CSE- 3004 LAB-8 Assignment

Academic year: 2020-2021 **Semester:** WIN

Faculty Name: Dr. D Sumathi Mam **Date:** 2 /9/2021

Student name: M.Taran **Reg. no.:** 19BCE7346

1Q)

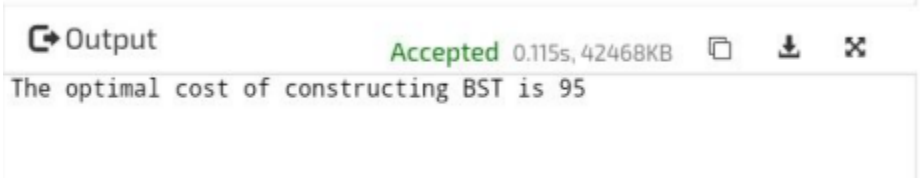
Solve optimal binary search tree using dynamic programming.

CODE :

```
Import java.util.HashMap;
Import java.util.Map;
Class Main
{
        Public static int findOptimalCost(int[] freq, int l, int j, int level,
        Map<String, Integer> lookup)
        {
                If (j < i) {
                        Return 0;
                }
                String key = l + "|" + j + "|" + level;
                If (!lookup.containsKey(key))
                {
                        Lookup.put(key, Integer.MAX_VALUE);
                        Int leftOptimalCost, rightOptimalCost;
                        For (int k = l; k <= j; k++)
                        {
        leftOptimalCost = findOptimalCost(freq, l, k – 1, level + 1, lookup);

                                rightOptimalCost = findOptimalCost(freq, k + 1, j, level + 1, lookup);
Int cost = freq[k] * level + leftOptimalCost + rightOptimalCost;
                                Lookup.put(key, Integer.min (lookup.get(key), cost));
                        }
                }
Return lookup.get(key);

        }
        Public static void main(String[] args)
        {
```

```
        Int[] freq = { 25, 10, 20 };
        Map<String, Integer> lookup = new HashMap<>();
        System.out.println("The optimal cost of constructing BST is
                + findOptimalCost(freq, 0, freq.length – 1, 1, lookup));

    }

}
```

OUTPUT:



2Q)

Suppose that we are designing a program to translate text from English to French. For each occurrence of each English word in the text, we need to look up its French equivalent. We could perform these lookup operations by building a binary search tree with n English words as keys and their French equivalents as satellite data. Because we will search the tree for each individual word in the text, we want the total time spent searching to be as low as possible. We could ensure an O.lg n/ search time per occurrence by using a red-black tree or any other balanced binary search tree. Words appear with different frequencies, however, and a frequently used word such as the may appear far from the root while a rarely used word such as machicolation appears near the root. Such an organization would slow down the translation, since the number of nodes visited when searching for a key in a binary search tree equals one plus the depth of the node containing the key. We want words that occur frequently in the text to be placed nearer the root.6Moreover, some words in the text might have no French translation,and such words would not appear in the binary search tree at all. How do we organize a binary search tree so as to minimize the number of nodes visited in all searches, given that we know how often each word occurs?

CODE :

```
Public final class Main{

   Public static void main(String[] args) {
      Final double[] p = new double[] { Double.NaN, 0.15, 0.10, 0.05, 0.1, 0.2 };
         Final double[] q = new double[] { 0.05, 0.1, 0.05, 0.05, 0.05, 0.1 };
         Final int n = 5;
         Final Tables tables = optimalBst(p, q, n);
         System.out.println(String.format("Search cost of Optimal BST is: %s", tables.e[1][n]));
```

```
        constructOptimalBst(tables.root);
}

Private static final Tables optimalBst(double[] p, double[] q, int n) {
    Final double[][] e = new double[n + 2][n + 1];
        Final double[][] w = new double[n + 2][n + 1];

        For (int l = 1; l <= n + 1; i++) {
            E[i][l – 1] = q[l – 1];
            W[i][l – 1] = q[l – 1];
        }

        Final int[][] root = new int[n + 1][n + 1];
        For (int l = 1; l <= n; l++) {
            For (int l = 1; l <= (n – l + 1); i++) {
                Final int j = l + l – 1;
                    E[i][j] = Double.MAX_VALUE;
                    W[i][j] = w[i][j – 1] + p[j] + q[j];
                    For (int r = l; r <= j; r++) {
                        Final double t = e[i][r – 1] + e[r + 1][j] + w[i][j];
                        If (t < e[i][j]) {
                            E[i][j] = t;
                                Root[i][j] = r;
                        }
                    }
            }
        }

        Return new Tables(e, root);
}

Private static void constructOptimalBstAux(int[][] root, int l, int j, int p) {
        String key = “d” + j;
        If (j >= i)
            Key = “k” + root[i][j];
        If (l == 1 && j == root.length – 1)
            System.out.println(key + “ is the root”);
        Else if (j < p)
            System.out.println(String.format(key + “ is the left child of k%d”, p));
        Else
            System.out.println(String.format(key + “ is the right child of k%d”, p));

        If (j >= i) {
```

```
            Final int r = root[i][j];
            constructOptimalBstAux(root, l, r – 1, r);
            constructOptimalBstAux(root, r + 1, j, r);
        }
    }

    Private static void constructOptimalBst(int[][] root) {
        constructOptimalBstAux(root, 1, root.length – 1, -1);
    }

    Static final class Tables {
        Final double[][] e;
        Final int[][] root;

        Public Tables(double[][] e, int[][] root) {
            Super();
            this.e = e;
            this.root = root;
        }
    }
}
```

OUTPUT: