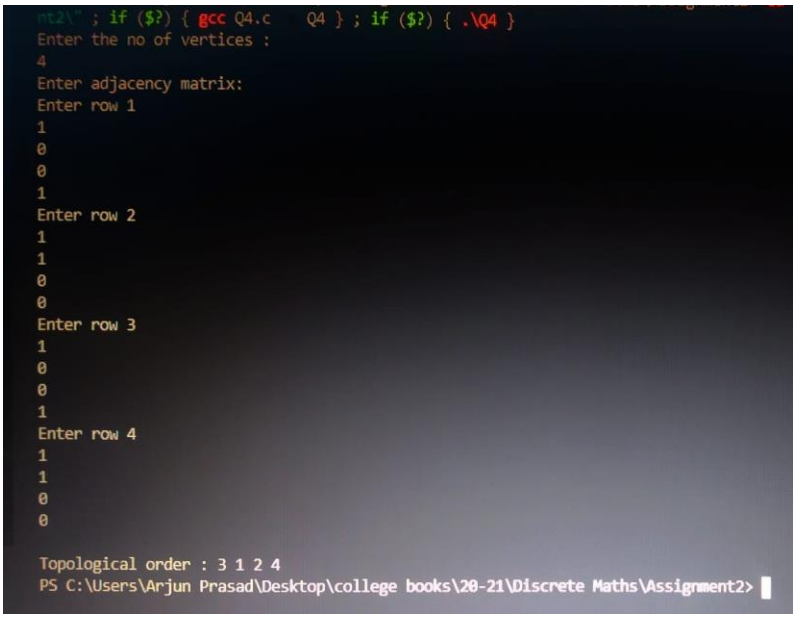
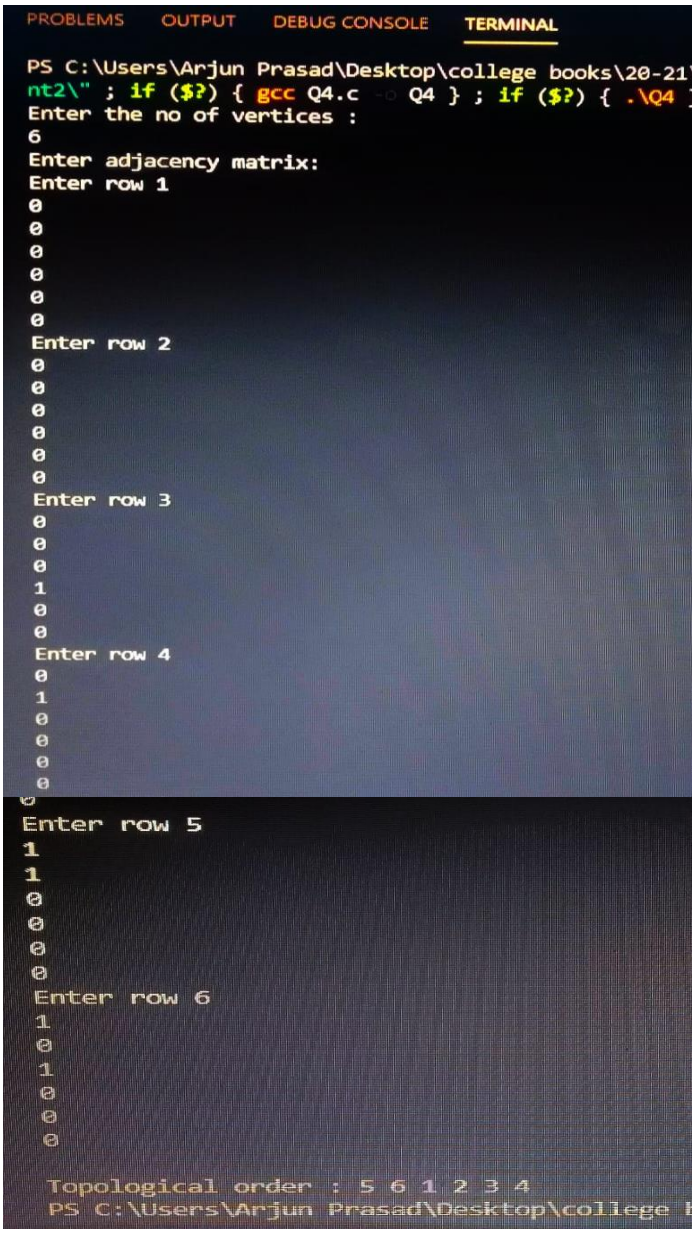
```c
// Ouestion 4

#include <stdio.h>

int main()
{

    int v, ctr = 0;
    int indeg[10], flag[10], adjacency_mat[10][10];

    printf("Enter the no of vertices : \n");
    scanf("%d", &v);

    printf("Enter adjacency matrix:\n");
    for(int i = 0; i < v ; i++)
    {
        printf("Enter row %d\n",i+1);

        for(int j = 0; j < v ; j++)
            {
                scanf("%d", &adjacency_mat[i][j]);
            }
    }

    for(int i = 0; i < v; i++)
    {
        indeg[i] = 0;
        flag[i] = 0;
    }

    for(int i = 0; i < v; i++)
    {
        for(int j = 0; j < v; j++)
        {
            indeg[i] +=  adjacency_mat[j][i];
        }
    }

    printf("\nTopological order : ");

    while(ctr < v)
    {
        for(int k = 0; k < v; k++)
        {
            if((indeg[k] == 0) && (flag[k] == 0))
```

```
            {
                printf("%d ",(k+1));
                flag[k] = 1;
            }

            for(int i = 0; i < v; i++)
            {
                if(adjacency_mat[i][k] == 1)
                {
                    indeg[k]--;
                }
            }
        }

        ctr++;
    }

    return 0;
}
```

// OUTPUT




PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PS C:\Users\Arjun Prasad\Desktop\college books\20-21
nt2\" ; if ($?) { gcc Q4.c -o Q4 } ; if ($?) { .\Q4
Enter the no of vertices :
6
Enter adjacency matrix:
Enter row 1
0
0
0
0
0
0
Enter row 2
0
0
0
0
0
0
Enter row 3
0
0
0
1
0
0
Enter row 4
0
1
0
0
0
0
0
Enter row 5
1
1
0
0
0
0
Enter row 6
1
0
1
0
0
0
Topological order : 5 6 1 2 3 4
PS C:\Users\Arjun Prasad\Desktop\college

```c
// Question 5

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>

#define N 1000

typedef struct Queue
{
    int array[N];
    int front, rear, size;
} Queue;

Queue *createQueue()
{
    Queue *q = (Queue *)malloc(sizeof(Queue));
    q->front = q->size = 0;
    q->rear = N-1;

    return q;
}

int enQueue(Queue *q, int val)
{
    if (q->rear == N)
    {
        printf("Queue Full");
        return -1;
    }

    q->rear = (q->rear + 1) % N;
    q->array[q->rear] = val;
    q->size++;
}

int deQueue(Queue *q)
{
    if (isEmpty(q))
    {
        printf("Queue is empty");
        return -1;
    }
```

```c
    int item = q->array[q->front];
    q->front = (q->front + 1) % N;
    q->size--;
    return item;
}

int isEmpty(Queue *q)
{
    return (!q->size);
}

void printQueue(Queue *q)
{
    if (isEmpty(q))
    {
        printf("Queue Empty");
        return;
    }

    for (int i = q->front; i < q->rear; ++i)
    {
        printf("%d ", q->array[i]);
    }
}

typedef struct Node
{
    int vertex;
    struct Node *next;
} Node;

Node *createNode(int v)
{
    Node *newNode = (Node *)malloc(sizeof(Node));

    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}

typedef struct Graph
{
    int numVertices;
    int *visited;
    Node **adjList;
```

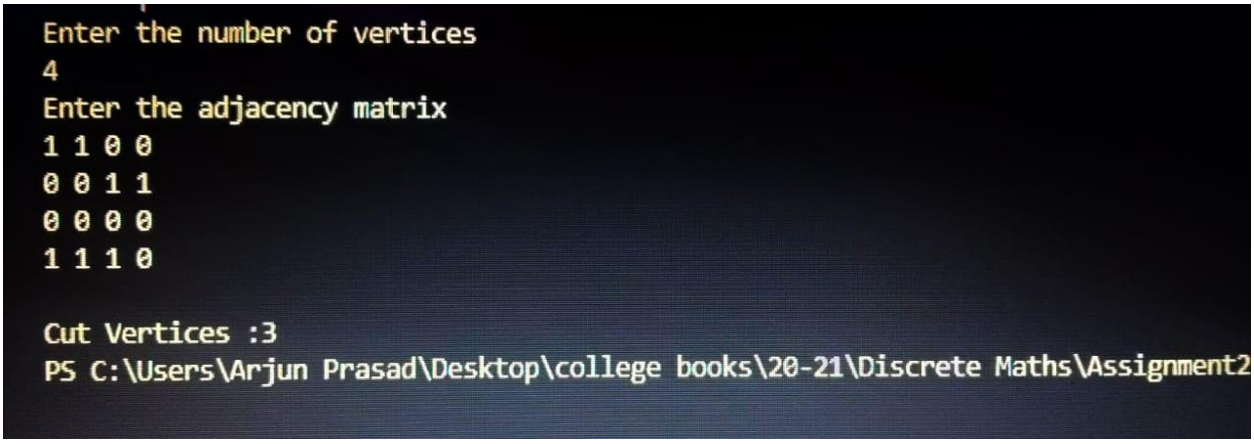
```
} Graph;

Graph *createGraph(int vertices, int adjMat[][vertices])
{

    Graph *graph = (Graph *)malloc(sizeof(Graph));
    graph->numVertices = vertices;
    graph->visited = (int *)malloc(vertices * sizeof(int));
    graph->adjList = (Node **)malloc(vertices * sizeof(Node *));

    for (int i = 0; i < vertices; ++i)
    {
        graph->adjList[i] = NULL;
        graph->visited[i] = 0;

        for (int j = 0; j < vertices; ++j)
        {
            if (adjMat[i][j])
            {
                Node *node = createNode(j);
                node->next = graph->adjList[i];
                graph->adjList[i] = node;
            }
        }
    }
    return graph;
}

void printGraph(Graph *graph)
{
    for (int i = 0; i < graph->numVertices; ++i)
    {
        printf("%d ", i);
        Node *temp = graph->adjList[i];

        while (temp)
        {
            printf(" -> %d", temp->vertex);
            temp = temp->next;
            // printf("in %d", i);
        }
        printf("\n");
    }
}
```

