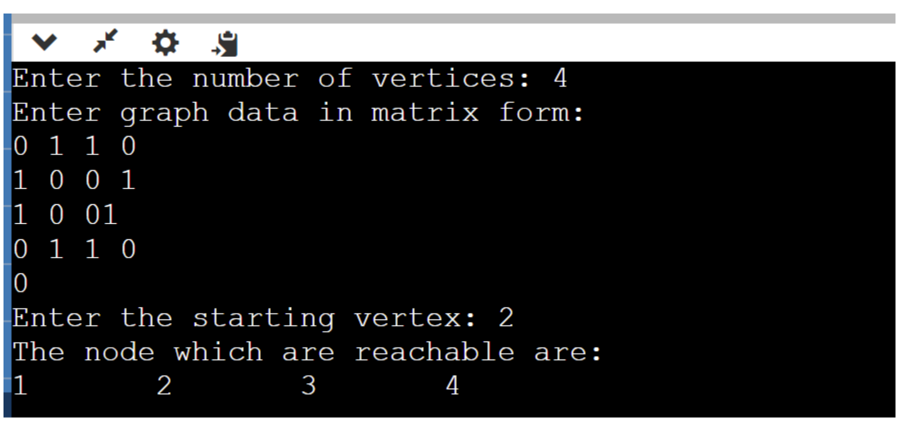
9a) Write a program to traverse a graph using BFS method.
9b) Write a program to check whether given graph is connected or not using DFS method.

BFS

```
#include <stdio.h>

 int n, i, j, visited[10], queue[10], front = -1, rear = -1;

int adj[10][10];


void bfs(int v)

{

   for (i = 1; i <= n; i++)

       if (adj[v][i] && !visited[i])

           queue[++rear] = i;

   if (front <= rear)

   {

       visited[queue[front]] = 1;

       bfs(queue[front++]);

   }

}


void main()

{

   int v;

   printf("Enter the number of vertices: ");

   scanf("%d", &n);

   for (i = 1; i <= n; i++)

   {

       queue[i] = 0;

       visited[i] = 0;
```

```
  }
  printf("Enter graph data in matrix form:    \n");
  for (i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
      scanf("%d", &adj[i][j]);
  printf("Enter the starting vertex: ");
  scanf("%d", &v);
  bfs(v);
  printf("The node which are reachable are:    \n");
  for (i = 1; i <= n; i++)
    if (visited[i])
      printf("%d\t", i);
    else
      printf("BFS is not possible. Not all nodes are reachable");

}
```

OUTPUT:



```
Enter the number of vertices: 4
Enter graph data in matrix form:
0 1 1 0
1 0 0 1
1 0 01
0 1 1 0
0
Enter the starting vertex: 2
The node which are reachable are:
1       2       3       4
```

DFS

```
#include<stdio.h>

#include<conio.h>

int a[20][20], reach[20], n;

void dfs(int v) {

   int i;

   reach[v] = 1;

   for (i = 1; i <= n; i++)

      if (a[v][i] && !reach[i]) {

         printf("\n %d->%d", v, i);

         dfs(i);

      }

}


int main(int argc, char **argv) {

   int i, j, count = 0;

   printf("\n Enter number of vertices:");

   scanf("%d", &n);

   for (i = 1; i <= n; i++) {

      reach[i] = 0;

      for (j = 1; j <= n; j++)

         a[i][j] = 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]);
```

```
    dfs(1);

    printf("\n");

    for (i = 1; i <= n; i++) {

        if (reach[i])

            count++;

    }

    if (count == n)

        printf("\n Graph is connected");

    else

        printf("\n Graph is not connected");

    return 0;

}
```

OUTPUT:

```
 Enter number of vertices:4

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

 1->2
 2->4
 4->3

 Graph is connected
```

```
 Enter number of vertices:4

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


 Graph is not connected
```

HACKERRANK QUESTION: (Reverse Doubly Linked List - B1)

```
DoublyLinkedListNode* reverse(DoublyLinkedListNode* llist) {
    DoublyLinkedListNode* temp = llist;
    DoublyLinkedListNode* curr = temp;
    DoublyLinkedListNode* prev = NULL;
    DoublyLinkedListNode* nextOne = NULL;

    while(curr != NULL) {
        nextOne = curr->next;
        curr->next = prev;
        prev = curr;
        curr = nextOne;
    }
    return prev;
}
```

HACKERRANK QUESTION: (Trees - B1)

Code:

```
void findLeaves(struct TreeNode* node, int** leafValues, int* size, int* capacity) {
    if (node == NULL) {
        return;
    }
```

```
    if (node->left == NULL && node->right == NULL) {
        if (*size >= *capacity) {
            *capacity *= 2;
            *leafValues = (int*) realloc(*leafValues, *capacity * sizeof(int));
        }
        (*leafValues)[(*size)++] = node->val;
    }

    findLeaves(node->left, leafValues, size, capacity);
    findLeaves(node->right, leafValues, size, capacity);
}

bool leafSimilar(struct TreeNode* root1, struct TreeNode* root2) {
    int *leaves1 = (int*) malloc(sizeof(int) * 10);
    int size1 = 0, capacity1 = 10;

    int *leaves2 = (int*) malloc(sizeof(int) * 10);
    int size2 = 0, capacity2 = 10;

    findLeaves(root1, &leaves1, &size1, &capacity1);
    findLeaves(root2, &leaves2, &size2, &capacity2);

    if (size1 != size2) {
        free(leaves1);
        free(leaves2);
        return false;
    }

    for (int i = 0; i < size1; i++) {
        if (leaves1[i] != leaves2[i]) {
            free(leaves1);
            free(leaves2);
            return false;
        }
    }

    free(leaves1);
    free(leaves2);
```

```
    return true;
}
```

Output: