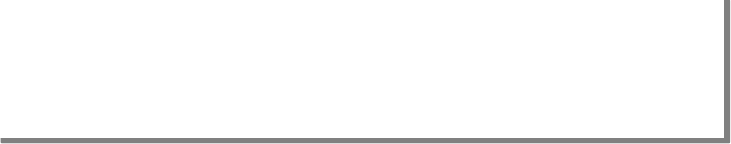
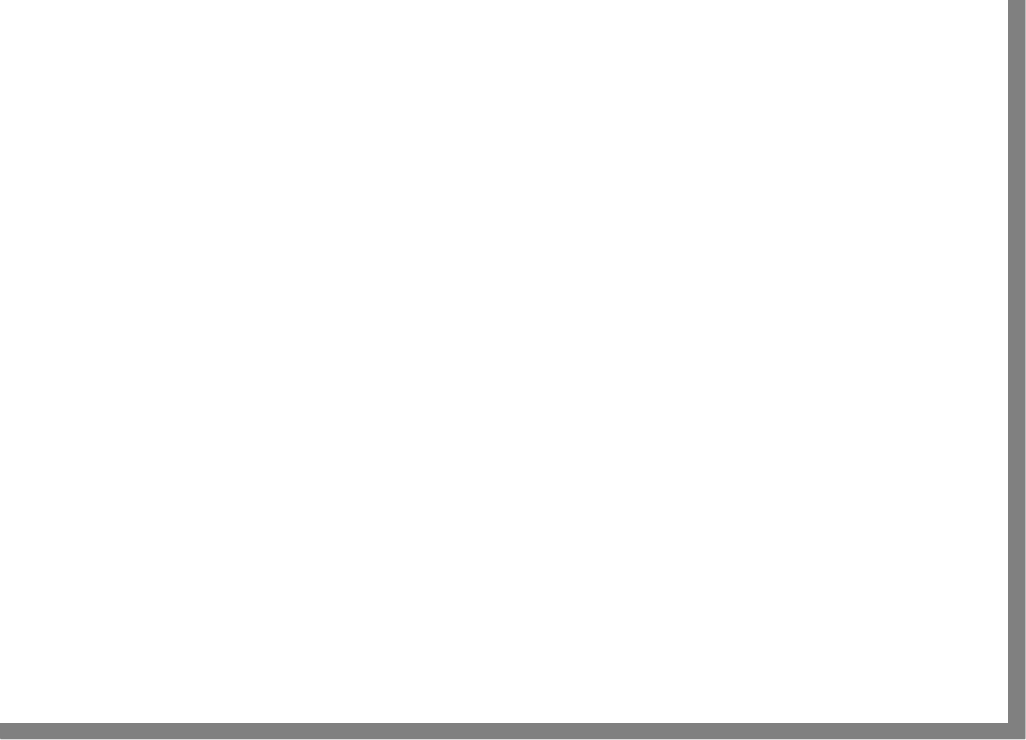
**B.Tech. CS&E Semester ( V )**

LAB RECORD



**BACHELOR OF TECHNOLOGY**

**(Academic Session – 2021 - 22)**

# Course Title

**Course Code Enrollment No. Name of Student Date of Submission Signature of Student**

# Grade/Marks Obtained

**: Analysis and Design ofAlgorithms**

# : CSE303

**: A7605220007**

# : Rakshita Rajan Singh

**Department of Computer Science & Engineering Amity School of Engineering & Technology**

**Amity University, Lucknow Campus**



# Department of Computer Science & Engineering Amity School of Engineering & Technology

**Amity University, Lucknow Campus**

# Index

|  |  |  |  |
| --- | --- | --- | --- |
| **Sno.** | **Lab Question** | **Date** | **Signature** |
| 1 | Implement recursive Binary Search and Linear Search. Determine the time required to search an element. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n. |  |  |
| 2 | Sort a given set of elements using the Quicksort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from afile or can be generated using the random  number generator. |  |  |
| 3 | Implement 0/1 Knapsack problem using Greedy Programming. |  |  |
| 4 | Implement 0/1 Knapsack problem using Dynamic Programming. |  |  |
| 5 | From a given starting node in a digraph, print all the nodes reachable byusing BFS/DFS method. |  |  |
| 6 | (a) Find Minimum Cost Spanning Tree of a given undirected graph usingKruskal'salgorithm. |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | (b) Find Minimum Cost Spanning Tree of a given undirected graph usingPrim’salgorithm. |  |  |
| 7 | From a given vertex in a weighted connected graph, find shortest pathsto other vertices using Dijkstra’s algorithm. |  |  |
| 8 | Consider the problem of N queen on an (NxN) chessboard. Two queens are said to attack each other if they are on the same row, column, or diagonal. Implements backtracking algorithm to solve the problem i.e. place N nonattacking queens on the board. |  |  |
| 9 | Implement Knapsack Problem based on Backtracking algorithm. |  |  |
| 10 | Implement Traveling Salesman problem based on Branch and Boundtechnique. |  |  |
|  | **OPEN ENDED PROGRAM** |  |  |
| 1 | Develop a Hamiltonian Path in an undirected graph is a path that visits each vertex exactly once. A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian Path such that there is an edge (in graph) from the last vertex to the first vertex of the Hamiltonian Path. Develop a program toimplement the solution of Travelling Salesman Problem by considering the Hamiltonian cycle approach. |  |  |
| 2 | Given two sequences, find the length of longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous. For example, “abc”, “abg”,“bdf”, “aeg”, ‘”acefg”, .. etc are subsequences of “abcdefg”. So a stringof length n has 2n different possible subsequences. |  |  |
| 3 | A road network can be considered as a graph with positive weights. The nodes represent road junctions and each edge of the graph is associated with a road segment between two junctions. The weight of an edge may correspond to the length of the associated road segment, the time needed to traverse the segment or the cost of traversing the segment. Using directed edges it is also possible to model one-way streets. Such graphs are special in the sense that some edges are more important than others for long distance travel (e.g. highways). This property has been formalized using the notion of highway dimension. There are a great number of algorithms that exploit this property and are therefore able to  compute the shortest path a lot quicker than would be possible on general |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | graphs. Develop a program to find the shortest path from each node tosolve the road network problem. Shortest path |  |  |
| 4 | There are a number of agents and a number of tasks. Any agent can be assigned to perform any task, incurring so me cost that may vary depending on the agenttask assignment. It is required to perform all tasksby assigning exactly one agent to each task and exactly one task to each agent in such a way that the total cost of the assignment is minimized. Ifthe numbers of agents and tasks are equal and the total cost of the assignment for all tasks is equal to the sum of the costs for each agent (or the sum of the costs for each task, which is the same thing in this case), then the problem is called the linear assignment problem. Develop a program to solve the job assignment problem. |  |  |
| 5 | Suppose that we are designing a program to simulate the storage and search in a dictionary. Words appear with different frequencies, however, and it may be the case that a frequentlyused word such as "the"appears far from the root while a rarely used word such as "conscientiousness" appears near the root. We want words that occur frequentlyin the text to be placed nearer to the root. Moreover, there may be words in the dictionary for which there is no definition. Write a program to organize an optimal binary search tree that simulates the storage and search of words in a dictionary. |  |  |