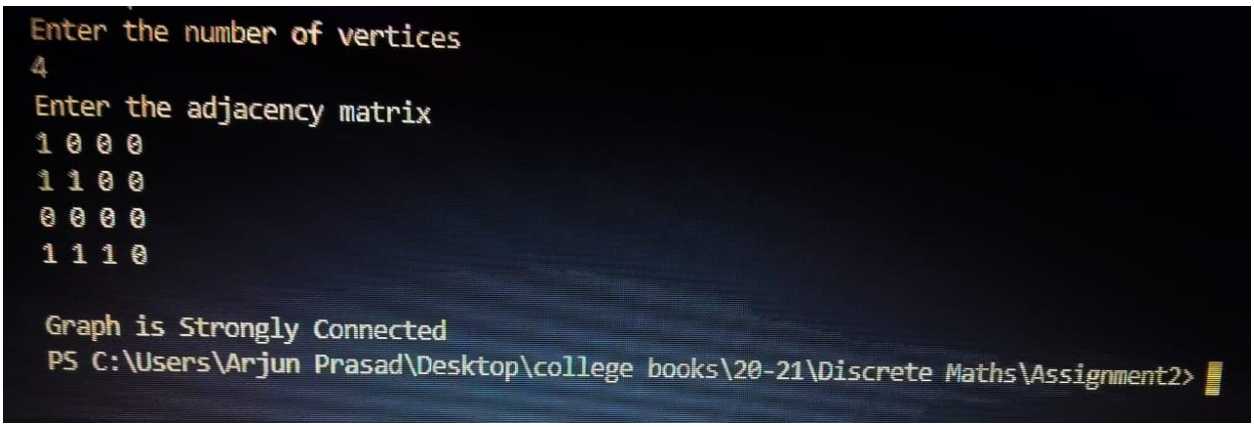
```c
int visited[N], tin[N], low[N], ap[N];
int timer = 0;
int stronglyConnected = 1;

int min(int a, int b)
{
    return (a < b) ? a : b;
}

void getMatrix(int n, int M[][n])
{
    printf("Enter the adjacency matrix \n");
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            scanf("%d", &M[i][j]);
}

void cutPoint(Graph *graph, int v, int p)
{
    visited[v] = 1;
    tin[v] = low[v] = timer++;

    int children = 0;

    Node *node = graph->adjList[v];
    while (node)
    {
        int to = node->vertex;

        if (visited[to])
        {
            low[v] = min(low[v], tin[to]);
        }
        else
        {
            cutPoint(graph, to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && ~p)
                ap[v] = 1, stronglyConnected = 0;
            ++children;
        }
        node = node->next;
    }

    if (p == -1 && children > 1)
```

```
        ap[v] = 1, stronglyConnected;
}

int main()
{
    int vertices;
    printf("Enter the number of vertices \n");
    scanf("%d", &vertices);

    int adjMat[vertices][vertices];
    getMatrix(vertices, adjMat);

    Graph *graph = createGraph(vertices, adjMat);
    Queue *cutPoints = createQueue();

    memset(visited, 0, sizeof(visited));
    memset(tin, -1, sizeof(tin));
    memset(low, -1, sizeof(low));
    memset(ap, 0, sizeof(ap));

    for (int i = 0; i < vertices; ++i)
    {
        if (!visited[i])
        {
            cutPoint(graph, i, -1);
        }
    }

    if (stronglyConnected)
    {
        printf("\nGraph is Strongly Connected\n");
    }

    else
    {
        printf("\nCut Vertices :");
        for (int i = 0; i < vertices; ++i)
        {
            if (ap[i])
            {
                printf("%d ", i);
            }
        }
        printf("\n");
    }
```

```
    return 0;
}
```

//OUTPUT

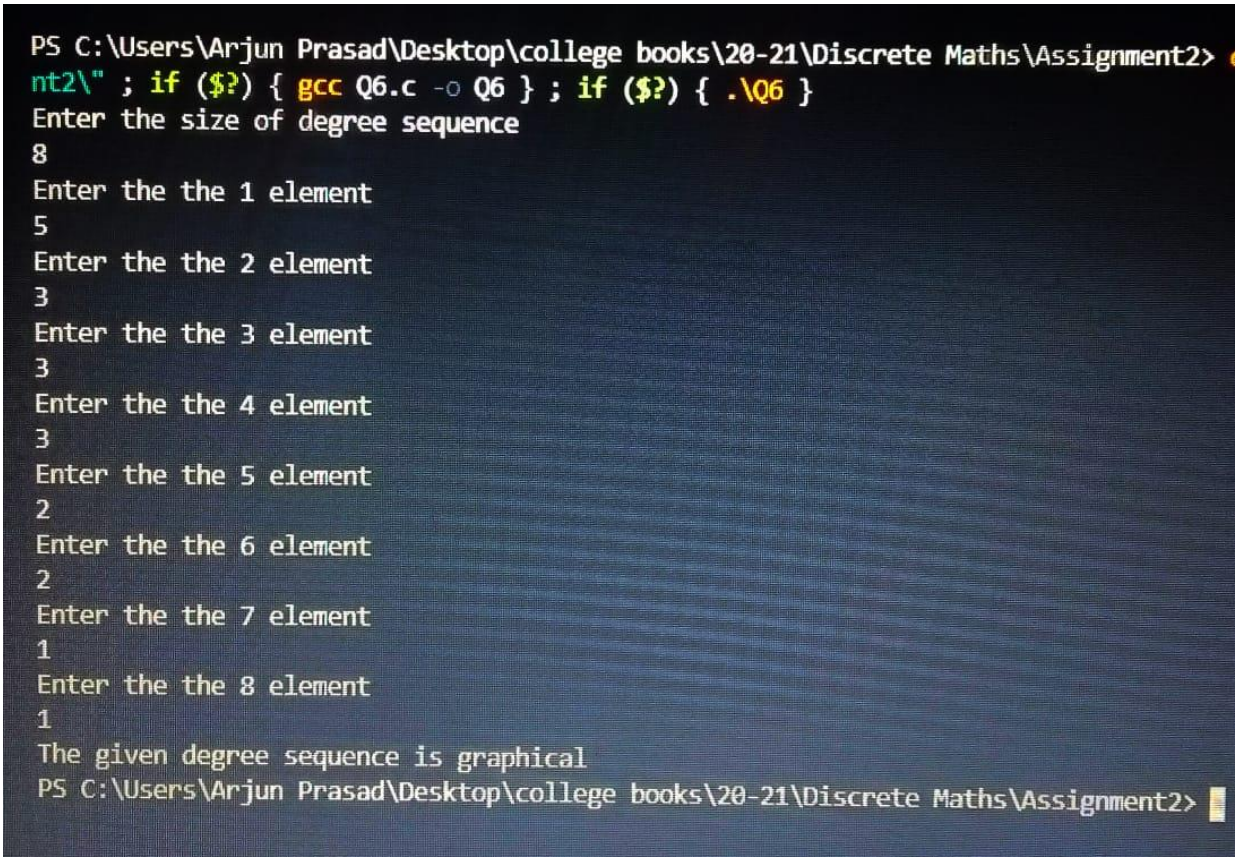


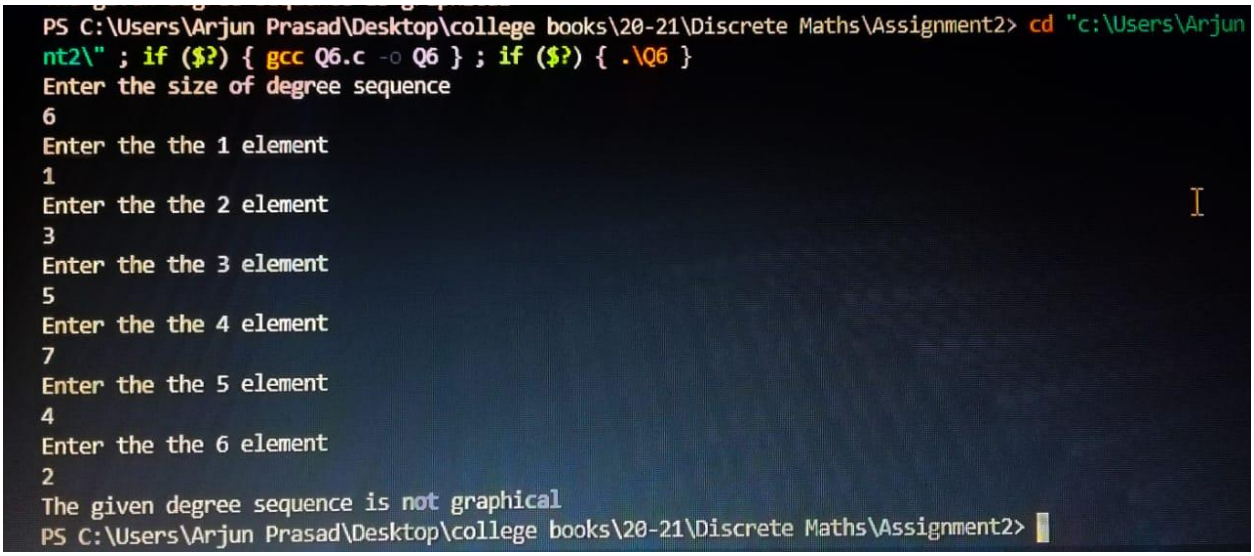


```c
// Question 6


#include<stdio.h>
int GraphExists(int arr[],int n){
    int a;
    while(1){

        for(int i = 0;i<n;++i)
        {
            for(int j = i + 1;j < n;++j)
            {
                if (arr[i] < arr[j])
                {
                    a = arr[i];
                    arr[i] = arr[j];
                    arr[j] = a;
                }
            }
        }

        if(arr[0] == 0 )
        {
            return 1;
        }

        int v = arr[0];
        n = n - 1;
        for(int j = 0;j<n;j++)
        {
            arr[j] = arr[j+1];
        }
        if(v > n )
        {
            return 0;
        }
        for (int i = 0; i < v; i++)
        {
            arr[i]--;

            // Checking for -ve elements encountered
            if (arr[i] < 0)
                return 0;
        }
```

```
    }
}
int main()
{
  int arr[20];
  int n;

  printf("Enter the size of degree sequence \n");
  scanf("%d",&n);

  for(int i = 0;i<n;i++)
  {
      int u, k = i+1;
      printf("Enter the the %d element \n",k);
      scanf("%d",&u);
      arr[i] = u;
  }

    int x = GraphExists(arr,n);
    if(x == 1)
    {
        printf("The given degree sequence is graphical");}
    else if(x == 0)
    {
        printf("The given degree sequence is not graphical");
    }


}
```

// OUTPUT

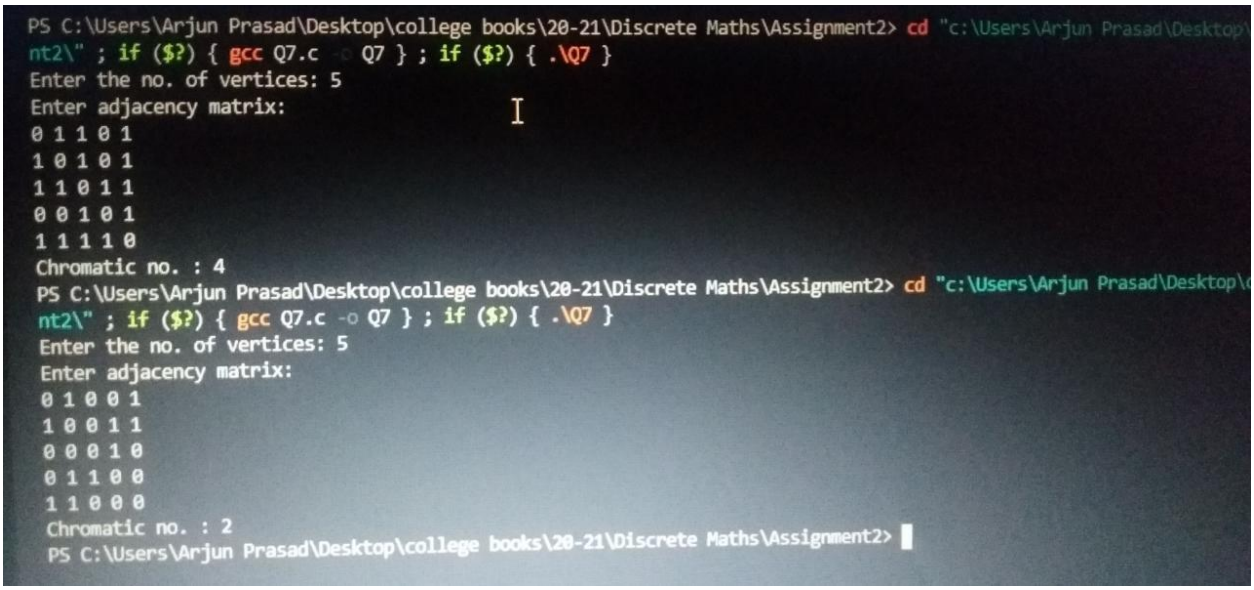

PS C:\Users\Arjun Prasad\Desktop\college books\20-21\Discrete Maths\Assignment2> c
nt2\" ; if ($?) { gcc Q6.c -o Q6 } ; if ($?) { .\Q6 }
Enter the size of degree sequence
8
Enter the the 1 element
5
Enter the the 2 element
3
Enter the the 3 element
3
Enter the the 4 element
3
Enter the the 5 element
2
Enter the the 6 element
2
Enter the the 7 element
1
Enter the the 8 element
1
The given degree sequence is graphical
PS C:\Users\Arjun Prasad\Desktop\college books\20-21\Discrete Maths\Assignment2>


PS C:\Users\Arjun Prasad\Desktop\college books\20-21\Discrete Maths\Assignment2> cd "c:\Users\Arjun
nt2\" ; if ($?) { gcc Q6.c -o Q6 } ; if ($?) { .\Q6 }
Enter the size of degree sequence
6
Enter the the 1 element
1
Enter the the 2 element
3
Enter the the 3 element
5
Enter the the 4 element
7
Enter the the 5 element
4
Enter the the 6 element
2
The given degree sequence is not graphical
PS C:\Users\Arjun Prasad\Desktop\college books\20-21\Discrete Maths\Assignment2>

```c
// Question 7


#include<stdio.h>
int main()
{
    int n;
    printf("Enter the no. of vertices: ");
    scanf("%d",&n);
    int a[n][n];
    printf("Enter adjacency matrix:\n");
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
   int  k=1;
    for(int i=0;i<n;i++)
    {
        int flag=0;
        if(i!=0)
        {
            for(int j=0;j<i;j++)
            {
            if(a[j][i]!=1)
             {
                a[i][i]=a[j][i];
                flag=1;
             }
            }
        }
        if(flag==0)
        {
            a[i][i]=k+1;
            k++;
        }
        for(int j=0;j<n;j++)
        {
            if(a[i][j]==0)
            a[i][j]=a[i][i];
        }
```

```
    }
    printf("Chromatic no. : %d",k-1);
    return 0;
}
```

// OUTPUT

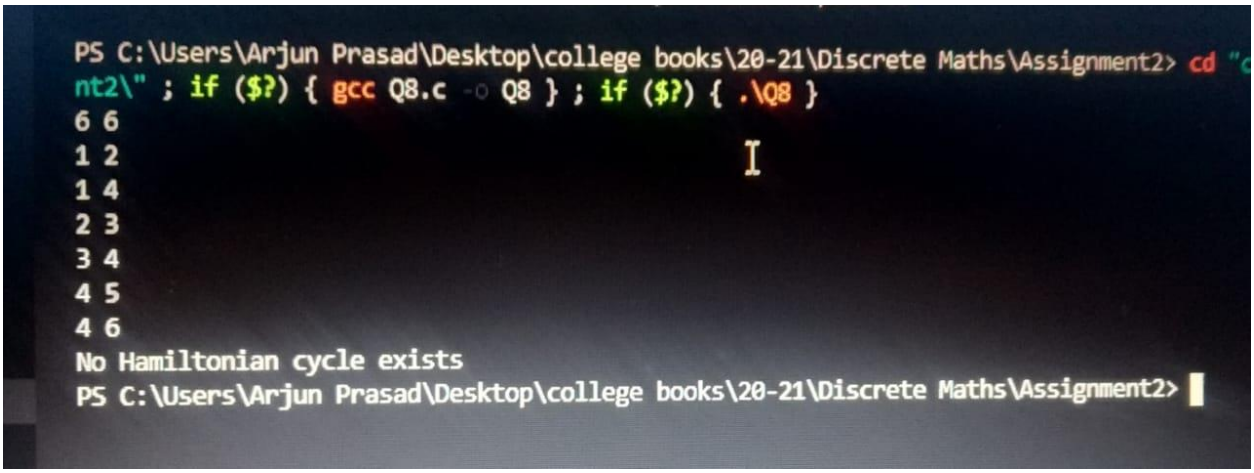


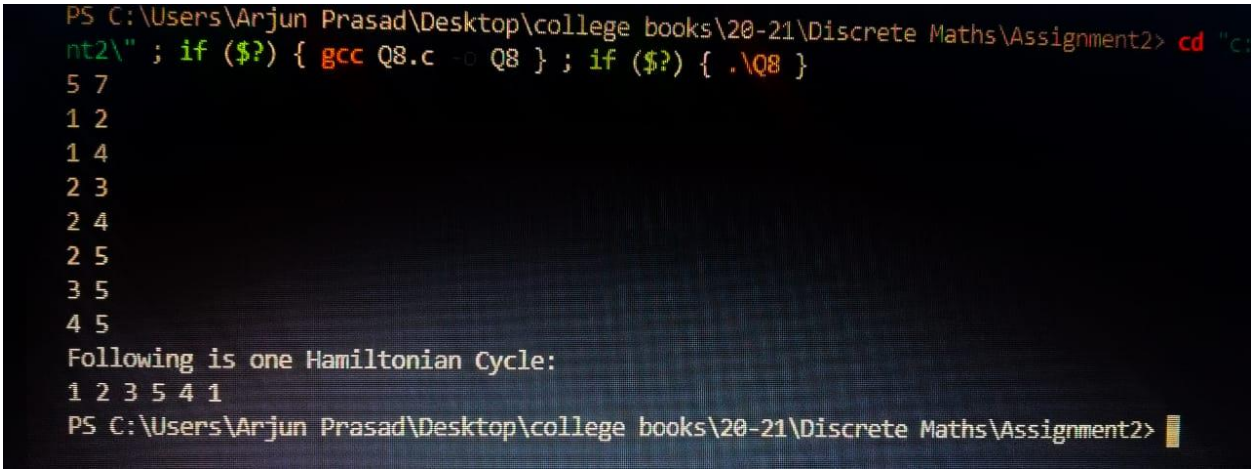
```c
// Question 8

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

#define V 1000
int matrix[V][V],num_vertices,num_edges;

bool check(int v, int path[], int pos)
{
    if (matrix[path[pos - 1]][ v ] == 0)  {
        return false;
    }
    for (int i = 0; i < pos; i++)  {
        if (path[i] == v)  {
            return false;
        }
    }
    return true;
}

bool HamiltonianCycle(int path[], int pos)
{

    if (pos == num_vertices)
    {
        if (matrix[path[pos - 1]][path[0]] == 1)
            return true;
        else
            return false;
    }

    for (int v = 1; v < num_vertices; v++)   {
        if (check(v, path, pos)) {
            path[pos] = v;
            if (HamiltonianCycle(path, pos + 1) == true) {
                return true;
            }
            path[pos] = -1;
        }
```

```
    }
    return false;
}

void Solution(int path[])
{
    printf("Following is one Hamiltonian Cycle: \n");
    for (int i = 0; i < num_vertices; i++) {
        printf("%d ",path[i]+1);
    }
    printf("%d\n",path[0]+1);
}

bool Hamiltonian()
{
    int path[num_vertices];
    for (int i = 0; i < num_vertices; i++) {
        path[i] = -1;
    }
    path[0] = 0;
    if (HamiltonianCycle(path, 1) == false )
    {
        printf("No Hamiltonian cycle exists");
        return false;
    }
    Solution(path);
    return true;
}


int main(){
    //taking number of vertices and edges as input
    scanf("%d%d",&num_vertices,&num_edges);
    for(int i=0;i<num_vertices;i++){
        for(int j=0;j<num_vertices;j++){
            matrix[i][j] = 0;
        }
    }
    int x,y;
    //taking list of edges as input
    for(int i=0;i<num_edges;i++){
       scanf("%d%d",&x,&y);
       x--;
       y--;
       matrix[x][y] = matrix[y][x]=1;
```

```
    }
    Hamiltonian();
    return 0;
}
```

// OUTPUT

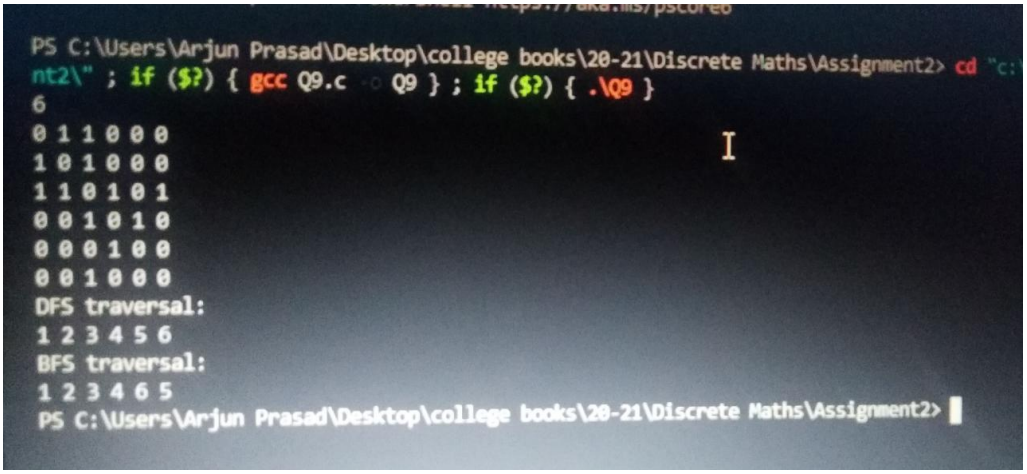


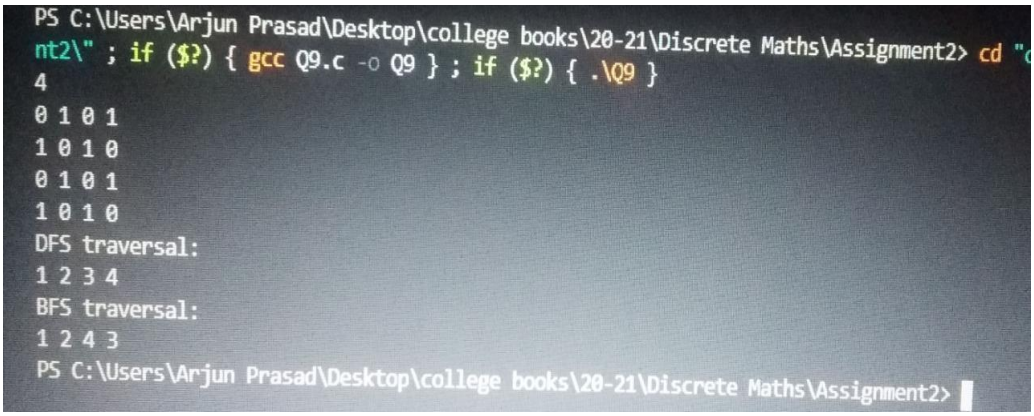


```c
// Question 9

#include <stdio.h>
#define N 1000
int n,visited_dfs[N],visited_bfs[N],matrix[N][N],queue[N],szq=0;

void erase(int arr[],int pos, int sz)
{
    for(int p=pos-1;p<sz;p++){
        arr[p]=arr[p+1];
    }
}

void push_back(int x)
{
    queue[szq]=x;
    ++szq;
}

void dfs(int v)
{
    visited_dfs[v]=1;
    printf("%d ",v+1);
    for(int i=0;i<n;i++)
    {
        if(matrix[v][i]&&!visited_dfs[i])
        {
            dfs(i);
        }
    }
}

void bfs(int st)
{
    visited_bfs[st]=1;
    push_back(st);
```

```
    queue[1]=0;
    int ve;
    while(szq)
    {
        ve = queue[0];
        printf("%d ",ve+1);
        erase(queue,1,szq);
        --szq;
        for(int i=0;i<n;i++)
        {
            if(matrix[ve][i]&&!visited_bfs[i])
            {
                push_back(i);
                visited_bfs[i]=1;
            }
        }
    }
}

int main()
{
    //taking value of n
    scanf("%d",&n);
    //adjacency matrix
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            scanf("%d",&matrix[i][j]);
        }
    }
    printf("DFS traversal:\n");
    dfs(0);
    printf("\n");
    printf("BFS traversal:\n");
    bfs(0);
    return 0;
}
```

// OUTPUT


PS C:\Users\Arjun Prasad\Desktop\college books\20-21\Discrete Maths\Assignment2> cd "c:
nt2\" ; if ($?) { gcc Q9.c -o Q9 } ; if ($?) { .\Q9 }
6
0 1 1 0 0 0
1 0 1 0 0 0
1 1 0 1 0 1
0 0 1 0 1 0
0 0 0 1 0 0
0 0 1 0 0 0
DFS traversal:
1 2 3 4 5 6
BFS traversal:
1 2 3 4 6 5
PS C:\Users\Arjun Prasad\Desktop\college books\20-21\Discrete Maths\Assignment2>


PS C:\Users\Arjun Prasad\Desktop\college books\20-21\Discrete Maths\Assignment2> cd "
nt2\" ; if ($?) { gcc Q9.c -o Q9 } ; if ($?) { .\Q9 }
4
0 1 0 1
1 0 1 0
0 1 0 1
1 0 1 0
DFS traversal:
1 2 3 4
BFS traversal:
1 2 4 3
PS C:\Users\Arjun Prasad\Desktop\college books\20-21\Discrete Maths\Assignment2>