**Rashtreeya Sikshana Samithi Trust**

**R V COLLEGE OF ENGINEERING**

***(Autonomous Institution Affiliated to VTU, Belgaum)***

**Department Of Computer Science & Engineering**

**Bangalore – 560059**

****

**DESIGN & ANALYSIS OF ALGORITHMS LAB**

**SUB CODE: 12CS/IS 45**

**IV SEMESTER B.E**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

***LAB INSTRUCTORS MANUAL***

**2015**

**DESIGN AND ANALYSIS OF ALGORITHMS – LAB(PROGRAM LIST)**

**General Guideline**

1. GOTO statements are not allowed

2. No global declarations allowed

3. Prototype for each user-defined-function must be provided before main

4. main should be the first function in any program

5. Students are encouraged to use user-defined-header files

6. Programs must be indented appropriately

7. Students are required to bring only the algorithms in the data sheet

**Implement the following using C++ Language**.

1. Write a program to sort a given set of elements using Merge sort method and find the time required to sort the elements.
2. Write a program to sort a given set of elements using Quick sort method and find the time required to sort the elements.
3. Write a program to print all the nodes reachable from a given starting node in a graph using Depth First Search method. Also check connectivity of the graph. If the graph is not connected, display the number of components in the graph.
4. Write a program to obtain the Topological ordering of vertices in a given digraph using
   1. Vertices deletion method b)DFS method
5. Write a program to print all the nodes reachable from a given starting node in a graph using Breadth First Search method. Also check connectivity of the graph. If the graph is not connected, display the number of components in the graph.
6. Write a program to sort a given set of elements using Heap sort method. Find the time complexity.

7a. Write a program to implement Horspool algorithm for String Matching

7b. Write a program to implement all pair shortest paths problem using Floyd’s algorithm.

1. Write a program to implement 0/1 Knapsack problem using dynamic programming
2. Write a program to find Minimum cost spanning tree of a given undirected graph using Prim’s algorithm.
3. Write a program to find Minimum cost spanning tree of a given undirected graph using Kruskal’s algorithm.
4. Write a program to find the shortest path using Dijkstra’s algorithm for a weighted connected graph.
5. Write a program to implement Subset-Sum problem using Back Tracking
6. Write a program to implement TSP using branch and bound algorithm
7. Write a program to implement n-queens problem

**SOLUTIONS TO LAB PROGRAMS WITH EXPECTED INPUT AND OUTPUT**

1. **Write a program to sort a given set of elements using Merge sort method and find the time required to sort the elements.**

#include<stdio.h>

#define MAX 1000

int count;

int main()

{

int i,j,n,a[MAX],b[MAX],c[MAX];

int c1,c2,c3;

printf("\n Enter n: ");

scanf("%d",&n);

printf("\nEnter elements: ");

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

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

count=0;

mergesort(a,0,n-1);

printf("\n Sorted elements: \n");

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

printf("%d\n",a[i]);

printf(“\n Number of counts : %d\n”,count);

printf(“\n SIZE\t ASC\t DESC\t RAND\n”);

for(i=16; i<550;i=i\*2)

{

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

{

a[j]=j;

b[j]=i-j;

c[j]=rand() % i;

}

count=0;

mergesort(a,0,i-1);

c1=count;

count=0;

mergesort(b,0,i-1);

c2=count;

count=0;

mergesort(c,0,i-1);

c3=count;

printf(“\n %d\t%d\t%d\t%d”,i,c1,c2,c3);

}

return 0;

}

void mergesort(int a[MAX], int low, int high)

{

int mid;

if(low < high)

{

count++;

mid = (low + high)/2;

mergesort(a,low,mid);

mergesort(a,mid+1,high);

merge(a,low,mid,high);

}

}

void merge(int a[MAX], int low, int mid, int high)

{

int i, j, k, b[MAX];

i = low;

j = mid+1;

k = low;

while( (i<=mid) && (j<=high))

{

count++;

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

b[k++] = a[i++];

else

b[k++] = a[j++];

}

while(i <= mid)

b[k++] = a[i++];

while(j <= high)

b[k++] = a[j++];

for(i=low; i<=high; i++)

a[i] = b[i];

}

**OUTPUT :**

Enter n: 6

Enter elements: 14 5 7 89 2 34

Sorted elements:

2

5

7

14

34

89

**2. Write a program to sort a given set of elements using Quick sort method and find the time required to sort the elements.**

#include <stdio.h>

#include <stdlib.h>

#define MAX 1000

//Function declarations

void quicksort(int a[MAX], int low, int high);

int partition(int a[MAX], int low, int high);

int count;

int main()

{

int n; //No. of elements

int a[MAX],b[MAX],c[MAX]; //Array to store elements

int i; //Index variable

int c1,c2,c3;

printf("\nEnter n: ");

scanf("%d",&n);

printf("\nEnter elements: \n");

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

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

count=0;

quicksort(a,0,n-1);

printf("Sorted elements: \n");

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

printf("%d\n",a[i]);

printf(“\n Number of counts : %d\n”,count);

printf(“\n SIZE\t ASC\t DESC\t RAND\n”);

for(i=16; i<550;i=i\*2)

{

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

{

a[j]=j;

b[j]=i-j;

c[j]=rand() % i;

}

count=0;

quciksort(a,0,i-1);

c1=count;

count=0;

quciksort(b,0,i-1);

c2=count;

count=0;

quciksort(c,0,i-1);

c3=count;

printf(“\n %d\t%d\t%d\t%d”,i,c1,c2,c3);

}

return 0;

}

void quicksort(int a[MAX],int low,int high)

{

int j;

count++;

if(low < high)//If there are more than one elements in the array

{

j = partition(a, low, high);

quicksort(a, low, j-1); //Sort left subarray

quicksort(a, j+1, high); //Sort right subarray

}

}

int partition(int a[MAX], int low, int high)

{

int i, j, key, temp;

i = low + 1; //Initialise lower index i

j = high; //Initialise higher index j

key = a[low]; //Make first element as key

while(1)

{

while ((key >= a[i]) && i < high)

i++;

while(key < a[j])

j--;

if(i < j)

{

temp = a[i]; a[i] = a[j]; a[j] = temp;

}

else

{

temp = a[low]; a[low] = a[j]; a[j] = temp;

return j;

}//end if

}//end while

}//end function

**OUTPUT :**

Enter n: 6

Enter elements: 14 5 7 89 2 34

Sorted elements:

2

5

7

14

34

89

1. **Write a program to print all the nodes reachable from a given starting node in a graph using Depth First Search method. Also check connectivity of the graph. If the graph is not connected, display the number of components in the graph.**

#include <stdio.h>

void dfs(int a[10][10], int n, int v[10], int source);

int main()

{

int n;

int a[10][10];

int v[10];

int source;

int i, j;

int count = 0;

printf("Enter no of nodes: ");

scanf("%d",&n);

printf(“\n Read Adjacency matrix \n”);

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

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

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

printf("Enter source: ");

scanf("%d",&source);

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

v[i] = 0;

dfs(a,n,v,source);

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

{

if(v[i] == 0)

{

dfs(i,a,n,v);

count++;

}

}

printf("Result: ");

if(count == 1)

printf("Graph is Connected");

else

printf("Graph is NOT Connected with %d Components\n",count);

return 0;

}

void dfs(int a[10][10], int n, int v[10], int source)

{

int i;

v[source] = 1;

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

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

dfs(a,n,v,i);

}

**4a. Write a program to obtain the Topological ordering of vertices in a given digraph using Vertices deletion method**

#include<stdio.h>

int main()

{

int n;

int a[10][10];

int i,j,k,node;

int in[10]={0};

int v[10]={0};

printf("Enter n: ");

scanf("%d",&n);

printf("Enter Adj matrix: \n");

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

{

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

{

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

if(a[i][j] == 1)

in[j]++;

}

}

printf("\nTopological order: ");

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

{

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

{

if(in[i] == 0 && v[i] == 0)

{

node = i;

printf("%5d",node);

v[node] = 1;

break;

}

}

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

if(a[node][i] == 1)

in[i]--;

}

printf("\n\n");

}

**4b. Write a program to obtain the Topological ordering of vertices in a given digraph using DFS method**

#include<stdio.h>

#include<stdlib.h>

int j=0;pop[10],v[10];

void dfs(int source,int n,int a[10][10])

{

int i,k,top=-1,stack[10];

v[source]=1;

stack[++top]= source+1;

while(top!=-1)

{

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

{

if( a[source][k] == 1 && v[k] == 1 )

{

for(i=top; i>=0;i--)

if(stack[i] == k+1 )

{

printf(“\n Topological order not possible”);

exit(0);

}

}

else

{

if( a[source][k] == 1 && v[k] == 0)

{

v[k]=1;

stack[++top]= k+1;

source = k;

k=0;

}

}

}

pop[j++]=source+1;

top --;

source = stack[top] – 1;

}

}

void topo(int n , int a[10][10])

{

int i,k;

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

v[i]=0;

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

if(v[k]== 0)

dfs(k,n,a);

}

int main()

{

int n,i,j,a[10][10];

printf(“\n Enter the no of Vertices : “);

scanf(“%d”,&n);

printf(“\n Enter the Adjacency matrix\n”);

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

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

scanf(“%d”,&a[i][j]);

topo(n,a);

printf(“\n The topological ordering is\n”);

for(i=n-1;i>=0;i--)

printf(“%d\t”,pop[i]);

}

1. **Write a program to print all the nodes reachable from a given starting node in a graph using Breadth First Search method. Also check connectivity of the graph. If the graph is not connected, display the number of components in the graph.**

#include <stdio.h>

void bfs(int a[10][10], int n, int v[10], int source);

int main()

{

int n;

int a[10][10];

int v[10];

int source;

int i, j,count=0;

printf("Enter no of nodes: ");

scanf("%d",&n);

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

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

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

printf("Enter source: ");

scanf("%d",&source);

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

v[i] = 0;

bfs(a,n,v,source);

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

{

if(v[i] == 0)

{

bfs(a,n,v,i);

count++;

}

}

printf("Result: ");

if(count == 1)

printf("Graph is Connected");

else

printf("Graph is NOT Connected with %d Components\n",count);

return 0;

}

void bfs(int a[10][10], int n, int v[10], int source)

{

int q[10], front=0, rear=-1;

int node, i;

v[source] = 1;

q[++rear] = source;

while(front <= rear)

{

node = q[front++];

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

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

{

v[i] = 1;

q[++rear] = i;

}

}//end while

}//end bfs

1. **Write a program to sort n elements using heap sort.**

#include<stdio.h>

#define MAX 1000

int count =0;

void heapcon(int a[MAX],int n)

{

int i,k,v,flag,j;

for(i=n/2; i>=1; i--)

{

k=i;

v=a[k];

flag = 0;

while ( !flag && (2\*k<=n) )

{

j=2\*k;

if(j<n)

if(a[j] < a[j+1])

{

j=j+1;

count ++;

}

if(v>=a[j])

flag = 1;

else

{

a[k]=a[j];

k=j;

}

}

a[k]=v;

}

}

void heapsort(int a[MAX], int n)

{

int i,j,temp;

for(i=n;i>=1;i--)

{

temp=a[1];

a[1]=a[i];

a[i]=temp;

heapcon(a,i-1);

}

}

void main()

{

int a[MAX],b[MAX],c[MAX];

int n,i,j,c1,c2,c3;

printf("\n enter the number of elements to be sorted : ");

scanf("%d",&n);

printf("\n Enter the elements to be sorted\n");

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

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

heapcon(a,n);

heapsort(a,n);

printf("\n Elements after sorting\n");

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

printf("%d ",a[i]);

printf(“\n Number of counts : %d\n”,count);

printf(“\n SIZE\t ASC\t DESC\t RAND\n”);

for(i=16; i<550;i=i\*2)

{

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

{

a[j]=j;

b[j]=i-j;

c[j]=rand() % i;

}

count=0;

mergesort(a,0,i-1);

c1=count;

count=0;

mergesort(b,0,i-1);

c2=count;

count=0;

mergesort(c,0,i-1);

c3=count;

printf(“\n %d\t%d\t%d\t%d”,i,c1,c2,c3);

}

return 0;

}

**7a. Write a program to implement Horspool algorithm for String Matching.**

#include<stdio.h>

int min(int a,int b)

{

if(a < b)

return a;

else

return b;

}

void floyd(int n,int d[10][10])

{

int i,j,k;

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

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

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

d[i][j] = min(d[i][j], d[i][k] + d[k][j]);

}

int main()

{

int n,a[10][10],d[10][10];

int i,j,k;

printf("Enter the no.of nodes: ");

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]);

d[i][j] = a[i][j];

