22/02/24 Week 9

a) Write a program to traverse a graph using BFS

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

int n, i, j, visited[MAX_VERTICES],
queue[MAX_VERTICES], front=0, rear=0;
void bfs(int v) {
  visited[v]=1;
  queue[rear++]=v;
  while(front<rear){
    int current = queue[front++];
    printf("%d\t", current);
    int i;
  for(i=0; i<n; i++) {
    if(adj[current][i] && !visited[i]) {
      visited[i]=1;
      queue[rear++]=i;
    }
  }
}
}
int main(){
  int v;
  printf("Enter the number of vertices");
  scanf("%d", &n);
  for(i=0; i<n; i++)
  visited[i]=0;
}
printf("Enter the graph data in matrix form
starting vertex: ");
for(i=0; i<n; i++)
  for(j=0; j<n; j++)
    scanf("%d", &adj[i][j]);
```

```
printf("Enter the starting vertex:");
scanf("%d", &v);
dfs(v);
for(i=0; i<n; i++){
  if(!visited[i]){
    printf("\nBFS is not possible. Not all
    nodes are reachable\n");
    return 0;
  }
}
return 0;
}
```

Output:

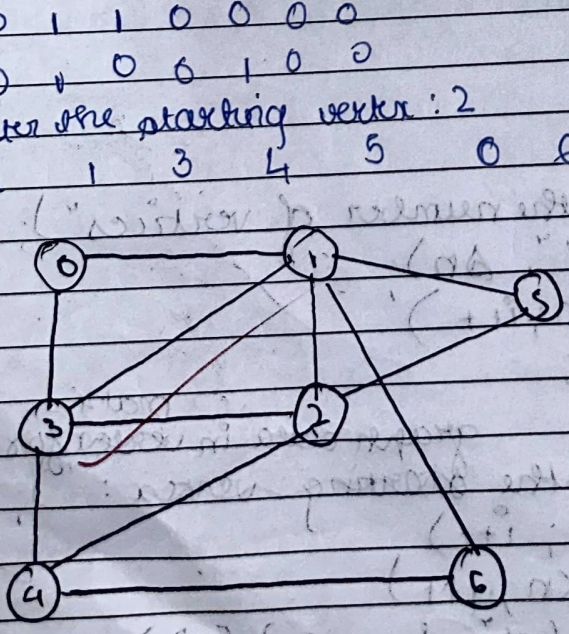
Enter the number of vertices: 7

Enter the adjacency matrix:

```
0 1 0 1 0 0 0
1 0 1 1 0 1 1
0 1 0 1 1 1 0
1 1 1 0 1 0 0
0 0 1 1 0 0 1
0 1 1 0 0 0 0
0 1 0 0 1 0 0
```

Enter the starting vertex: 2

2 1 3 4 5 0 6



8) Write a program
program is conn

```
#include<stdio
#include<stdb
#define MAX
void dfs(int q
int n, int
int vertex){
visited[ver
int i;
for(i=0;
if(adj
dfs(adj
}
}
bool isConnec
[MAX_VERTICE
bool visite
dfs(adj
int i;
for(i=0
if(!
ret
}
return
}
```

Q) Write a program to check whether the given program is connected or not using DFS method

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_VERTICES 10

void dfs (int graph [MAX_VERTICES] [MAX VERTICES],
int num_vertices, bool visited [MAX_vertices],
int vertex) {
    visited [vertex] = true;
    int i;
    for (i=0; i< numVertices ; ++i ) {
      if (graph [vertex [i]] == 1 && ! visited [i]) {
        dfs (graph, num_vertices, visited, i );
    }
  }
}
bool isconnected (int graph [MAX_VERTICES]
[MAX_VERTICES], int num_vertices ) {
    bool visited [MAX_VERTICES] = {false};
    dfs ( graph, num_vertices, visited, 0);
    int i;
    for (i=0; i< num_vertices ; ++i) {
      if ( ! visited [i]) {
        return false;
    }
  }
  return true;
}
```

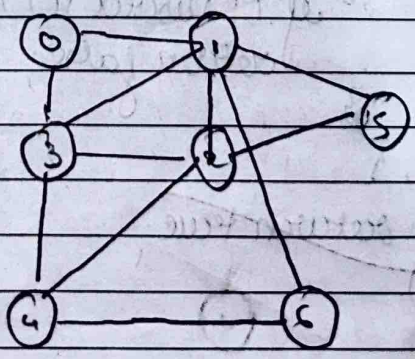
```
int main() {
    int num_vertices;
    printf("Enter the number of vertices: ");
    scanf("%d", &num_vertices);
    int i, j;
    int graph[MAX_VERTICES][MAX_VERTICES];
    printf("Enter the adjacency matrix:\n");
    for (int i=0; i<num_vertices; ++i) {
        for (j=0; j<num_vertices; ++j) {
            scanf("%d", &graph[i][j]);
        }
    }
    if (isconnected(graph, num_vertices)) {
        printf("The graph is connected.\n");
    }
    else {
        printf("The graph is not connected\n");
    }
    return 0;
}
```

Output:

Enter the number of vertices: 7

Enter the adjacency matrix:

```
0 1 0 1 0 0 0
1 0 1 1 0 1 1
0 1 0 1 1 1 0
1 1 1 0 1 0 0
0 0 1 1 0 0 1
0 1 1 0 0 0 0
0 1 0 0 1 0 0
```



The graph is connected

Rotate list

```
int Getlength (struct listNode * head)
{
  if (head == NULL)
    return 0;
  return 1 + Getlength (head -> next);
}
struct listNode * rotateRight (struct listNode *head, int k)
{
  if (head == NULL || k == 0)
    return head;
int length = Getlength (head);
if (length == 1)
    return head;
for (int i = 0; i < k % length; i++)
{
  struct listNode * p = head;
  while (p -> next -> next != NULL)
{
  p = p -> next;
}
struct listNode * a = (struct listNode *) malloc
  (sizeof (struct listNode));
a -> val = p -> next -> val;
a -> next = head;
head = a;
p -> next = NULL;
}
return head;
}
```

Output

head = [1, 2, 3, 4, 5] k = 2

Output: [4, 5, 1, 2, 3]

head = [0, 1, 2] k = 4

Output: [2, 0, 1]