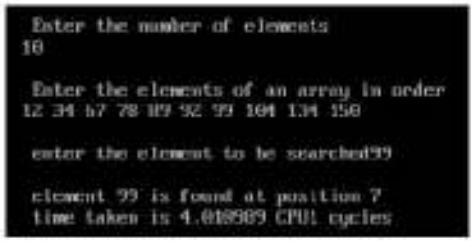
// **LabCycle 1**– WAP to implement linear search algorithm. Repeat the experiment for different values of n, // the number of elements in the list to be searched and plot a graph of the time taken versus n

```
#include <stdio.h>
#include <conio.h>
#include <time.h>
#include <stdlib.h>
#define max 20
int pos;
int LinSearch(int,int[],int);

void main()
{

        int choice =1;
        double t;
        int n,i,a[max],k,op,low,high,pos,ch;
        clock_t begin,end;
        clrscr();
         printf("\n Enter the number of elements \n");
          scanf("%d",&n);
         printf("\n Enter the elements of an array in order \n");
         for(i=0;i<n;i++)
          scanf("%d", &a[i]);
         printf(" \n enter the element to be searched");
          scanf("%d", &k);
                begin = clock();
                pos = LinSearch(n,a,k);
                end = clock();
                if (pos == -1)
                        printf("\n Unsuccessful search \n");
                else
                        printf(" \n element %d is found at position %d", k, pos + 1);
                        printf("\n time taken is %1f CPU1 cycles\n", (end - begin) / CLK_TCK);
                        getch();
}

int LinSearch(int n, int a[], int k)
{
        delay(1000);
        if(n<0)
        return -1;
        if(k == a[n-1])
        return (n-1);
        else
        return LinSearch(n-1, a,k);

}
```

Output:



Run the program with number of elements as 20, 40 and find out the time taken to perform linear search always maintaining the element to be searched in the middle position.

Plot a graph of number of elements (x-axis) versus the time taken to search the element (y-axis)

//**Labcycle 2**: WAP to implement binary search algorithm. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.

```
#include <stdio.h>
#include <conio.h>
#include <time.h>
#include <stdlib.h>
#define max 20
int pos;
int BinSearch(int, int[], int,int,int);

void main()
{
        double t;
        int n,i,a[max],k,op,low,high,pos,ch;
        clock_t begin,end;
        clrscr();
        printf("\n Enter the number of elements \n");
        scanf("%d",&n);
        printf("\n Enter the elements of an array in order \n");
        for(i=0;i<n;i++)
        scanf("%d", &a[i]);
        printf(" \n enter the element to be searched");
        scanf("%d", &k);
        low = 0;
        high = n-1;
        begin = clock();
        pos = BinSearch(n,a,k,low,high);
        end = clock();
        if (pos == -1)
        {
          printf("\n Unsuccessful search \n");
        }
        else
        {
          printf(" \n element %d is found at position %d\n", k, (pos + 1));
          printf("\n time taken is %1f CPU1 cycles\n", (end - begin)/CLK_TCK);
        }
        getch();
}

int BinSearch(int n, int a[], int k, int low, int high)
{
        int mid;
        delay(1000);
        mid = (low + high)/ 2;
        if(low > high)
        return -1;
```

```
        if(k == a[mid])
        return mid;
        else if(k>a[mid])
        return BinSearch(n,a,k,(mid+1),high);
        else
        return BinSearch(n,a,k,low,mid-1);
}
```

Output:

Run the program with number of elements as 20, 40 and find out the time taken to perform linear search always maintaining the element to be searched in the middle position.

Plot a graph of number of elements (x-axis) versus the time taken to search the element (y-axis)

//**Labcycle 3**: WAP to solve towers of Hanoi problem and execute it for different number of disks

```
#include<stdio.h>
void toh(int n, char A, char C, char B)
{
 if (n == 1)
 {
   printf("\n Move disk 1 from pole %c to pole %c", A,C);
   return;
 }
 toh(n-1,A,B,C);
 printf("\n Move disk %d from pole %c to pole %c", n,A,C);
 toh(n-1,B,C,A);
}
int main()
{
int n;
clrscr();
printf("enter the number of discs \n");
scanf("%d", &n);
toh(n,'A','C','B');
getch();
return(0);
}
```

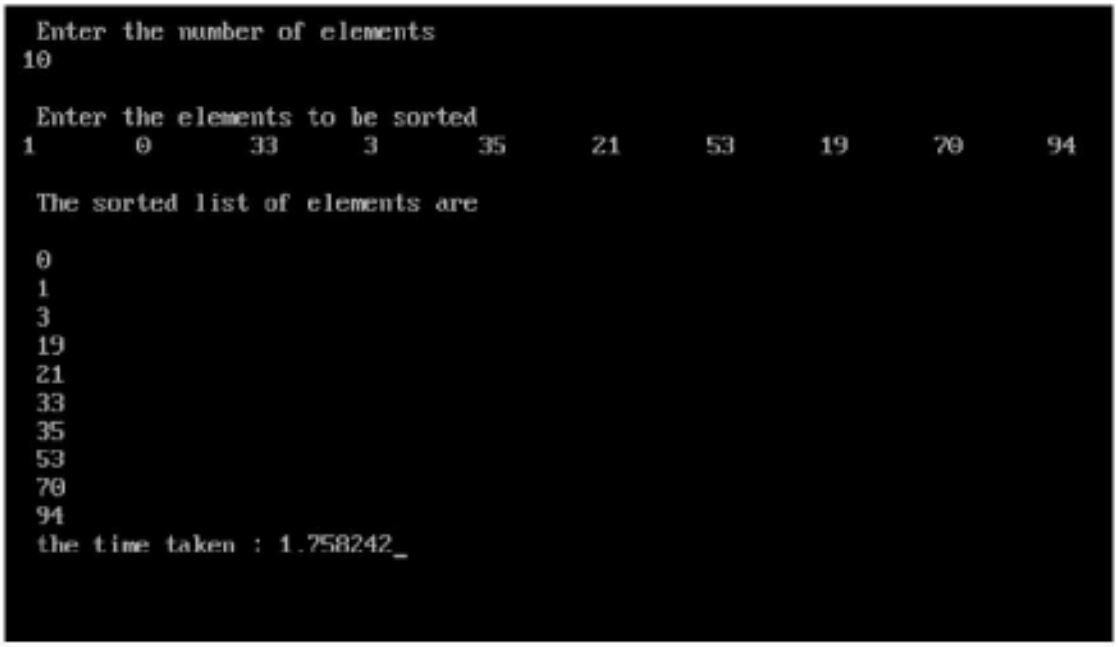
Output:

```
enter the number of discs
3

 Move disk 1 from pole A to pole C
 Move disk 2 from pole A to pole B
 Move disk 1 from pole C to pole B
 Move disk 3 from pole A to pole C
 Move disk 1 from pole B to pole A
 Move disk 2 from pole B to pole C
 Move disk 1 from pole A to pole C_
```

//**Labcycle 4**: WAP to sort a given set of numbers using selection sort algorithm. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<time.h>

void main()
{
int i,j,n, min,temp,k,a[1000];
clock_t begin,end;
clrscr();

 printf("\n Enter the number of elements\n");
 scanf("%d", &n);
 printf("\n Enter the elements to be sorted \n");
 for(k=0;k<n;k++)
 {
  a[k]= random(100);
  printf("%d\t", a[k]);
  }
 begin=clock();
 for(i=0;i<(n-1);i++)
 {
  min=i;
  delay(200);
  for(j=(i+1);j<n;j++)
  {
    if(a[j] < a[min])
          min =j;
  }
  if(min!=i)
  {
    temp = a[i];
    a[i] =a[min];
    a[min] = temp;
  }
 }
 end = clock();
 printf("\n The sorted list of elements are \n");
 for(i=0;i<n;i++)
 printf("\n %d", a[i]);
 printf("\n the time taken : %f", (end-begin) / CLK_TCK);
 getch();
 }
```

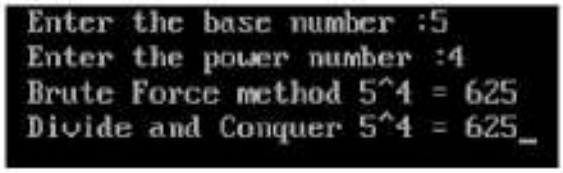
Output:


Enter the number of elements
10
Enter the elements to be sorted
1 0 33 3 35 21 53 19 70 94
The sorted list of elements are
0
1
3
19
21
33
35
53
70
94
the time taken : 1.758242

//**Labcycle 5**: WAP to find the value of a power n (where a and n are integers) using both brute force algorithm and divide and conquer based method

```
#include <stdio.h>
int brute_power(int a, int n);
int divide_power(int a, int n);
int main()
{
int a, n, result;
clrscr();
printf("Enter the base number :");
scanf("%d", &a);
printf("Enter the power number :");
scanf("%d", &n);
result = brute_power(a,n);
printf("Brute Force method %d^%d = %d\n", a,n,result);
result = divide_power(a,n);
printf("Divide and Conquer %d^%d = %d", a,n,result);
getch();
return 0;
}
int brute_power(int a,int n) // brute force method
{
if (n!=0)
 return(a * brute_power(a,(n-1)));
else
 return 1;
}

int divide_power(int a, int n) //divide and conquer method
{
   int temp;
   if (n == 0)
      return 1;
   temp = divide_power(a, n / 2);
   if (n % 2 == 0)
      return temp * temp;
   else
         return a * temp * temp;
}
```

Output:


Enter the base number :5
Enter the power number :4
Brute Force method 5^4 = 625
Divide and Conquer 5^4 = 625_

//**Labcycle 6:** WAP to sort a given set of elements using quick sort algorithm. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n.

```
#include <stdio.h>
#include<stdlib.h>
#include <conio.h>
#include <time.h>

void Exchange(int *p, int *q)
{
 int temp = *p;
 *p = *q;
 *q = temp;
}

void QuickSort(int a[], int low, int high)
{
 int i,j,pivot;
 delay(100);
 if(low >= high)
 return;
 pivot = low;
 i = low + 1;
 j = high;
 while(i <= j)
 {
  while(a[i] <= a[pivot])
   i++;
   while(a[j] > a[pivot])
   j--;
   if(i<j)
   {
   Exchange(&a[i],&a[j]);
   }
 }
 Exchange(&a[j],&a[pivot]);
 QuickSort(a,low,j-1);
 QuickSort(a,j+1,high);
 }

 void main()
 {
  int n, a[1000],k;
  clock_t begin, end;
  float timetaken;
  clrscr();
```

```
 printf("\n Enter how many elements \n");
 scanf("%d", &n);
 printf("\n Enter the random numbers :\n");
 for(k=1; k<=n;k++)
 {
  scanf("%d", &a[k]);
 }
 begin = clock();
 QuickSort(a,1,n);
 end = clock();
 timetaken =(double)(end - begin) / CLOCKS_PER_SEC;
 printf("\n sorted numbers are :\n");
 for(k=1;k<=n;k++)
  printf("%d \t", a[k]);
  printf("\n Time taken is %1f", timetaken);
  getch();
}
```

Output:

```
Enter the number of elements: 8

The elements are 46 40 89 40 54 63 25 13
        25      13      40      46      54      63      89      40
        13      25      40      46      54      63      89      40
        13      25      40      46      40      63      89      54
        13      25      40      46      40      54      89      63
        13      25      40      46      40      54      63      89

The sorted array is 13 25 40 46 40 54 63 89

Time taken is 0.000000 seconds_
```

//**Labcycle 7**: WAP to find the binomial co-efficient C(n,k), [where n and k are integers and n > k] using brute force based algorithm and also dynamic programming based algorithm

Brute force method #The value of C(n, k) can be recursively calculated using the following standard
formula for Binomial Coefficients. C(n, k) = C(n-1, k-1) + C(n-1, k) # C(n, 0) = C(n, n) = 1

```
#include<stdio.h>
#include<conio.h>
int min(int a, int b);
// Returns value of Binomial Coefficient C(n, k)
int binomialCoeff_BF(int n, int k);
int binomialCoeff_DC(int n, int k);

int binomialCoeff_BF(int n, int k)
{
 if (k > n)
 return 0;
 if(( k == 0) || ( k == n))
 return 1;
return binomialCoeff_BF(n - 1, k - 1) + binomialCoeff_BF(n - 1, k) ;
}

// Divide and Conquer method
//re-computations of the same subproblems can be avoided by constructing a temporary 2D-array C[][]
// in a bottom-up manner.
// uses Overlapping Subproblems concept

int binomialCoeff_DC(int n, int k)
{
   int C[20][20];
   int i, j;
  /* loop to clear junk values in the C matrix */
   for(i=0;i<=n;i++)
   {
    for(j=0;j<=k;j++)
    C[i][j]=0;
   }
   /************************************************/
   // Calculate value of Binomial Coefficient in bottom up manner
   for (i = 0; i <= n; i++) {
          for (j = 0; j <= min(i, k); j++) {
             // Base Cases
```

```
            if (j == 0 || j == i)
                C[i][j] = 1;
            // Calculate value using previously stored values
            else
                C[i][j] = C[i - 1][j - 1] + C[i - 1][j];
        }
    }
    /*************************Pascal Triangle************/
    for(i=0;i<=n;i++)
     {
          for(j=0;j<=k;j++)
           if(C[i][j] !=0)
           printf("\t%d ", C[i][j]);
           printf("\n");
     }
    /*****************************************************/
    return C[n][k];
}

// A utility function to return minimum of two integers
int min(int a, int b)
{
 if(a < b)
  return a;
 else
  return b;
 }


/* The program to test above function*/
void main()
{
    int n, k;
    clrscr();
    printf("Enter the values of n and k\n");
    scanf("%d  %d", &n, &k);
    printf("Binomial Coefficient using Brute Force of C(%d, %d) is %d ", n, k, binomialCoeff_BF(n, k));
    printf("Binomial Coefficient using Divide and Conquer of C(%d, %d) is %d ", n, k, binomialCoeff_DC(n,
k));
    getch();
}
```

Output:

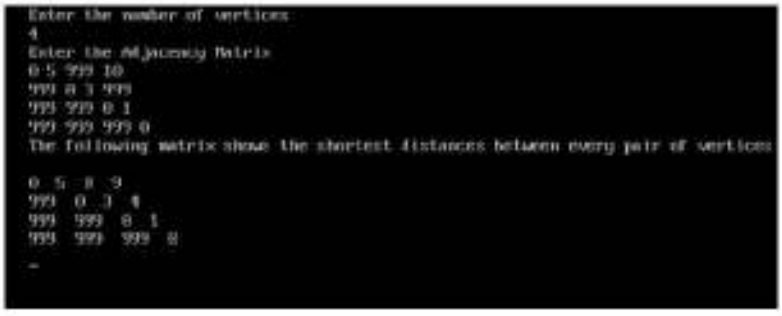
```
Enter the values of n and k
5 2
Binomial Coefficient using Brute Force of C(5, 2) is 10         1
        1       1
        1       2       1
        1       3       3
        1       4       6
        1       5       10
Binomial Coefficient using Dynamic programming of C(5, 2) is 10
```

// **LabCycle 08** WAP to implement Floyd's algorithm and find the lengths of the shortest paths from every pair of vertices in a given weighted graph.

```
#include <stdio.h>
#define INF 99

int dist[20][20],n;
// A function to print the solution matrix
void printSolution();
void floyd ();
void floyd ()
{
  int i, j, k;
  for (k = 0; k < n; k++) {
        // Pick all vertices as source one by one
        for (i = 0; i < n; i++) {
          // Pick all vertices as destination for the
          // above picked source
          for (j = 0; j < n; j++) {
                if (dist[i][j] > (dist[i][k] + dist[k][j])
                  && (dist[k][j] != INF)
                        && (dist[i][k] != INF))
                  dist[i][j] = dist[i][k] + dist[k][j];
          }
        }
  }

  // Print the shortest distance matrix
  printSolution();
}

/* A utility function to print solution */
void printSolution()
{
 int i, j;
  printf("The following matrix shows the shortest distances between every pair of vertices \n");
  for ( i = 0; i < n; i++) {
```

```
            for (j = 0; j < n; j++) {
               if (dist[i][j] == 99)
                        printf("INF  ");
               else
                        printf("%d  ",dist[i][j]);
            }
            printf("\n");
   }
}

// Driver's code
void main()
{
 int i,j;
 clrscr();
 printf("Enter the number of vertices\n");
 scanf("%d", &n);

printf("Enter the Adjacency Matrix\n");
 for(i=0;i<n;i++)
  for(j=0;j<n;j++)
    scanf("%d",&dist[i][j]);
 floyd();
 getch();
}
```

Output:



//Labcycle 9:WAP to evaluate a polynomial using brute-force based algorithm and using Horner's rule and compare their performances

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

// returns value of poly[0]x(n-1) + poly[1]x(n-2) + .. + poly[n-1]
int homer (int a[], int n, int x);
int polynomial_BF(int z[], int x, int n);

int polynomial_BF(int poly[],int x,int n)
{
 int i,j, Sum, result=0;
 for (i=0;i<n;i++)
 {
 Sum = poly[i];
 for (j=0;j<(n-i-1);j++)
  {
    Sum = Sum * x;
  }
 // Adding the sum to the result
 result = result + Sum;
}
 return result;
}

int homer(int poly[], int n, int x)
{
   int i;
   int result = poly[0]; // Initialize result
   clrscr();
   // Evaluate value of polynomial using Homer's method
   for (i=1; i<n; i++)
   {
          result = result    *x + poly[i];
  }
   return result;
}

// Driver program to test above function.
int main()
{
   // Let us evaluate value of 2x3 - 6x2 + 2x - 1 for x = 3
   int poly[] = {2, -6, 2, -1};
   int x = 3;
   int n = 4;
   printf("Value of polynomial using Horner method is  %d", homer(poly, n, x));
   printf("Value of polynomial using Brute force method is  %d", polynomial_BF(poly, x, n));
```

```
    getch();
    return 0;
}
```

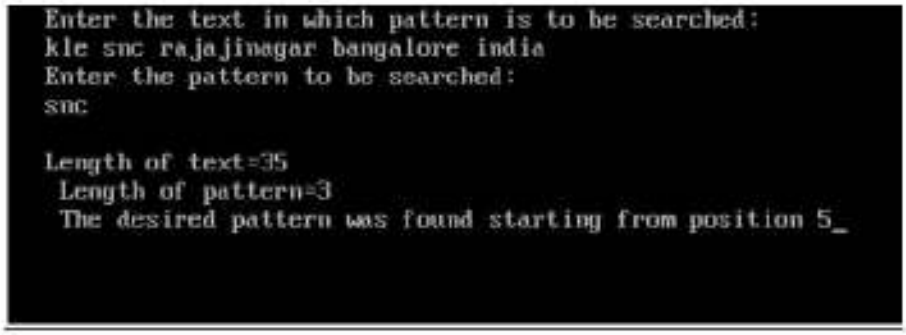
Output:

//Labcycle 10:WAP to solve the string matching problem using Boyer Moore approach.

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
#define MAX 500
int t[MAX];
void shifttable(char p[]) {
        int i,j,m;
        m=strlen(p);
        for (i=0;i<MAX;i++)
         t[i]=m;
        for (j=0;j<m-1;j++)
         t[p[j]]=m-1-j;
}
int horspool(char src[],char p[]) {
        int i,j,k,m,n;
        n=strlen(src);
        m=strlen(p);
        printf("\nLength of text=%d",n);
        printf("\n Length of pattern=%d",m);
        i=m-1;
        while(i<n) {
                k=0;
                while((k<m)&&(p[m-1-k]==src[i-k]))
                  k++;
                if(k==m)
                  return(i-m+1); else
                  i+=t[src[i]];
        }
        return -1;
}
void main() {
        char src[100],p[100];
        int pos;
        clrscr();
        printf("Enter the text in which pattern is to be searched:\n");
        gets(src);
        printf("Enter the pattern to be searched:\n");
        gets(p);
        shifttable(p);
        pos=horspool(src,p);
```

```
        if(pos>=0)
          printf("\n The desired pattern was found starting from position %d",pos+1); else
          printf("\n The pattern was not found in the given text\n");
        getch();
}
```

Output:



// **Labcycle 11:** Write a program to solve the String Matching Problem using KMP Algorithm

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

void computeLPSArray(char* pat, int M, int* lps);

// Prints occurrences of txt[] in pat[]
void KMPSearch(char* pat, char* txt)
{
	int lps[25];
	int i=0,j=0;
	int M = strlen(pat);
	int N = strlen(txt);

	computeLPSArray(pat, M, lps);

	while ((N - i) >= (M - j)) {
		if (pat[j] == txt[i]) {
			j++;
			i++;
		}

		if (j == M) {
			printf("Found pattern at index %d \n", ((i - j)+1));
			j = lps[j - 1];
		}

		// mismatch after j matches
		else if (i < N && pat[j] != txt[i]) {
			if (j != 0)
				j = lps[j - 1];
			else
				i = i + 1;
		}
	}
}

void computeLPSArray(char* pat, int M, int* lps)
{
	int len = 0;
	int i=1;
	lps[0] = 0;
	while (i < M) {
		if (pat[i] == pat[len]) {
			len++;
			lps[i] = len;
```

```
                i++;
            }
            else
            {
                if (len != 0) {
                    len = lps[len - 1];
                }
                else
                {
                    lps[i] = 0;
                    i++;
                }
            }
        }
}

void main()
{
        char txt[100],pat[100];
        clrscr();
        printf("Enter the main string:\n");
        gets(txt);
        printf("Enter the sub string that you want to search in the main string:\n");
        gets(pat);
        KMPSearch(pat, txt);
        getch();
}
```

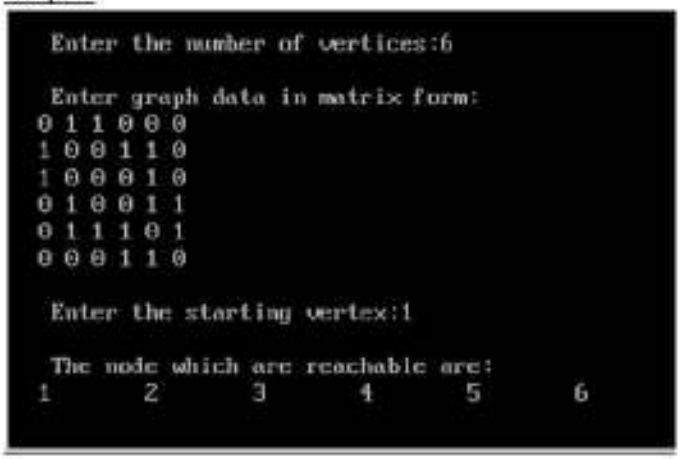
Output:

```
Enter the main string:
welcome to the world of programming!. welcome to the world of programming
Enter the sub string that you want to search in the main string:
welcome
Found pattern at index 1
Found pattern at index 39
```

// **LabCycle – 12** Program to print all the nodes reachable from a given starting node in a digraph using
// BFS

```
#include<stdio.h>
#include<conio.h>
int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;

void bfs(int v)
{
for(i=1;i<=n;i++)
 if(a[v][i] && !visited[i])
 q[++r]=i;
if(f<=r)
{
 visited[q[f]]=1;
 bfs(q[f++]);
}
}
void main()
{
int v;
clrscr();
printf("\n Enter the number of vertices:");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
 q[i]=0;
 visited[i]=0;
}
printf("\n Enter graph data in matrix form:\n");
for(i=1;i<=n;i++)
 for(j=1;j<=n;j++)
 scanf("%d",&a[i][j]);
printf("\n Enter the starting vertex:");
scanf("%d",&v);
bfs(v);
printf("\n The node which are reachable are:\n");
for(i=1;i<=n;i++)
 if(visited[i])
 printf("%d\t",i);
getch();
}
```

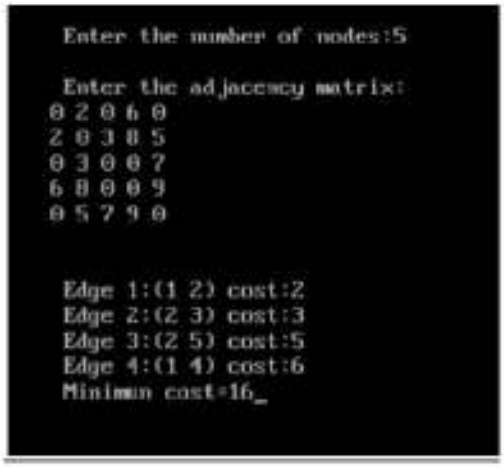
Output:


Enter the number of vertices:6
Enter graph data in matrix form:
0 1 1 0 0 0
1 0 0 1 1 0
1 0 0 0 1 0
0 1 0 0 1 1
0 1 1 1 0 1
0 0 0 1 1 0
Enter the starting vertex:1
The node which are reachable are:
1 2 3 4 5 6

// **Lab Cycle 13**. Write a program to find minimum cost spanning tree of a given undirected graph 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];
void main()
{
clrscr();
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 Minimun cost=%d",mincost);
getch();
}
```

Output:



//LabCycle 14: Write a Program to obtain the topological ordering of vertices in a given digraph. Compute the transitive closure of a given directed graph using Warshall's algorithm.

//LabCycle 14a: Write a Program to obtain the topological ordering of vertices in a given digraph.

```
#include<stdio.h>
#include<conio.h>
int temp[10],k=0;
void topo(int n,int indegree[10],int a[10][10])
  {
  int i,j;
  for(i=1;i<=n;i++)
   {
         if(indegree[i]==0)
          {
          indegree[i]=1;
           temp[++k]=i;
                  for(j=1;j<=n;j++)
                 {
                  if(a[i][j]==1&&indegree[j]!=-1)
                   indegree[j]--;
                 }
                 i=0;
          }
    }
  }
 void main()
  {
  int i,j,n,indegree[10],a[10][10];
  clrscr();
  printf("enter the number of vertices:");
  scanf("%d",&n);
  for(i=1;i<=n;i++)
  indegree[i]=0;
```

```
printf("\n enter the adjacency matrix\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{
  scanf("%d",&a[i][j]);
  if(a[i][j]==1)
  indegree[j]++;
}

topo(n,indegree,a);
if(k!=n)
printf("topological ordering is not possible\n");

else
 {
   printf("\n topological ordering is :\n");
   for(i=1;i<=k;i++)
   printf("v%d\t",temp[i]);
 }
getch();
}
```

Output:

```
 enter the number of vertices:6

  enter the adjacency matrix
0 0 0 0 0 0
1 0 1 0 0 0
0 0 0 1 1 0
0 0 0 0 0 0
0 0 0 1 0 1
0 0 0 0 0 0

  topological ordering is :
v2      v1      v3      v5      v4      v6
```

//LabCycle 14b: Compute the transitive closure of a given directed graph using Floyd Warshall's algorithm.

```
# include <stdio.h>
# include <conio.h>
int n,a[10][10],p[10][10];
void path()
{
int i,j,k;
for(i=0;i<n;i++)
for(j=0;j<n;j++)
p[i][j]=a[i][j];
for(k=0;k<n;k++)
for(i=0;i<n;i++)
for(j=0;j<n;j++)
if(p[i][k]==1&&p[k][j]==1) p[i][j]=1;
}
void main()
{
int i,j;
clrscr();
printf("Enter the number of nodes:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&a[i][j]);
path();
printf("\n The path matrix is shown below\n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
printf("%d ",p[i][j]);
printf("\n");
}
getch();
}
```

Output:

```
Enter the number of nodes:5

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

 The path matrix is shown below
0 1 1 1 1
0 0 1 1 1
0 0 0 0 1
0 0 0 0 1
0 0 0 0 0
```

//LabCycle 15: Write a program to find subset of a given set S={s1,s2,....sn} of n positive integers whose sum is equal to given positive integer d. For example if S={1,2,5,6,8} and d=9 then two solutions {1,2,6} and {1,8}. A suitable message is to be displayed if given problem doesn't have solution.

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

static int total_nodes;
void printValues(int A[], int size){
 int i;
  for ( i = 0; i < size; i++) {
    printf("%*d", 5, A[i]);
  }
  printf("\n");
}

void subset_sum(int s[], int t[], int s_size, int t_size, int sum, int ite, int const target_sum)
{
  int i;
  total_nodes++;
  if (target_sum == sum) {
    printValues(t, t_size);
    subset_sum(s, t, s_size, t_size - 1, sum - s[ite], ite + 1, target_sum);
    return;
  }
  else {
    for (i = ite; i < s_size; i++) {
          t[t_size] = s[i];
          subset_sum(s, t, s_size, t_size + 1, sum + s[i], i + 1, target_sum);
    }
  }
}

void generateSubsets(int s[], int size, int target_sum){
  int* tuplet_vector = (int*)malloc(size * sizeof(int));
```

```
  subset_sum(s, tuplet_vector, size, 0, 0, 0, target_sum);
  free(tuplet_vector);
}

int main(){
  int set[50],n=0,i,sum=0;
  int size =0;
  clrscr();
  printf("Enter the size of set of numbers\n");
  scanf("%d", &n);
  printf("Enter the elements of the set \n");
  for(i=0;i<n;i++)
    scanf("%d", &set[i]);
  size = n;
  printf("Enter the sum for creating subsets\n");
  scanf("%d", &sum);
  printf("The set is ");
  printValues(set , size);
  generateSubsets(set, size, sum);
  printf("Total Nodes generated %d\n", total_nodes);
  getch();
  return 0;
}
```

Output:

```
Enter the size of set of numbers
6
Enter the elements of the set
3
4
5
6
7
8
Enter the sum for creating subsets
9
The set is     3    4    5    6    7    8
     3    6
     4    5
Total Nodes generated 60
_
```