**PROGRAM NO. 01**

**Implement recursive Binary Search and Linear Search. Determine the time required to search an element. Repeat the experiment for different values of *n*, the number of elements in the list to be searched and plot a graph of the time taken versus n.**

**CODE:**

#include <iostream> using namespace std;

int binarySearch(int A[], int low, int high, int x)

{

if (low > high)return -1;

int mid = (low + high)/ 2;if (x == A[mid]) return mid;

else if (x < A[mid])

return binarySearch(A, low, mid - 1, x);

else

return binarySearch(A, mid + 1, high, x);

}

int linearSearch(int A[],int key,intsize,inti)

{

if(key==A[i]

)return i; else

return linearSearch(A,key,size,i+1);

}

int main()

{

cout<<"Enter The Size Of Array: "; int n;

cin>>n;

int arr[n], key; cout<<"enter array in sorted

order\n";for(int j=0;j<n;j++)

{

cout<<"Enter "<<j<<" Element: ";cin>>arr[j];

}

cout<<"Enter Key To Search in Array";cin>>key;

int low = 0, high = n - 1;int choice;

cout<<"enter 1 for binary and 2 for linear search\n";cin>>choice;

switch(choice)

{

case 1:

{int index=binarySearch(arr, low, high, key);if (index != -1)

cout<<"Element found at index "<<index;else

cout<<"Element not found in the array";break;

}

case 2:

{

int index=linearSearch(arr,key,n,0); if(index != n)

cout<<"KEY FOUND at index : "<<index;else

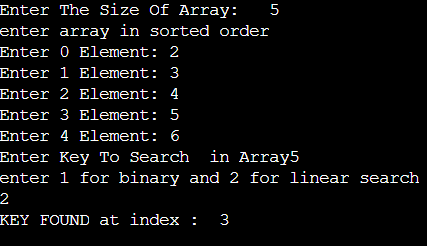
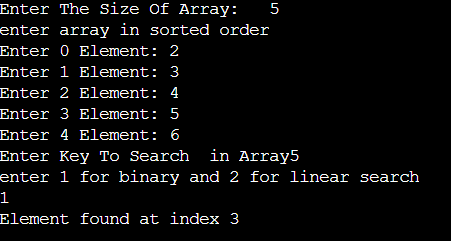
cout<<"KEY NOT FOUND in Array ";break;

}}

return 0;

}

**OUTPUT:**



# PROGRAM NO. 02

**Sort a given set of elements using the Quicksort method and determine the time required to sortthe elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from a file orcan be generated using the random number generator.**

**CODE:**

#include<iostream

>#include<cstdlib>

using namespace std; void swap(int \*a, int \*b)

{int temp; temp = \*a;

\*a = \*b;

\*b = temp;

}

int Partition(int a[], int l, int h)

{int pivot, index, i; index =

l;pivot = h;

for(i = l; i< h; i++)

{if(a[i] < a[pivot]) { swap(&a[i], &a[index]);index++;

}

}

swap(&a[pivot], &a[index]);return index;

}

int RandomPivotPartition(int a[], int l, int h)

{int pvt, n, temp; n = rand();

pvt = l + n%(h-l+1); swap(&a[h], &a[pvt]);return Partition(a, l, h);

}

int QuickSort(int a[], int l, int h)

{int pindex; if(l < h) {

pindex = RandomPivotPartition(a, l, h);QuickSort(a, l, pindex-1);

QuickSort(a, pindex+1, h);

}

return 0;

}

int main()

{int n, i;

cout<<"\nEnter the number of data element to be sorted: ";cin>>n;

int arr[n];

for(i = 0; i< n; i++) { cout<<"Enter element "<<i+1<<": ";cin>>arr[i];

}

QuickSort(arr, 0, n-1); cout<<"\nSorted Data ";

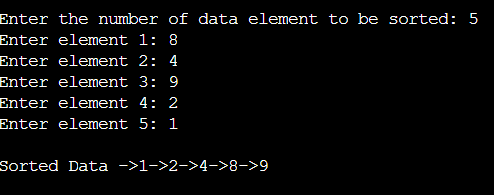
for (i = 0; i< n; i+

+)cout<<"-

>"<<arr[i]; return 0;

}

**OUTPUT:**



# PROGRAM NO. 03

**Implement 0/1 Knapsack problem using Greedy Programming. CODE:**

#include<stdio.h

>int main()

{

float weight[50],profit[50],ratio[50],Totalvalue,temp,capacity,amount; int n,i,j;

printf("Enter the number of items :");scanf("%d",&n); for (i = 0; i < n; i++)

{

printf("Enter Weight and Profit for item[%d] :

\n",i);scanf("%f %f", &weight[i], &profit[i]);

}

printf("Enter the capacity of knapsack :

\n");scanf("%f",&capacity);

for(i=0;i<n;i++) ratio[i]=profit[i]/ weight[i];

for (i = 0; i < n; i++) for (j = i + 1; j < n; j+

+)if (ratio[i] < ratio[j])

{

temp = ratio[j]; ratio[j] = ratio[i];ratio[i] = temp;

temp = weight[j]; weight[j] = weight[i];weight[i] = temp;

temp = profit[j]; profit[j] = profit[i];

profit[i] = temp;

}

printf("Knapsack problems using Greedy Algorithm:

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

{

if (weight[i] >

capacity)break; else

{

Totalvalue = Totalvalue + profit[i]; capacity = capacity - weight[i];

}

}

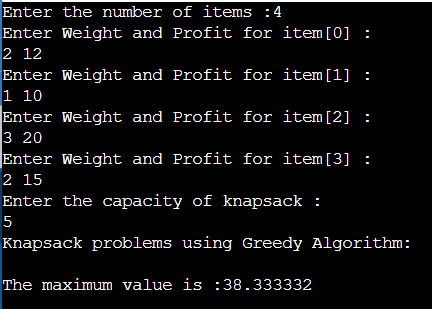
if (i < n)

Totalvalue = Totalvalue + (ratio[i]\*capacity); printf("\nThe maximum value is :

%f\n",Totalvalue);return 0;

}

**OUTPUT:**



# PROGRAM NO. 04

**Implement 0/1 Knapsack problem using Dynamic Programming. CODE:**

#include<stdio.h>

int max(int a, int b) {

if(a>b){

return a;

} else {

return b;

}

}

int knapsack(int W, int wt[], int val[], int n) {

int i, w;

int knap[n+1][W+1];

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

for (w = 0; w <= W; w++) {

if (i==0 || w==0)

knap[i][w] = 0;

else if (wt[i-1] <= w)

knap[i][w] = max(val[i-1] + knap[i-1][w-wt[i-1]], knap[i-1][w]);

else

knap[i][w] = knap[i-1][w];

}

}

return knap[n][W];

}

int main() {

int val[] = {20, 25, 40};

int wt[] = {25, 20, 30};

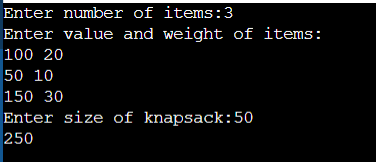
int W = 50;

int n = sizeof(val)/sizeof(val[0]);

printf("The solution is : %d", knapsack(W, wt, val, n));

return 0;

}**OUTPUT:**



**PROGRAM NO. 05**

**From a given starting node in a digraph, print all the nodes reachable by using BFS/DFS method.**

**CODE:**

#include<stdio.h> #include<stdlib.h>

int a[50][50], n, visited[50]; intq[20], front = -1,rear = -1; int s[20], top = -1, count=0;

void bfs(int v)

{

int i, cur; visited[v] = 1; q[++rear] = v; while(front!=rear)

{

cur = q[+

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

{

if((a[cur][i]==1)&&(visited[i]==0))

{

q[++rear] = i;

visited[i] = 1; printf("%d ", i);

}

}

}

}

void dfs(int v)

{

int i; visited[v]=1

;s[++top] = v;

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

{

if(a[v][i] == 1&& visited[i] == 0 )

{

printf("%d ",

i);dfs(i);

}

}

}

int main()

{

int ch, start, i,j;

printf("\nEnter the number of vertices in graph: ");scanf("%d",&n);

printf("\nEnter the adjacency matrix:

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

{

for(j=1;j<=n;j+

+)scanf("%d",&a[i][j]);

}

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

+)visited[i]=0;

printf("\nEnter the starting vertex: ");scanf("%d",&start);

printf("\n==>1. BFS: Print all nodes reachable from a given starting node");printf("\n==>2. DFS: Print all nodes reachable from a given starting node"); printf("\n==>3:Exit"); printf("\nEnter your choice: "); scanf("%d", &ch);

switch(ch)

{

case 1: printf("\nNodes reachable from starting vertex %d are: ", start);bfs(start);

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

{

break;

if(visited[i]==0)

printf("\nThe vertex that is not reachable is %d" ,i);

}

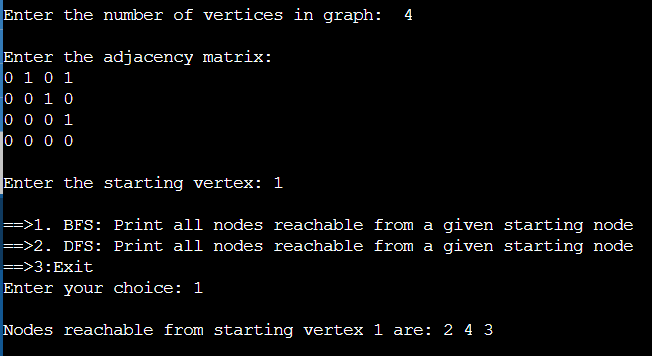
case 2: printf("\nNodes reachable from starting vertex %d are:

\n",start);dfs(start); break; case 3: exit(0); default: printf("\nPlease enter valid choice:");

}

}

# OUTPUT:



**PROGRAM NO. 06**

**Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.CODE:**

// Kruskal's algorithm in C #include <stdio.h> #define MAX 30

typedef struct edge {

## int u, v, w;

} edge;

## typedef struct edge\_list {

edge data[MAX];

## int n;

} edge\_list; edge\_list elist;

## int Graph[MAX][MAX], n; edge\_list spanlist;

void kruskalAlgo();

## int find(int belongs[], int vertexno);

void applyUnion(int belongs[], int c1, int c2); void sort();

## void print();

// Applying Krushkal Algo void kruskalAlgo() {

## int belongs[MAX], i, j, cno1, cno2;

elist.n = 0;

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

for (j = 0; j < i; j++) {

## if (Graph[i][j] != 0) {

elist.data[elist.n].u = i; elist.data[elist.n].v = j;

elist.data[elist.n].w = Graph[i][j];

## elist.n++;

}

## }

sort();

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

belongs[i] = i;

## spanlist.n = 0;

for (i = 0; i < elist.n; i++) {

cno1 = find(belongs, elist.data[i].u); cno2 = find(belongs, elist.data[i].v);

## if (cno1 != cno2) {

spanlist.data[spanlist.n] = elist.data[i];

## spanlist.n = spanlist.n + 1;

applyUnion(belongs, cno1, cno2);

## }

}

## }

int find(int belongs[], int vertexno) {

return (belongs[vertexno]);

## }

void applyUnion(int belongs[], int c1, int c2) {

## int i;

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

## if (belongs[i] == c2)

belongs[i] = c1;

}

## // Sorting algo void sort() {

int i, j;

## edge temp;

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

## for (j = 0; j < elist.n - 1; j++)

if (elist.data[j].w > elist.data[j + 1].w) {

## temp = elist.data[j];

elist.data[j] = elist.data[j + 1];

## elist.data[j + 1] = temp;

}

}

## // Printing the result void print() {

int i, cost = 0;

## for (i = 0; i < spanlist.n; i++) {

printf("\n%d - %d : %d", spanlist.data[i].u, spanlist.data[i].v, spanlist.data[i].w);

## cost = cost + spanlist.data[i].w;

}

printf("\nSpanning tree cost: %d", cost);

## }

int main() {

## int i, j, total\_cost;

n = 6;

Graph[0][0] = 0;

Graph[0][1] = 4;

Graph[0][2] = 4;

Graph[0][3] = 0;

Graph[0][4] = 0;

Graph[0][5] = 0;

Graph[0][6] = 0;

Graph[1][0] = 4;

Graph[1][1] = 0;

Graph[1][2] = 2;

Graph[1][3] = 0;

Graph[1][4] = 0;

Graph[1][5] = 0;

Graph[1][6] = 0;

Graph[2][0] = 4;

Graph[2][1] = 2;

Graph[2][2] = 0;

Graph[2][3] = 3;

Graph[2][4] = 4;

Graph[2][5] = 0;

## Graph[2][6] = 0;

Graph[3][0] = 0;

Graph[3][1] = 0;

Graph[3][2] = 3;

Graph[3][3] = 0;

Graph[3][4] = 3;

Graph[3][5] = 0;

Graph[3][6] = 0;

Graph[4][0] = 0;

Graph[4][1] = 0;

Graph[4][2] = 4;

Graph[4][3] = 3;

Graph[4][4] = 0;

Graph[4][5] = 0;

Graph[4][6] = 0;

Graph[5][0] = 0;

Graph[5][1] = 0;

Graph[5][2] = 2;

Graph[5][3] = 0;

Graph[5][4] = 3;

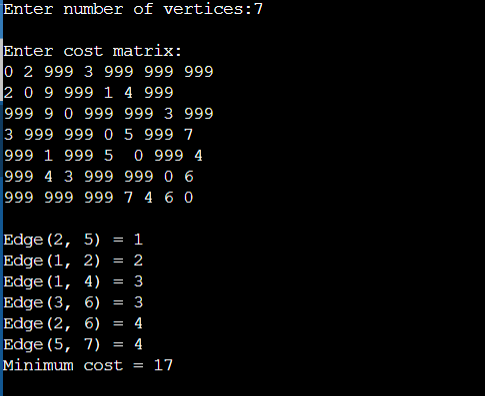
Graph[5][5] = 0;

Graph[5][6] = 0;

kruskalAlgo();

## print();

}**OUTPUT:**



**(b) Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm.**

#include<stdio.h> #include<conio.h>

void main()

{ int n, v, u,cost[10][10], visited[10]={0}, i, j; int count=1, mincost=0, min, a, b;

printf("Enter number of vertices:"); scanf("%d",&n); printf("\nEnter cost matrix (For infinity, put 999):\n");

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

{

scanf("%d",&cost[i][j]); if(cost[i][j]==0) cost[i][j]=999;

}

visited[1]=1;

printf("\nThe edges of spanning tree are: \n");

while(count<n)

{

min=999; for(i=1;i<=n;i+

+) for(j=1;j<=n;j++) if(cost[i][j]<min) if(visited[i]==0) continue; else

{

min=cost[i] [j];a=u=i; b=v=j;

}

if(visited[u]==0 || visited[v]==0)

{ count++;

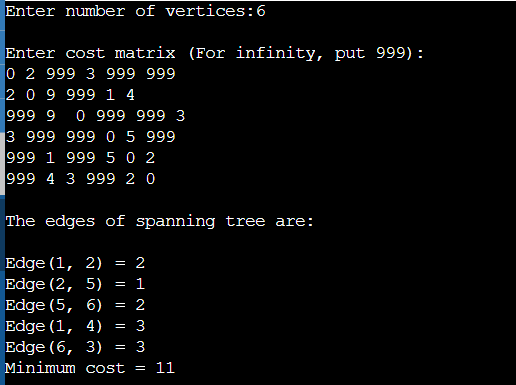
printf("\nEdge(%d, %d) = %d", a, b, min); mincost+=min; visited[b]=1;

}

cost[a][b]=cost[b][a]=999; }

printf("\nMinimum cost = %d", mincost); }

**OUTPUT:**



# PROGRAM NO. 07

**From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm.**

**CODE:**

#include<stdio.h> #include<conio.h> #define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode); int main()

{

int G[MAX][MAX],i,j,n,u; printf("Enter no. of vertices:");scanf("%d",&n); printf("\nEnter the adjacency matrix:

\n");for(i=0;i<n;i++) for(j=0;j<n;j++) scanf("%d",&G[i] [j]); printf("\nEnter the starting node:");scanf("%d",&u); dijkstra(G,n,u)

;return 0;

}

void dijkstra(int G[MAX][MAX],int n,int startnode)

{

int cost[MAX] [MAX],distance[MAX],pred[MAX];int visited[MAX],count,mindistance,nextnode,i,j;

*//pred[] stores the predecessor of each node*

*//count gives the number of nodes seen so far*

*//create the cost matrix*for(i=0;i<n;i++) for(j=0;j<n;j++) if(G[i][j]==0) cost[i] [j]=INFINITY; else

cost[i][j]=G[i][j];

*//initialize pred[],distance[] and visited[]*

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

{

distance[i]=cost[startnode][i];

pred[i]=startnode

;visited[i]=0;

}

distance[startnode]=0

;visited[startnode]=1; count=1; while(count<n-1)

{

mindistance=INFINITY;

*//nextnode gives the node at minimum distance*for(i=0;i<n;i++) if(distance[i]<mindistance&&!visited[i])

{

mindistance=distance[i]; nextnode=i;

}

*//check if a better path exists through nextnode*

visited[nextnode]=1; for(i=0;i<n;i++) if(! visited[i])

if(mindistance+cost[nextnode][i]<distance[i])

{

distance[i]=mindistance+cost[nextnode][i]; pred[i]=nextnode;

}

count++;

}

*//print the path and distance of each node*

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

+)if(i!

=startnode)

{

printf("\nDistance of node%d=%d",i,distance[i]); printf("\nPath=%d",i);

j=i; do

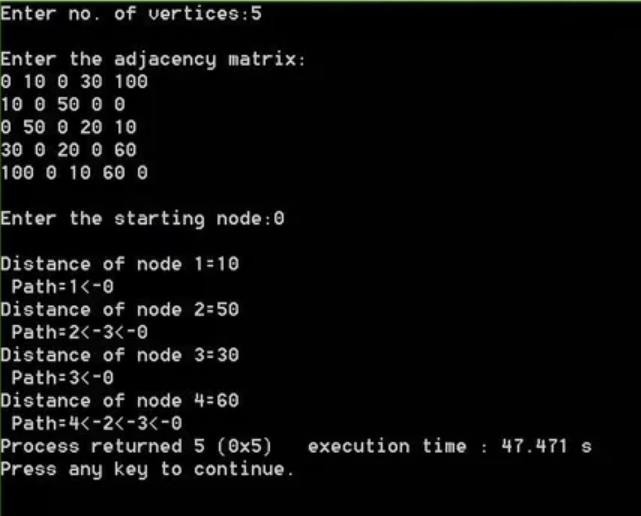
{

j=pred[j]; printf("<-%d",j);

}while(j!=startnode);

}}

**OUTPUT:**



# PROGRAM NO. 08

**Consider the problem of N queen on an (NxN) chessboard. Two queens are said to attack each other if they are on the same row, column, or diagonal. Implements backtracking algorithm to solve the problem i.e. place N nonattacking queens on the board.**

**CODE:**

#include<stdio.h> #include<math.h

>

#include<stdlib.h>

int place(int m[50], int k)

{

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

if((m[i]==m[k])||(abs(m[i]-m[k])==abs(i-k))) return 0;

return 1;

}

void display(int m[50],int n)

{

int i, j;

int s[10][10]={0};

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

+)s[i][m[i]]=1;

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

{

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

+)if(s[i][j]) printf("Q\t")

;else printf("x\t");

printf("\n");

}

exit(1);

}

int main()

{

int m[50], s[50][50], n, k; printf("Enter number of Queens:");scanf("%d",&n);

printf("\nThe solution for the problem is:

\n");n--;

for(m[0]=0, k=0;k>=0; m[k]+=1)

{

while((m[k]<=n) &&(!

place(m,k)))m[k]+=1;

if(m[k]<=n)

if(k==n) display(m,n+1)

;else

{ k+

+;

m[k]=-1;

}

els e

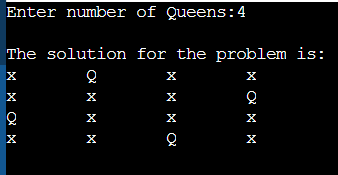
k--;

}

return 0;

}

**OUTPUT:**



# PROGRAM NO. 09

**Implement Knapsack Problem based on Backtracking algorithm CODE:**

/\* knapsack problem using backtracking\*/

/\* Variable Description….

n – Total no. of items available

w[] – Weight of each item

p[] – Profit of each item

m – Maximum Capacity of the Sack

unit[] – Profit of each item per Unit p[]/w[]

x[] – Final list of items put into the Sack

y[] – Intermediate list of selected items

fp – Final Profit fw – Final Weight cp – Current Profit

cw – Current Weight

\*/

#include <stdio.h> #include <conio.h> #define max 10

int w[max],i,j,p[max];

int n,m; Advertisements

REPORT THIS AD

float unit[max]; int

y[max],x[max],fp=

-1,fw; void get()

{

printf(“\n Enter total number of items: “);

scanf(“%d”,&n);

printf(“\n Enter the Maximum capacity of the Sack: “);

scanf(“%d”,&m); for(i=0;i<n;i++)

{

printf(“\n Enter the weight of the item # %d : “,i+1);

scanf(“%d”,&w[i])

;

printf(“\n Enter the profit of the item #

%d : “, i+1);

scanf(“%d”,

&p[i]);

}

}

void show()

{

float s=0.0;

printf(“\n\tItem\tW eight\tCost\tUnit Profit\tSelected “);

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

printf(“\n\t%d\t%d\ t%d\t%f\t%d”,i+1, w[i],p[i],unit[i],x[i]

);

printf(“\n\n The Sack now holds following items : “);

for(i=0;i<n;i++) if(x[i]==1)

{

printf(“%d\t”,i+1);

s += (float) p[i] \* (float) x[i];

}

printf(“\n

Maximum Profit:

%f “,s);

}

/\*Arrange the item based on high profit per Unit\*/

void sort()

{

int t,t1; float t2;

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

unit[i] = (float) p[i]

/ (float) w[i]; for(i=0;i<n-1;i++)

{

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

{

if(unit[i] < unit[j])

{

t2 = unit[i]; unit[i] = unit[j]; unit[j] = t2;

t = p[i]; p[i] = p[j]; p[j] = t;

t1 = w[i];

w[i] = w[j];

w[j] =t1;

}

}

}

}

float bound(float cp,float cw,int k)

{

float b = cp; float c = cw;

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

{

c = c+w[i]; if( c < m)

b = b +p[i]; else

return (b+(1-(c-m)/ (float)w[i])\*p[i]);

}

return b;

}

void knapsack(int k,float cp,float cw)

{

if(cw+w[k] <= m)

{

y[k] = 1;

if(k <= n)

knapsack(k+1,cp+ p[k],cw+w[k]);

if(((cp+p[k]) > fp) && ( k == n))

{

fp = cp+p[k]; fw = cw+w[k];

for(j=0;j<=k;j++) x[j] = y[j];

}

}

if(bound(cp,cw,k)

>= fp)

{

y[k] = 0;

if( k <= n)

knapsack(k+1,cp,c w);

if((cp > fp) && (k

== n))

{

fp = cp; fw = cw;

for(j=0;j<=k;j++)

x[j] = y[j];

}

}

}

void main()

{

clrscr(); printf(“\n\n\n\t\t

\*\*\*\*\*\*\*\* KNAPSACK PROBLEM

\*\*\*\*\*\*\*\*”);

printf(“\n\t\t

———————

——————–“);

get();

printf(“\n The Sack is arranged in the order…\n”);

sort(); knapsack(0,0.0,0.0

);

show();

getch();

}

Output:

\*\*\*\*\*\*\*\* KNAPSACK PROBLEM

\*\*\*\*\*\*\*\*

———————

———————-

Enter total number of items: 3

Enter the Maximum capacity of the Sack: 25

Enter the weight of the item # 1 : 1

Enter the profit of the item # 1 : 11

Enter the weight of the item # 2 : 11

Enter the profit of the item # 2 : 21

Enter the weight of the item # 3 : 21

Enter the profit of the item # 3 : 31

The Sack is arranged in the order…

Item Weight Cost Unit Profit Selected

1 1 11

11.000000 1

2 11 21

1.909091 0

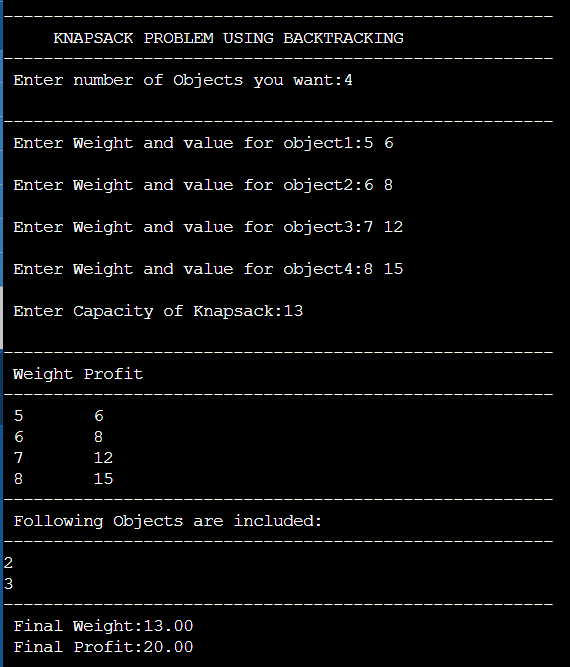
3 21 31

1.476190 1

The Sack now holds following items : 1 3

Maximum Profit: 42.000000

**OUTPUT:**



# PROGRAM NO. 10

**Implement Traveling Salesman problem based on Branch and Bound technique.**

**CODE:**

#include <stdio.h> #include <iostream> #include <vector> #include <queue> #include <utility> #include <cstring> #include <climits> using namespace std;

// number of total nodes#define N 5 #define INF INT\_MAXclass Node

{

public:

v e c t o r < p a i r < i n t , i n t > > p a t h ; i n t matrix\_reduced[N][N]; int cost;

int vertex;int level;

};

Node\* newNode(int matrix\_parent[N][N], vector<pair<int, int>>const&path,int level, int i, int j)

{

Node\* node = new Node;node->path = path; if (level != 0)

node->path.push\_back(make\_pair(i, j)); memcpy(node->matrix\_reduced, matrix\_parent,sizeof node->matrix\_reduced); for (int k = 0; level != 0 && k < N; k++)

{

node->matrix\_reduced[i][k] = INF;node->matrix\_reduced[k][j] = INF;

}

node->matrix\_reduced[j][0] = INF;node->level = level; node->vertex =

j;return node;

}

void reduce\_row(int matrix\_reduced[N][N], int row[N])

{

fill\_n(row, N, INF); for (int i = 0; i< N; i+

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

+)

if (matrix\_reduced[i][j] < row[i])

row[i] = matrix\_reduced[i] [j];for (int i = 0; i< N; i++) for (int j = 0; j < N; j++)

if (matrix\_reduced[i][j] != INF && row[i] != INF)matrix\_reduced[i][j] -= row[i];

}

void reduce\_column(int matrix\_reduced[N][N], int col[N])

{

fill\_n(col, N, INF);

for (int i = 0; i< N; i++) for (int j = 0; j < N; j++) if (matrix\_reduced[i][j] < col[j])col[j] =

matrix\_reduced[i][j]; for (int i = 0; i< N; i++)

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

if (matrix\_reduced[i][j] != INF && col[j] != INF)matrix\_reduced[i][j] -= col[j];

}

int cost\_calculation(int matrix\_reduced[N][N])

{

int cost = 0;int row[N];

reduce\_row(matrix\_reduced, row); int col[N]; reduce\_column(matrix\_reduced, col);for (int i = 0; i< N; i++)

cost += (row[i] != INT\_MAX) ? row[i] : 0,

cost += (col[i] != INT\_MAX) ? col[i] :

0;return cost;

}

void printPath(vector<pair<int, int>>const&list)

{

for (int i = 0; i<list.size(); i+

+)cout<< list[i].first + 1 << "

-> "

<< list[i].second + 1 <<endl;

}

class comp

{public:

bool operator()(const Node\* lhs, const Node\* rhs) const

{

return lhs->cost >rhs->cost; }

};

int solve(int adjacensyMatrix[N][N])

{

priority\_queue<Node\*, std::vector<Node\*>, comp>pq;vector<pair<int, int>>v;

Node\* root = newNode(adjacensyMatrix, v, 0, -1, 0);root->cost = cost\_calculation(root-

>matrix\_reduced);

pq.push(root); while (!pq.empty())

{

Node\* min = pq.top();pq.pop(); int i = min->vertex;

if (min->level == N - 1)

{

min->path.push\_back(make\_pair(i, 0));printPath(min->path);

return min->cost;

}

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

{

if (min->matrix\_reduced[i][j] != INF)

{

Node\* child = newNode(min->matrix\_reduced, min-

>path,min->level + 1, i, j);

child->cost = min->cost + min->matrix\_reduced[i][j]

+ cost\_calculation(child-

>matrix\_reduced);pq.push(child);

}

}

delete min;

}

}

int main()

{

int adjacensyMatrix[N][N] =

{

{ INF, 20, 30, 10, 11 },

{ 15, INF, 16, 4, 2 },

{ 3, 5, INF, 2, 4 },

{ 19, 6, 18, INF, 3 },

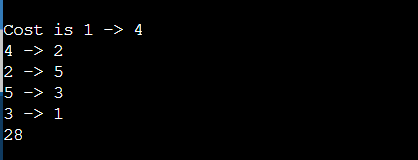
{ 16, 4, 7, 16, INF }

};

cout<< "\nCost is " << solve(adjacensyMatrix);return 0;

}

**OUTPUT:**



# OPEN ENDED PROBLEMS

**Q1) Develop a Hamiltonian Path in an undirected graph is a path that visits each vertex exactlyonce. A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian Path such that there is an edge (in graph) from the last vertex to the first vertex of the Hamiltonian Path. Develop a program to implement the solution of Travelling Salesman Problem by considering the Hamiltonian cycle approach.**

**CODE:**

#include <bits/stdc+

+.h>using namespace std; #define V 4

// implementation of traveling Salesman Problem int travllingSalesmanProblem(int graph[][V], int s)

{

// store all vertex apart from source vertexvector<int> vertex;

for (int i = 0; i < V; i++) if (i != s)

vertex.push\_back(i);

// store minimum weight Hamiltonian Cycle. int min\_path = INT\_MAX;

do {

// store current Path weight(cost)int current\_pathweight = 0;

// compute current path weightint k = s;

for (int i = 0; i < vertex.size(); i++)

{ current\_pathweight += graph[k] [vertex[i]];k = vertex[i];

}

current\_pathweight += graph[k][s];

// update minimum

min\_path = min(min\_path, current\_pathweight);

} while (

next\_permutation(vertex.begin(), vertex.end()));

return min\_path;

}

// Driver Code int main()

{

// matrix representation of graph int graph[][V] = { { 0, 10, 15, 20 },

{ 10, 0, 35, 25 },

{ 15, 35, 0, 30 },

{ 20, 25, 30, 0 } };

int s = 0;

cout << travllingSalesmanProblem(graph, s) << endl; return 0;

}

**OUTPUT:**



# Q2) Given two sequences, find the length of longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous. For example, “abc”, “abg”, “bdf”, “aeg”, ‘”acefg”, .. etc are subsequences of “abcdefg”. So a string of length n has 2n different possible subsequences. Examples: LCS for input Sequences “ABCDGH” and “AEDFHR” is “ADH” of length 3. LCS for input Sequences “AGGTAB” and “GXTXAYB” is “GTAB” of length 4.

**CODE:**

#include <bits/stdc+

+.h>using namespace std;

int max(int a, int b);

/\* Returns length of LCS for X[0..m-1], Y[0..n-1]

\*/int lcs( char \*X, char \*Y, int m, int n )

{

if (m == 0 || n == 0)

return 0;

if (X[m-1] == Y[n-1])

return 1 + lcs(X, Y, m-1, n-1);

else

}

return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));

/\* Utility function to get max of 2 integers

\*/int max(int a, int b)

{

return (a > b)? a : b;

}

/\* Driver code

\*/int main()

{

char X[] = "ABCDGH";char Y[] = "AEDFHR";

int m = strlen(X);int n = strlen(Y);

cout<<"Length of LCS is "<< lcs( X, Y, m, n ) ; return 0;

}

**OUTPUT:**



**Q3) A road network can be considered as a graph with positive weights. The nodes represent road junctions and each edge of the graph is associated with a road segment between two junctions. The weight of an edge may correspond to the length of the associated road segment, the time needed to traverse the segment or the cost of traversing the segment. Using directed edges it is also possible to model one-way streets. Such graphs are special in the sense that some edges are more important than others for long distance travel (e.g. highways). This property has been formalized using the notion of highway dimension. There are a great number of algorithms that exploit this property and are therefore able to compute the shortest path a lot quicker than would be possible on general graphs. Develop a program to find the shortest path from each node to solve the road network problem. Shortest path.**

**CODE:**

#include <bits/stdc+

+.h>using namespace std;

// Number of vertices in the graph#define V 4

/\* Define Infinite as a large enough value.This value will be used for vertices not connected to each other

\*/#define INF 99999

// A function to print the solution matrixvoid printSolution(int dist[][V]);

// Solves the all-pairs shortest path

// problem using Floyd Warshall algorithmvoid floydWarshall(int graph[] [V])

{

/\* dist[][] will be the output matrixthat will finally have the shortest

distances between every pair of vertices

\*/int dist[V][V], i, j, k;

/\* Initialize the solution matrix same as input graph matrix. Or we can say the initial values of shortest distances are based on shortest paths consideringno intermediate vertex. \*/ for (i = 0; i < V; i++)

for (j = 0; j < V; j++)

dist[i][j] = graph[i][j];

/\* Add all vertices one by one to

the set of intermediate vertices.

---> Before start of an iteration,

we have shortest distances between allpairs of vertices such that the shortest distances consider only thevertices in set {0, 1, 2, .. k-1} as intermediate vertices.

----> After the end of an iteration,vertex no. k is added to the set of

intermediate vertices and the set becomes {0, 1, 2, ..k} \*/

for (k = 0; k < V; k++) {

// Pick all vertices as source one by onefor (i = 0; i < V; i++) {

// Pick all vertices as destination for the

// above picked sourcefor (j = 0; j < V; j++) {

// If vertex k is on the shortest path from

// i to j, then update the value of

// dist[i][j]

if (dist[i][j] > (dist[i][k] + dist[k]

[j])&& (dist[k][j] != INF

&& dist[i][k] != INF)) dist[i][j] = dist[i][k] + dist[k][j];

}

}

}

// Print the shortest distance matrixprintSolution(dist);

}

/\* A utility function to print solution

\*/void printSolution(int dist[][V])

{

cout << "The following matrix shows the shortest

""distances"

" between every pair of vertices

\n";for (int i = 0; i < V; i++) { for (int j = 0; j < V; j++)

{ if (dist[i][j] == INF)

cout << "INF"

<< " ";

else

}

cout << endl;

}

cout << dist[i][j] << " ";

}

// Driver codeint main()

{

/\* Let us create the following weighted graph

10

(0)------->(3)

| /|\

5 | |

| | 1

\|/ |

(1)------->(2)

3 \*/

int graph[V][V] = { { 0, 5, INF, 10 },

{ INF, 0, 3, INF },

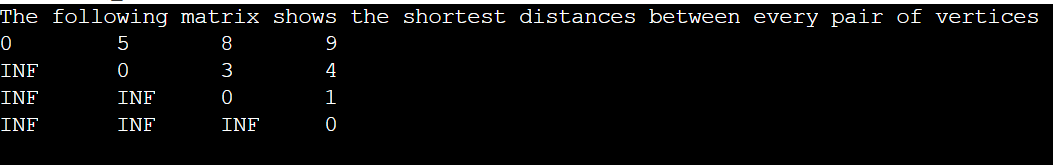
{ INF, INF, 0, 1 },

{ INF, INF, INF, 0 } };

// Print the solution floydWarshall(graph); return 0;

}

**OUTPUT:**



**Q4) There are a number of agents and a number of tasks. Any agent can be assigned to perform any task, incurring so me cost that may vary depending on the agenttask assignment. It is required to perform all tasks by assigning exactly one agent to each task and exactly one task to each agent in such a way that the total costof the assignment is minimized. If the numbers of agents and tasks are equal and the total cost of the assignment for all tasks is equal to the sum of the costs for each agent (or the sum of the costs for each task, which is the same thing in this case), then the problem is called the linear assignment problem. Develop a program to solve this problem.**

**CODE:**

#include <bits/stdc+

+.h>usingnamespacestd; #define N 4

structNode

{

Node\* parent; intpathCost; intcost; intworkerID; intjobID; boolassigned[N];

};

Node\* newNode(intx, inty,

boolassigned[],Node\* parent)

{

Node\* node = newNode;

for(intj = 0; j < N; j++) node->assigned[j] =

assigned[j];node->assigned[y] = true;

node->parent = parent;node-

>workerID = x; node-

>jobID = y;

returnnode;

}

intcalculateCost(intcostMatrix[N][N], intx, inty, boolassigned[])

{

intcost = 0;

boolavailable[N] = {true};

for(inti = x + 1; i < N; i++)

{

intmin = INT\_MAX, minIndex = -1;

for(intj = 0; j < N; j++)

{

if(!assigned[j] && available[j] &&costMatrix[i][j] < min)

{

minIndex = j;

min = costMatrix[i][j];

}

}

cost += min;

available[minIndex] = false;

}

returncost;

}

structcomp

{

booloperator()(constNode\* lhs,

constNode\* rhs) const

{

returnlhs->cost > rhs->cost;

}

};

voidprintAssignments(Node \*min)

{

if(min->parent==NULL) return;

printAssignments(min->parent);

cout << "Assign Worker "<<char(min->workerID + 'A')

<< " to Job "<< min->jobID << endl;

}

intfindMinCost(intcostMatrix[N][N])

{

priority\_queue<Node\*, std::vector<Node\*>, comp> pq;

boolassigned[N] = {false};

Node\* root = newNode(-1, -1, assigned, NULL);root->pathCost = root->cost = 0; root->workerID =

-1;pq.push(root);

while(!pq.empty())

{

Node\* min =

pq.top();pq.pop();

inti = min->workerID + 1;

if(i == N)

{

printAssignments(min)

;returnmin->cost;

}

for(intj = 0; j < N; j++)

{

if(!min->assigned[j])

{

Node\* child = newNode(i, j, min->assigned, min);

child->pathCost = min->pathCost + costMatrix[i][j];

child->cost = child->pathCost + calculateCost(costMatrix, i, j, child->assigned);

pq.push(child);

}

}

}

}

// Driver code intmain()

{

intcostMatrix[N][N] =

{

{9, 2, 7, 8},

{6, 4, 3, 7},

{5, 8, 1, 8},

{7, 6, 9, 4}

};

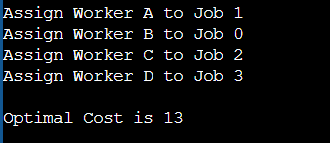
cout << "\nOptimal Cost is "

<< findMinCost(costMatrix);

return0;

}

**OUTPUT:**



**Q5) Suppose that we are designing a program to simulate the storage and search in a dictionary. Words appear with different frequencies, however, and it may be the case that a frequently used word such as "the" appears far from the root while a rarely used word such as "conscientiousness" appears near the root. We want words that occur frequently in the text to be placed nearer to the root. Moreover, there may be words in the dictionary for which there is no definition. Write a program to organize an optimal binary search treethat simulates the storage and search of words in a dictionary.**

**CODE:**

#include<stdio.h> #include<stdlib.h> #define NMAX 20 typedef struct OBST

{

int KEY;

struct OBST \*left, \*right;

}

OBST;

int C[NMAX][NMAX]; i n t W [ N M A X ] [NMAX];int R[NMAX] [NMAX];

int q[NMAX];int p[NMAX];

int NUMBER\_OF\_KEYS;int KEYS[NMAX];

OBST \*ROOT;

void COMPUTE\_W\_C\_R()

{

int X, min; int i, j,k,h,m;

for(i=0; i<=NUMBER\_OF\_KEYS; i++)

{

W[i][i]=q[i];

for(j=i+1; j<= NUMBER\_OF\_KEYS; j+

+)W[i][j]=W[i][j-1]+p[j]+q[j];

}

for(i=0; i<=NUMBER\_OF\_KEYS; i+

+)C[i][i]=W[i][i];

for(i=0; i<=NUMBER\_OF\_KEYS - 1; i++)

{

j=i+1;

C[i][j] = C[i][i] + C[j][j] + W[i] [j];R[i][j] = j;

}

for(h=2; h<=NUMBER\_OF\_KEYS; h++) for(i=0; i<=NUMBER\_OF\_KEYS - h; i++ )

{

j = i+h;

m = R[i][j-1];

min = C[i][m-1]+C[m][j]; for(k =m+1; k <= R[i+1][j]; k++)

{

X =C[i][k-1] + C[k][j];

if(X < min)

{

m = k; min = X;

}

}

C[i][j] = W[i][j] + min;

R[i][j] = m;

}

printf("\nThe weight matrix W:\n ");

for (i=0; i<= NUMBER\_OF\_KEYS; i++)

{

for(j=i; j<= NUMBER\_OF\_KEYS; j+

+)printf("%d", W[i][j]); printf("\n");

}

printf("\nThe cost matrix C:\n");

for (i=0; i<= NUMBER\_OF\_KEYS; i++)

{

for(j=i; j<= NUMBER\_OF\_KEYS; j+

+)printf("%d", C[i][j]); printf("\n");

}

printf("\nThe root matrix R:\n");

for (i=0; i<= NUMBER\_OF\_KEYS; i++)

{

for(j=i; j<= NUMBER\_OF\_KEYS; j+

+)printf("%d", R[i][j]); printf("\n");

}

}

OBST \*CONSTRUCT\_OBST(int i, int j)

{

OBST \*p;

if(i ==j)

p=NULL;

else

{

p = new OBST ;

p->KEY= KEYS[R[i][j]];

p->left = CONSTRUCT\_OBST(i, R[i][j]-1);

p->right = CONSTRUCT\_OBST(R[i][j],j);

}

return p;

}

void DISPLAY (OBST \*ROOT, int nivel)

{

int i; if(ROOT !=0)

{

DISPLAY(ROOT->right, nivel+1); for(i=0; i<= nivel; i++)

printf(" ");

printf("%d\n", ROOT->KEY); DISPLAY(ROOT->left, nivel + 1);

}

}

void OPTIMAL\_BINARY\_SEARCH\_TREE()

{

float average\_cost\_per\_weight; COMPUTE\_W\_C\_R();

printf("C[0]=%d w[0] = %d\n", C[0][NUMBER\_OF\_KEYS], W[0][NUMBER\_OF\_KEYS]);

average\_cost\_per\_weight= C[0][NUMBER\_OF\_KEYS]/(float)W[0][NUMBER\_OF\_KEYS]; printf("The cost per weight ratio is :%f\n", average\_cost\_per\_weight);

ROOT = CONSTRUCT\_OBST(0, NUMBER\_OF\_KEYS);

}

int main()

{

int i, k;

printf("Input number of keys:"); scanf("%d", &NUMBER\_OF\_KEYS);

for(i =1; i<= NUMBER\_OF\_KEYS; i++)

{

printf("key[%d]=",i); scanf("%d",

&KEYS[i]);

printf("frequency="); scanf("%d", &p[i]);

}

for(i=0; i<=NUMBER\_OF\_KEYS; i++)

{

printf("q[%d]=",i);

scanf("%d",&q[i]);

}

while(1)

{

printf("1.Construct tree \n2.Display tree\n3.Exit\n");

scanf("%d",&k);

switch(k)

{

case 1: OPTIMAL\_BINARY\_SEARCH\_TREE();

break;

case 2: DISPLAY(ROOT,0);

break;

exit(0);

case 3:

break;

}

}

system("PAUSE");

}

**OUTPUT:**

