

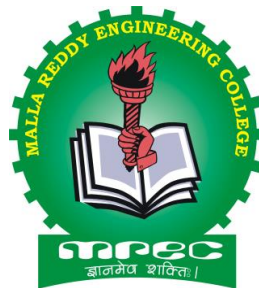
# **Department of Computer Science and Engineering (IOT)**

## **II B. Tech II Semester**

**Subject Name: Design & Analysis of Algorithms Lab Manual**

**Subject Code: A0521**

**Regulations: MR-20**



**Academic Year: 2021-22**



### **MALLAREDDY ENGINEERING COLLEGE**

(An UGC Autonomous Institution, Approved by AICTE and Affiliated to JNTUH Hyderabad,  
Recognized under section 2(f) & 12(B) of UGC Act 1956,  
Accredited by NAAC with 'A' Grade (II Cycle) and NBA  
Maisammaguda, Dhulapally (Post Via Kompally), Secunderabad-500 100

<b>2020-21 Onwards (MR-20)</b>	<b>MALLA REDDY ENGINEERING COLLEGE (Autonomous)</b>	<b>B.Tech. III Semester</b>		
<b>Code: A0521</b>	<b>Design and Analysis of Algorithms Lab (Common for CSE, CSE (Cyber Security), CSE (AI and ML), CSE (DS), CSE (IOT) and IT)</b>	<b>L</b>	<b>T</b>	<b>P</b>
<b>Credits: 1.5</b>		<b>-</b>	<b>-</b>	<b>3</b>

#### **COURSE OBJECTIVES:**

This course will make students

1. To analyze asymptotic performance of algorithms, understand different methods postfix, infix expressions, spanning tree algorithms, Strassen's matrix multiplication.
2. To develop solutions to Job sequencing problems, Knapsack algorithm, shortest path algorithms.
3. To implement solutions traveling sales person.
4. To apply dynamic programming method N-Queen's Problem.
5. To learn and apply synthesizing branch and bound, NP problems.

#### **Software Requirements: Turbo C**

#### **LIST OF PROGRAMS:**

1. Write a program to evaluate a postfix expression E. Assume E is presented dataString.
2. Write a program to obtain the postfix form of an infix expression E. Assume E has only the binary operators +, -, \*, /, ^.
3. Implement the minimum cost spanning tree algorithm (Kruskal's algorithm).
4. Implement the minimum cost spanning tree algorithm (Prim's algorithm).
5. Implement Strassen's matrix multiplication.
6. Implement Job sequencing problem with deadlines.
7. Implement the Knapsack Algorithm.
8. Implement the shortest path Dijkstra's Algorithm.
9. Implement SSSP (Single Source Shortest Path) in DAG (Directed Acyclic Graphs).
10. Implement travelling sales person problem.
11. Implement N-Queen's Problem using Backtracking.
12. Implement sum of subsets problem.

#### **TEXTBOOKS**

1. Ellis Horowitz, Satrajit Sahn and Rajasekharan, "**Fundamentals of Computer Algorithms**" Galgotia publications pvt. Ltd
2. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, "**Introduction to Algorithms**", second edition, PHI Pvt. Ltd./ Pearson Education
3. Parag Himanshu Dave, Himanshu Balchandra Dave, "**Design and Analysis of algorithms**" Pearson.

## REFERENCES

1. M.T.Goodrich and R.Tomassia "**Algorithm Design, Foundations, Analysis and Internet examples**", John Wiley and Sons.
2. R.C.T.Lee, S.S.Tseng, R.C.Chang and T.Tsai, "**Introduction to Design and Analysis of Algorithms A strategic approach**", Mc Graw Hill.

## COURSE OUTCOMES:

At the end of the course, students will be able to

1. **Analyze** asymptotic performance of algorithms, understand different methods
2. **Develop** solutions to Job sequencing problems, Knapsack algorithm, shortest path algorithms,
3. **Implement** solutions traveling sales person.
4. **Apply** dynamic programming method N-Queen's Problem.
5. **Apply** synthesizing branch and bound NP problems.

CO- PO, PSO Mapping (3/2/1 indicates strength of correlation) 3-Strong, 2-Medium, 1-Weak															
COs	Programme Outcomes (POs)												PSOs		
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	3	3	2									3	3	
CO2	3	3	3	2									3	2	
CO3	3	3	3	2									3	2	
CO4	3	3	3	2									3	2	
CO5	3	3	3	2									3	2	

<b>S.No</b>	<b>Experiment</b>
<b>1</b>	<b>EVALUATE POSTFIX EXPRESSION</b>
<b>2</b>	<b>POSTFIX FROM INFIX EPRESSION</b>
<b>3</b>	<b>MINIMUM COST SPANNING TREE-KRUSKALS ALGORITHM</b>
<b>4</b>	<b>MINIMUM COST SPANNING TREE-PRIMS ALGORITHM</b>
<b>5</b>	<b>STRASSENS MATRIX MULTIPLICATION</b>
<b>6</b>	<b>JOB SEQUENCING WITH DEADLINES</b>
<b>7</b>	<b>KNAPSACK PROBLEM</b>
<b>8</b>	<b>DIJKSTRA'S ALGORITHM</b>
<b>9</b>	<b>SINGLE SOURCE SHORTEST PATHS ALGORITHM</b>
<b>10</b>	<b>TRAVELLING SALES PERSON PROBLEM</b>
<b>11</b>	<b>N QUEENS PROBLEM</b>
<b>12</b>	<b>SUM OF SUB SETS PROBLEM</b>

## POSTFIX EVALUATION

```
#include<stdio.h>

#define MAX 20

typedef struct stack
{
    int data[MAX];
    int top;
}stack;

void init(stack *);
int empty(stack *);
int full(stack *);
int pop(stack *);
void push(stack *,int);
int evaluate(char x,int op1,int op2);

int main()
{
    stack s;
    char x;
    int op1,op2,val;
    init(&s);
    printf("Enter the expression(eg: 59+3*)\nSingle digit operand and operators only:");
    while((x=getchar())!='\n')
    {
        if(isdigit(x))
            push(&s,x-48); //x-48 for removing the effect of ASCII
        else
        {
            op2=pop(&s);
            op1=pop(&s);
            val=evaluate(x,op1,op2);
            push(&s,val);
        }
    }
    val=pop(&s);
    printf("\nValue of expression=%d",val);

    return 0;
}

int evaluate(char x,int op1,int op2)
{
    if(x=='+')
        return(op1+op2);
    if(x=='-')
```

```

return(op1-op2);
if(x=='*')
return(op1*op2);
if(x=='/')
return(op1/op2);
if(x=='%')
return(op1%op2);
}

```

```

void init(stack *s)
{
s->top=-1;
}

```

```

int empty(stack *s)
{
if(s->top==-1)
return(1);
return(0);
}

```

```

int full(stack *s)
{
if(s->top==MAX-1)
return(1);
return(0);
}

```

```

void push(stack *s,int x)
{
s->top=s->top+1;
s->data[s->top]=x;
}

```

```

int pop(stack *s)
{
int x;
x=s->data[s->top];
s->top=s->top-1;
return(x);
}

```

OUTPUT:

Enter the expression :: 245+\*

The result of expression 245+\* = 18

## INFIX TO POSTFIX CONVERSION

```
#include<stdio.h>
#include<string.h>
char stack[50];
int top=-1;
void post(char infix[]);
void push(char);
char pop();

void main()
{
    char infix[25];
    printf("\nEnter the infix expression = ");
    gets(infix);
    post(infix);
    getch();
}

void push(char symb)
{
    if(top>=49)
    {
        printf("\nStack overflow");
        getch();
        return;
    }
    else
    {
        top=top+1;
        stack[top]=symb;
    }
}

char pop()
{
    char item;
    if(top== -1)
    {
```

```

        printf("\nSTACK IS EMPTY");
        getch();
        return(0);
    }
    else
    {
        item=stack[top];
        top--;
    }
    return(item);
}
int preced(char ch)
{
    if(ch==47)
    {
        return(5);
    }
    else if(ch==42)
    {
        return(4);
    }
    else if(ch==43)
    {
        return(3);
    }
    else
        return(2);
}
void post(char infix[])
{
    int l;
    int index=0,pos=0;
    char symbol,temp;
    char postfix[40];
    l=strlen(infix);
    push('#');
    while(index<l)
    {

```

```

symbol=infix[index];
switch(symbol)
{
    case '(': push(symbol);
    break;
    case ')': temp=pop();
    while(temp!='(')
    {
        postfix[pos]=temp;
        pos++;
        temp=pop();
    }
    break;
    case '+':
    case '-':
    case '*':
    case '/':
    case '^':
    while(preced(stack[top])>=preced(symbol))
    {
        temp=pop();
        postfix[pos]=temp;
        pos++;
    }
    push(symbol);
    break;
    default: postfix[pos++]=symbol;
    break;
}
index++;
}
while(top>0)
{
    temp=pop();
    postfix[pos++]=temp;
}
postfix[pos++]='\0';
puts(postfix);

```

```
    return;  
}
```

**Output:**

First Run:

Enter Infix expression :  $A+(B*C-(D/E^F)*G)*H$

Postfix Expression:  $ABC*DEF^/G*-H*+$

Second Run:

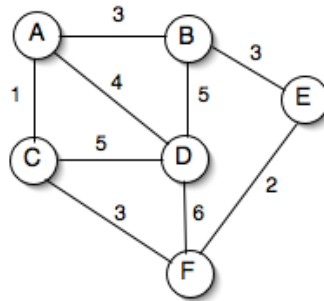
Enter Infix expression :  $(3^2*5)/(3*2-3)+5$

Postfix Expression:  $32^5*32*3-/5+$

## MINIMUM COST SPANNING TREE

### OBJECTIVE:

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



### PROGRAM LOGIC:

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.