}

floyd(n,a);

printf("\n\nThe distance matrix is \n");

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

{

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

printf("%5d",d[i][j]);

printf("\n");

}

return 0;

}

**7b. Write a c program to implement horspool string matching algorithm**

#include<stdio.h>

#include<conio.h>

#include<string.h>

#define MAX 256

int t[MAX];

int count=1;

void shifttable(char pat[])

{

int i,j,m;

m=strlen(pat);

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

t[i]=m;

for(j=0;j<m-1;j++)

t[pat[j]]=m-1-j;

}

int horspool(char src[],char pat[])

{

int i,j,k,m,n;

n=strlen(src);

m=strlen(pat);

i=m-1;

while(i<n)

{ k=0;

while((k<m) && (pat[m-1-k]==src[i-k]))

k++;

if(k==m)

return (i-m+1);

else

{

i=i+t[src[i]];

count=count+1;

}

}

return -1;

}

int main()

{

char src[100],pat[10];

int pos;

printf("\n Enter the main source string\n");

gets(src);

printf("\n Enter the pattern to be searched\n");

gets(pat);

shifttable(pat);

pos=horspool(src,pat);

if(pos>=0)

{

printf("\n Found at %d position ",pos+1);

printf("\n number of shifts are %d",count);

}

else

printf("\n String match failed");

return 0;

}

**8. Write a program to implement 0/1 Knapsack problem using dynamic programming.**

#include <stdio.h>

#define MAX 150

//Function declarations

int knap(int n,int m);

int big(int a,int b);

//Global variables

int w[MAX]; //Array to store weights of each item

int p[MAX]; //Array to store profits of each item

int v[MAX][MAX]; //Optimal solution of 'i' items with 'j' capacity

int main()

{

int i, j, profit, n, m;

printf("\n Enter n (no. of items): ");

scanf("%d",&n);

printf("\n Enter the knapsack capacity:");

scanf("%d",&m);

printf("\n enter the weights and profits :\n");

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

{

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

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

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

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

}

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

v[i][0]=0;

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

v[0][j]=0;

profit = knap(n,m);

printf("\n goal = %d\n\n",profit);

return 0;

}

int knap(int n,int m)

{

int i, j;

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

for(j = 1; j <= m; j++)

{

if( (j - w[i]) < 0)

v[i][j] = v[i-1][j];

else

v[i][j] = big(v[i-1][j], p[i] + v[i-1][j-w[i]] );

}

return v[n][m];

}

int big(int a,int b)

{

if (a > b) return a; else return b;

}

**9. Write a program to find Minimum cost spanning tree of a given undirected graph using Prim’s algorithm.**

#include<stdio.h>

#define INFINITY 999

void prims(int n, int cost[10][10], int source);

int main()

{

int n; //no. of nodes

int cost[10][10]; //Adjacency matrix of graph

int source; //source node

int i, j; //index variables

printf("Enter n (no. of nodes): ");

scanf("%d",&n);

printf("Enter cost matrix:\n ");

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

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

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

printf("Enter Source: ");

scanf("%d",&source);

prims(n,cost,source);

return 0;

}

void prims(int n,int cost[10][10],int source)

{

int v[10];

int d[10];

int i, j;

int vertex[10];

int u, least, sum=0;

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

{

v[i] = 0;

d[i] = cost[source][i];

vertex[i] = source;

}

v[source] = 1;

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

{

least = INFINITY;

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

{

if(v[j] == 0 && d[j] < least)

{

least = d[j];

u = j;

}

}

v[u] = 1;

sum += d[u];

printf("%d --> %d = %d Sum = %d\n\n",vertex[u],u, d[u],sum);

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

{

if(v[j] == 0 && cost[u][j] < d[j])

{

d[j] = cost[u][j];

vertex[j] = u;

}

}

}

printf("Total cost: %d",sum);

}

/\*

Output1:

Enter n (no. of nodes): 4

Enter cost matrix:

0 20 10 50

20 0 60 999

10 60 0 40

50 999 40 0

Enter Source: 1

1 --> 1 = 0 Sum = 0

1 --> 2 = 20 Sum = 20

1 --> 3 = 10 Sum = 30

3 --> 4 = 40 Sum = 70

Total cost: 70

\*/

**10. Write a program to find Minimum cost spanning tree of a given undirected graph using Kruskal’s algorithm.**

#include<stdio.h>

#define INFINITY 999

#define MAX 10

//Function declarations

void kruskal(int n);

int get\_parent(int v);

void join(int i,int j);

void sort\_edges();

void display();

struct EDGE

{

int x, y, wt;

}e[MAX];

int parent[MAX];

int cost[MAX][MAX]; //cost matrix

int t[MAX][2]; //Result: edges in spanning tree

int nedges; //no. of edges

int eno; //edge number (used as index in e[])

int main()

{

int i,j;

int n; //no. of nodes

//1. Read no. of nodes

printf("\nEnter the no.of vertices: ");

scanf("%d",&n);

//2. Initialize each element of parent[] to zero

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

parent[i] = 0;

//3. Read cost matrix of graph and Identify edges and store in e

eno = 1;

printf("\nEnter the cost adjacency matrix: 0 = self loop & 999 = no edge\n");

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

{

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

{

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

if(i == j || cost[i][j] == INFINITY)

continue;

//add edge

e[eno].x = i;

e[eno].y = j;

e[eno].wt = cost[i][j];

eno++; nedges++;

}

}

//4. Sort the edges in e[]

sort\_edges();

//5. Call kruskals function

kruskal(n);

return 0;

}

//Function to return top level parent of a given node v.

int get\_parent(int v)

{

while(parent[v])

v = parent[v];

return v;

}

//Function to update parent array after edge added to spanning tree

void join(int i, int j)

{

parent[j] = i;

}

//Function to obtain minimum cost spanning tree

void kruskal(int n)

{

int i,j,k,sum=0;

int eno = 1;

struct EDGE nextedge;

//a. Select n-1 edges to connect all nodes

for(k=1; k < n; )

{

nextedge = e[eno++]; //b. Get next edge

i = get\_parent( nextedge.x );//c. Find parents of i and j

j = get\_parent( nextedge.y );

if(i != j) //d. If parents are different

{ // include the edge in spanning tree

//else ignore the edge

join(nextedge.x, j); //e. parent[j] = nextedge.x;

t[k][1] = nextedge.x; //f.Store the edge in t[][]

t[k][2] = nextedge.y;

sum = sum + nextedge.wt; //g. Add cost on edge to sum

k++;

}

}

//h. Display result

printf("\nCost of the spaning tree is: %d\n",sum);

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

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

printf("%d -> %d\n",t[i][1],t[i][2]);

}

//Function to sort(bubble sort) edges based on cost of edges

void sort\_edges()

{

int i,j;

struct EDGE temp;

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

for(j=1; j < nedges-i; j++)

if(e[j].wt > e[j+1].wt)

{

temp = e[j]; e[j] = e[j+1]; e[j+1] = temp;

}

}

/\* OUTPUT:

Run1:

enter the number of vertices:4

enter the cost adjacency matrix

0 20 2 999

20 0 15 5

2 15 0 25

999 5 25 0

cost of spanning tree is 22

edges of spanning tree are

1->3

2->4

2->3

Run 2:

Enter the no.of vertices: 5

Enter the cost adjacency matrix: 0 = self loop & 999 = no edge

0 999 10 7 999

999 0 999 32 999

10 999 0 9 999

7 32 9 0 23

999 999 999 23 0

Cost of the spaning tree is: 71

The edges of the spanning tree are:

1 -> 4

3 -> 4

4 -> 5

2 -> 4

\*/

**11. Write a program to find the shortest path using Dijkstra’s algorithm for a weighted connected graph.**

#include <stdio.h>

#define INFINITY 999

void dijk(int cost[10][10], int n, int source, int v[10], int d[10]);

int main()

{

int n; //no. of nodes

int cost[10][10]; //Adjacency matrix of graph

int source; //source node

int v[10]; //visited array. keeps track to nodes visited

int d[10]; //distance array.shortest distance from source node

int i, j; //index variables

//1. Read no. of nodes

printf("Enter n: ");

scanf("%d",&n);

//2. Read cost adjacency matrix of graph

printf("Enter Cost matrix: \n");

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

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

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

//3. Read source

printf("Enter Source: ");

scanf("%d",&source);

//4. Initialise d[] to distance from source to each node

//Initialise v[] to 0, indicating none of the nodes are visited

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

{

d[i] = cost[source][i];

v[i] = 0;

}

//5. Call function to compute shortest distance

dijk(cost, n, source, v, d);

//6. Print Shortest distance from source to all other nodes

printf("Shortest distance from source %d\n\n",source);

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

printf("%d --> %d = %d\n\n",source,i,d[i]);

return 0;

}

//Function to implement dijkstra algorithm

void dijk(int cost[10][10],int n,int source,int v[10],int d[10])

{

int least, i, j, u;

//A. Mark source node as visited

v[source] = 1;

//B. From each node find shortest distance to nodes not visited

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

{

//B1. Assume least as infinity

least = INFINITY;

//B2. Find u and d(u) such that d(u) is minimum i.e., Find //the next nearest node

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

{

if(v[j] == 0 && d[j] < least)

{

least = d[j];

u = j;

}

}

//B3. Mark u as visited (mark nearest node as visited)

v[u] = 1;

//B4. For remaining nodes, find shortest distance through u

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

{

if(v[j] == 0 && (d[j] > (d[u] + cost[u][j])) )

d[j] = d[u] + cost[u][j];

}

}//end for outer

}//end function

**12. Write a program to implement Subset-Sum problem using Back Tracking.**

#include <stdio.h>

void subset(int n, int d, int s[]);

int main()

{

int n; //No. of elements in set

int d; //Required subset sum

int s[10]; //Array: Elements in the set

int i; //index variable

int sum = 0;

//1. Read no. of elements in set

printf("Enter the value of n");

scanf("%d",&n);

//2. Read the elements in the set

printf("Enter the set in increasing order\n");

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

{

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

sum += s[i];

}

//3. Read required subset sum

printf("Enter the maximum subset value of d: ");

scanf("%d",&d);

//4. Call function

if(sum < d)

printf("Solution NOT possible.\n");

else

subset(n,d,s);

return 0;

}

void subset(int n, int d, int s[])

{

int x[10]; //Shows elements in subset (0 - Absent 1 - Present)

int sum; //Stores current sumset sum

int i, k; //index variables

//Initialise x[] to 0. (None of the elements in set are selected)

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

x[i] = 0;

sum = 0;

k = 1; //Take first element

x[k] = 1; //Add first element to subset

while(1)

{

if(k <= n && x[k] == 1)

{

if(sum+s[k] == d)

{

printf("Solution is \n");

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

{

if(x[i] == 1)

printf("%5d", s[i]);

}

printf("\n");

x[k] = 0;

}

else if(sum + s[k] < d)

sum += s[k];

else

x[k] = 0;

}

else

{

k--;

while(k > 0 && x[k] == 0)

k--;

if(k == 0) break;

x[k] = 0;

sum = sum - s[k];

}

k = k + 1;

x[k] = 1;

}

}

/\*

Run1:

Enter the value of n5

Enter the set in increasing order

1

2

3

4

5

Enter the maximum subset value of d: 7

Solution is

1 2 4

Solution is

2 5

Solution is

3 4

\*/

**13. Write a program to implement TSP using branch and bound algorithm.**

#include<stdio.h>

//Function declarations

int tsp\_dp(int source,int v[10]);

int tsp\_nn(int source,int v[10]);

int g(int source,int s[10]);

int setempty(int s[10]);

//Global variables

int n,cost[10][10],start;

//Main function

int main()

{

int v[10] = {0}; //Initialise all elements of v[] = 0

int i, j;

int mincost1, mincost2;

//Read No. of cities

printf("Enter no. of cities: ");

scanf("%d",&n);

//Read cost matrix

printf("Enter cost matrix:\n");

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

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

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

//Read starting node (to start journey)

printf("Enter Source: ");

scanf("%d",&start);

//Solve TSP using dynamic programming and find least path

mincost1 = tsp\_nn(start,v);

//Initialise all elements of v[] = 0

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

v[i] = 0;

//Solve TSP using nearest neighbour and find least path

mincost2 = tsp\_dp(start,v);

//Print result

printf("\n\nCost using NN = %5d\n\n",mincost1);

printf("\n\nCost using DP = %5d\n\n",mincost2);

printf("Deviation: %f\n\n",(float)mincost1/mincost2);

return 0;

}

//Function to check set is empty or not

//returns 1 - if set is empty else returns 0

int setempty(int s[10])

{

int i;

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

{

if(s[i] == 0) return 0;

}

return 1;

}

//Function to find the optimal path from source to source through all //the remaining nodes(k)

int g(int source,int s[10])

{

int k,sum,least;

//If set empty return c(1,k)

if(setempty(s))

return cost[source][start];

//Compute least cost path from source to source through all the //remaining nodes(k)

//for all combinations of remaining(k) nodes

least = 999;

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

{

if(s[k] == 1) //If node k already visited then ignore

continue;

s[k] = 1;

sum = cost[source][k] + g(k,s);

if(sum < least)

{

least = sum;

}

s[k] = 0;

}// end for

return least;

}// end g

//Function to find optimal path using Dynamic programming

int tsp\_dp(int source,int v[10])

{

int sum;

v[source] = 1; //mark source node as visited

sum = g(source,v); //get optimal path cost

return sum;

}

//Function to find optimal path using Nearest neighbour (Approximation technique)

int tsp\_nn(int source,int v[10])

{

int sum=0;

int least=0;

int nextnode;

int i,j;

//Make diagonal elements as infinity (999)

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

{

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

if(i == j)

cost[i][j] = 999;

}

printf("TSP Solution using Nearest neighbour:\n\n");

printf("Path : %5d",source);

//Find least cost neighbour and visit it.

//Repeat the process for n-1 times

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

{

v[source] = 1;

least = 999;

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

{

if(cost[source][j] < least && v[j] == 0)

{

least = cost[source][j];

nextnode = j;

}

}

sum += least;

printf(" --> %5d",nextnode);

source = nextnode;

}

//add cost from last node to start node

sum += cost[nextnode][start];

printf(" --> %5d\n\n",start);

return sum;

}

/\*

Run 1:

Enter no. of cities: 4

Enter cost matrix:

0 30 6 4

30 0 5 10

6 5 0 20

4 10 20 0

Enter Source: 2

TSP Solution using Nearest neighbour:

Path : 2 --> 3 --> 1 --> 4 --> 2

Cost using NN = 25

Cost using DP = 25

Deviation: 1.000000

Run 2:

Enter no. of cities: 4

Enter cost matrix:

0 10 15 20

5 0 9 10

6 13 0 12

8 8 9 0

Enter Source: 4

TSP Solution using Nearest neighbour:

Path : 4 --> 1 --> 2 --> 3 --> 4

Cost using NN = 39

Cost using DP = 35

Deviation: 1.114286

\*/

**14. Write a program to implement n-queens problem.**

#include <stdio.h>

//Function declarations

void nqueens(int n);

int can\_place(int c[10],int r);

void display(int c[10],int r);

//Global variable

int count = 0;

int main()

{

int n;

//1. Read no. of queens

printf("Enter n (no of queens): ");

scanf("%d",&n);

//2. Call function if solution exist

if(n == 2 || n == 3)

printf("Solution doesnot exist.");

else

{

nqueens(n);

printf("Total no. of solutions: %d\n",count);

}

return 0;

}

void nqueens(int n)

{

int r; //Contains row no.

int c[10]; //Stores queens positions in each row

int i;

r = 0; //Select first queen (place queen in first row)

c[r] = -1; //Initial position of queen

while(r >= 0) //As long as there are solutions

{

c[r]++; //Place queen in r th coloumn

//verify there is no attack from any of t previous queens placed

while(c[r] < n && !can\_place(c,r))

c[r]++;

if(c[r] < n)

{

if(r == n-1) /if all n queens - display

{

printf("Solution %d: ",++count);

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

printf("%4d",c[i]+1);

display(c,n);

}

else //else place the next queen in next row

{

r++;

c[r] = -1;

}

}

else

r--; //backtracking (go to previous row)

}

}

//Function to check attack on queen r from 0-(r-1) queens

//return 0: if there is attack, other wise return 1;

int can\_place(int c[10],int r)

{

int i;

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

{

if( (c[i] == c[r]) || (abs(i-r) == abs(c[i] - c[r])) )

return 0;

}

return 1;

}

//Function to create chessboard with queens placed and display

void display(int c[10],int n)

{

char cb[10][10];

int i, j;

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

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

cb[i][j] = '-';

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

cb[i][c[i]] = 'Q';

//Display the chess board

printf("\n\nChessboard: \n");

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

{

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

printf("%4c",cb[i][j]);

printf("\n\n");

}

}