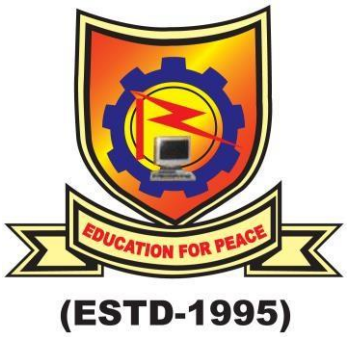
DESIGN AND ANALYSIS OF ALGORITHMS LAB

(A0598204)

LAB MANUAL

RGM R-20 Regulations



Department of Computer Science and Engineering

Rajeev Gandhi Memorial College of Engineering and Technology
(Autonomous)
Nandyal, Kurnool – Dist. A.P

Experiment No: 1

Write a program to perform operation count for a given pseudo code

```
#include<stdio.h>
#include<conio.h>
int main()
{
int count=0,sum=0,n,i,a[50];
//clrscr();
count=count+1;
printf("\n Enter the n value:");
scanf("%d",&n); count=count+1;
printf("\n Enter %d values to sum:",n);
for(i=0;i<n;i++)
{
count=count+1;
scanf("%d",&a[i]);
}
count=count+1;
for(i=0;i<n;i++)
{
count=count+1;
sum=sum+a[i];
count=count+1;
}
count= count+1;
printf("\n The sum of %d values is:%d and count is=%d",n,sum,count);
//getch();
}
```

Output:

```
Enter the n value:3
 Enter 3 values to sum:1
2
3
 The sum of 3 values is:6 and count is=13
```

Experiment No: 2

Aim: Write a program to perform Bubble sort for any given list of numbers.

Algorithm:

```
Step 1: Start
Step 2: Read n, a[i] values as integers
Step 3: for i: 1 to n do increment i by 1
        begin
          for j: 0 to n - i - 1 increment j by 1
          begin
            if(a[j] > a[j + 1])
            begin
             t = a[j];
             a[j] = a[j + 1];
             a[j + 1] = t;
            end
          end
        end
Step 4: for i: 0 to n
          Print a[i]
Step 5: Stop
```

Program:

```
/* Bubble sort code */
#include <stdio.h>
int main()
{
  int array[100], n, c, d, swap;
  printf("Enter number of elements\n");
  scanf("%d", &n);
  printf("Enter %d integers\n", n);
  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);
  for (c = 0 ; c < n - 1; c++)
  {
    for (d = 0 ; d < n - c - 1; d++)
    {
      if (array[d] > array[d+1]) /* For decreasing order use '<' instead of '>' */
      {
        swap       = array[d];
        array[d]   = array[d+1];
        array[d+1] = swap;
      }
    }
  }
  printf("Sorted list in ascending order:\n");
  for (c = 0; c < n; c++)
     printf("%d\n", array[c]);

  return 0;
}
```

Output of program:

```
Enter number of elements
5
Enter 5 integers
6
3
9
-1
7
Sorted list in ascending order:
-1
3
6
7
9
```

Experiment No: 3

Aim: Write a program to perform Insertion sort for any given list of numbers.

```
/* Insertion sort ascending order */

#include <stdio.h>

int main()
{
  int n, array[1000], c, d, t, flag = 0;

  printf("Enter number of elements\n");
  scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

  for (c = 1 ; c <= n - 1; c++) {
    t = array[c];

    for (d = c - 1 ; d >= 0; d--) {
      if (array[d] > t) {
        array[d+1] = array[d];
        flag = 1;
      }
      else
        break;
    }
    if (flag)
      array[d+1] = t;
  }

  printf("Sorted list in ascending order:\n");

  for (c = 0; c <= n - 1; c++) {
    printf("%d\n", array[c]);
  }

  return 0;
}
```

Output of program:

```
Enter number of elements
5
Enter 5 integers
4
3
-1
2
1
Sorted list in ascending order:
-1
1
2
3
4
```

Experiment No: 4

Aim: Write a program to perform Quick sort for any given list of numbers.

```
#include<stdio.h>
void quicksort(int [10],int,int);
int main(){
  int x[20],size,i;
  printf("Enter size of the array: ");
  scanf("%d",&size);

  printf("Enter %d elements: ",size);
  for(i=0;i<size;i++)
    scanf("%d",&x[i]);

  quicksort(x,0,size-1);
  printf("Sorted elements: ");
  for(i=0;i<size;i++)
    printf(" %d",x[i]);

  return 0;
}
void quicksort(int x[10],int first,int last){
    int pivot,j,temp,i;
     if(first<last){
         pivot=first;
         i=first;
         j=last;
         while(i<j){
             while(x[i]<=x[pivot]&&i<last)
                 i++;
             while(x[j]>x[pivot])
                 j--;
             if(i<j){
                 temp=x[i];
                  x[i]=x[j];
                  x[j]=temp;
             }
         }
         temp=x[pivot];
         x[pivot]=x[j];
         x[j]=temp;
```

```
        quicksort(x,first,j-1);
        quicksort(x,j+1,last);

    }
}
```

Output:

Enter size of the array: 5

Enter 5 elements: 3 8 0 1 2

Sorted elements: 0 1 2 3 8

Experiment No: 5

Aim: Write a program to find Maximum and Minimum of the given set of integer values.

```
#include<stdio.h>
int main()
{
   int i,size,max=0,min=0;
   printf("Enter size of the array\n");
   scanf("%d",&size);
   int a[size];
   printf("Enter elements in array\n");
   for(i=0;i<size;i++)
   {
      scanf("%d",&a[i]);
   }
   min=a[0];
   max=a[0];
   for(i=0;i<size;i++)
   {
      if(a[i]>max)
      {
         max=a[i];
      }
      if(a[i]<min)
      {
         min=a[i];
      }
   }
   printf("The maximum number is %d\n",max);
   printf("The minimum number is %d\n",min);
}
```

Output:

```
Enter size of the array
5
Enter elements in array
2 66 7 55 4
The maximum number is 66
The minimum number is 2
```

.

Experiment No: 6

Aim: Write a Program to perform Merge Sort on the given two lists of integer values.

```
#include<stdio.h>
#define MAX 50
void mergeSort(int arr[],int low,int mid,int high);
void partition(int arr[],int low,int high);
int main(){
   int merge[MAX],i,n;
   printf("Enter the total number of elements: ");
   scanf("%d",&n);
   printf("Enter the elements which to be sort: ");
   for(i=0;i<n;i++){
      scanf("%d",&merge[i]);
   }
   partition(merge,0,n-1);
   printf("After merge sorting elements are: ");
   for(i=0;i<n;i++){
      printf("%d ",merge[i]);
   }
  return 0;
}
void partition(int arr[],int low,int high){
   int mid;
   if(low<high){
      mid=(low+high)/2;
      partition(arr,low,mid);
      partition(arr,mid+1,high);
      mergeSort(arr,low,mid,high);
   }
}
void mergeSort(int arr[],int low,int mid,int high){
   int i,m,k,l,temp[MAX];
   l=low;
   i=low;
   m=mid+1;
   while((l<=mid)&&(m<=high)){
      if(arr[l]<=arr[m]){
         temp[i]=arr[l];
         l++;
      }
      else{
         temp[i]=arr[m];
         m++;
      }
      i++;
   }

   if(l>mid){
      for(k=m;k<=high;k++){
```

```
            temp[i]=arr[k];
            i++;
        }
    }
    else{
        for(k=l;k<=mid;k++){
            temp[i]=arr[k];
            i++;
        }
    }

    for(k=low;k<=high;k++){
        arr[k]=temp[k];
    }
}
```

Output :

Enter the total number of elements: 5
Enter the elements which to be sort: 2 5 0 9 1
After merge sorting elements are: 0 1 2 5 9

Experiment No: 7

Aim: Write a Program to perform Binary Search for a given set of integer values recursively andnon-recursively.

