Ayush Thada
16BCE1333
Lab 11(BFS & DFS)

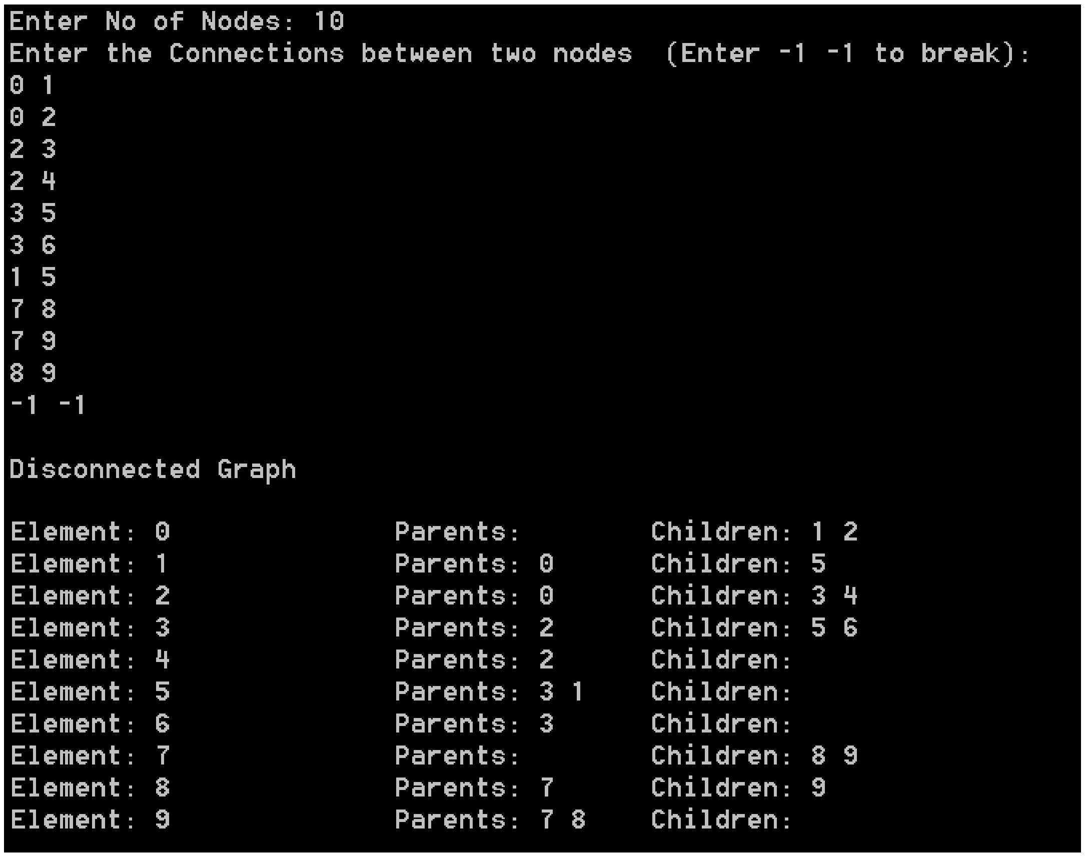
<u>Question 1:</u> Let G=(V,E) be a graph, where V is a vertex set and E is an edge set. A graph is connected when there is a path between every pair of vertices. Design an algorithm to determine whether the graph G is connected or not. Implement your algorithm in any programming language.

CODE

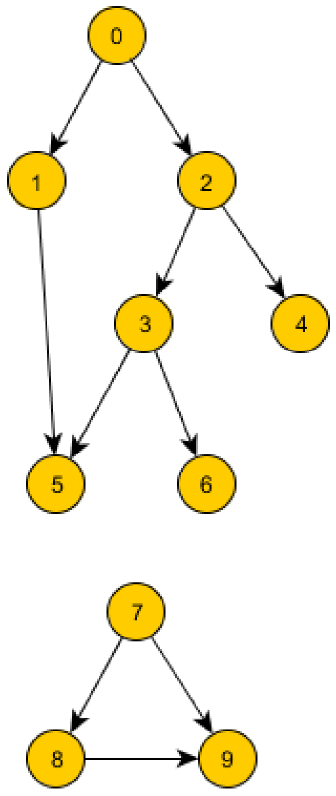
```
#include<stdio.h>
#include<stdlib.h>
#define N 50
typedef struct node
{
    int key;
    int nop;
    struct node **parents;
    int noc;
    struct node **children;
    int dist;
} NODE;

NODE arr[N];

void graphCreate(int n)
{
    int i,j,origin,destin;
    for(i=0 ; i<n ; i++)
    {
        arr[i].key=i;
        arr[i].nop=0;
        arr[i].parents = (NODE**)malloc(sizeof(NODE*) * arr[i].nop);
        arr[i].noc=0;
        arr[i].children = (NODE**)malloc(sizeof(NODE*) * arr[i].noc);
       arr[i].dist = 0;
    }

    printf("Enter the Connections between two nodes  (Enter -1 -1 to break):\n");
    for(i=0 ; i<((n*(n-1))/2) ; i++)
    {
        scanf("%d%d",&origin , &destin);
        if(origin<0 || destin<0)
            break;
        else
        {
            /*Setting Parent for Destin node */
            NODE *Ptemp[++(arr[destin].nop)];    //Copy previous Parent Elements
            for(j=0 ; j<arr[destin].nop -1  ; j++)
                Ptemp[j] = arr[destin].parents[j];

            Ptemp[j] = (&arr[origin]);          //Add new Parent

            free(arr[destin].parents);
            arr[destin].parents = (NODE**)malloc(sizeof(NODE*) * arr[i].nop);

            for(j=0 ; j<arr[destin].nop ;j++)
                arr[destin].parents[j] = Ptemp[j];  //Copy all parents to main
parents list

            free(Ptemp);

            /*Setting Children for origin node*/
            NODE *Ctemp[++(arr[origin].noc)];     //Copy previous Child Elements
            for(j=0 ; j<arr[origin].noc -1  ; j++)
                Ctemp[j] = arr[origin].children[j];

            Ctemp[j] = (&arr[destin]);               //Add new Child

            free(arr[origin].children);
            arr[origin].children = (NODE**)malloc(sizeof(NODE*) * arr[i].noc);

            for(j=0 ; j<arr[origin].noc ;j++)
                arr[origin].children[j] = Ctemp[j];  //Copy all children to main
children list

            free(Ctemp);
        }
    }
}

void graphDiscCheck(int n)
{
    int i,count=0;
    for(i=0 ; i<n ;i++)
    {
        if(arr[i].nop<=0)
            count++;

        if(count>=2)
        {
            printf("\n\nDisconnected Graph\n\n");
            return;
        }
```

```c
    }
    printf("\nConnected Graph\n");
}

void graphDisplay(int n)
{
    int i,j;
    for(i=0 ; i<n ;i++)
    {
        printf("Element: %d\t",arr[i].key);

        printf("\tParents: ");
            for(j=0 ; j<arr[i].nop ; j++)
                printf("%d ",(arr[i].parents[j])->key);

        printf("\tChildren: ");
            for(j=0 ; j<arr[i].noc ; j++)
                printf("%d ",(arr[i].children[j])->key);

        printf("\n");
    }
}

int main()
{
    int n;
    printf("Enter No of Nodes: ");
    scanf("%d",&n);
    graphCreate(n);
    graphDiscCheck(n);
    graphDisplay(n);
}
```

OUTPUT: **Input:**



```
Enter No of Nodes: 10
Enter the Connections between two nodes  (Enter -1 -1 to break):
0 1
0 2
2 3
2 4
3 5
3 6
1 5
7 8
7 9
8 9
-1 -1

Disconnected Graph

Element: 0              Parents:        Children: 1 2
Element: 1              Parents: 0      Children: 5
Element: 2              Parents: 0      Children: 3 4
Element: 3              Parents: 2      Children: 5 6
Element: 4              Parents: 2      Children:
Element: 5              Parents: 3 1    Children:
Element: 6              Parents: 3      Children:
Element: 7              Parents:        Children: 8 9
Element: 8              Parents: 7      Children: 9
Element: 9              Parents: 7 8    Children:
```



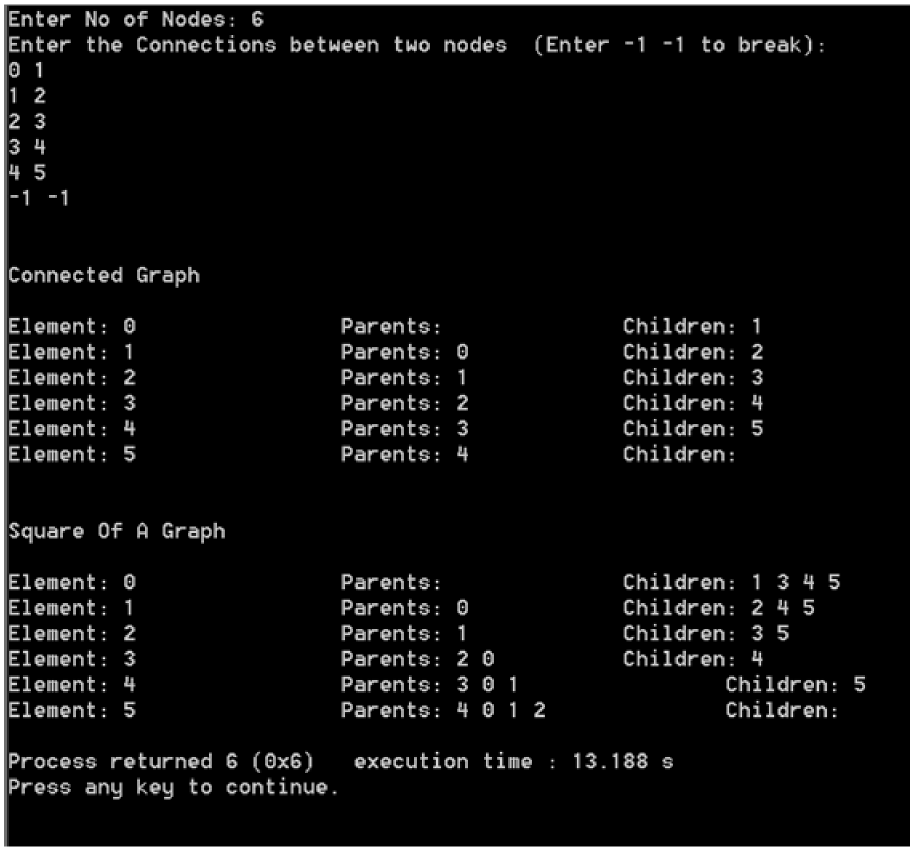
Question 2: The SQUARE of a directed graph G=(V,E) is the graph G2=(V,E2) such that (u,v)εE2 iff G contains a path with at most two edges between u and v. Design an algorithm for computing G2 from G f. Implement your algorithm any programming language.

CODE

In above Code I have added a distance attribute in structure NODE to keep record of distance from main root node.

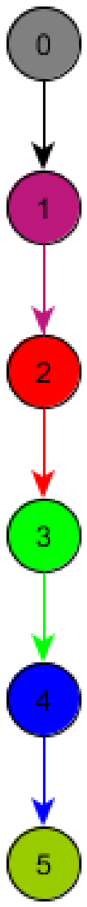
```
/*Function to calculate any power of graph .Here q attribute represents power.*/

void graphPower(int n, int q)
{
    int i, j, k;
    int save[n-q],count=0;
    for(i=0 ; i<n ; i++)
    {
        for(k=i+1 ; k<n ; k++)
        {
            if(arr[k].dist -arr[i].dist > q)
            {
                /*Setting Parent for a[k] node */
                NODE *Ptemp[++(arr[k].nop)];        //Copy previous Parent
Elements
                for(j=0 ; j<arr[k].nop -1  ; j++)
                    Ptemp[j] = arr[k].parents[j];

                Ptemp[j] = (&arr[i]);               //Add new Parent

                free(arr[k].parents);
                arr[k].parents = (NODE**)malloc(sizeof(NODE*) * arr[k].nop);

                for(j=0 ; j<arr[k].nop ;j++)
                    arr[k].parents[j] = Ptemp[j];  //Copy all parents to main
parents list

                free(Ptemp);

                /*Setting Children for arr[i] node */
                NODE *Ctemp[++(arr[i].noc)];        //Copy previous Child
Elements
                for(j=0 ; j<arr[i].noc -1  ; j++)
                    Ctemp[j] = arr[i].children[j];

                Ctemp[j] = (&arr[k]);               //Add new Child

                free(arr[i].children);
                arr[i].children = (NODE**)malloc(sizeof(NODE*) * arr[i].noc);

                for(j=0 ; j<arr[i].noc ;j++)
                    arr[i].children[j] = Ctemp[j];  //Copy all children to main
children list

                free(Ctemp);

                save[count++]=k;
            }
        }
    }
    for(i=0 ; i< count ; i++)
        arr[save[i]].dist=1;
}


/* Rewrite the main function in this way */

int main()
{
    int n;
    printf("Enter No of Nodes: ");
    scanf("%d",&n);
    graphCreate(n);
    graphDiscCheck(n);
    graphDisplay(n);
    graphPower(n, 2);
    printf("\n\nSquare Of A Graph\n\n");
    graphDisplay(n);
}
```

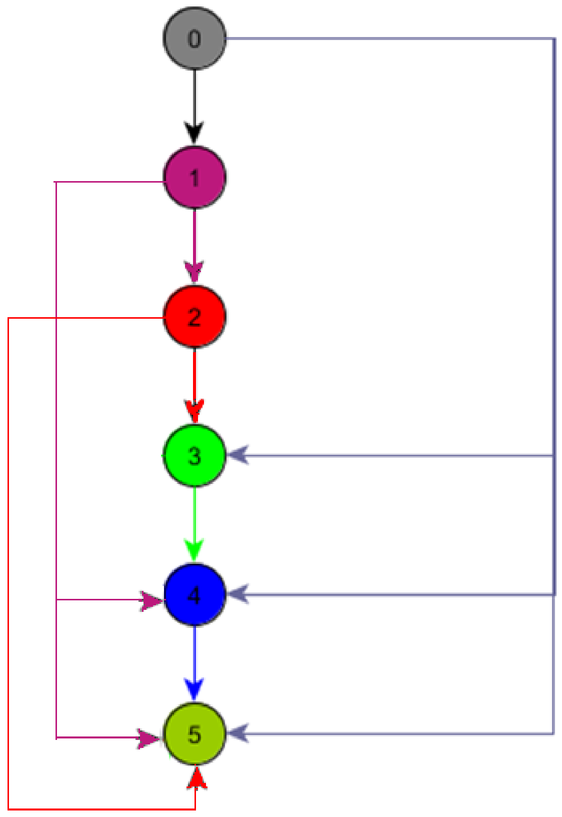
OUTPUT



```
Enter No of Nodes: 6
Enter the Connections between two nodes  (Enter -1 -1 to break):
0 1
1 2
2 3
3 4
4 5
-1 -1


Connected Graph

Element: 0                Parents:              Children: 1
Element: 1                Parents: 0            Children: 2
Element: 2                Parents: 1            Children: 3
Element: 3                Parents: 2            Children: 4
Element: 4                Parents: 3            Children: 5
Element: 5                Parents: 4            Children:


Square Of A Graph

Element: 0                Parents:              Children: 1 3 4 5
Element: 1                Parents: 0            Children: 2 4 5
Element: 2                Parents: 1            Children: 3 5
Element: 3                Parents: 2 0          Children: 4
Element: 4                Parents: 3 0 1                Children: 5
Element: 5                Parents: 4 0 1 2              Children:

Process returned 6 (0x6)   execution time : 13.188 s
Press any key to continue.
```

INPUT **OUTPUT**


0
1
2
3
4
5


0
1
2
3
4
5