### PROCEDURE:

1. Create: Open Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

### SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
inti,j,k,a,b,u,v,n,ne=1;
intmin,mincost=0,cost[9][9],parent[9];
int find(int);
intuni(int,int);
void main() {
    printf("\n Implementation of Kruskal's algorithm\n\n");
    printf("\nEnter the no. of vertices\n");
    scanf("%d",&n);
    printf("\nEnter the cost adjacency matrix\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;
        }
    }
```

```

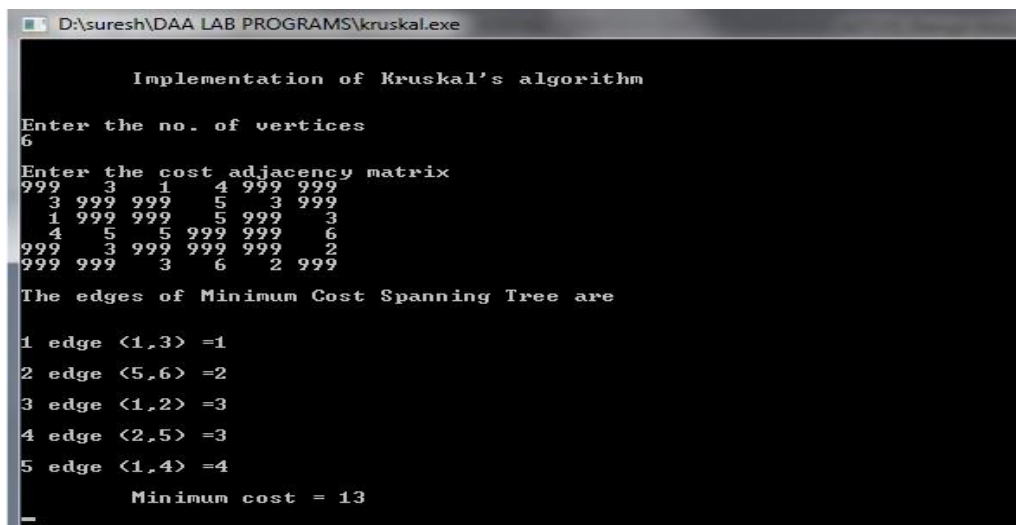
    }
    printf("\nThe edges of Minimum Cost Spanning Tree are\n\n");
    while(ne<n) {
        for(i=1,min=999;i<=n;i++) {
            for(j=1;j<=n;j++) {
                if(cost[i][j]<min) {
                    min=cost[i][j];
                    a=u=i;
                    b=v=j;
                }
            }
        }
        u=find(u);
        v=find(v);
        if(uni(u,v)) {
            printf("\n%d edge (%d,%d) =%d\n",ne++,a,b,min);
            mincost +=min;
        }
        cost[a][b]=cost[b][a]=999;
    }
    printf("\n\tMinimum cost = %d\n",mincost);
}

int find(int i) {
    while(parent[i])
        i=parent[i];
    return i;
}

int uni(inti,int j){
    if(i!=j) {
        parent[j]=i;
        return 1;
    }
    return 0;
}

```

## INPUT/ OUTPUT



```

D:\suresh\DAALAB PROGRAMS\kruskal.exe

Implementation of Kruskal's algorithm

Enter the no. of vertices
6
Enter the cost adjacency matrix
999 3 1 4 999 999
3 999 999 5 3 999
1 999 999 5 999 3
4 5 5 999 999 6
999 3 999 999 999 2
999 999 3 6 2 999

The edges of Minimum Cost Spanning Tree are

1 edge <1,3> =1
2 edge <5,6> =2
3 edge <1,2> =3
4 edge <2,5> =3
5 edge <1,4> =4

Minimum cost = 13

```

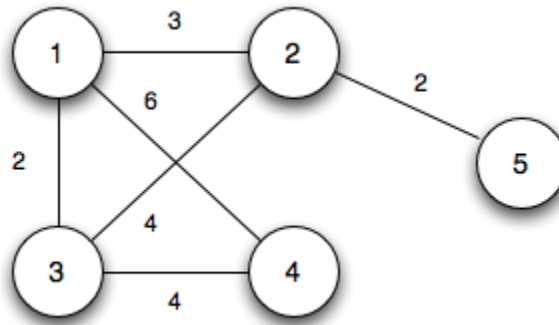
**LAB VIVA QUESTIONS:**

1. What is the time complexity of Kruskal's algorithm.
2. Define spanning tree.
3. Define minimum cost spanning tree.

## MINIMUM COST SPANNING TREE

### OBJECTIVE:

Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.



### PROGRAM LOGIC:

- 1) Create a set  $S$  that keeps track of vertices already included in MST.
- 2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
- 3) While  $S$  doesn't include all vertices.
  - a) Pick a vertex  $u$  which is not there in  $S$  and has minimum key value.
  - b) Include  $u$  to  $S$ .
  - c) Update key value of all adjacent vertices of  $u$ .

To update the key values, iterate through all adjacent vertices. For every adjacent vertex  $v$ , if weight of edge  $u-v$  is less than the previous key value of  $v$ , update the key value as weight of  $u-v$ .

The idea of using key values is to pick the minimum weight edge from cut. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.

### SOURCE CODE.

```
#include<stdio.h>
inta,b,u,v,n,i,j,ne=1
;
int
visited[10]={0},min,mincost=0,cost[10][10]
; void main()
{
    printf("\n Enter the number of
    nodes:"); scanf("%d",&n);
```

```

printf("\n Enter the adjacency
matrix:\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("\n");
while(ne<n)
{
    for(i=1,min=999;i<=n;i++)
        for(j=1;j<=n;j++)
            if(cost[i][j]<min)
                if(visited[i]!=0)
                {
                    min=cost[i][j];
                    a=u=i;
                    b=v=j;
                }
    if(visited[u]==0 || visited[v]==0)
    {
        printf("\n Edge %d:(%d %d)
cost:%d",ne++,a,b,min); mincost+=min;
        visited[b]=1;
    }
    cost[a][b]=cost[b][a]=999;
}
printf("\n Minimum cost=%d",mincost);
}

```

## INPUT/ OUTPUT

```

D:\suresh\DAALAB PROGRAMS\prims.exe
Enter the number of nodes:5
Enter the adjacency matrix:
999 3 2 6 999
3 999 4 999 2
2 4 999 4 999
6 999 4 999 999
999 2 999 999 999

Edge 1:(1 3) cost:2
Edge 2:(1 2) cost:3
Edge 3:(2 5) cost:2
Edge 4:(3 4) cost:4
Minimum cost=11
-----
Process exited after 247.9 seconds with return value 17
Press any key to continue . . .

```

**LAB VIVA QUESTIONS:**

1. What is Minimum Cost spanning Tree.
2. Explain Prim's ALGORITHM.
3. What is time complexity of Prim's algorithm.

## STRASSEN'S MATRIX MULTIPLICATION

```
#include<stdio.h>
int main(){
    int a[2][2],b[2][2],c[2][2],i,j;
    int m1,m2,m3,m4,m5,m6,m7;

    printf("Enter the 4 elements of first matrix: ");
    for(i=0;i<2;i++)
        for(j=0;j<2;j++)
            scanf("%d",&a[i][j]);

    printf("Enter the 4 elements of second matrix: ");
    for(i=0;i<2;i++)
        for(j=0;j<2;j++)
            scanf("%d",&b[i][j]);

    printf("\nThe first matrix is\n");
    for(i=0;i<2;i++){
        printf("\n");
        for(j=0;j<2;j++)
            printf("%d\t",a[i][j]);
    }

    printf("\nThe second matrix is\n");
    for(i=0;i<2;i++){
        printf("\n");
        for(j=0;j<2;j++)
            printf("%d\t",b[i][j]);
    }

    m1= (a[0][0] + a[1][1])*(b[0][0]+b[1][1]);
    m2= (a[1][0]+a[1][1])*b[0][0];
    m3= a[0][0]*(b[0][1]-b[1][1]);
    m4= a[1][1]*(b[1][0]-b[0][0]);
    m5= (a[0][0]+a[0][1])*b[1][1];
    m6= (a[1][0]-a[0][0])*(b[0][0]+b[0][1]);
    m7= (a[0][1]-a[1][1])*(b[1][0]+b[1][1]);

    c[0][0]=m1+m4-m5+m7;
    c[0][1]=m3+m5;
    c[1][0]=m2+m4;
    c[1][1]=m1-m2+m3+m6;

    printf("\nAfter multiplication using \n");
    for(i=0;i<2;i++){
        printf("\n");
        for(j=0;j<2;j++)
```

```
        printf("%d\t",c[i][j]);  
    }  
  
    return 0;  
}
```

Sample output:

Enter the 4 elements of first matrix: 1

2

3

4

Enter the 4 elements of second matrix: 5

6

7

8

The first matrix is

1    2

3    4

The second matrix is

5    6

7    8

After multiplication using

19   22

43   50

## JOB SEQUENCING WITH DEADLINES

```
#include <stdio.h>

#define MAX 100

typedef struct Job {
    char id[5];
    int deadline;
    int profit;
} Job;

void jobSequencingWithDeadline(Job jobs[], int n);

int minValue(int x, int y) {
    if(x < y) return x;
    return y;
}

int main(void) {
    //variables
    int i, j;

    //jobs with deadline and profit
    Job jobs[5] = {
```

```
{ "j1", 2, 60},  
{ "j2", 1, 100},  
{ "j3", 3, 20},  
{ "j4", 2, 40},  
{ "j5", 1, 20},  
};  
  
//temp  
Job temp;  
  
//number of jobs  
int n = 5;  
  
//sort the jobs profit wise in descending order  
for(i = 1; i < n; i++) {  
    for(j = 0; j < n - i; j++) {  
        if(jobs[j+1].profit > jobs[j].profit) {  
            temp = jobs[j+1];  
            jobs[j+1] = jobs[j];  
            jobs[j] = temp;  
        }  
    }  
}  
  
printf("%10s %10s %10s\n", "Job", "Deadline", "Profit");
```

```

for(i = 0; i < n; i++) {
    printf("%10s %10i %10i\n", jobs[i].id, jobs[i].deadline, jobs[i].profit);
}

jobSequencingWithDeadline(jobs, n);

return 0;
}

void jobSequencingWithDeadline(Job jobs[], int n) {
    //variables
    int i, j, k, maxprofit;

    //free time slots
    int timeslot[MAX];

    //filled time slots
    int filledTimeSlot = 0;

    //find max deadline value
    int dmax = 0;
    for(i = 0; i < n; i++) {
        if(jobs[i].deadline > dmax) {
            dmax = jobs[i].deadline;
        }
    }
}

```

```

}

//free time slots initially set to -1 [-1 denotes EMPTY]

for(i = 1; i <= dmax; i++) {
    timeslot[i] = -1;
}

printf("dmax: %d\n", dmax);

for(i = 1; i <= n; i++) {
    k = minValue(dmax, jobs[i - 1].deadline);
    while(k >= 1) {
        if(timeslot[k] == -1) {
            timeslot[k] = i-1;
            filledTimeSlot++;
            break;
        }
        k--;
    }

    //if all time slots are filled then stop
    if(filledTimeSlot == dmax) {
        break;
    }
}

```

```

//required jobs

printf("\nRequired Jobs: ");

for(i = 1; i <= dmax; i++) {

    printf("%s", jobs[timeslot[i]].id);

    if(i < dmax) {

        printf(" --> ");

    }

}

//required profit

maxprofit = 0;

for(i = 1; i <= dmax; i++) {

    maxprofit += jobs[timeslot[i]].profit;

}

printf("\nMax Profit: %d\n", maxprofit);

}

```

## Output

Job	Deadline	Profit
-----	----------	--------

j2	1	100
----	---	-----

j1	2	60
----	---	----

j4	2	40
----	---	----

j3	3	20
----	---	----

j5	1	20
----	---	----

dmax: 3

Required Jobs: j2 --> j1 --> j3

Max Profit: 180

## KNAPSACK PROBLEM

### OBJECTIVE:

Implement 0/1 Knapsack problem using Dynamic Programming.

### SOURCE CODE:

```
#include<stdio.h>
int w[10],p[10],v[10][10],n,i,j,cap,x[10]={0};
int max(int i,int j){
    return ((i>j)?i:j);
}
int knap(int i,int j) {
    int value;
    if(v[i][j]<0) {
        if(j<w[i])
            value=knap(i-1,j);
        else
            value=max(knap(i-1,j),p[i]+knap(i-1,j-w[i]));
        v[i][j]=value;
    }
    return(v[i][j]);
}
int main() {
    int profit,count=0;
    printf("\nEnter the number of objects ");
    scanf("%d",&n);
    printf("Enter the profit and weights of the elements \n ");
    for(i=1;i<=n;i++) {
        printf("\nEnter profit and weight For object no %d :",i);
        scanf("%d%d",&p[i],&w[i]);
    }
    printf("\nEnter the capacity ");
    scanf("%d",&cap);
    for(i=0;i<=n;i++)
        for(j=0;j<=cap;j++)
            if((i==0)||j==0)
                v[i][j]=0;
            else
                v[i][j]=-1;
    profit=knap(n,cap);
    i=n;
    j=cap;
    while(j!=0&& i!=0) {
        if(v[i][j]!=v[i-1][j]) {
            x[i]=1;
            j=j-w[i];
            i--;
        }
    }
}
```

```

        }
        else
            i--;
    }
    printf("object included are \n ");
    printf("Sl.no\tweight\tprofit\n");
    for(i=1;i<=n;i++)
        if(x[i])
            printf("%d\t%d\t%d\n",++count,w[i],p[i]);
    printf("Total profit = %d\n",profit);
}

```

## INPUT/ OUTPUT

```

C:\Users\Administrator\Desktop\suresh\DAAB LAB PROGRAMS\KNAPSACK .exe
Enter the number of objects 3
Enter the profit and weights of the elements
Enter profit and weight For object no 1 :1 2
Enter profit and weight For object no 2 :2 3
Enter profit and weight For object no 3 :5 4
Enter the capacity 6
object included are
Sl.no  weight  profit
1       2       1
2       4       5
Total profit = 6

-----
Process exited after 24.4 seconds with return value 0
Press any key to continue . . .

```

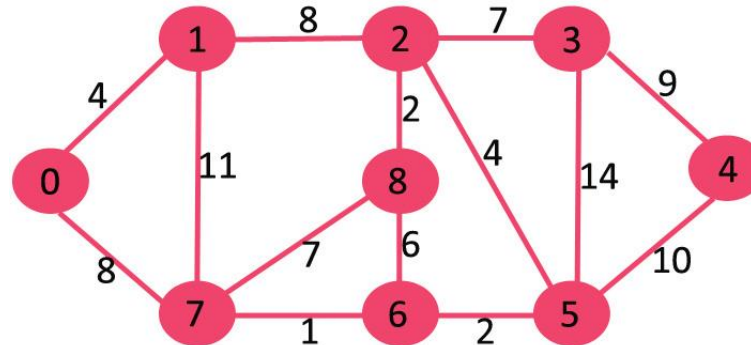
## LAB VIVA QUESTIONS:

1. Define knapsack problem.
2. Define principle of optimality.
3. What is the optimal solution for knapsack problem?
4. What is the time complexity of knapsack problem?

## SHORTEST PATHS ALGORITHM

### OBJECTIVE:

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



### PROGRAM LOGIC:

- 1) Create a set  $S$  that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.
- 2) Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.
- 3) While  $S$  doesn't include all vertices
  - a) Pick a vertex  $u$  which is not there in  $S$  and has minimum distance value.
  - b) Include  $u$  to  $S$ .
  - c) Update distance value of all adjacent vertices of  $u$ .

To update the distance values, iterate through all adjacent vertices. For every adjacent vertex  $v$ , if sum of distance value of  $u$  (from source) and weight of edge  $u-v$ , is less than the distance value of  $v$ , then update the distance value of  $v$ .

### SOURCE CODE:

```
#include<stdio.h>
#define infinity 999
void dij(int n, int v, int cost[20][20], int dist[]) {
    int i, u, count, w, flag[20], min;
    for(i=1; i<=n; i++)
        flag[i]=0, dist[i]=cost[v][i];
    count=2;
    while(count<=n) {
        min=99;
        for(w=1; w<=n; w++)
            if(dist[w]<min && !flag[w]) {
                min=dist[w];
                u=w;
            }
        flag[u]=1;
        for(i=1; i<=n; i++)
            if(flag[i]==0)
                dist[i]=min+cost[u][i];
        count++;
    }
}
```

```

        flag[u]=1;
        count++;
        for(w=1;w<=n;w++)
            if((dist[u]+cost[u][w]<dist[w]) && !flag[w])
                dist[w]=dist[u]+cost[u][w];
    }
}
int main() {
    int n,v,i,j,cost[20][20],dist[20];
    printf("enter the number of nodes:");
    scanf("%d",&n);
    printf("\n enter the cost matrix:\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]=infinity;
        }
    printf("\n enter the source matrix:");
    scanf("%d",&v);
    dij(n,v,cost,dist);
    printf("\n shortest path : \n");
    for(i=1;i<=n;i++)
        if(i!=v)
            printf("%d->%d,cost=%d\n",v,i,dist[i]);
}

```

## INPUT/ OUTPUT

```

C:\Users\Administrator\Desktop\dijkstra.exe
enter the number of nodes:9

enter the cost matrix:
0 4 0 0 0 0 0 8 0
4 0 8 0 0 0 0 11 0
0 8 0 7 0 4 0 0 2
0 0 7 0 9 14 0 0 0
0 0 0 9 0 10 0 0 0
0 0 4 14 10 0 2 0 0
0 0 0 0 0 2 0 1 6
8 11 0 0 0 0 1 0 7
0 0 2 0 0 0 6 7 0

enter the source matrix:1

shortest path :
1->2,cost=4
1->3,cost=12
1->4,cost=19
1->5,cost=21
1->6,cost=11
1->7,cost=9
1->8,cost=8
1->9,cost=14

-----
Process exited after 148.6 seconds with return value 9
Press any key to continue . . . _

```

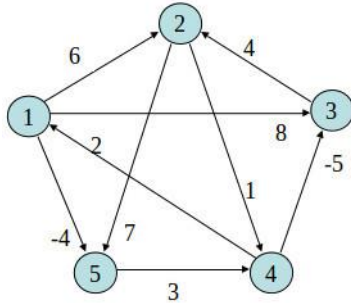
**LAB VIVA QUESTIONS:**

1. What is the time complexity of Dijkstra's algorithm?
2. Define cost matrix.
3. Define directed graph.
4. Define connected graph.

## ALL PAIRS SHORTEST PATHS

### OBJECTIVE:

Implement All-Pairs Shortest Paths Problem using Floyd's algorithm.



	1	2	3	4	5
1	0	6	8	$\infty$	-4
2	$\infty$	0	$\infty$	1	7
3	$\infty$	4	0	$\infty$	$\infty$
4	2	$\infty$	-5	0	$\infty$
5	$\infty$	$\infty$	$\infty$	3	0

### PROGRAM LOGIC:

Initialize the solution matrix same as the input graph matrix as a first step. Then we update the solution matrix by considering all vertices as an intermediate vertex. The idea is to one by one pick all vertices and update all shortest paths which include the picked vertex as an intermediate vertex in the shortest path.

When we pick vertex number  $k$  as an intermediate vertex, we already have considered vertices  $\{0, 1, 2, \dots, k-1\}$  as intermediate vertices.

For every pair  $(i, j)$  of source and destination vertices respectively, there are two possible cases.

- 1)  $k$  is not an intermediate vertex in shortest path from  $i$  to  $j$ . We keep the value of  $\text{dist}[i][j]$  as it is.
- 2)  $k$  is an intermediate vertex in shortest path from  $i$  to  $j$ . We update the value of  $\text{dist}[i][j]$  as  $\text{dist}[i][k] + \text{dist}[k][j]$ .

### SOURCE CODE.

```
#include<stdio.h>
int min(int,int);
void floyds(int p[10][10],int n) {
    int i,j,k;
    for(k=1;k<=n;k++)
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                if(i==j)
                    p[i][j]=0;
                else
                    p[i][j]=min(p[i][j],p[i][k]+p[k][j]);
}
int min(int a,int b) {
```

```

        if(a<b)
            return(a);
        else
            return(b);
    }
    main() {
        int p[10][10],w,n,e,u,v,i,j;
        printf("\n Enter the number of vertices:");
        scanf("%d",&n);
        printf("\n Enter the number of edges:\n");
        scanf("%d",&e);
        for(i=1;i<=n;i++){
            for(j=1;j<=n;j++)
                p[i][j]=999;
        }
        for(i=1;i<=e;i++) {
            printf("\n Enter the end vertices of edge%d with its weight \n",i);
            scanf("%d%d%d",&u,&v,&w);
            p[u][v]=w;
        }
        printf("\n Matrix of input data:\n");
        for(i=1;i<=n;i++) {
            for(j=1;j<=n;j++)
                printf("%d \t",p[i][j]);
            printf("\n");
        }
        floyds(p,n);
        printf("\n Transitive closure:\n");
        for(i=1;i<=n;i++) {
            for(j=1;j<=n;j++)
                printf("%d \t",p[i][j]);
            printf("\n");
        }
        printf("\n The shortest paths are:\n");
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++){
                if(i!=j)
                    printf("\n <%d,%d>=%d",i,j,p[i][j]);
            }
    }
}

```

## INPUT/ OUTPUT

```
D:\suresh\DAALAB PROGRAMS\allpairs.exe

Enter the number of vertices:5
Enter the number of edges:
9
Enter the end vertices of edge1 with its weight
1 2 6
Enter the end vertices of edge2 with its weight
1 3 8
Enter the end vertices of edge3 with its weight
1 5 -4
Enter the end vertices of edge4 with its weight
2 4 1
Enter the end vertices of edge5 with its weight
2 5 7
Enter the end vertices of edge6 with its weight
3 2 4
Enter the end vertices of edge7 with its weight
4 1 2
Enter the end vertices of edge8 with its weight
4 3 -5
Enter the end vertices of edge9 with its weight
5 4 3

Matrix of input data:
999    6    8    999    -4
999    999    999    1    7
999    4    999    999    999
2    999    -5    999    999
999    999    999    3    999

Transitive closure:
0    -2    -6    -1    -4
3    0    -4    1    -1
7    4    0    5    3
2    -1    -5    0    -2
5    2    -2    3    0

The shortest paths are:
<1,2>=-2
<1,3>=-6
<1,4>=-1
<1,5>=-4
<2,1>=3
<2,3>=-4
<2,4>=1
<2,5>=-1
<3,1>=7
<3,2>=4
<3,4>=5
<3,5>=3
<4,1>=2
<4,2>=-1
<4,3>=-5
<4,5>=-2
<5,1>=5
<5,2>=2
<5,3>=-2
<5,4>=3

-----
Process exited after 91.57 seconds with return value 5
Press any key to continue . . .
```

## LAB VIVA QUESTIONS:

1. What is Floyd's algorithm?
2. What is the time complexity of Floyd's algorithm?
3. Define Distance Matrix.

## TRAVELLING SALES PERSON PROBLEM

### OBJECTIVE:

Implement any scheme to find the optimal solution for the Traveling Sales Person problem and then solve the same problem instance using any approximation algorithm and determine the error in the approximation

### PROGRAM LOGIC:

1. Check for the disconnection between the current city and the next city
2. Check whether the travelling sales person has visited all the cities
3. Find the next city to be visited
4. Find the solution and terminate

### SOURCE CODE:

```
#include<stdio.h>
int s,c[100][100],ver;
float optimum=999,sum;
/* function to swap array elements */
void swap(int v[], int i, int j) {
    int t;
    t = v[i];
    v[i] = v[j];
    v[j] = t;
}
/* recursive function to generate permutations */
void brute_force(int v[], int n, int i) {
    // this function generates the permutations of the array from element i to element
    n-1
    int j,sum1,k;
    //if we are at the end of the array, we have one permutation
    if (i == n) {
        if(v[0]==s) {
            for (j=0; j<n; j++)
                printf ("%d ", v[j]);
            sum1=0;
            for( k=0;k<n-1;k++) {
                sum1=sum1+c[v[k]][v[k+1]];
            }
            sum1=sum1+c[v[n-1]][s];
            printf("sum = %d\n",sum1);
            if (sum1<optimum)
                optimum=sum1;
        }
    }
}
```

```

        }
    }
    else
    // recursively explore the permutations starting at index i going through index n-
    1*/
        for (j=i; j<n; j++) { /* try the array with i and j switched */
            swap (v, i, j);
            brute_force (v, n, i+1);
            /* swap them back the way they were */
            swap (v, i, j);
        }
    }

void nearest_neighbour(intver) {
    int min,p,i,j,vis[20],from;
    for(i=1;i<=ver;i++)
        vis[i]=0;
    vis[s]=1;
    from=s;
    sum=0;
    for(j=1;j<ver;j++) {
        min=999;
        for(i=1;i<=ver;i++)
            if(vis[i] !=1 && c[from][i]<min && c[from][i] !=0 ) {
                min= c[from][i];
                p=i;
            }
        vis[p]=1;
        from=p;
        sum=sum+min;
    }
    sum=sum+c[from][s];
}

void main () {
    int ver,v[100],i,j;
    printf("Enter n : ");
    scanf("%d",&ver);
    for (i=0; i<ver; i++)
        v[i] = i+1;
    printf("Enter cost matrix\n");
    for(i=1;i<=ver;i++)
        for(j=1;j<=ver;j++)
            scanf("%d",&c[i][j]);
    printf("\nEnter source : ");
    scanf("%d",&s);
    brute_force (v, ver, 0);
    printf("\nOptimum solution with brute force technique is=%f\n",optimum);
    nearest_neighbour(ver);
    printf("\nSolution with nearest neighbour technique is=%f\n",sum);
}

```

```

        printf("The approximation val is=%f",((sum/optimum)-1)*100);
        printf(" % ");
    }

```

### INPUT/ OUTPUT

```

D:\suresh\DAA LAB PROGRAMS\traveling.exe
Enter n : 4
Enter cost matrix
0 10 15 20
5 0 9 10
6 13 0 12
8 8 9 0

Enter source : 1
1 2 3 4 sum = 39
1 2 4 3 sum = 35
1 3 2 4 sum = 46
1 3 4 2 sum = 40
1 4 3 2 sum = 47
1 4 2 3 sum = 43

Optimum solution with brute force technique is=35.000000
Solution with nearest neighbour technique is=39.000000
The approximation val is=11.428571
-----
Process exited after 59.19 seconds with return value 1
Press any key to continue . . . _

```

### LAB VIVA QUESTIONS:

1. Define Optimal Solution.
2. Explain Travelling Sales Person Problem.
3. What is the time complexity of Travelling Sales Person Problem?

# N QUEENS PROBLEM

## OBJECTIVE:

Implement N Queen's problem using Back Tracking.

## PROGRAM LOGIC:

- 1) Start in the leftmost column
- 2) If all queens are placed return true
- 3) Try all rows in the current column. Do following for every tried row.
  - a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
  - b) If placing queen in [row, column] leads to a solution then return true.
  - c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.
- 3) If all rows have been tried and nothing worked, return false to trigger Backtracking.

## SOURCE CODE:

```
#include<stdio.h>
#include<math.h>
int a[30],count=0;
int place(intpos) {
    int i;
    for(i=1;i<pos;i++) {
        if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))
            return 0;
    }
    return 1;
}
void print_sol(int n) {
    inti,j; count++;
    printf("\n\nSolution #d:\n",count);
    for(i=1;i<=n;i++) {
        for(j=1;j<=n;j++) {
            if(a[i]==j)
                printf("Q\t");
            else
                printf("*\t");
        }
        printf("\n");
    }
}
void queen(int n) {
    int k=1;
    a[k]=0;
    while(k!=0) {
```

```

        a[k]=a[k]+1;
    while((a[k]<=n)&&!place(k))
        a[k]++;
    if(a[k]<=n) {
        if(k==n)
            print_sol(n);
        else {
            k++;
            a[k]=0;
        }
    }
    else
        k--;
}
}

void main() {
    int i,n;
    printf("Enter the number of Queens\n");
    scanf("%d",&n);
    queen(n);
    printf("\nTotal solutions=%d",count);
}

```

## INPUT/ OUTPUT

```

D:\suresh\DAALAB PROGRAMS\nqueens.exe
Enter the number of Queens
4

Solution #1:
*   Q   *   *
*   *   *   Q
Q   *   *   *
*   *   Q   *

Solution #2:
*   *   Q   *
Q   *   *   *
*   *   *   Q
*   Q   *   *

Total solutions=2
-----
Process exited after 1.669 seconds with return value 18
Press any key to continue . . .

```

## LAB VIVA QUESTIONS:

1. Define backtracking.
2. Define live node, dead node.
3. Define implicit and explicit constraints.
4. What is the time complexity of n-queens problem.

## SUM OF SUB SETS PROBLEM

### OBJECTIVE:

Find a subset of a given set  $S = \{s_1, s_2, \dots, s_n\}$  of  $n$  positive integers whose sum is equal to a given positive integer  $d$ . For example, if  $S = \{1, 2, 5, 6, 8\}$  and  $d = 9$  there are two solutions  $\{1, 2, 6\}$  and  $\{1, 8\}$ . A suitable message is to be displayed if the given problem instance doesn't have a solution.

### PROGRAM LOGIC:

Given a set of non-negative integers, and a value *sum*, determine if there is a subset of the given set with sum equal to given *sum*.

### SOURCE CODE:

```
#include<stdio.h>
#define TRUE 1
#define FALSE 0
int inc[50],w[50],sum,n;
void sumset(int ,int ,int);
int promising(inti,intwt,int total) {
    return (((wt+total)>=sum)&&((wt==sum)||((wt+w[i+1]<=sum))));
}
void main() {
    inti,j,n,temp,total=0;
    printf("\n Enter how many numbers: ");
    scanf("%d",&n);
    printf("\n Enter %d numbers : ",n);
    for (i=0;i<n;i++) {
        scanf("%d",&w[i]);
        total+=w[i];
    }
    printf("\n Input the sum value to create sub set: ");
    scanf("%d",&sum);
    for (i=0;i<=n;i++)
        for (j=0;j<n-1;j++)
            if(w[j]>w[j+1]) {
                temp=w[j];
                w[j]=w[j+1];
                w[j+1]=temp;
            }
    printf("\n The given %d numbers in ascending order: ",n);
    for (i=0;i<n;i++)
        printf("%3d",w[i]);
    if((total<sum))
        printf("\n Subset construction is not possible");
    else {
```

```

        for (i=0;i<n;i++)
            inc[i]=0;
        printf("\n The solution using backtracking is:\n");
        sumset(-1,0,total);
    }

}

void sumset(int i,int wt, int total) {
    int j;
    if(promising(i,wt,total))    {
        if(wt==sum) {
            printf("\n{ ");
            for (j=0;j<=i;j++)
                if(inc[j])
                    printf("%3d",w[j]);
            printf(" }\n");
        } else {
            inc[i+1]=TRUE;
            sumset(i+1,wt+w[i+1],total-w[i+1]);
            inc[i+1]=FALSE;
            sumset(i+1,wt,total-w[i+1]);
        }
    }
}
}

```

## INPUT/ OUTPUT

```

Enter how many numbers: 5
Enter 5 numbers : 1 2 5 6 8
Input the sum value to create sub set: 9
The given 5 numbers in ascending order: 1 2 5 6 8
The solution using backtracking is:
< 1 2 6 >
< 1 8 >

```

## LAB VIVA QUESTIONS:

1. Define is Back-Tracking.
2. Explain Sum of subset problem.
3. What is time complexity of sum of subset problem?