NEW HORIZON
COLLEGE OF ENGINEERING

Design and Analysis of Algorithms Laboratory Manual

Department of Information Science and Engineering

Course Code:22ISL52

Semester: V 2024-25

Prepared By	Approved By	Authorized By
Mrs. Neha Jadhav **Mrs. Thamari Selvi**	**Mrs. Divya K V**	**Dr.Vandana CP**

New HorizonCollegeofEngineering

VISION

To emerge as an institute of eminence in the fields of engineering, technology and management in serving the industry and the nation by empowering students with a high degree of technical, managerial and practical competence.

MISSION

1. To strengthen the theoretical, practical and ethical dimensions of the learning process by fostering a culture of research and innovation among faculty members and students.
2. To encourage long-term interaction between the academia and industry through their involvement in the design of curriculum and its hands-on implementation.
3. To strengthen and mould students in professional, ethical, social and environmental dimensions by encouraging participation in co-curricular and extracurricular activities.

Department of Information Science & Engineering

VISION

To emerge as a Department of eminence in Information Science and Engineering in serving the Information Technology industry and the nation by empowering students with a high degree of technical and practical competence.

MISSION

1. To strengthen the theoretical, practical and ethical dimensions of the learning process by continuous learning and establishing a culture of research and innovation among faculty members and students, in the field of Information Science and Engineering.
2. To build long-term interaction between the academia and Information Technology industry, through their involvement in the design of curriculum and its hands-on implementation.
3. To strengthen and mould students in professional, ethical, social and environmental dimensions by encouraging participation in co-curricular and extracurricular activities

PROGRAMEDUCATIONAL OBJECTIVES

PEO1: To excel as Information Science Engineers with ability to solve wide range of computational problems in IT industry, government or other work environments.

PEO2: To pursue higher studies with profound knowledge enriched with academia and industrial skill sets.

PEO3: To exhibit adaptive skills to develop computing systems using modern tools and technologies in multi-disciplinary areas to meet technical and managerial challenges, which meet societal requirements.

PEO4: To possess the ability to collaborate as a team a member and leader with professional ethics to make a positive impact on society.

DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY

CourseCode: 22ISL52	**Credits : 01**
L:T:P:S 0:0:1:0	**CIE Marks : 50**
ExamHours:3	**SEE Marks : 50**

Course Outcomes: At the end of the Course,the Student will be able to:

22ISL52 .1	Examine the problems using brute force, divide and conquer and decrease and conquer techniques.
22ISL52 .2	Analyze the problems using greedy and dynamic programming techniques.
22ISL52 .3	Investigate the problems using backtracking and online approaches.
22ISL52 .4	Analyze the different string-matching algorithms

Mapping of Course Outcomes to Program Outcomes:

CO/PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
22ISL52 .1	3	3	3	2	3	-	-	-	-	-	-	1	3	3
22ISL52 .2	3	3	3	2	3	-	-	-	-	-	-	1	3	3
22ISL52 .3	3	3	3	2	3	-	-	-	-	-	-	1	3	3
22ISL52 .4	3	3	3	2	3	-	-	-	-	-	-	1	3	3

Exp. No.	**Experiment**
PART-A	
1	Implement and analyze quick sort algorithm.
2	Implement and analyze merge sort algorithm
3	Implement and analyze topological sorting in a given directed graph.
4	Implement and analyze Kruskal`s algorithm and find minimum cost Spanning tree of a given connected undirected graph.
5	Implement and analyze Prim`s algorithm and find minimum cost spanning Tree of a given connected undirected graph.
6	Implement and analyze Dijkstra’s algorithm to find the shortest path from a given source.
PART-B	
7	Implement travelling salesman problem using dynamic programming.
8	Implement 0/1 Knapsack problem.

9	Implement N-Queens problem using backtracking.
10	Implement sum of sub set problem using backtracking.
11	Implement and compare Simple string matching and KMP string matching Algorithm.
12	Implement and analyze k-server problem.

For SEE Examination:

- One experiment from part A & One experiment from part B to be given
- Examination will be conducted for 50 marks
- Marks Distribution: Procedure write-up –20%
  Conduction – 60%
  Viva– Voce – 20%
- Change of the experiment is allowed only once and procedure write-up marks will be considered as '0'

CIE- Continuous Internal Evaluation (50Marks)

Bloom's Category	**Tests(20Marks)**	**Weekly Assessment**
Remember	-	-
Understand	-	5
Apply	5	10
Analyze	5	10
Evaluate	10	5
Create	-	-

SEE–Semester End Examination (50Marks)

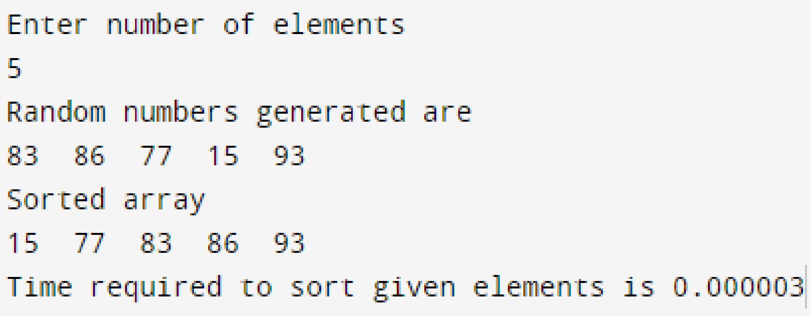
Bloom's Taxonomy	**Marks**
Remember	-
Understand	-
Apply	20
Analyze	20
Evaluate	10
Create	-

PART-A

Program1: Implement and analyze quicksort algorithm.

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int partition(inta[], intlow, inthigh)
{
    int pivot=a[low], i=low,
    j=high+1;int temp;
    while(i<j)
    {
            do
            {
                i++;
            }while(pivot>=a[i]
            &&i<high);do
            {
                j--;
            }while(pivot<a[j]);
            if(i<j)
            {
               temp=a[i];
               a[i]=a[j];a
               [j]=temp;
            }
    }
    a[low]=a[j];
    a[j]=pivot;r
    eturnj;
}
voidquick_sort(inta[],intlow,int high)
{
    int s;
    if(low<high)
    {
            s=partition(a,low,high);
            quick_sort(a,low,s-1);

            quick_sort(a,s+1,high);
    }
}
int main()
{
    int
    a[10000],n,low,high,i;clo
    ck_tst,end;
    printf("Enter number of elements\n");
    scanf("%d",&n);
    printf("Random numbers generated are\n");
    for(i=0;i<n;i++)
    {
            a[i]=rand()%100;

            printf("%d\t",a[i]);
    }
    low=0;

    high=n-1;

    st=clock();
    quick_sort(a,low,high);
    end=clock();
    printf("\nSorted array\n");
    for(i=0;i<n;i++)
    {
            printf("%d\t",a[i]);
    }
    printf("\nTimerequiredtosortgiven elementsis%f",(float)(end-st)/CLOCKS_PER_SEC);
}
```

OUTPUT



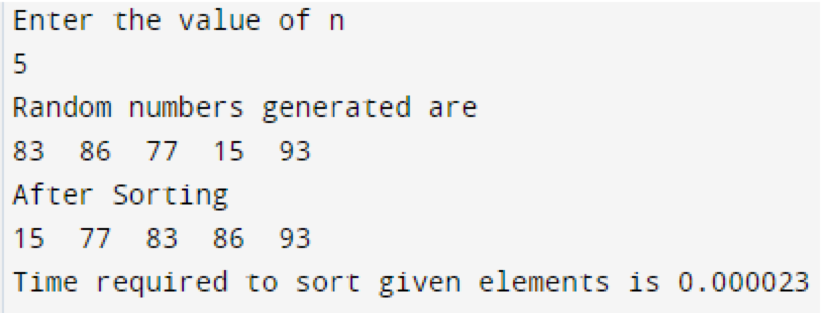
```
Enter number of elements
5
Random numbers generated are
83  86  77  15  93
Sorted array
15  77  83  86  93
Time required to sort given elements is 0.000003
```

Program2:Implement and analyze mergesort algorithm.

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void simple_merge (inta[],intlow,intmid,inthigh)
{
   inti=low,j=mid+1,k=low,c[10000];
   while(i<=mid&&j<=high)
   {
         if(a[i]<a[j])
         {
            c[k]=a[i];
            i++;
            k++;
         }
         else
         {
            c[k]=a[j];
            j++;
            k++;
         }
   }
   while(i<=mid)
         c[k++]=a[i++];
   while(j<=high)
         c[k++]=a[j++];
   for(i=low;i<=high;i++)
         a[i]=c[i];
}
void merge_sort (inta[],intlow,inthigh)
{
   int mid;
   if(low<high)
   {
         mid=(low+high)/2;
```

```
          merge_sort(a,low,mid);
          merge_sort(a,mid+1,high);
          simple_merge(a,low,mid,high);
    }
}
int main()
{
    int a[10000],i=0,n;
    clock_tst,end;
    printf("Enter the value of n\n");
    scanf("%d",&n);
    printf("Random numbers generated are\n");
    for(i=0;i<n;i++)
    {
          a[i]=rand()%100;
          printf("%d\t",a[i]);
    }
    st=clock();merge_sort(a,0,
    n-1);
    end=clock();
    printf("\nAfter Sorting\n");
    for(i=0;i<n;i++)
    printf("%d\t",a[i]);
    printf("\nTimerequiredtosortgiven elementsis%f",(float)(end-st)/CLOCKS_PER_SEC);
}
```

OUTPUT

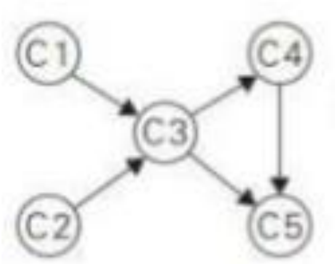


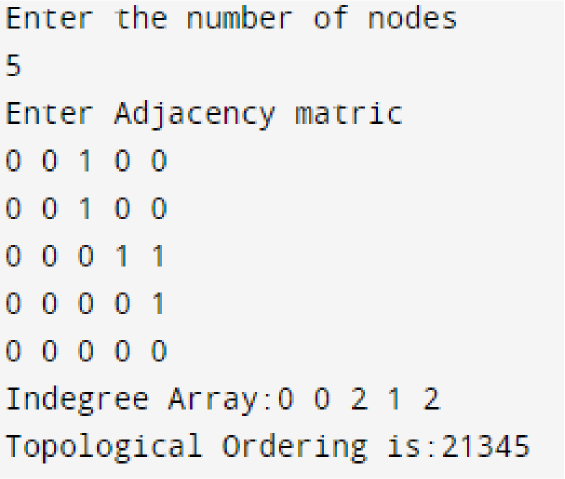
```
Enter the value of n
5
Random numbers generated are
83  86  77  15  93
After Sorting
15  77  83  86  93
Time required to sort given elements is 0.000023
```

Program3: Implement and analyze topological sorting in a given directed graph.

```
#include<stdio.h>
void ts(int a[20][20], int n)
{
 int t[10],vis[10],stack[10],i,j,indeg[10],top=0,ele,k=1;
 for(i=1;i<=n;i++)
 {
   t[i]=0;
   vis[i]=0;
   indeg[i]=0;
 }
 for(i=1;i<=n;i++)
 {
   for(j=1;j<=n;j++)
   {
       if(a[i][j]==1)
       {
           indeg[j]=indeg[j]+1;
      }
   }
 }
 printf("Indegree Array:");
 for(i=1;i<=n;i++)
   printf("%d ",indeg[i]);
   for(i=1;i<=n;i++)
 {
   if(indeg[i]==0)
   {
       stack[++top]=i;
       vis[i]=1;
    }
 }
 while(top>0)
 {
   ele=stack[top--
   ];t[k++]=ele;
   for(j=1;j<=n;j++)
   {
      if(a[ele][j]==1&&vis[j]==0)
      {
         indeg[j]=indeg[j]-1;
         if(indeg[j]==0)
         {
            stack[++top]=j;
            vis[j]=1;
         }
      }
   }
 }
 printf("\nTopological Ordering is:");
 for(i=1;i<=n;i++)
   printf("%d",t[i]);
}
int main()
{
 int n,a[20][20],i,j;
 printf("Enter the number of nodes\n");
 scanf("%d",&n);
 printf("Enter Adjacency matric\n");
 for(i=1;i<=n;i++)
 {
   for(j=1;j<=n;j++)
   {
       scanf("%d",&a[i][j]);
   }
 }
 ts(a,n);
}
```

OUTPUT





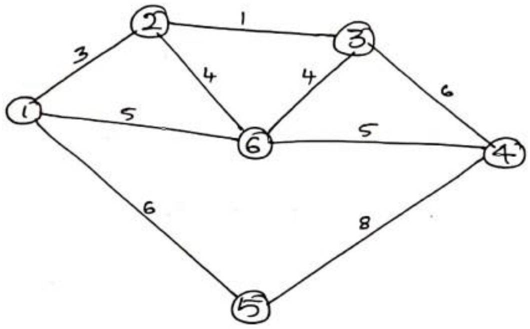
```
Enter the number of nodes
5
Enter Adjacency matric
0 0 1 0 0
0 0 1 0 0
0 0 0 1 1
0 0 0 0 1
0 0 0 0 0
Indegree Array:0 0 2 1 2
Topological Ordering is:21345
```

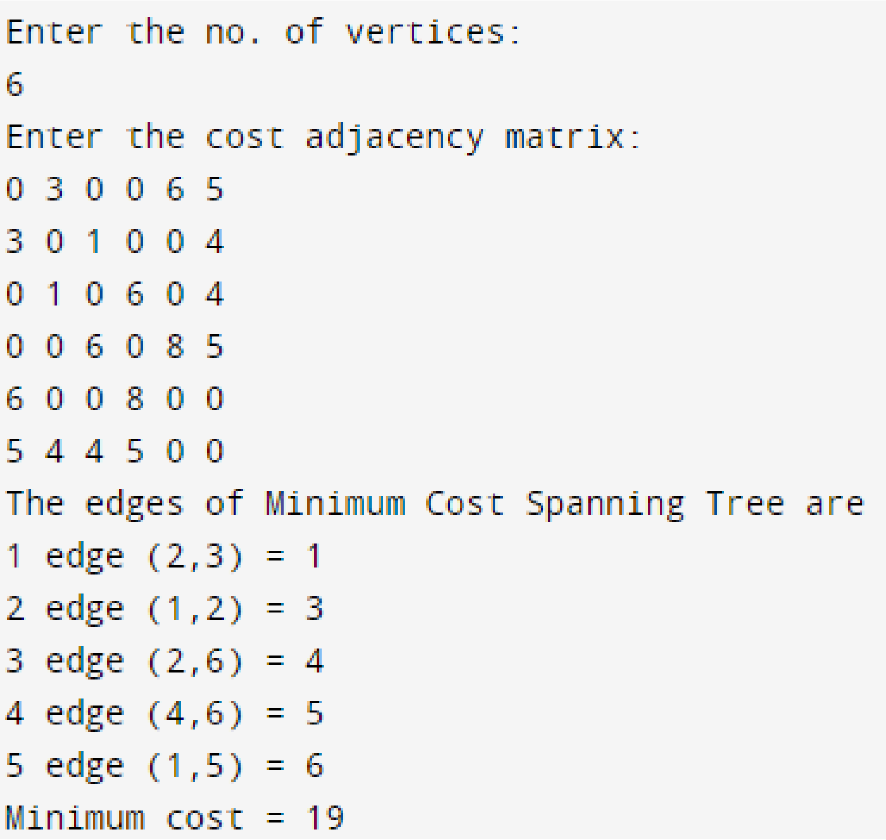
Program4:Implement and analyze Kruskal`s algorithm and find minimum cost spanning tree of a given connected undirected graph.

```
#include<stdio.h>
int i,j,k,a,b,u,v,n,ne=1;
int min, mincost=0, cost[9][9],
parent[9] ;

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

   return i;
}
int uni(int i, int j)
{
   if(i!=j)
   {
      parent[j]=i;
         return1;
   }
   return0;
}
int main()
{
   printf("Enter the no. of vertices:\n");

   scanf("%d",&n);
   printf("Enter 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;
      }
   }
```

```
    for(i=1;i<=n;i++)
    {
         parent[i]=0;
    }
    printf("The edges of Minimum Cost Spanning Tree are\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);i
        f(uni(u,v))
        {
            printf("%d edge (%d,%d) = %d\n",ne++,a,b,min);

            mincost+=min;
        }
        cost[a][b]=cost[b][a]=999;
    }
    printf("Minimumcost =%d\n",mincost);
}
```

OUTPUT



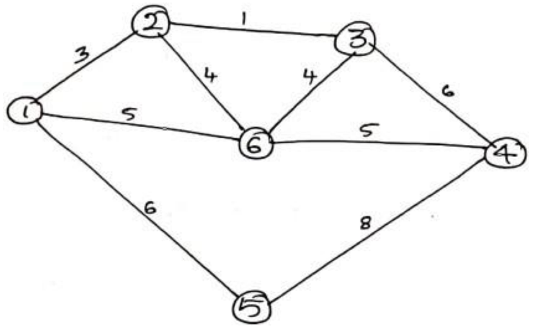


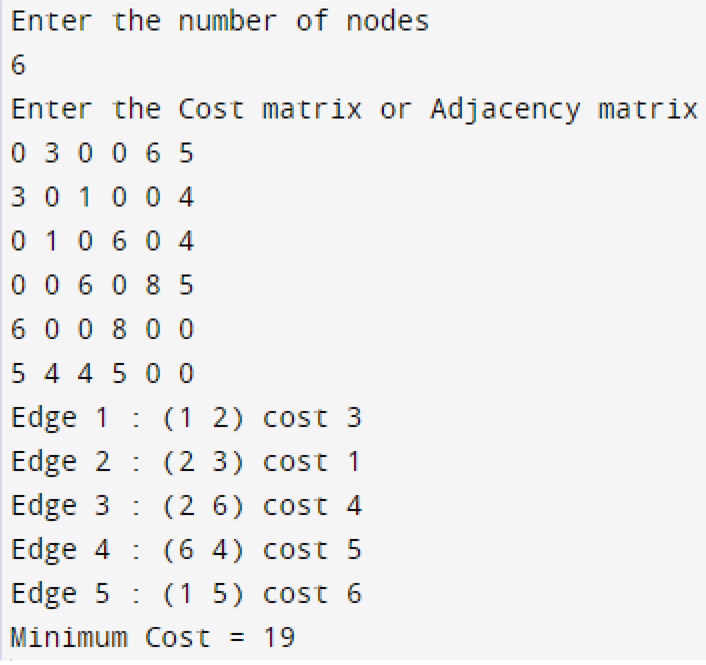
```
Enter the no. of vertices:
6
Enter the cost adjacency matrix:
0 3 0 0 6 5
3 0 1 0 0 4
0 1 0 6 0 4
0 0 6 0 8 5
6 0 0 8 0 0
5 4 4 5 0 0
The edges of Minimum Cost Spanning Tree are
1 edge (2,3) = 1
2 edge (1,2) = 3
3 edge (2,6) = 4
4 edge (4,6) = 5
5 edge (1,5) = 6
Minimum cost = 19
```

Program 5: Implement and analyze Prim`s algorithm and find minimum cost spanning tree of a given connected undirected graph.

```
#include<stdio.h>
intmain()
{
    intn,a[20][20],i,j,min,mincost,u,v,ne,vis[20];
    printf("Enter the number of
    nodes\n");scanf("%d",&n);
    for(i=1;i<=n;i++)
        vis[i]=0;
    printf("Enter the Cost matrix or Adjacency
    matrix\n");for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%d",&a[i][j]);
            if(a[i][j]==0)
            {
                a[i][j]=999;
            }
        }
    }
    vis[1]=1;ne
    =1;mincost
    =0;while(ne
    <n)
    {
        for(i=1,min=999;i<=n;i++)
        {
            for(j=1;j<=n;j++)
            {
                if((a[i][j]<min)&&(vis[i]!=0))
                {
                    min=a[i][j];
                    u=i;
                    v=j;
                }
            }
        }
        if(vis[v]==0)
        {
            printf("Edge %d : (%d %d) cost %d\n",
            ne,u,v,a[u][v]);mincost+=a[u][v];
            ne+=1;vi
            s[v]=1;
        }
        a[u][v]=a[v][u]=999;
    }
    printf("MinimumCost=%d\n",mincost);
}
```

OUTPUT





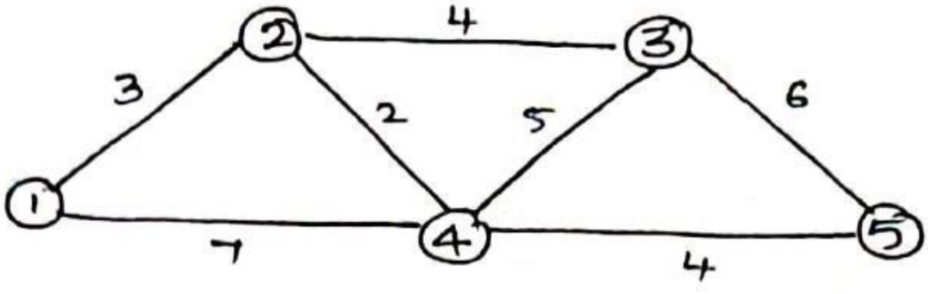
```
Enter the number of nodes
6
Enter the Cost matrix or Adjacency matrix
0 3 0 0 6 5
3 0 1 0 0 4
0 1 0 6 0 4
0 0 6 0 8 5
6 0 0 8 0 0
5 4 4 5 0 0
Edge 1 : (1 2) cost 3
Edge 2 : (2 3) cost 1
Edge 3 : (2 6) cost 4
Edge 4 : (6 4) cost 5
Edge 5 : (1 5) cost 6
Minimum Cost = 19
```

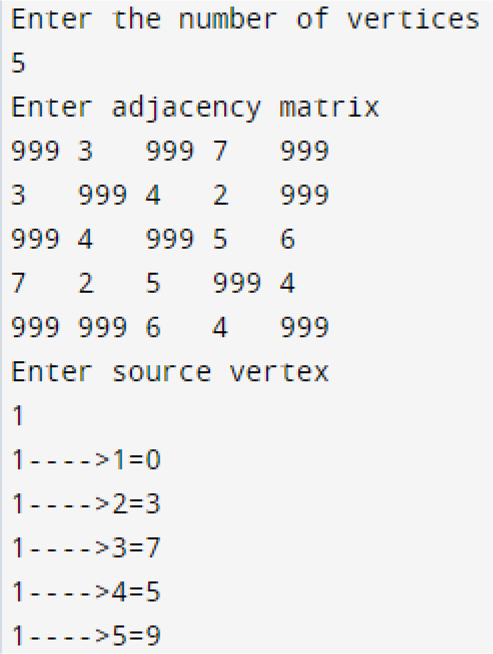
Program 6: Implement and analyze Dijkstra’s algorithm to find the shortest path froma given source.

```
#include<stdio.h>
intmain()
{
    intn,a[20][20],i,j,min,u,v,s[10],d[10],k;
    printf("Enter      the      number      of
    vertices\n");scanf("%d",&n);
    printf("Enter adjacency
    matrix\n");for(i=1;i<=n;i++)
    {
      for(j=1;j<=n;j++)
      {
         scanf("%d",&a[i][j]);
      }
    }
    printf("Enter source
    vertex\n");scanf("%d",&v);for(
    i=1;i<=n;i++)
   {
      s[i]=0;
      d[i]=a[v][i];
   }
   d[v]=0;
   s[v]=1;
   for(k=2;k<=n;k++)
   {
      min=999;for(i=1;
      i<=n;i++)
      {
         if(d[i]<min&&s[i]==0)
         {
            min=d[i];
            u=i;
         }
      }
```

```
        s[u]=1;
        for(i=1;i<=n;i++)
        {
            if(s[i]==0)
            {
                if(d[i]>d[u]+a[u][i])
                {
                    d[i]=d[u]+a[u][i];
                }
            }
        }
    }
    for(i=1;i<=n;i++)
    {
        printf("%d--- >%d=%d\n",v,i,d[i]);
    }
}
```

OUTPUT





```
Enter the number of vertices
5
Enter adjacency matrix
999 3   999 7   999
3   999 4   2   999
999 4   999 5   6
7   2   5   999 4
999 999 6   4   999
Enter source vertex
1
1---->1=0
1---->2=3
1---->3=7
1---->4=5
1---->5=9
```

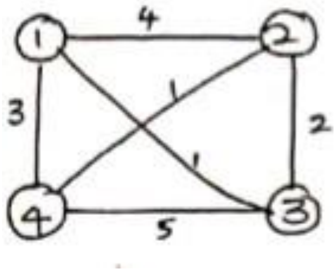
PART–B

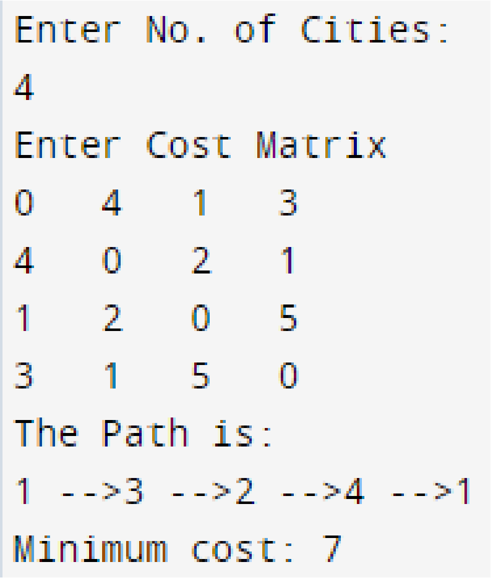
Program7:Implement travelling salesman problem using dynamic programming.

```
#include<stdio.h>
int a[10][10], visited[10], n,
cost=0;intleast(int c)
{
    inti,nc=999;
    int
    min=999,kmin;for
    (i=0;i<n;i++)
    {
        if((a[c][i]!=0)&&(visited[i]==0))
        if(a[c][i]+a[i][c]<min)
        {
            min=a[i][0]+a[c][i];
            kmin=a[c][i];
            nc=i;
        }
    }
    if(min!=999)
    cost+=kmin;
    returnnc;
}
voidmincost(intcity)
{
    inti,ncity;visited[city]=
    1;printf("%d --
    >",city+1);ncity=least(c
    ity);if(ncity==999)
    {
        ncity=0;printf("%d",
        ncity+1);cost+=a[city
        ][ncity];return;
    }
    mincost(ncity);
```

```
 }
intmain()
 {
    inti,j;
    printf("Enter No. of
    Cities:\n");scanf("%d",&n);
    printf("Enter Cost
    Matrix\n");for(i=0;i<n;i++)
    {
        for(
        j=0;j<n;j++)scanf("%d",&a[i][j]
        );visited[i]=0;
    }
    printf("The Path
    is:\n");mincost(0);
    printf("\nMinimumcost:%d",cost);
 }
```

OUTPUT



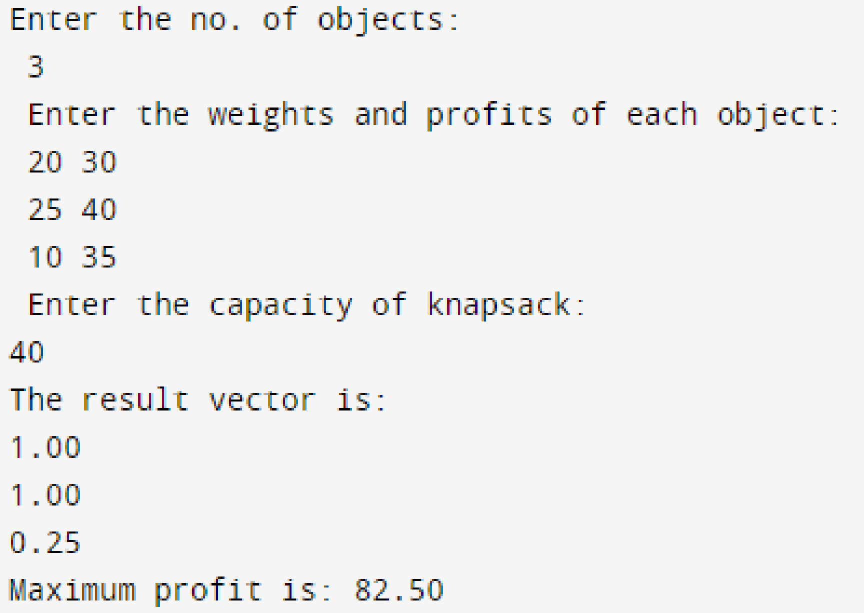


```
Enter No. of Cities:
4
Enter Cost Matrix
0   4   1   3
4   0   2   1
1   2   0   5
3   1   5   0
The Path is:
1 -->3 -->2 -->4 -->1
Minimum cost: 7
```

Program8: Implement Knapsack problem.

```
#include<stdio.h>
voidknapsack(intn,float weight[],floatprofit[],floatcapacity)
{
   float x[20], tp=
   0;inti, j,
   rc;rc=capacity;for
   (i=0;i<n;i++)x[i]
   =0.0;
   for(i=0;i<n;i++)
   {
      if(weight[i]>rc)
         break;
      else
      {
         x[i]=1.0;
         tp=
         tp+profit[i];rc=
         rc-weight[i];
      }
   }
   if(i<n)
      x[i]=rc/weight[i];tp=
   tp+(x[i]*profit[i]);
   printf("The result vector
   is:\n");for(i=0;i<n;i++)
      printf("%0.2f\n",x[i]);printf("Maximu
   mprofitis:%0.2f\n",tp);
}
intmain()
{
   float weight[20], profit[20], capacity, ratio[20],
   temp;int n, i ,j;
   printf ("Enter the no. of objects:\n
   ");scanf("%d",&n);
   printf ("Enter the weights and profits of each object:\n
   ");for(i=0;i<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]=t
             emp;

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

             temp=
             profit[j];profit[j]=
             profit[i];profit[i]=t
             emp;
         }
      }
   }
   knapsack(n,weight,profit,capacity);
}
```

OUTPUT

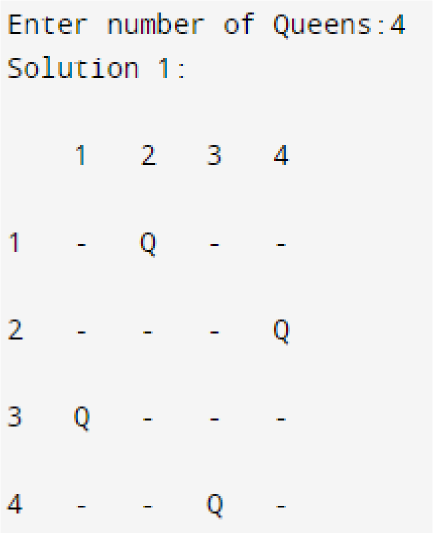


```
Enter the no. of objects:
 3
 Enter the weights and profits of each object:
 20 30
 25 40
 10 35
 Enter the capacity of knapsack:
40
The result vector is:
1.00
1.00
0.25
Maximum profit is: 82.50
```

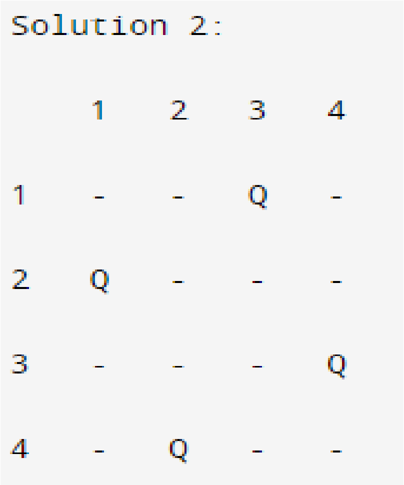
Program9:Implement N-Queens problem using backtracking.

```
#include<stdio.h>#i
nclude<stdlib.h>intb
oard[20], count;void
print(int n)
{
    inti, j;
    printf("\n\nSolution%d:\n\n",++count);for(i
    =1;i<=n;i++)
    printf("\t%d",i);f
    or(i=1;i<=n;i++)
    {
        printf("\n\n%d",i);
        for(j=1;j<=n;j++)
        {
            if(board[i]==j)
                printf("\tQ");
            else
                printf("\t-");
        }
    }
}
intplace(introw,intcolumn)
{
    inti;
    for(i=1;i<=row-1;i++)
    {
        if(board[i]==column)
            return0;
        else if(abs(board[i]-column)==abs(i-
            row))return0;
    }
    return1;
}
voidqueen(introw,intn)
{
    int
    column;for(column=1;column<=n;col
    umn++)
    {
        if(place(row,column))
        {
            board[row]=column;
            if(row==n)
                print(n);
            else
                queen(row+1,n);
        }
    }
}
intmain()
{
    int n;
    printf("Enter number of
    Queens:");scanf("%d",&n);
    queen(1,n);
}
```

OUTPUT



```
Enter number of Queens:4
Solution 1:

   1  2  3  4

1  -  Q  -  -

2  -  -  -  Q

3  Q  -  -  -

4  -  -  Q  -
```

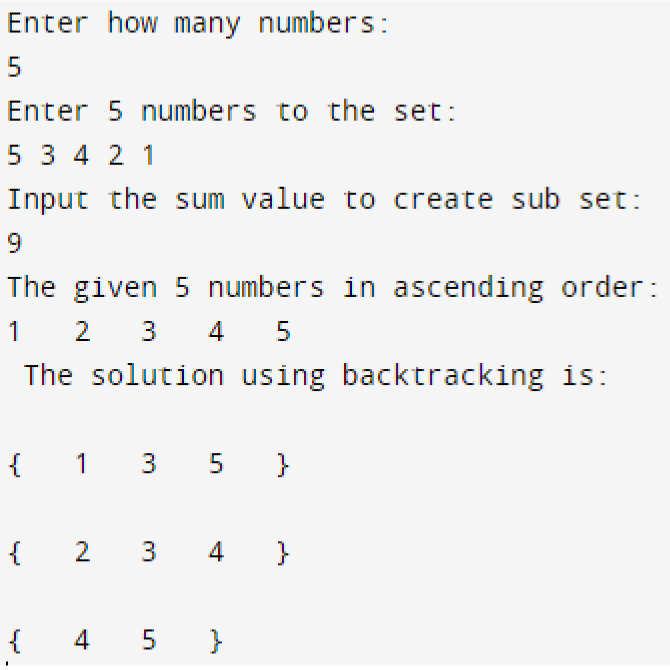


```
Solution 2:

    1   2   3   4

1   -   -   Q   -

2   Q   -   -   -

3   -   -   -   Q

4   -   Q   -   -
```

Program10: Implement sum of subset problem using backtracking.

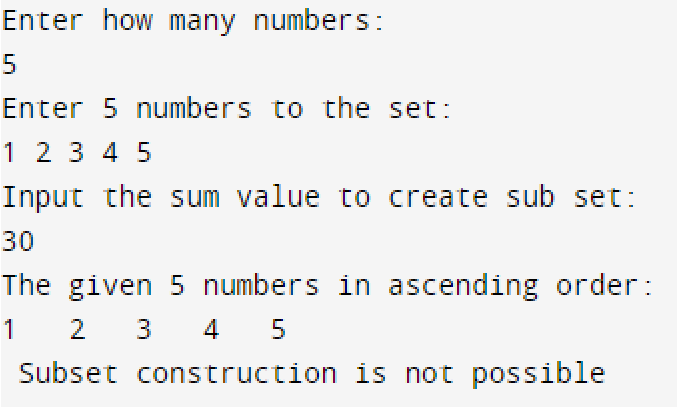
```
#include<stdio.h>
#defineTRUE 1
#defineFALSE0
intinc[50],w[50],sum,n;
intpromising(inti,intwt,inttotal)
{
    return(((wt+total)>=sum)&&((wt==sum)||(wt+w[i+1]<=sum)));
}
voidsumset(inti,intwt,int total)
{
    int
    j;if(promising(i,wt,total
    ))
    {
        if(wt==sum)
        {
            printf("\n{\t");for
            (j=0;j<=i;j++)
                if(inc[j])printf("%d\t"
                    ,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]);
        }
    }
}
intmain()
{
    inti,j, n,temp,total=0;
    printf("\n Enter how many
    numbers:\n");scanf("%d",&n);
    printf("\n Enter %d numbers to the
    set:\n",n);for(i=0;i<n;i++)
    {
    scanf("%d",&w[i]);
    total+=w[i];
    }
    printf("\n Input the sum value to create sub
    set:\n");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",n);for(i=0;i<n;i++)
        printf("%d
    \t",w[i]);if((total<sum)
    )
        printf("\nSubsetconstructionisnotpossible");else
    {
        for
        (i=0;i<n;i++)inc
        [i]=0;
        printf("\n The solution using backtracking
        is:\n");sumset(-1,0,total);
    }
}
```

OUTPUT1:



```
Enter how many numbers:
5
Enter 5 numbers to the set:
5 3 4 2 1
Input the sum value to create sub set:
9
The given 5 numbers in ascending order:
1    2    3    4    5
 The solution using backtracking is:

{    1    3    5    }

{    2    3    4    }

{    4    5    }
```

OUTPUT2:



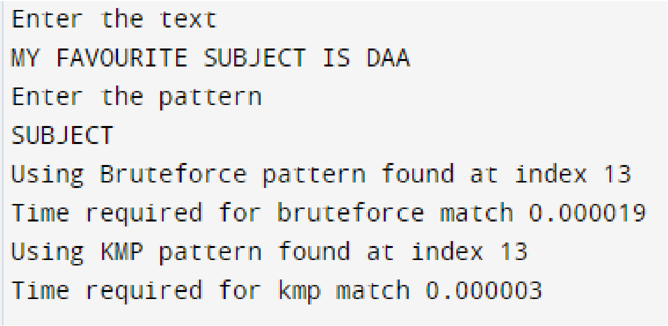
```
Enter how many numbers:
5
Enter 5 numbers to the set:
1 2 3 4 5
Input the sum value to create sub set:
30
The given 5 numbers in ascending order:
1    2    3    4    5
 Subset construction is not possible
```

Program11: Implement and compare Simple string matching and KMP string matching algorithm.

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>#
include<string.h>i
ntlps[100];
voidcomputeLPSArray(charpattern[])
{
   intlen=0, i=1, m,
   n;lps[0] =0;
   m=
   strlen(pattern);whi
   le(i<m)
   {
      if(pattern[i]==pattern[len])
      {
         len++;lps[i]
         = len;i++;
      }
      else
      {
         if(len!=0)
            len = lps[len-
         1];else
         {
            lps[i] =
            0;i++;
         }
      }
   }
}
voidkmp(chartext[],charpattern[])
{
   int
   j=0,i=0,m,n;m=str
```

```
    len(pattern);
    n =
    strlen(text);computeLPSArray(patt
    ern);while(i<n)
    {
        if(pattern[j] ==text[i])
        {
            j++;
            i++;
        }
        if(j ==m)
        {
            printf("Using KMP pattern found at index %d \n", i-
            j);j = lps[j-1];
        }
        elseif(pattern[j] !=text[i])
        {
            if(j!=0)
                j = lps[j-
            1];else
                i= i+1;
        }
    }
}
voidbruteforce(chartext[],charpattern[])
{
    inti,j,k,m,n,flag=1;n
    =strlen(text);m=strl
    en(pattern);for(i=0;i
    <=n-m;i++)
    {
        j=0;
        while(j<m&&pattern[j]==text[j+i])
        {
            j++;
            if(j==m)
            {
               flag=1;
               k=i;
            }
            else
               flag=0;
         }
      }
      if(flag==1)
         printf("Using Bruteforce pattern found at index
      %d\n",k);else
         printf("UsingBruteforce:NoMatchfound");
}
int main()
{
      char
      text[50],pattern[50];clock
      _t
      st1,st2,end1,end2;printf("
      Enter the
      text\n");gets(text);
      printf("Enter the
      pattern\n");gets(pattern);
      st1=clock();bruteforce(t
      ext,pattern);end1=clock
      ();
      printf("Time required for bruteforce match %f\n",(float)(end1-
      st1)/CLOCKS_PER_SEC);st2=clock();
      kmp(text,pattern);
      end2=clock();
      printf("Timerequired forkmpmatch%f\n",(float)(end2-st2)/CLOCKS_PER_SEC);
}
```

OUTPUT



```
Enter the text
MY FAVOURITE SUBJECT IS DAA
Enter the pattern
SUBJECT
Using Bruteforce pattern found at index 13
Time required for bruteforce match 0.000019
Using KMP pattern found at index 13
Time required for kmp match 0.000003
```

Program12: Implement and analyze k-server problem.

```
#include <stdio.h>
#include <stdbool.h>

#define MAX_LOCATIONS 100

typedefstruct {
int k;
int locations[MAX_LOCATIONS];
intserver_positions[MAX_LOCATIONS];
} KServer;

void initialize(KServer *server, int k, int *locations) {
server->k = k;
inti;
for (i = server->k - 1; i> 0; i--) {
server->server_positions[i] = server->server_positions[i - 1];
}

server->locations[i] = locations[i];
server->server_positions[i] = i< k ? i : -1;  // Initialize servers at locations 0, 1, ..., k-1
    }

intfind_server_index(KServer *server, int location) {
            inti;
for (i = 0; i< server->k; i++) {
if (server->server_positions[i] == location) {
returni;
        }
    }
return -1;  // Server not found at the location
}

voidserve_request(KServer *server, int request) {
if (request < 0 || request >= MAX_LOCATIONS) {
printf("Invalid request: %d\n", request);
return;
    }

intserver_index = find_server_index(server, request);

if (server_index != -1) {
        // Requested page is already being served, update its position to the most recent
int j;
                        for (j = server_index; j > 0; j--) {
server->server_positions[j] = server->server_positions[j - 1];
        }
server->server_positions[0] = request;
    } else {
        // Move the server to the location of the most recently requested page
inti;
for ( i = server->k - 1; i> 0; i--) {
server->server_positions[i] = server->server_positions[i - 1];
        }
server->server_positions[0] = request;
    }
}

voidprint_server_positions(KServer *server) {
printf("Current server positions:");
inti;
for (i = 0; i< server->k; i++) {
printf(" %d", server->server_positions[i]);
    }
printf("\n");
}

int main() {
int k = 3;  // Number of servers
int locations[MAX_LOCATIONS] = {0, 1, 2, 3, 4, 5};  // Possible page locations

KServer server;
initialize(&server, k, locations);

    // Serve some requests
serve_request(&server, 2);
print_server_positions(&server);

serve_request(&server, 4);
print_server_positions(&server);

serve_request(&server, 1);
print_server_positions(&server);

serve_request(&server, 5);
print_server_positions(&server);

return 0;
}
```

OUTPUT:

```
Current server positions: 2 0 1
Current server positions: 4 2 0
Current server positions: 1 4 2
Current server positions: 5 1 4
```

```
--------------------------------
Process exited after 0.01138 seconds with return value
```

Programs beyond the Syllabus for Academic Year 2024-25

Program1: Implement mergesort for the demonstration of parallel algorithm.

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

void merge(intarr[], int l, int m, int r) {
inti, j, k;
int n1 = m - l + 1;
int n2 = r - m;

    // Create temporary arrays
int L[n1], R[n2];

    // Copy data to temporary arrays L[] and R[]
for (i = 0; i< n1; i++)
        L[i] = arr[l + i];
for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    // Merge the temporary arrays back into arr[l..r]
i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
while (i< n1 && j < n2) {
if (L[i] <= R[j]) {
arr[k] = L[i];
i++;
        } else {
arr[k] = R[j];
j++;
        }
k++;
    }

    // Copy the remaining elements of L[], if there are any
while (i< n1) {
arr[k] = L[i];
i++;
k++;
    }

    // Copy the remaining elements of R[], if there are any
while (j < n2) {
arr[k] = R[j];
j++;
k++;
    }
}

voidmergeSortParallel(intarr[], int l, int r, int depth) {
if (l < r) {
        // Same as the sequential version until a certain depth is reached
if (depth > 0) {
#pragma omp parallel sections
            {
#pragma omp section
mergeSortParallel(arr, l, (l + r) / 2, depth - 1);

#pragma omp section
mergeSortParallel(arr, (l + r) / 2 + 1, r, depth - 1);
            }
        } else {
            // Continue sequentially
mergeSortParallel(arr, l, (l + r) / 2, depth);

mergeSortParallel(arr, (l + r) / 2 + 1, r, depth);
        }

merge(arr, l, (l + r) / 2, r);
    }
}

voidprintArray(int A[], int size) {
for (inti = 0; i< size; i++)
printf("%d ", A[i]);
printf("\n");
}
```

```
int main() {
intarr[] = {12, 11, 13, 5, 6, 7};
intarr_size = sizeof(arr) / sizeof(arr[0]);

printf("Given array is \n");
printArray(arr, arr_size);

    // Choose the depth to control the parallelism level
int depth = 2;

    // Perform parallel merge sort
mergeSortParallel(arr, 0, arr_size - 1, depth);

printf("Sorted array is \n");
printArray(arr, arr_size);

return 0;
}
```

Program2:Implement prefix computation for the demonstration of parallel algorithm.

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

voidparallelPrefixSum(int *arr, int n) {
int *temp = malloc(n * sizeof(int));

    // Perform parallel prefix sum using OpenMP
#pragma omp parallel
    {
intthread_id = omp_get_thread_num();
intnum_threads = omp_get_num_threads();

        // Calculate chunk size for each thread
intchunk_size = (n + num_threads - 1) / num_threads;
int start = thread_id * chunk_size;
int end = (start + chunk_size<= n) ? start + chunk_size : n;

        // Compute local prefix sum
temp[start] = (start > 0) ? arr[start - 1] : 0;
for (inti = start + 1; i< end; i++) {
temp[i] = arr[i - 1] + arr[i];
        }

#pragma omp barrier

        // Update the original array with the local prefix sum
for (inti = start; i< end; i++) {
arr[i] = temp[i];
        }
    }

free(temp);
}

int main() {
int n = 8;
intarr[] = {3, 1, 7, 0, 4, 1, 6, 3};

printf("Original array:\n");
```

```
    for (inti = 0; i< n; i++) {
    printf("%d ", arr[i]);
        }
    printf("\n");

        // Perform parallel prefix sum
    parallelPrefixSum(arr, n);

    printf("Prefix sum array:\n");
    for (inti = 0; i< n; i++) {
    printf("%d ", arr[i]);
        }
    printf("\n");

    return 0;
}
```