/* Binary search program in C using both recursive and non recursive functions */

```
#include <stdio.h>
#define MAX_LEN 10

/* Non-Recursive function*/
void b_search_nonrecursive(int l[],int num,int ele)
{
  int l1,i,j, flag = 0;
  l1 = 0;
  i = num-1;
  while(l1 <= i)
  {
    j = (l1+i)/2;
    if( l[j] == ele)
    {
  printf("\nThe element %d is present at position %d in list\n",ele,j);
                                        flag =1;
                                        break;
    }
    else
                                    if(l[j] < ele)
                                          l1 = j+1;
                                    else
                                          i = j-1;
  }
  if( flag == 0)
  printf("\nThe element %d is not present in the list\n",ele);
}

/* Recursive function*/
int b_search_recursive(int l[],int arrayStart,int arrayEnd,int a)
{
 int m,pos;
 if (arrayStart<=arrayEnd)
 {
   m=(arrayStart+arrayEnd)/2;
   if (l[m]==a)
     return m;
   else if (a<l[m])
     return b_search_recursive(l,arrayStart,m-1,a);
   else
     return b_search_recursive(l,m+1,arrayEnd,a);
  }
  return -1;
```

```
}

void read_list(int l[],int n)
{
  int i;
  printf("\nEnter the elements:\n");
  for(i=0;i<n;i++)
     scanf("%d",&l[i]);
}

void print_list(int l[],int n)
{
   int i;
  for(i=0;i<n;i++)
     printf("%d\t",l[i]);
}

/*main function*/
main()
{
  int l[MAX_LEN], num, ele,f,l1,a;
  int ch,pos;

  //clrscr();

  printf("=====================================================");
  printf("\n\t\t\tMENU");
  printf("\n=====================================================");
  printf("\n[1] Binary Search using Recursion method");
  printf("\n[2] Binary Search using Non-Recursion method");
  printf("\n\nEnter your Choice:");
  scanf("%d",&ch);

  if(ch<=2 & ch>0)
  {
    printf("\nEnter the number of elements : ");
    scanf("%d",&num);
    read_list(l,num);
    printf("\nElements present in the list are:\n\n");
    print_list(l,num);
    printf("\n\nEnter the element you want to search:\n\n");
    scanf("%d",&ele);

  switch(ch)
  {
    case 1:printf("\nRecursive method:\n");
                                        pos=b_search_recursive(l,0,num,ele);
                                        if(pos==-1)
                                        {
                                            printf("Element is not found");
```

```
                                        }
                                        else
                                        {
                                            printf("Element is found at %d position",pos);
                                        }
                                        //getch();
                                        break;

    case 2:printf("\nNon-Recursive method:\n");
                                        b_search_nonrecursive(l,num,ele);
                                        //getch();
                                        break;
    }
  }
//getch();
}
```

Output:

```
=====================================================
                MENU
=====================================================
[1] Binary Search using Recursion method
[2] Binary Search using Non-Recursion method

Enter your Choice:1

Enter the number of elements: 5

Enter the elements:
12
22
32
42
52

Elements present in the list are:

12      22      32      42      52

Enter the element you want to search:

42

Recursive method:
Element is found at 3 position
```

Experiment No: 8

Aim: Write a program to find solution for knapsack problem using greedy method.

```
# include<stdio.h>

void knapsack(int n, float weight[], float profit[], float capacity) {
  float x[20], tp = 0;
  int i, j, u;
  u = capacity;

  for (i = 0; i < n; i++)
    x[i] = 0.0;

  for (i = 0; i < n; i++) {
    if (weight[i] > u)
      break;
    else {
      x[i] = 1.0;
      tp = tp + profit[i];
      u = u - weight[i];
    }
  }

  if (i < n)
    x[i] = u / weight[i];

  tp = tp + (x[i] * profit[i]);

  printf("\nThe result vector is:- ");
  for (i = 0; i < n; i++)
    printf("%f\t", x[i]);

  printf("\nMaximum profit is:- %f", tp);

}

int main() {
  float weight[20], profit[20], capacity;
  int num, i, j;
  float ratio[20], temp;

  printf("\nEnter the no. of objects:- ");
  scanf("%d", &num);

  printf("\nEnter the wts and profits of each object:- ");
  for (i = 0; i < num; i++) {
    scanf("%f %f", &weight[i], &profit[i]);
  }
```

```
  printf("\nEnter the capacityacity of knapsack:- ");
  scanf("%f", &capacity);

  for (i = 0; i < num; i++) {
    ratio[i] = profit[i] / weight[i];
  }

  for (i = 0; i < num; i++) {
    for (j = i + 1; j < num; j++) {
      if (ratio[i] < ratio[j]) {
        temp = ratio[j];
        ratio[j] = ratio[i];
        ratio[i] = temp;

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

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

  knapsack(num, weight, profit, capacity);
  return(0);
}
```

Output

Enter the no. of objects:- 7

Enter the wts and profits of each object:-
2 10
3 5
5 15
7 7
1 6
4 18
1 3

Enter the capacity of knapsack:- 15

The result vector is:- 1.000000 1.000000 1.000000 1.000000
1.000000 0.666667 0.000000

Maximum profit is:- 55.333332

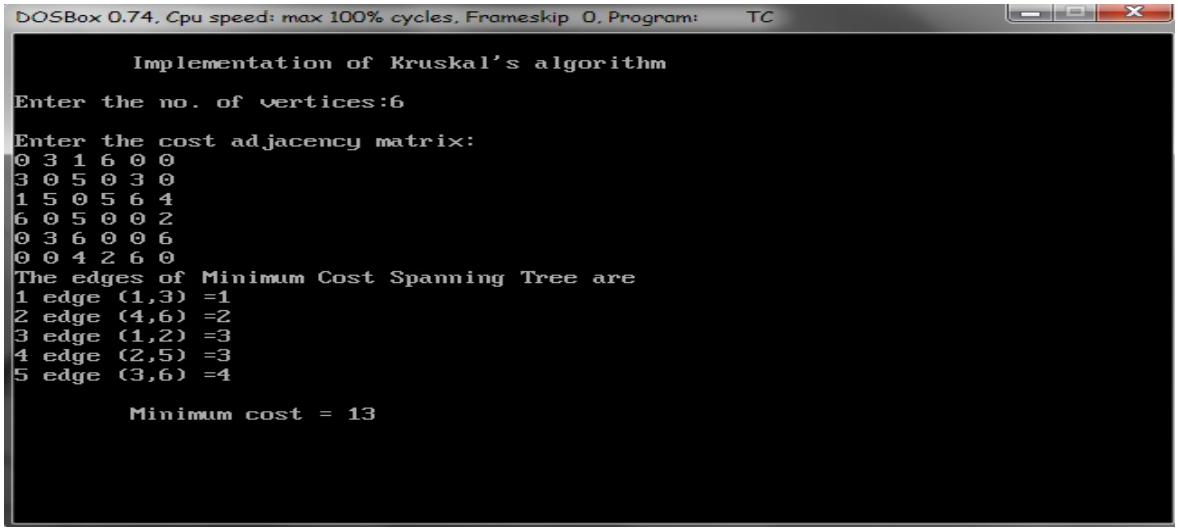
Experiment No: 9

Aim: **Write a program to find minimum cost spanning tree using Prim's Algorithm**

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]={0},min,mincost=0,cost[10][10];
int main()
{
        printf("\nEnter the number 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",&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 Minimun cost=%d",mincost);
        getch();
}
```

Output:



Experiment No: 10

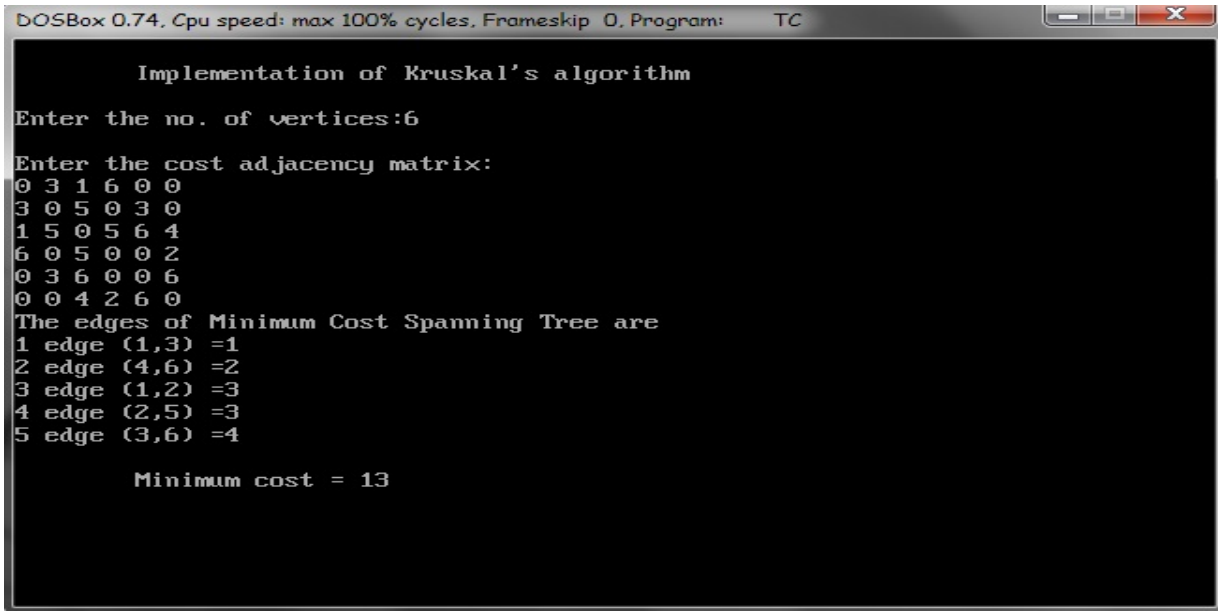
Aim: Write a program to find minimum cost spanning tree using Kruskal's Algorithm.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
int main()
{
        //clrscr();
        printf("\n\tImplementation of Kruskal's algorithm\n");
        printf("\nEnter the no. of vertices:");
        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("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);
                if(uni(u,v))
                {
                        printf("%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);
        getch();
}
int find(int i)
{
        while(parent[i])
        i=parent[i];
        return i;
}
int uni(int i,int j)
{
        if(i!=j)
        {
                parent[j]=i;
                return 1;
        }
        return 0;
}
```

Output:



Experiment No: 11(C IMPLEMETATION of Dijkstra's Algorithm)

Aim: Write a program to perform Single source shortest path problem for a given graph.

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10

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

int main(){
	int G[MAX][MAX], i, j, n, u;
	//clrscr();
	printf("\nEnter the no. of vertices:: ");
	scanf("%d", &n);
	printf("\nEnter the adjacency matrix::\n");
	for(i=0;i < n;i++)
		for(j=0;j < n;j++)
			scanf("%d", &G[i][j]);
	printf("\nEnter the starting node:: ");
	scanf("%d", &u);
	dijikstra(G,n,u);
	getch();
}

void dijikstra(int G[MAX][MAX], int n, int startnode)
{
	int cost[MAX][MAX], distance[MAX], pred[MAX];
	int visited[MAX], count, mindistance, nextnode, i,j;
	for(i=0;i < n;i++)
		for(j=0;j < n;j++)
			if(G[i][j]==0)
				cost[i][j]=INFINITY;
			else
				cost[i][j]=G[i][j];

	for(i=0;i< n;i++)
	{
		distance[i]=cost[startnode][i];
		pred[i]=startnode;
		visited[i]=0;
	}
	distance[startnode]=0;
	visited[startnode]=1;
	count=1;
	while(count < n-1){
		mindistance=INFINITY;
		for(i=0;i < n;i++)
```

```
                if(distance[i] < mindistance&&!visited[i])
                {
                        mindistance=distance[i];
                        nextnode=i;
                }
        visited[nextnode]=1;
        for(i=0;i < n;i++)
                if(!visited[i])
                        if(mindistance+cost[nextnode][i] < distance[i])
                        {
                                distance[i]=mindistance+cost[nextnode][i];
                                pred[i]=nextnode;
                        }
                count++;
    }

    for(i=0;i < n;i++)
        if(i!=startnode)
        {
                printf("\nDistance of %d = %d", i, distance[i]);
                printf("\nPath = %d", i);
                j=i;
                do
                {
                        j=pred[j];
                        printf(" <-%d", j);
                }
                while(j!=startnode);
        }
}
```

Output:

```
Enter the no. of vertices:: 4

Enter the adjacency matrix::
0 1 1 1
1 0 1 0
1 1 0 1
1 0 1 0

Enter the starting node:: 1

Distance of 0 = 1
Path = 0<-1
Distance of 2 = 1
Path = 2<-1
Distance of 3 = 2
Path = 3<-0<-1
```

Experiment No: 12

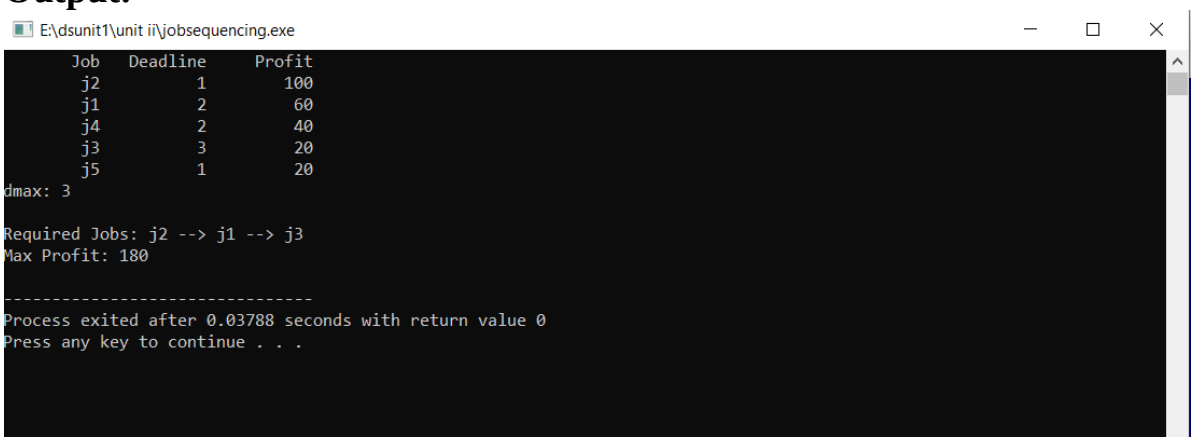
Aim: Write a program to find solution for job sequencing with deadlines problem.

```
#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:



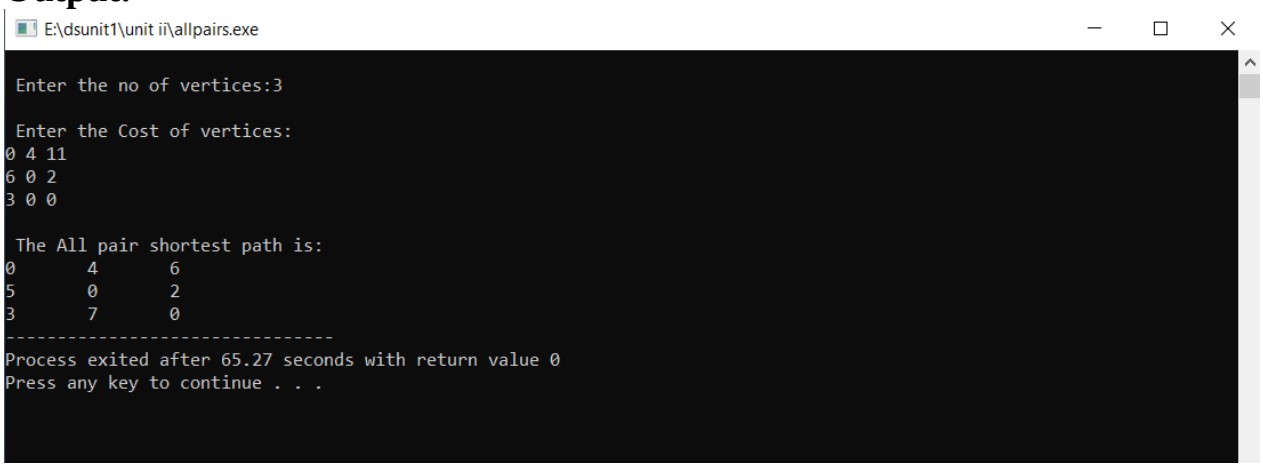
Experiment No: 13

Aim: Write a program for all pairs shortest path problem.

```
#include<stdio.h>
#include<conio.h>
 void readf();
void amin();
int cost[20][20],a[20][20]; int i,j,k,n;
void readf()
{
printf("\n Enter the no of vertices:"); scanf("%d",&n);
printf("\n Enter the Cost of vertices:"); for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&cost[i][j]);
if(cost[i][j]==0 && (i!=j)) cost[i][j]=999;
a[i][j]=cost[i][j];
}
}
}
void amin()
{
for(k=0;k<n;k++)
{
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
if(a[i][j]>a[i][k]+a[k][j])
{
a[i][j]=a[i][k]+a[k][j];
}
}
}
}
printf("\n The All pair shortest path is:"); for(i=0;i<n;i++)
{
printf("\n");
for(j=0;j<n;j++)
{
printf("%d\t",a[i][j]);
```

```
}
}
}
int main()
{
//clrscr();
readf();
amin();
//getch();
}
```

Output:



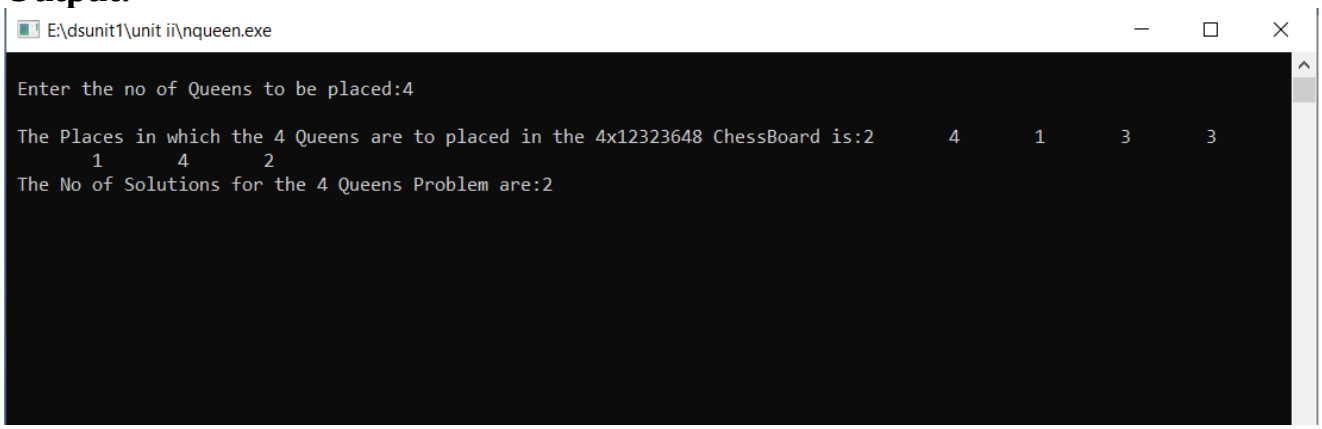
Experiment No: 14

Aim: Write a program to solve N-QUEENS problem.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void readv();
void nqueen(int,int);
int place(int,int);
int x[25],count=0;
void readv()
{
int n;
printf("\n Enter the no of Queens to be placed:");
scanf("%d",&n);
printf("\n The Places in which the %d Queens are to placed in the %dx%d ChessBoard
is:",n,n);
nqueen(1,n);
printf("\n The No of Solutions for the %d Queens Problem are:%d",n,count);
}
void nqueen(int k,int n)
{
int i,j;
 for(i=1;i<=n;i++)
{
if(place(k,i))
{
x[k]=i; if(k==n)
{
count++;
if(count%10 == 0)
//getch();
printf("\n");
for(j=1;j<=n;j++)
{
printf("%d\t",x[j]);
}
}
else
{
nqueen(k+1,n);
}
}
}
}
int place(int k,int i)
{
int j;
for(j=1;j<=k-1;j++)
{
```

```
if((x[j]==i)||(abs(x[j]-i)==abs(j-k)))
{
return 0;
}
}
return 1;
}
int main()
{
//clrscr();
readv();
getch();
}
```

Output:



Experiment No: 15

Aim: Write a program to solve Sum of subsets problem for a given set of distinct numbers.

```
#include<stdio.h>
#include<conio.h>
void SumOfSub(int,int,int);
int x[25],n,m=0,sum=0,w[25];;
 void readf()
{
```

```
int i;
printf("\n Enter the no of values in the set:");
scanf("%d",&n);
printf("\n Enter the %d weights of the values in the set:",n);
 for(i=1;i<=n;i++)
{
scanf("%d",&w[i]);
 sum=sum+w[i];
x[i]=0;
}
printf("\n Enter the required sum of the values in the subset:");
scanf("%d",&m);
printf("\n The Total sum of the weights is:%d",sum);
SumOfSub(0,1,sum);
}
void SumOfSub(int s,int k,int r)
{
int i,j;
x[k]=1;
if(sum>=m)
{
if(s+w[k]==m)
{
printf("\n");
for(j=1;j<=n;j++)
{
printf("%d\t",x[j]);
}
printf("\n-->");
for(j=1;j<=k;j++)
{
if(x[j] == 1)
printf("%d\t",w[j]);
}
}
else if(s+w[k]+w[k+1]<=m)
 SumOfSub(s+w[k],k+1,r-w[k]);
if((s+r-w[k]>=m) && (s+w[k+1]<=m))
{
x[k]=0;
SumOfSub(s,k+1,r-w[k]);
}
}
```

```
else
{
printf("\n No Solutions Available because sum of all weights is %d less than
required sum%d",sum,m);
}
}
int main()
{
//clrscr();
readf();
//getch();
}
```

Output: