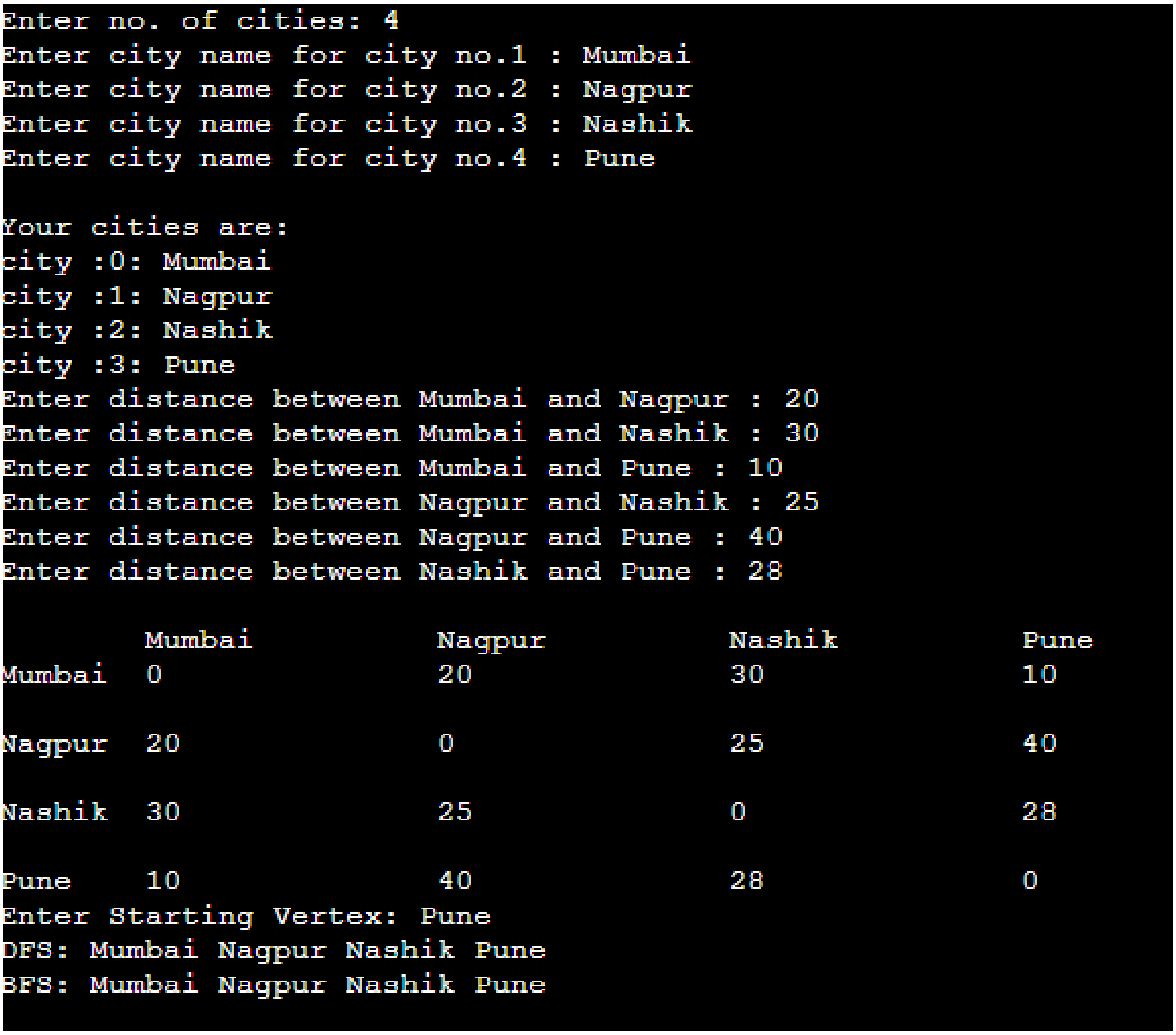
```
#include <iostream>
#include <queue>
using namespace std;

int adj_mat[50][50] = {0, 0};
int visited[50] = {0};

void dfs(int s, int n, string arr[])
{
visited[s] = 1;
cout << arr[s] << " ";
for (int i = 0; i < n; i++)
{
if (adj_mat[s][i] && !visited[i])
dfs(i, n, arr);
}
}

void bfs(int s, int n, string arr[])
{
bool visited[n];
for (int i = 0; i < n; i++)
visited[i] = false;
int v;
queue<int> bfsq;
if (!visited[s])
{
cout << arr[s] << " ";
bfsq.push(s);
visited[s] = true;
while (!bfsq.empty())
{
v = bfsq.front();
for (int i = 0; i < n; i++)
{
if (adj_mat[v][i] && !visited[i])
{
```

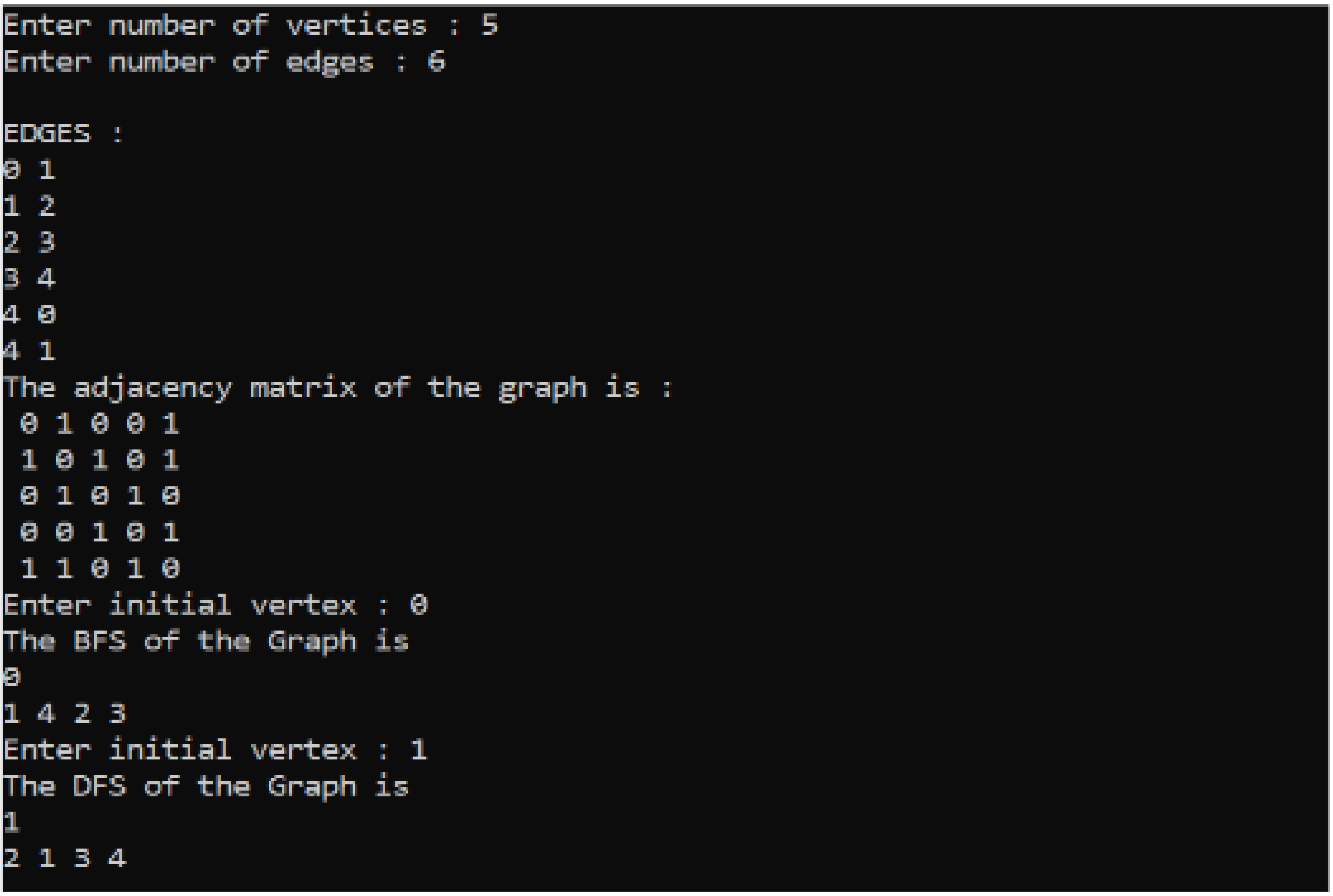
```
cout << arr[i] << " ";
visited[i] = true;
bfsq.push(i);
}
}
bfsq.pop();
}
}
}

int main()
{
cout << "Enter no. of cities: ";
int n, u;
cin >> n;
string cities[n];
for (int i = 0; i < n; i++)
{
cout << "Enter city name for city no." << i+1 <<" : ";
cin >> cities[i];
}
cout << "\nYour cities are: " << endl;
for (int i = 0; i < n; i++)
cout << "city :" << i << ": " << cities[i] << endl;
for (int i = 0; i < n; i++)
{
for (int j = i + 1; j < n; j++)
{
cout << "Enter distance between " << cities[i] << " and " << cities[j] << " : ";
cin >> adj_mat[i][j];
adj_mat[j][i] = adj_mat[i][j];
}
}
cout << endl;
for (int i = 0; i < n; i++)
cout << "\t" << cities[i] << "\t";
for (int i = 0; i < n; i++)
```

```
{

cout << "\n"

<< cities[i];

for (int j = 0; j < n; j++)

cout << "\t" << adj_mat[i][j] << "\t";

cout << endl;

}

cout << "Enter Starting Vertex: ";

cin >> u;

cout << "DFS: ";

dfs(u, n, cities);

cout << endl;

cout << "BFS: ";

bfs(u, n, cities);

return 0;

}
```



```
Enter no. of cities: 4
Enter city name for city no.1 : Mumbai
Enter city name for city no.2 : Nagpur
Enter city name for city no.3 : Nashik
Enter city name for city no.4 : Pune

Your cities are:
city :0: Mumbai
city :1: Nagpur
city :2: Nashik
city :3: Pune
Enter distance between Mumbai and Nagpur : 20
Enter distance between Mumbai and Nashik : 30
Enter distance between Mumbai and Pune : 10
Enter distance between Nagpur and Nashik : 25
Enter distance between Nagpur and Pune : 40
Enter distance between Nashik and Pune : 28

        Mumbai          Nagpur          Nashik          Pune
Mumbai  0               20              30              10

Nagpur  20              0               25              40

Nashik  30              25              0               28

Pune    10              40              28              0
Enter Starting Vertex: Pune
DFS: Mumbai Nagpur Nashik Pune
BFS: Mumbai Nagpur Nashik Pune
```

```
#include <iostream>
#include <stdlib.h>
using namespace std;

int cost[10][10], i, j, k, n, qu[10], front, rear, v, visit[10], visited[10];
int stk[10], top, visit1[10], visited1[10];

int main()
{
int m;
cout << "Enter number of vertices : ";
cin >> n;
cout << "Enter number of edges : ";
cin >> m;
cout << "\nEDGES :\n";
for (k = 1; k <= m; k++)
{
cin >> i >> j;
cost[i][j] = 1;
cost[j][i] = 1;
}
//display function
cout << "The adjacency matrix of the graph is : " << endl;
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
cout << " " << cost[i][j];
}
cout << endl;
}
cout << "Enter initial vertex : ";
cin >> v;
cout << "The BFS of the Graph is\n";
cout << v<<endl;
visited[v] = 1;
k = 1;
while (k < n)
{
for (j = 1; j <= n; j++)
if (cost[v][j] != 0 && visited[j] != 1 && visit[j] != 1)
{
visit[j] = 1;
qu[rear++] = j;
}
v = qu[front++];
cout << v << " ";
k++;
visit[v] = 0;
visited[v] = 1;
}
cout <<endl<<"Enter initial vertex : ";
cin >> v;
cout << "The DFS of the Graph is\n";
cout << v<<endl;
visited[v] = 1;
k = 1;
while (k < n)
{
for (j = n; j >= 1; j--)
if (cost[v][j] != 0 && visited1[j] != 1 && visit1[j] != 1)
{
visit1[j] = 1;
stk[top] = j;
top++;
}
v = stk[--top];
cout << v << " ";
k++;
visit1[v] = 0;
visited1[v] = 1;
}
return 0;
}
```



```
Enter number of vertices : 5
Enter number of edges : 6

EDGES :
0 1
1 2
2 3
3 4
4 0
4 1
The adjacency matrix of the graph is :
 0 1 0 0 1
 1 0 1 0 1
 0 1 0 1 0
 0 0 1 0 1
 1 1 0 1 0
Enter initial vertex : 0
The BFS of the Graph is
0
1 4 2 3
Enter initial vertex : 1
The DFS of the Graph is
1
2 1 3 4
```

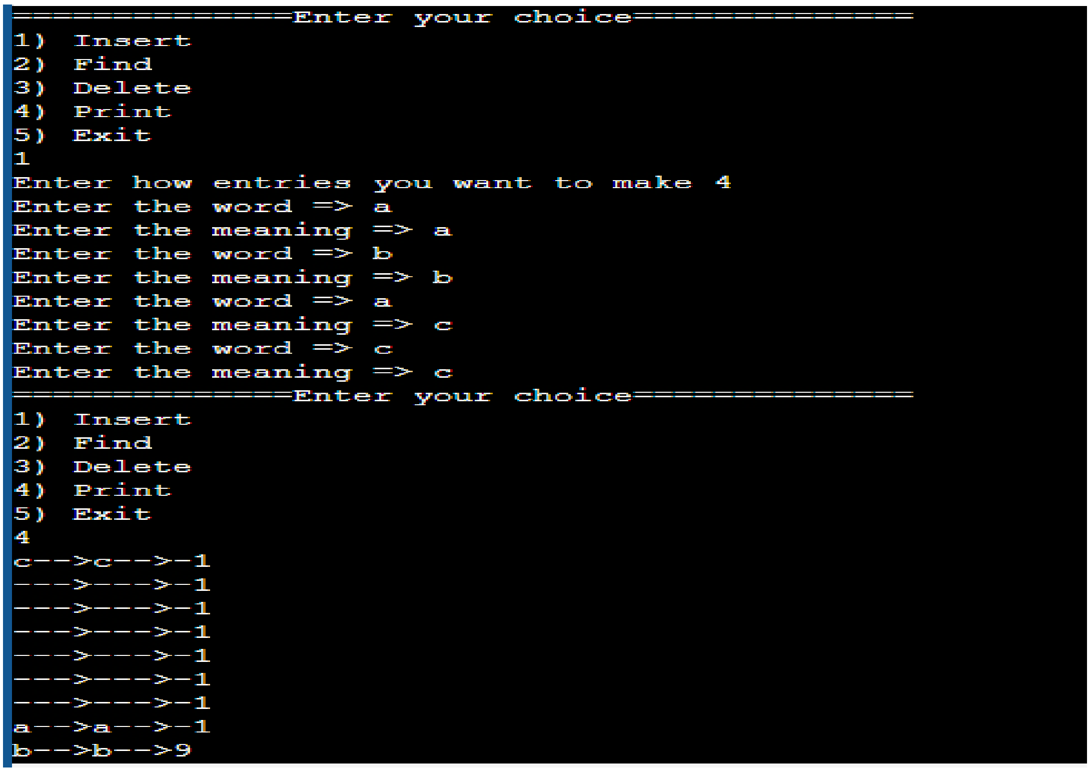
```
#include <iostream>

#include <cstring>

using namespace std;

struct hash

{

string word;

string meaning;

int chain;

} obj[10];

void hash_initialization()

{

for (int i = 0; i < 10; i++)

{

obj[i].word = "-";

obj[i].meaning = "-";

obj[i].chain = -1;

}

}

void display()

{

for (int i = 0; i < 10; i++)

{

cout << obj[i].word << "-->" << obj[i].meaning << "-->" << obj[i].chain << endl;

}

}

int calculate(string word)

{

int key = 0;

for (int i = 0; i < word.length(); i++)

{

key = key + word[i];

}

return key % 10;

}

void collision(int key, string word, string meaning)

{

int i = 1;
```

```
while (((key + i) % 10) < 10)
{
if (obj[(key + i) % 10].word == "-")
{
obj[(key + i) % 10].word = word;
obj[(key + i) % 10].meaning = meaning;
obj[(key + i - 1) % 10].chain = (key + i) % 10;
break;
}
else
{
i++;
}
}
}
void insert()
{
string wd, mg;
cout << "Enter the word => ";
cin >> wd;
cout << "Enter the meaning => ";
cin >> mg;
int hash_key = calculate(wd);
if (obj[hash_key].word == "-")
{
obj[hash_key].word = wd;
obj[hash_key].meaning = mg;
}
else
{
collision(hash_key, wd, mg);
}
}
void find(string wd)
{
int hash_key = calculate(wd);
if (obj[hash_key].word == wd)
```

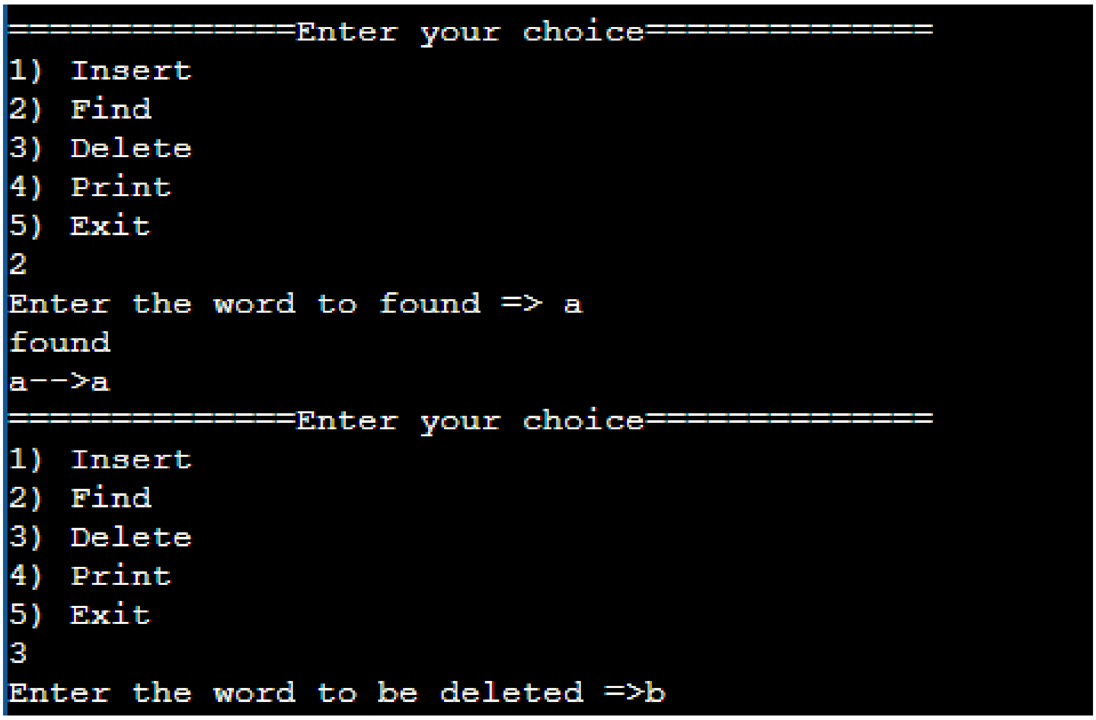
```
{
cout << "found" << endl;
cout << obj[hash_key].word << "-->" << obj[hash_key].meaning << endl;
}
else if (obj[hash_key].chain != -1)
{
int temp = obj[hash_key].chain;
while (true)
{
if (obj[temp].word == wd)
{
cout << "found" << endl;
cout << obj[temp].word << "-->" << obj[temp].meaning << endl;
break;
}
temp = obj[temp].chain;
}
}
else
{
cout << "Not Found" << endl;
}
}
void Del(string wd)
{
int hash_key = calculate(wd);
if (obj[hash_key].word == wd)
{
obj[hash_key].word = "-";
obj[hash_key].meaning = "-";
obj[hash_key].chain = -1;
}
else if (obj[hash_key].chain != -1)
{
int temp = obj[hash_key].chain;
while (true)
{
```

```
if (obj[temp].word == wd)

{

obj[temp].word = "-";

obj[temp].meaning = "-";

obj[temp].chain = -1;

break;

}

temp = obj[temp].chain;

}

}

else

{

cout << "Word Not Found" << endl;

}

}

int main()

{

int choice, n;

string wd_find, wd_Del;

hash_initialization();

do

{

cout << "==============Enter your choice==============" << endl;

cout << "1) Insert" << endl;

cout << "2) Find" << endl;

cout << "3) Delete" << endl;

cout << "4) Print" << endl;

cout << "5) Exit" << endl;

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter how entries you want to make ";

cin >> n;

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

{

insert();
```

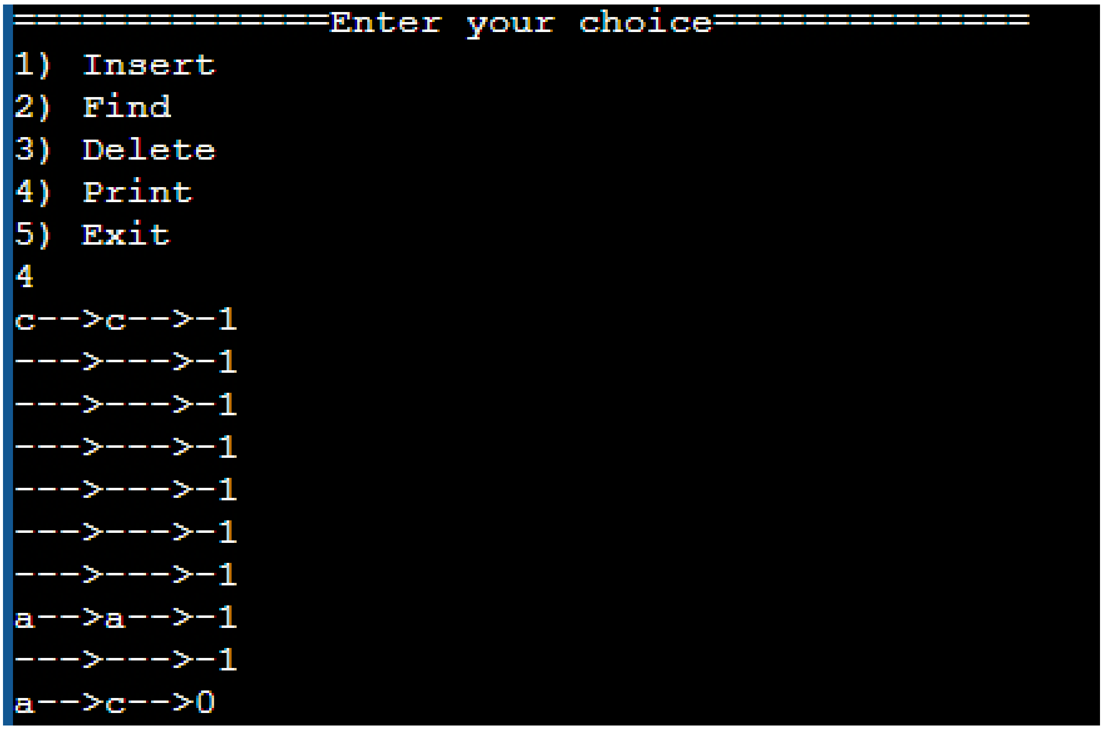
```
}
break;
case 2:
cout << "Enter the word to found => ";
cin >> wd_find;
find(wd_find);
break;
case 3:
cout << "Enter the word to be deleted =>";
cin >> wd_Del;
Del(wd_Del);
break;
case 4:
display();
break;
case 5:
break;
default:
cout << "Invalid choice" << endl;
break;
}
} while (choice < 5);
return 0;}}
```



```
==============Enter your choice==============
1)  Insert
2)  Find
3)  Delete
4)  Print
5)  Exit
1
Enter how entries you want to make 4
Enter the word => a
Enter the meaning => a
Enter the word => b
Enter the meaning => b
Enter the word => a
Enter the meaning => c
Enter the word => c
Enter the meaning => c
==============Enter your choice==============
1)  Insert
2)  Find
3)  Delete
4)  Print
5)  Exit
4
c-->c-->-1
--->--->-1
--->--->-1
--->--->-1
--->--->-1
--->--->-1
--->--->-1
a-->a-->-1
b-->b-->9
```



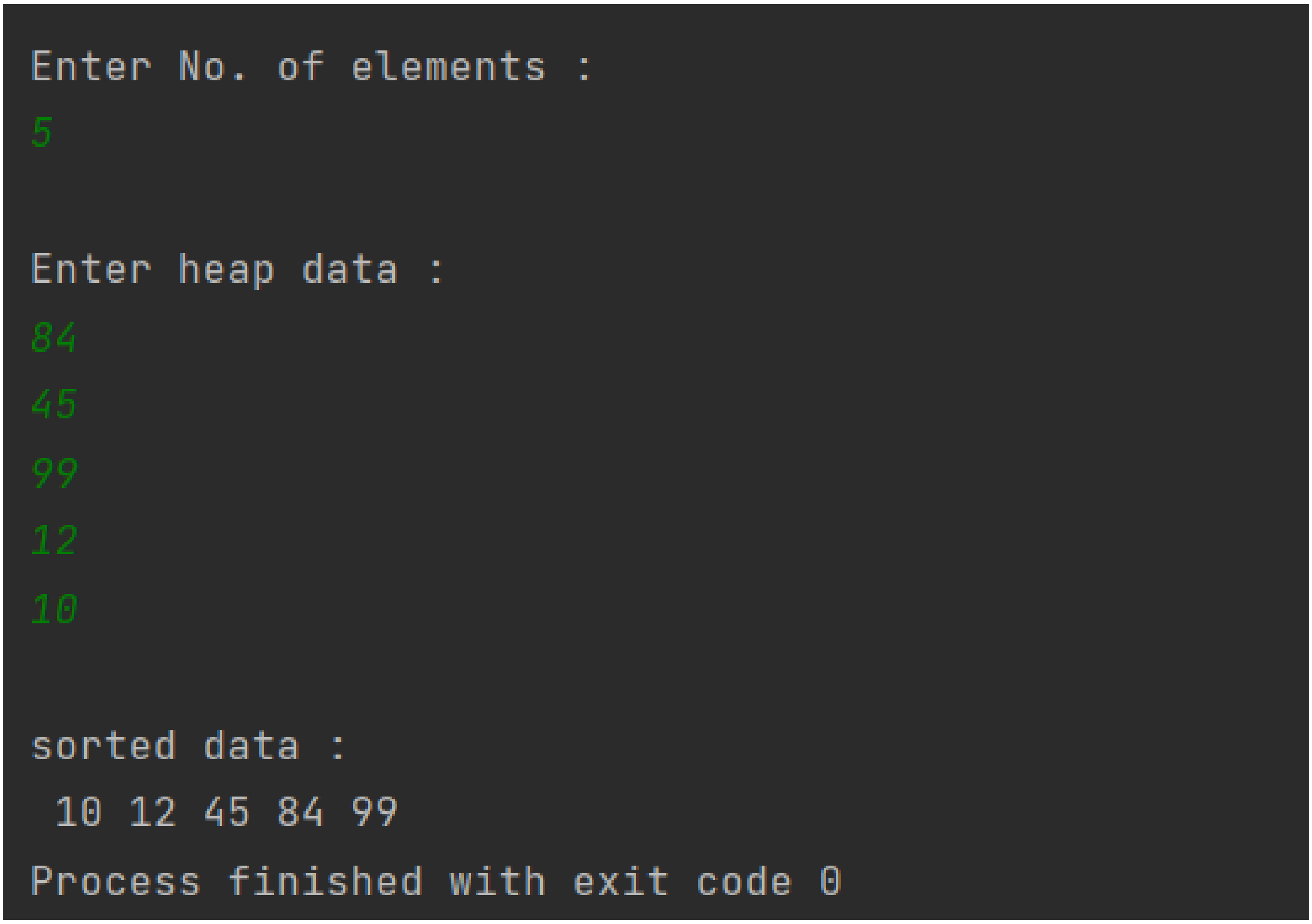
```
==============Enter your choice==============
1) Insert
2) Find
3) Delete
4) Print
5) Exit
2
Enter the word to found => a
found
a-->a
==============Enter your choice==============
1) Insert
2) Find
3) Delete
4) Print
5) Exit
3
Enter the word to be deleted =>b
```



```
==============Enter your choice==============
1) Insert
2) Find
3) Delete
4) Print
5) Exit
4
c-->c-->-1
--->--->-1
--->--->-1
--->--->-1
--->--->-1
--->--->-1
--->--->-1
a-->a-->-1
--->--->-1
a-->c-->0
```

```
import java.io.*;
import java.util.*;
public class heapsort {
    public int[] heap;
    public int count;
    public void downadjust(int i) {
        int j, temp, n;
        n = heap[0];
        if (2 * i <= n) {
            j = 2 * i;// j on left child of i
            if (j + 1 <= n && heap[j + 1] > heap[j]) // j points to larger of left and right child
                    j = j + 1;
            if (heap[i] < heap[j]) {
                    temp = heap[i];
                    heap[i] = heap[j];
                    heap[j] = temp;
                    downadjust(j);
            }
        }
     }
    public void upadjust(int i) {
        int temp;
        while (i > 1 && heap[i] > heap[i / 2]) {
            temp = heap[i];
            heap[i] = heap[i / 2];
            heap[i / 2] = temp;
            i = i / 2;
        }
     }
    public void insert(int x) {
        heap[++heap[0]] = x;
        upadjust(heap[0]);
     }
    public void create() {
        int i, x, n;
        heap = new int[30];
        heap[0] = 0;
```

```
      Scanner reader = new Scanner(System.in);
      System.out.println("\nEnter No. of elements : ");
      n = reader.nextInt();
      count = n;
      System.out.println("\nEnter heap data : ");
      for (i = 0; i < n; i++) {
        x = reader.nextInt();
        insert(x);
      }
     }
    public void sort() {
      int last, temp;
      while (heap[0] > 1) {
        last = heap[0];
        temp = heap[1];
        heap[1] = heap[last];
        heap[last] = temp;
        heap[0]--;
        downadjust(1);
      }
     }
    public void print() {
      int n, i;
       n = count;
      System.out.println("\nsorted data : ");
      for (i = 1; i <= n; i++)
        System.out.print(" " + heap[i]);
     }
    public static void main(String[] args) {
      int x;
      heapsort myobject = new heapsort();
      myobject.create();
      myobject.sort();
      myobject.print();
     }
}
```

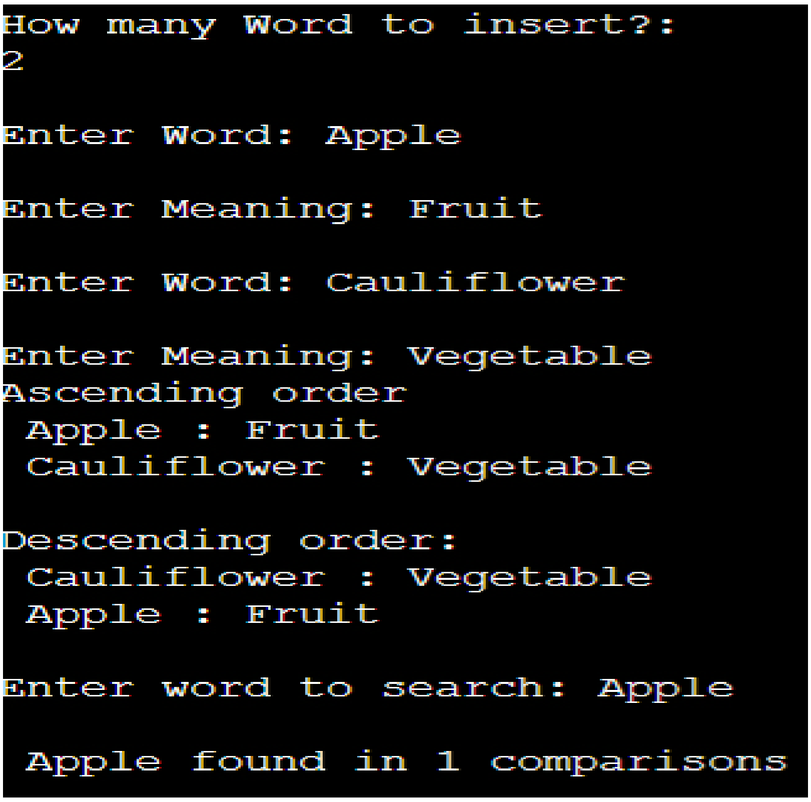


```
Enter No. of elements :
5


Enter heap data :
84
45
99
12
10


sorted data :
 10 12 45 84 99
Process finished with exit code 0
```

```
#include <iostream>
#include<string>
using namespace std;
class dictionary;
class node
{
 string word,meaning;
 node *left,*right;
public:
 friend class dictionary;
 node()
 {
  left=NULL;
  right=NULL;
 }
 node(string word, string meaning)
 {
  this->word=word;
  this->meaning=meaning;
  left=NULL;
  right=NULL;
 }
};
class dictionary
{
 node *root;
public:
 dictionary()
{
  root=NULL;
}
 void create();
 void inorder_rec(node *rnode);
 void postorder_rec(node *rnode);
 void inorder()
 {
  inorder_rec(root);
 }
 void postorder();
 bool insert(string word,string meaning);
 int search(string key);
};
int dictionary::search(string key)
{
 node *tmp=root;
 int count;
 if(tmp==NULL)
 {
  return -1;
 }
 if(root->word==key)
  return 1;
 while(tmp!=NULL)
 {
  if((tmp->word)>key)
  {
   tmp=tmp->left;
   count++;
  }
  else if((tmp->word)<key)
  {
   tmp=tmp->right;
   count++;
  }
  else if(tmp->word==key)
  {
   return ++count;
  }
 }
 return -1;
}
void dictionary::postorder()
{
 postorder_rec(root);
}
void dictionary::postorder_rec(node *rnode)
{
 if(rnode)
 {
  postorder_rec(rnode->right);
  cout<<" "<<rnode->word<<" : "<<rnode->meaning<<endl;
  postorder_rec(rnode->left);
 }
}
void dictionary::create()
{
 int n;
 string wordI,meaningI;
 cout<<"\nHow many Word to insert?:\n";
 cin>>n;
 for(int i=0;i<n;i++)
 {
  cout<<"\nENter Word: ";
  cin>>wordI;
  cout<<"\nEnter Meaning: ";
  cin>>meaningI;
  insert(wordI,meaningI);
 }
}
void dictionary::inorder_rec(node *rnode)
{
 if(rnode)
 {
  inorder_rec(rnode->left);
  cout<<" "<<rnode->word<<" : "<<rnode->meaning<<endl;
  inorder_rec(rnode->right);
 }
}
bool dictionary::insert(string word, string meaning)
{
```

```
node *p=new node(word, meaning);
if(root==NULL)
{
 root=p;
 return true;
}
node *cur=root;
node *par=root;
while(cur!=NULL) //traversal
{
 if(word>cur->word)
 {par=cur;
 cur=cur->right;
 }
 else if(word<cur->word)
 {
  par=cur;
  cur=cur->left;
 }
 else
 {
  cout<<"\nWord is already in the dictionary.";
  return false;
 }
}
if(word>par->word) //insertion of node
{
 par->right=p;
 return true;
}
else
{
 par->left=p;

 return true;
}
}
```

```
int main() {
 string word;
 dictionary months;
 months.create();
 cout<<"Ascending order\n";
 months.inorder();


 cout<<"\nDescending order:\n";
 months.postorder();


 cout<<"\nEnter word to search: ";
 cin>>word;
 int comparisons=months.search(word);
 if(comparisons==-1)
 {
  cout<<"\nNot found word";
 }
 else
 {
  cout<<"\n "<<word<<" found in "<<comparisons<<" comparisons";
 }
 return 0;}
```



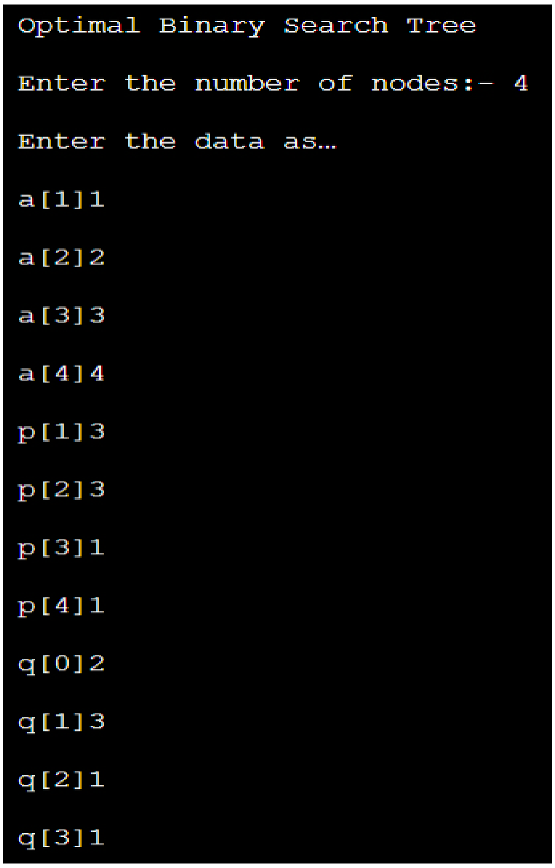
```
How many Word to insert?:
2

Enter Word: Apple

Enter Meaning: Fruit

Enter Word: Cauliflower

Enter Meaning: Vegetable
Ascending order
 Apple : Fruit
 Cauliflower : Vegetable

Descending order:
 Cauliflower : Vegetable
 Apple : Fruit

Enter word to search: Apple

 Apple found in 1 comparisons
```

```
#include <iostream>
using namespace std;
#define SIZE 10
class OBST
{
    int p[SIZE];          // Probabilities with which we search for an element
    int q[SIZE];      // Probabilities that an element is not found
    int a[SIZE];          // Elements from which OBST is to be built
    int w[SIZE][SIZE]; // Weight ‘w[i][j]’ of a tree having root
    //’r[i][j]’
    int c[SIZE][SIZE]; // Cost ‘c[i][j] of a tree having root ‘r[i][j]
    int r[SIZE][SIZE]; // represents root
    int n;                  // number of nodes
public:
    /* This function accepts the input data */
    void get_data()
     {
        int i;
        cout << "\n Optimal Binary Search Tree \n";
        cout << "\n Enter the number of nodes";
        cin >> n;
        cout << "\n Enter the data as…\n";
        for (i = 1; i <= n; i++)
        {
            cout << "\n a[" << i << "]";
            cin >> a[i];
        }
        for (i = 1; i <= n; i++)
        {
            cout << "\n p[" << i << "]";
            cin >> p[i];
        }
        for (i = 0; i <= n; i++)
        {
            cout << "\n q[" << i << "]";
            cin >> q[i];
```

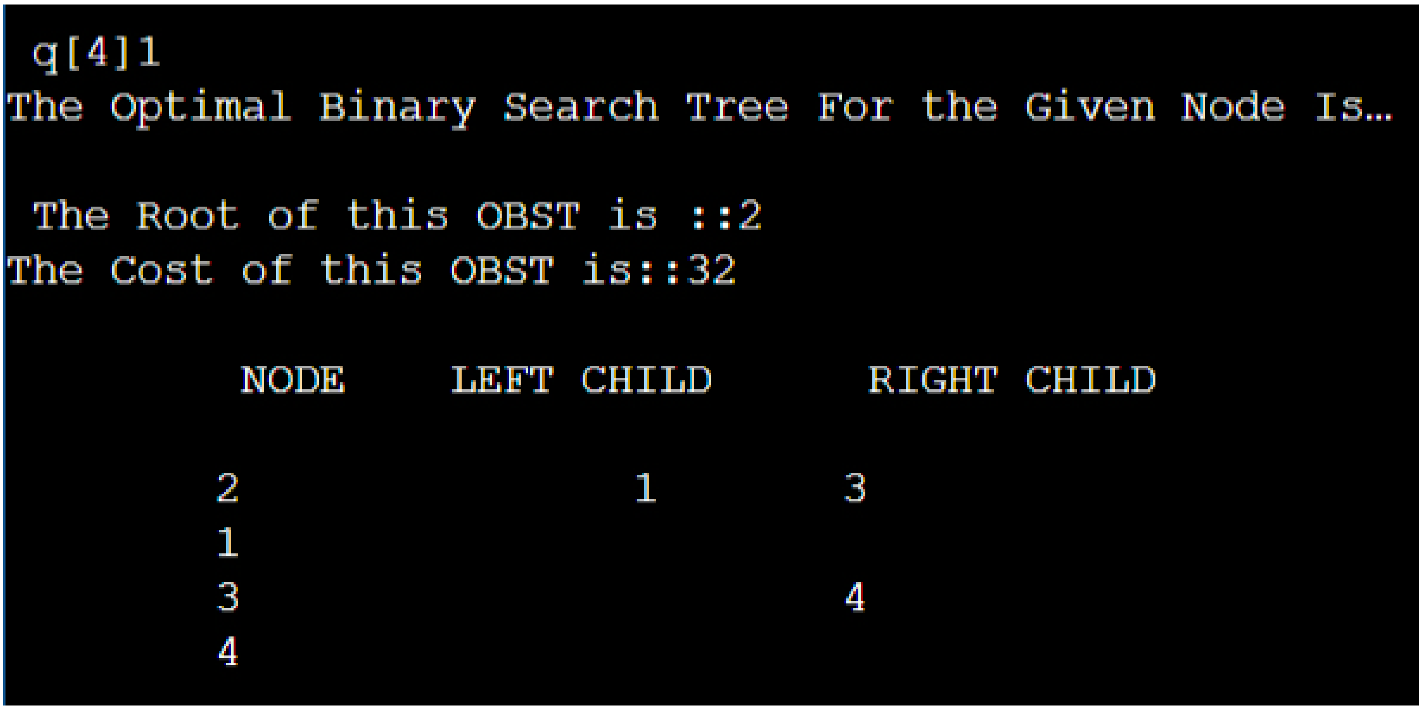
```
   }
 }
/* This function returns a value in the range ‘r[i][j-1]’ to ‘r[i+1][j]’so
that the cost ‘c[i][k-1]+c[k][j]’is minimum */
int Min_Value(int i, int j)
 {
   int m, k;
   int minimum = 32000;
   for (m = r[i][j - 1]; m <= r[i + 1][j]; m++)
   {
      if ((c[i][m - 1] + c[m][j]) < minimum)
      {
            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 */
void build_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 */
void build_tree()
 {
  int i, j, k;
  int queue[20], front = -1, rear = -1;
  cout << "The Optimal Binary Search Tree For the Given Node Is...\n";
  cout << "\n The Root of this OBST is ::" << r[0][n];
  cout << "\nThe Cost of this OBST is::" << c[0][n];
  cout << "\n\n\t NODE \t LEFT CHILD \t RIGHT CHILD ";
  cout << "\n";
  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 << "\t\t" << r[i][k - 1];
        queue[++rear] = i;
        queue[++rear] = k - 1;
    }
    else
        cout << "\t\t";
```

```
      if (r[k][j] != 0)
      {
          cout << "\t" << r[k][j];
          queue[++rear] = k;
          queue[++rear] = j;
      }
      else
          cout < "\t";
    } // end of while
    cout << "\n";
   }
}; // end of the class
/*This is the main function */
int main()
{
  OBST obj;
  obj.get_data();
  obj.build_OBST();
  obj.build_tree();
  return 0;
}
```



```
Optimal Binary Search Tree
Enter the number of nodes:- 4
Enter the data as…
a[1]1
a[2]2
a[3]3
a[4]4
p[1]3
p[2]3
p[3]1
p[4]1
q[0]2
q[1]3
q[2]1
q[3]1
```

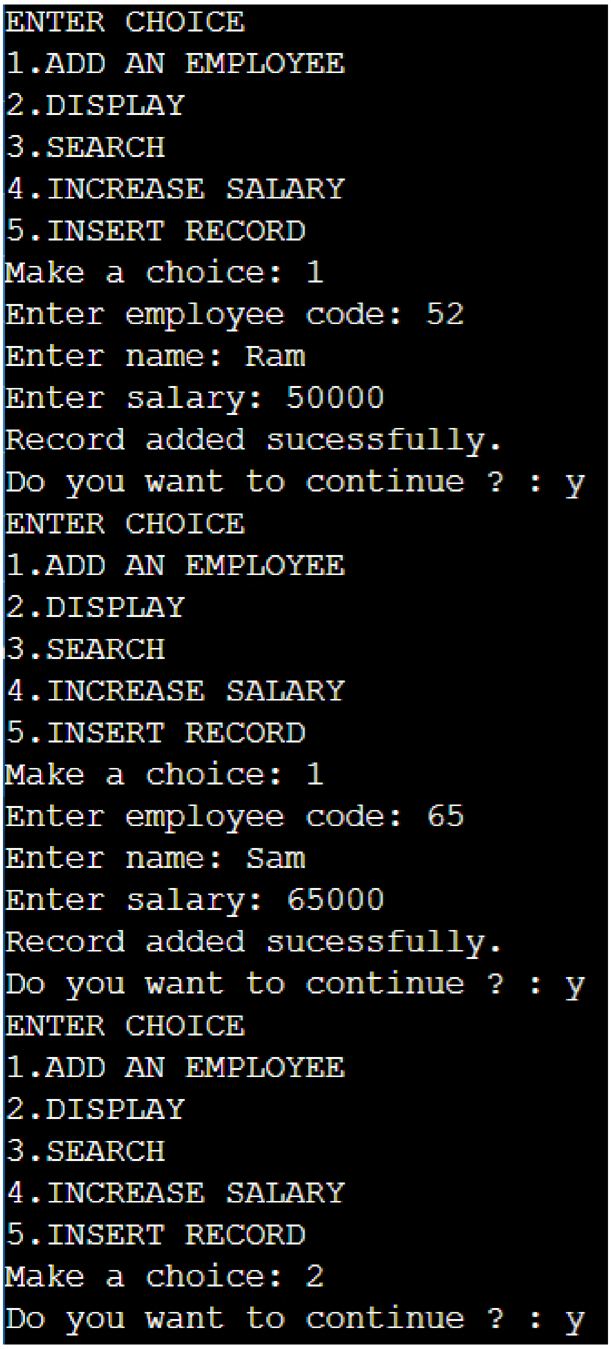


```
 q[4]1
The Optimal Binary Search Tree For the Given Node Is…

 The Root of this OBST is ::2
The Cost of this OBST is::32

         NODE     LEFT CHILD      RIGHT CHILD

        2               1       3
        1
        3                       4
        4
```

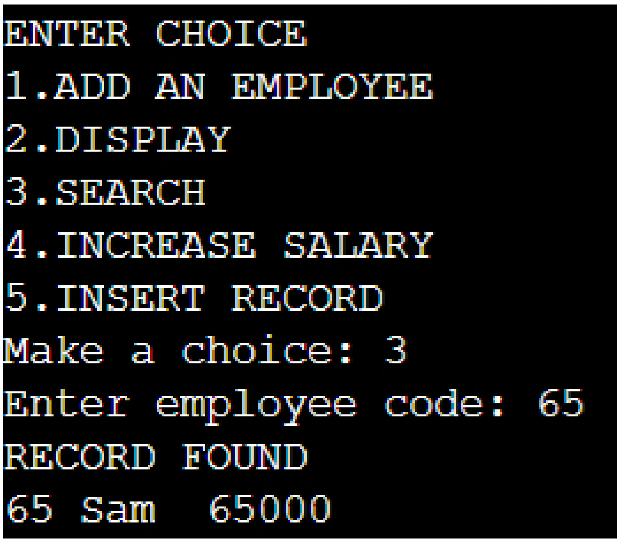
```
#include<iostream>
#include<fstream>
#include<stdio.h>
using namespace std;
class Employee{
   private:
      int code;
      char name[20];
      float salary;
   public:
      void read();
      void display();
      int getEmpCode()          { return code;}
      int getSalary()           { return salary;}
      void updateSalary(float s)  { salary=s;}
};

void Employee::read(){
   cout<<"Enter employee code: ";
   cin>>code;
   cout<<"Enter name: ";
   cin.ignore(1);
   cin.getline(name,20);
   cout<<"Enter salary: ";
   cin>>salary;
}
void Employee::display()
{
   cout<<code<<" "<<name<<"\t"<<salary<<endl;
}
fstream file;
 void deleteExistingFile(){
   remove("EMPLOYEE.DAT");
}
 void appendToFille(){
   Employee    x;
```

```
        x.read();
       file.open("EMPLOYEE.DAT",ios::binary|ios::app);
    if(!file){
        cout<<"ERROR IN CREATING FILE\n";
        return;
    }
file.write((char*)&x,sizeof(x));
    file.close();
    cout<<"Record added sucessfully.\n";
}
void displayAll(){
    Employee    x;
    file.open("EMPLOYEE.DAT",ios::binary|ios::in);
    if(!file){
        cout<<"ERROR IN OPENING FILE \n";
        return;
    }
    while(file){
    if(file.read((char*)&x,sizeof(x)))
        if(x.getSalary()>=10000 && x.getSalary()<=20000)
            x.display();
    }
  file.close();
}
void searchForRecord(){
    //read employee id
    Employee    x;
    int c;
    int isFound=0;
    cout<<"Enter employee code: ";
    cin>>c;
 file.open("EMPLOYEE.DAT",ios::binary|ios::in);
    if(!file){
        cout<<"ERROR IN OPENING FILE \n";
        return;
    }
    while(file){
        if(file.read((char*)&x,sizeof(x))){
            if(x.getEmpCode()==c){
                cout<<"RECORD FOUND\n";
                x.display();
                isFound=1;
                break;
            }
        }
    }
    if(isFound==0){
        cout<<"Record not found!!!\n";
    }
    file.close();
}
 void increaseSalary(){
    //read employee id
    Employee    x;
    int c;
    int isFound=0;
    float sal;
    cout<<"enter employee code \n";
    cin>>c;
    file.open("EMPLOYEE.DAT",ios::binary|ios::in);
    if(!file){
        cout<<"ERROR IN OPENING FILE \n";
        return;
    }
    while(file){
        if(file.read((char*)&x,sizeof(x))){
            if(x.getEmpCode()==c){
                cout<<"Salary hike? ";
                cin>>sal;
                x.updateSalary(x.getSalary()+sal);
                isFound=1;
                break;
            }
        }
    }
    if(isFound==0){
        cout<<"Record not found!!!\n";
    }
    file.close();
    cout<<"Salary updated successfully."<<endl;
}
void insertRecord(){
    //read employee record
    Employee    x;
    Employee newEmp;
    newEmp.read();
    fstream fin;
    file.open("EMPLOYEE.DAT",ios::binary|ios::in);
    //open file in write mode
    fin.open("TEMP.DAT",ios::binary|ios::out);

    if(!file){
        cout<<"Error in opening EMPLOYEE.DAT file!!!\n";
        return;
    }
    if(!fin){
        cout<<"Error in opening TEMP.DAT file!!!\n";
        return;
    }
    while(file){
        if(file.read((char*)&x,sizeof(x))){
            if(x.getEmpCode()>newEmp.getEmpCode()){
                fin.write((char*)&newEmp, sizeof(newEmp));
            }
            //no need to use else
            fin.write((char*)&x, sizeof(x));
        }
    }
    fin.close();
    file.close();
    rename("TEMP.DAT","EMPLOYEE.DAT");
```

```
  remove("TEMP.DAT");

  cout<<"Record inserted successfully."<<endl;

}

int main()

{

   char ch;

  //if required then only remove the file

   deleteExistingFile();

   do{

   int n;

   cout<<"ENTER CHOICE\n"<<"1.ADD AN EMPLOYEE\n"<<"2.DISPLAY\n"<<"3.SEARCH\n"<<"4.INCREASE
SALARY\n"<<"5.INSERT RECORD\n";

   cout<<"Make a choice: ";

   cin>>n;

   switch(n){

      case 1:

       appendToFille();

       break;

      case 2 :

       displayAll();

       break;

      case 3:

       searchForRecord();

       break;

     case 4:

       increaseSalary();

       break;

     case 5:

       insertRecord();

       break;

      default :

         cout<<"Invalid Choice\n";

   }

   cout<<"Do you want to continue ? : ";

   cin>>ch;

   }while(ch=='Y'||ch=='y');

  return 0;}
```



```
ENTER CHOICE
1.ADD AN EMPLOYEE
2.DISPLAY
3.SEARCH
4.INCREASE SALARY
5.INSERT RECORD
Make a choice: 1
Enter employee code: 52
Enter name: Ram
Enter salary: 50000
Record added sucessfully.
Do you want to continue ? : y
ENTER CHOICE
1.ADD AN EMPLOYEE
2.DISPLAY
3.SEARCH
4.INCREASE SALARY
5.INSERT RECORD
Make a choice: 1
Enter employee code: 65
Enter name: Sam
Enter salary: 65000
Record added sucessfully.
Do you want to continue ? : y
ENTER CHOICE
1.ADD AN EMPLOYEE
2.DISPLAY
3.SEARCH
4.INCREASE SALARY
5.INSERT RECORD
Make a choice: 2
Do you want to continue ? : y
```



```
ENTER CHOICE
1.ADD AN EMPLOYEE
2.DISPLAY
3.SEARCH
4.INCREASE SALARY
5.INSERT RECORD
Make a choice: 3
Enter employee code: 65
RECORD FOUND
65 Sam  65000
```