**ASSIGNMENT NO.-08**

**Harshal V. Sawale**

**COB222**

**Title:**

Given sequence k = k1 <k2< … <kn of n sorted keys, with a search probability pi for each key ki. Build the Binary search tree that has the least search cost given the access probability for each key?

#include <iostream>

#define SIZE 10

using namespace std;

class optimal

{

public:

int p[SIZE];

int q[SIZE];

int a[SIZE];

int w[SIZE][SIZE];

int c[SIZE][SIZE];

int r[SIZE][SIZE];

int n;

int front,rear,queue[20];

optimal() //default constructor

{

front=rear=-1;

}

void getdata();

int minvalue(int,int);

void OBST();

void buildtree();

};

void optimal::getdata()

{

int i;

cout<<"\n Optimal Binary search tree";

cout<<"\n Enter the number of nodes :";

cin>>n;

cout<<"\n Enter the data : \n";

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

{

cout<<"\n a["<<i<<"]:";

cin>>a[i];

}

cout<<"\n Enter probalities for successful search \n";

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

{

cout<<"p["<<i<<"]:";

cin>>p[i];

}

cout<<"\n Enter probalities for unsuccessful search \n";

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

{

cout<<"q["<<i<<"]:";

cin>>q[i];

}

}

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

int optimal::minvalue(int i,int j)

{

int m,k;

int min=32000;

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

{

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

{

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

k=m;

}

}

return k;

}

/\* This function builds table from all given probalities. it basically computes C,r,w value \*/

void optimal::OBST()

{

int i,j,k,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=minvalue(i,j);

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

r[i][j]=k;

}

}

}

/\* This function builds tree from table made by OBST function \*/

void optimal::buildtree()

{

int i,j,k;

cout<<"\n The optimal Binary search tree for given nodes is : \n";

cout<<"\n The root of this OBST is :"<<r[0][n];

cout<<"\n The cost of this OBST is: "<<c[0][n];

cout<<"\n\n Node \t Left child \t Right child";

cout<<"\n \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;

queue[++rear]=0;

queue[++rear]=n;

while(front!=rear)

{

i=queue[++front];

j=queue[++front];

k=r[i][j];

cout<<"\n\t"<<k;

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

{

cout<<" "<<r[i][k-1];

queue[++rear]=i;

queue[++rear]=k-1;

}

else

cout<<" ";

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

{

cout<<" "<<r[k][j];

queue[++rear]=k;

queue[++rear]=j;

}

else

cout<<" ";

}

cout<<endl;

}

/\* This is main function \*/

int main() {

optimal obj;

obj.getdata();

obj.OBST();

obj.buildtree();

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Output:

