x0 + x1 + x2 + x3 + x4* mod *N*

For a polynomial hash code, we need to choose a number *z* which should also be a prime (for example, *z=37*). Then the hash funtion is

*x4 z4 + x3 z3 + x2 z2 + x1 z + x0* mod *N*

which can be calculated using "Horner's Rule" as

*x0 + z ( x1 + z ( x2 + z ( x3 + z ( x4* )))) mod *N*

(Note: "mod *N*" can be applied after each multiplication or addition, thus keeping the intermediate values between 0 and *N*.

1. Write a binary search tree method that takes two keys, *low* and *high*, and prints all elements with keys in the range between *low* and *high*. Your program shoud run in **O**(*k* + log *n*) time, where *n* is the size of the tree and *k* is the number of elements printed. (Thus if *k* is small, you should be examining only a small part of the tree.)  
   Explain why this kind of method could not be implemented efficiently if the data are stored in a hash table rather than a BST.

The inefficient way to do it is:

void printRange( int low, int high )

{

if( left != null ) {

left.printRange( low, high );

}

if(( key >= low )&&( key <= high )) {

[print this node]

}

if( right != null ) {

right.printRange( low, high );

}

}

The efficient way is:

void printRange( int low, int high )

{

if( key >= low ) {

if( left != null ) {

left.printRange( low, high );

}

if( key <= high ) {

[print this node]

}

}

if( key <= high ) {

if( right != null ) {

right.printRange( low, high );

}

}

}

Given a Graph of V vertices and E edges and another edge(c - d), the task is to find if the given edge is a Bridge**.** i.e., removing the edge disconnects the graph.

**Example 1:**

**Input:**

**c** = 1, **d** = 2

**Output:**

1

**Explanation**:

From the graph, we can clearly see that

blocking the edge 1-2 will result in

disconnection of the graph. So, it is

a Bridgeand thus the Output 1.

**Example 2:**

**Input:**

**c** = 0, **d** = 2

**Output:**

0

**Explanation**:

blocking the edge between nodes 0 and 2

won't affect the connectivity of the graph.

So, it's not a Bridge Edge. All the Bridge

Edges in the graph are marked with a blue

line in the above image.