**WEEK 10: Write a java program to implement Dynamic Programming algorithm for the Optimal Binary Search Tree Problem.**

import java.io.\*;

import java.util.\*;

class Optimal

{

public int p[];

public int q[];

public int a[];

public int w[][];

public int c[][];

public int r[][];

public int n;

int front,rear,queue[];

public Optimal(int SIZE)

{

p=new int[SIZE];

q= new int[SIZE];

a=new int[SIZE];

w=new int[SIZE][SIZE];

c=new int[SIZE][SIZE];

r=new int[SIZE][SIZE];

queue=new int[SIZE];

front=rear=-1;

}

/\* This function returns a value in the range r[i][j-1] to r[i+1][j] SO that the cost c[i][k-1] + c[k][j] is minimum \*/

public int Min\_Value(int i, int j)

{

int m,k=0;

int minimum = 32000;

for (m=r[i][j-1] ; m<=r[i+1][j] ; m++)

{

if ((c[i][m-1]+c[m][j]) < minimum)

{

minimum = c[i][m-1] + c[m][j];

k = m;

}

}

return k;

}

/\* This function builds the table from all the given probabilities It basically computes C,r,W values \*/

public void OBST()

{

int i, j, k, l, m;

for (i=0 ; i<n ; i++)

{

// Initialize

w[i][i] = q[i];

r[i][i] = c[i][i] = 0;

// Optimal trees with one node

w[i][i+1] = q[i] + q[i+1] + p[i+1];

r[i][i+1] = i+1;

c[i][i+1] = q[i] + q[i+1] + p[i+1];

}

w[n][n] = q[n];

r[n][n] = c[n][n] = 0;

// Find optimal trees with m nodes

for (m=2 ; m<=n ; m++)

{

for (i=0 ; i<=n-m ; i++)

{

j = i+m;

w[i][j] = w[i][j-1] + p[j] + q[j];

k = Min\_Value(i,j);

c[i][j] = w[i][j] + c[i][k-1] + c[k][j];

r[i][j] = k;

}

}

}

/\*This function builds the tree from the tables made by the OBST function \*/

public void build\_tree()

{

int i, j, k;

System.out.print("The Optimal Binary Search Tree For The Given Nodes Is ....\n");

System.out.print("\n The Root of this OBST is :: "+r[0][n]);

System.out.print("\n The Cost Of this OBST is :: "+c[0][n]);

System.out.print("\n\n\tNODE\tLEFT CHILD\tRIGHT CHILD");

System.out.println("\n -------------------------------------------------------");

queue[++rear] = 0;

queue[++rear] = n;

while(front != rear)

{

i = queue[++front];

j = queue[++front];

k = r[i][j];

System.out.print("\n "+k);

if (r[i][k-1] != 0)

{

System.out.print(" "+r[i][k-1]);

queue[++rear] = i;

queue[++rear] = k-1;

}

else

System.out.print(" -");

if(r[k][j] != 0)

{

System.out.print(" "+r[k][j]);

queue[++rear] = k;

queue[++rear] = j;

}

else

System.out.print(" -");

}

System.out.println("\n");

}

}

/\* This is the main function \*/

class OBSTDemo

{

public static void main (String[] args )throws IOException,NullPointerException

{

Optimal obj=new Optimal(10);

int i;

System.out.print("\n Optimal Binary Search Tree \n");

System.out.print("\n Enter the number of nodes ");

obj.n=getInt();

System.out.print("\n Enter the data as ....\n");

for (i=1;i<=obj.n;i++)

{

System.out.print("\n a["+i+"]");

obj.a[i]=getInt();

}

for (i=1 ; i<=obj.n ; i++)

{

System.out.println("p["+i+"]");

obj.p[i]=getInt();

}

for (i=0 ; i<=obj.n ; i++)

{

System.out.print("q["+i+"]");

obj.q[i]=getInt();

}

obj.OBST();

obj.build\_tree();

}

public static String getString() throws IOException

{

InputStreamReader input = new InputStreamReader(System.in);

BufferedReader b = new BufferedReader(input);

String str = b.readLine();//reading the string from console

return str;

}

public static char getChar() throws IOException

{

String str = getString();

return str.charAt(0);//reading first char of console string

}

public static int getInt() throws IOException

{

String str = getString();

return Integer.parseInt(str);//converting console string to numeric value

}

}