Problem: Graph connectivity using dfs

```
#include<stdio.h>


int visit[100];
int a[100][100];
int n;



void dfs(int v)
{
        int i;
        visit[v]=1;
        for (i=1;i<=n;i++)
        {
                if((a[v][i]==1) &&(visit[i]==0))
                        dfs(i);
        }
}
int check(int v)
{
  if(n==v)
  return 1;
  if(visit[v]==0)
  return 0;
  check(v+1);
}
void create_graph()
{
    int i;int j;
    while(1)
    {
```

```
    printf("Enter the source and the destination vertex of the edge\n");
    scanf("%d %d",&i,&j);
    if(i==0 && j==0) //to stop taking input
     break;
    a[i][j]=1;       //for undirected graph: Its should be a[i][j]=a[j][i]=1;
   }
}



int main()
{
        int i;int v;int k;

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

        create_graph();
        printf("Enter the source vertex\n");
        scanf("%d",&v);
        dfs(v);
 int success=check(0);
 if(success)
 printf("%s\n","All nodes are reachable");
 else
 printf("%s\n","Nodes arnt reachable");
        return 0;
 }
```

```
graphs1
Enter the number of vertices
4
Enter the source and the destination vertex of the edge
0 1
Enter the source and the destination vertex of the edge
0 2
Enter the source and the destination vertex of the edge
1 2
Enter the source and the destination vertex of the edge
0 0
Enter the source vertex
0
Nodes arnt reachable

Press any key to continue . . .
```

Problem: Graph transversal using bfs

```
// BFS algorithm in C


#include <stdio.h>

#include <stdlib.h>

#define SIZE 40


struct queue {

  int items[SIZE];

  int front;

  int rear;

};


struct queue* createQueue();

void enqueue(struct queue* q, int);

int dequeue(struct queue* q);

void display(struct queue* q);

int isEmpty(struct queue* q);
```

```
void printQueue(struct queue* q);

struct node {
  int vertex;
  struct node* next;
};
struct node* createNode(int);
struct Graph {
  int numVertices;
  struct node** adjLists;
  int* visited;
};
void bfs(struct Graph* graph, int startVertex) {
  struct queue* q = createQueue();

  graph->visited[startVertex] = 1;
  enqueue(q, startVertex);

  while (!isEmpty(q)) {
    printQueue(q);
    int currentVertex = dequeue(q);
    printf("\nVisited %d\n", currentVertex);

    struct node* temp = graph->adjLists[currentVertex];

    while (temp) {
      int adjVertex = temp->vertex;

      if (graph->visited[adjVertex] == 0) {
        graph->visited[adjVertex] = 1;
        enqueue(q, adjVertex);
```

```
      }
      temp = temp->next;
    }
  }
}
struct node* createNode(int v) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->vertex = v;
  newNode->next = NULL;
  return newNode;
}
struct Graph* createGraph(int vertices) {
  struct Graph* graph = malloc(sizeof(struct Graph));
  graph->numVertices = vertices;
  graph->adjLists = malloc(vertices * sizeof(struct node*));
  graph->visited = malloc(vertices * sizeof(int));
  int i;
  for (i = 0; i < vertices; i++) {
    graph->adjLists[i] = NULL;
    graph->visited[i] = 0;
  }
  return graph;
}
void addEdge(struct Graph* graph, int src, int dest) {
  struct node* newNode = createNode(dest);
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;
  newNode = createNode(src);
  newNode->next = graph->adjLists[dest];
  graph->adjLists[dest] = newNode;
}
```

```
struct queue* createQueue() {
  struct queue* q = malloc(sizeof(struct queue));
  q->front = -1;
  q->rear = -1;
  return q;
}
int isEmpty(struct queue* q) {
  if (q->rear == -1)
    return 1;
  else
    return 0;
}
void enqueue(struct queue* q, int value) {
  if (q->rear == SIZE - 1)
    printf("\nQueue is Full!!");
  else {
    if (q->front == -1)
      q->front = 0;
    q->rear++;
    q->items[q->rear] = value;
  }
}
int dequeue(struct queue* q) {
  int item;
  if (isEmpty(q)) {
    printf("Queue is empty");
    item = -1;
  } else {
    item = q->items[q->front];
    q->front++;
    if (q->front > q->rear) {
```

```
      printf("Resetting queue ");
      q->front = q->rear = -1;
    }
  }
  return item;
}
void printQueue(struct queue* q) {
  int i = q->front;

  if (isEmpty(q)) {
    printf("Queue is empty");
  } else {
    printf("\nQueue contains \n");
    for (i = q->front; i < q->rear + 1; i++) {
      printf("%d ", q->items[i]);
    }
  }
}
int main() {
  int n;
  printf("Enter the size of graph:");
  scanf("%d",&n);
  struct Graph* graph = createGraph(n);
  int p,q;
  while(1)
  {
    printf("Enter source and destination:");
    scanf("%d %d",&p,&q);
    if(p<0 || p>n || q<0 || q>n)
    break;
    addEdge(graph,p,q);
```

```
  }
  bfs(graph, 0);
  return 0;
}
```

graphs

```
Enter the size of graph:10
Enter source and destination:0 1
Enter source and destination:0 2
Enter source and destination:0 3
Enter source and destination:1 4
Enter source and destination:4 7
Enter source and destination:7 9
Enter source and destination:3 5
Enter source and destination:3 6
Enter source and destination:5 7
Enter source and destination:5 2
Enter source and destination:6 7
Enter source and destination:6 8
Enter source and destination:8 9
Enter source and destination:-1 1-

Queue contains
0 Resetting queue
Visited 0

Queue contains
3 2 1
Visited 3

Queue contains
2 1 6 5
Visited 2

Queue contains
1 6 5
Visited 1

Queue contains
6 5 4
Visited 6

Queue contains
5 4 8 7
Visited 5

Queue contains
4 8 7
Visited 4

Queue contains
8 7
Visited 8

Queue contains
7 9
Visited 7
```