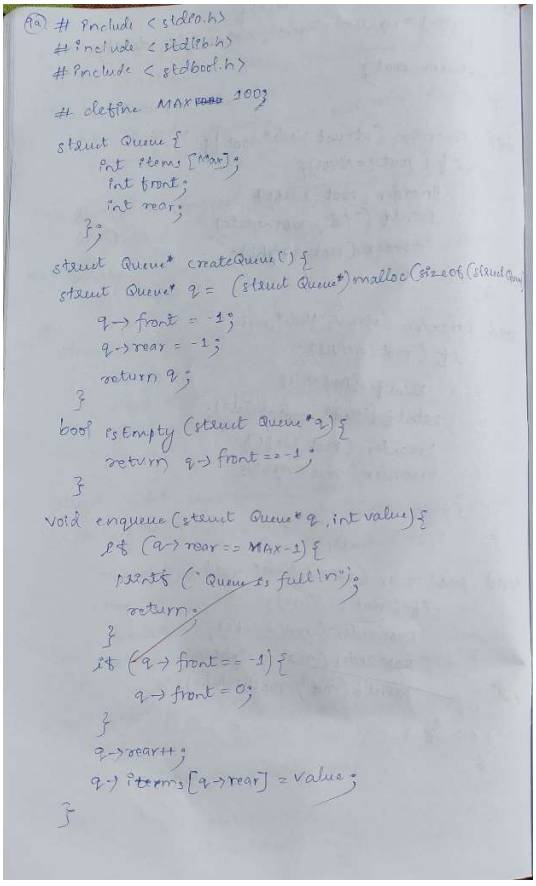
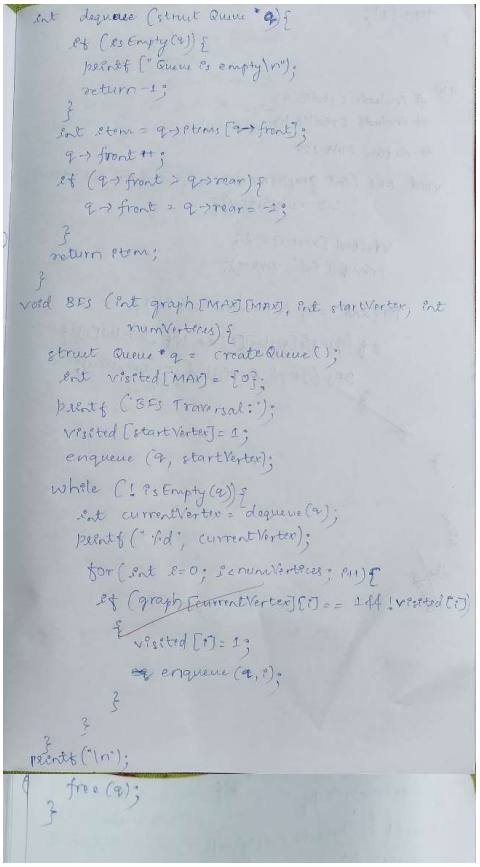
Program 10:

9a) Write a program to traverse a graph using BFS method.

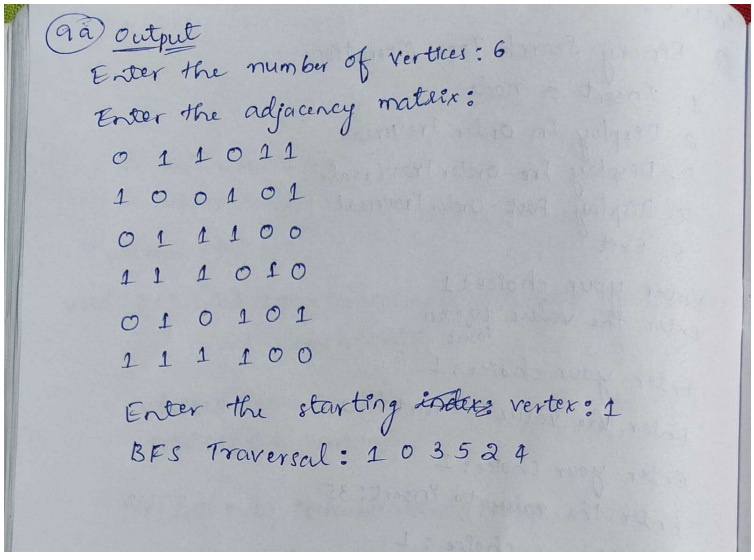
Observation:



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

#define MAX 100

struct Queue {
    int items[MAX];
    int front;
    int rear;
};

struct Queue* createQueue() {
    struct Queue* q = (struct Queue*)malloc(sizeof(struct Queue));
    q->front = -1;
    q->rear = -1;
    return q;
}

bool isEmpty(struct Queue* q) {
    return q->front == -1;
}

void enqueue(struct Queue* q, int value) {
    if (q->rear == MAX-1) {
        printf("Queue is full\n");
        return;
    }
    if (q->front == -1) {
        q->front = 0;
    }
    q->rear++;
    q->items[q->rear] = value;
}
```



```
int dequeue(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return -1;
    }
    int item = q->items[q->front];
    q->front++;
    if (q->front > q->rear) {
        q->front = q->rear = -1;
    }
    return item;
}

void BFS(int graph[MAX][MAX], int startVertex, int numVertices) {
    struct Queue* q = createQueue();
    int visited[MAX] = {0};
    printf("BFS Traversal: ");
    visited[startVertex] = 1;
    enqueue(q, startVertex);
    while (!isEmpty(q)) {
        int currentVertex = dequeue(q);
        printf("%d ", currentVertex);
        for (int i=0; i<numVertices; i++) {
            if (graph[currentVertex][i] == 1 && !visited[i]) {
                visited[i] = 1;
                enqueue(q, i);
            }
        }
    }
    printf("\n");
    free(q);
}
```



9a) Output

```
Enter the number of vertices: 6
Enter the adjacency matrix:
0 1 1 0 1 1
1 0 0 1 0 1
0 1 1 1 0 0
1 1 1 0 1 0
0 1 0 1 0 1
1 1 1 1 0 0
Enter the starting vertex: 1
BFS Traversal: 1 0 3 5 2 4
```

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define MAX 100
struct Queue {
    int items[MAX];
    int front;
    int rear;
};

struct Queue* createQueue() {
    struct Queue* q = (struct Queue*)malloc(sizeof(struct Queue));
    q->front = -1;
    q->rear = -1;
    return q;
}

bool isEmpty(struct Queue* q) {
    return q->front == -1;
}

void enqueue(struct Queue* q, int value) {
    if (q->rear == MAX - 1) {
        printf("Queue is full\n");
        return;
    }
    if (q->front == -1) {
        q->front = 0;
    }
    q->rear++;
    q->items[q->rear] = value;
}

int dequeue(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return -1;
    }
    int item = q->items[q->front];
    q->front++;
    if (q->front > q->rear) {
        q->front = q->rear = -1;
    }
    return item;
}
void BFS(int graph[MAX][MAX], int startVertex, int numVertices) {
    struct Queue* q = createQueue();
```

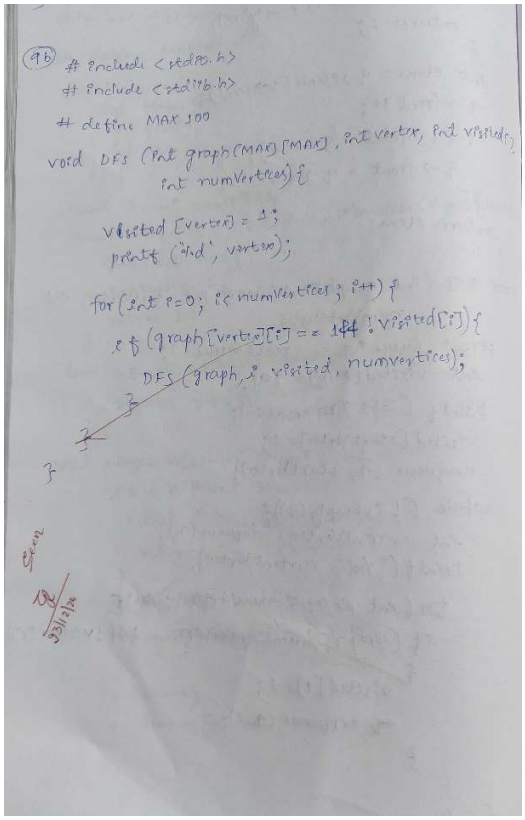
```c
    int visited[MAX] = {0};

    printf("BFS Traversal: ");
    visited[startVertex] = 1;
    enqueue(q, startVertex);

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

        for (int i = 0; i < numVertices; i++) {
            if (graph[currentVertex][i] == 1 && !visited[i]) {
                visited[i] = 1;
                enqueue(q, i);
            }
        }
    }
    printf("\n");

    free(q);
}

int main() {
    int graph[MAX][MAX], numVertices, startVertex;

    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);

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

    printf("Enter the starting vertex: ");
    scanf("%d", &startVertex);

    BFS(graph, startVertex, numVertices);

    return 0;
}
```

Output:

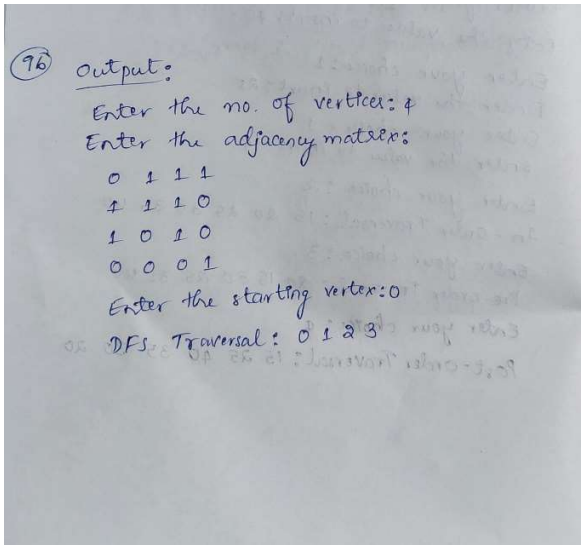
```
Enter the number of vertices: 6
Enter the adjacency matrix:
0 1 1 0 1 1
1 0 0 1 0 1
0 1 1 1 0 0
1 1 1 0 1 0
0 1 0 1 0 1
1 1 1 1 0 0
Enter the starting vertex: 1
BFS Traversal: 1 0 3 5 2 4
```

9b) Write a program to traverse a graph using DFS method.

Observation:



```
(9b) #include <stdio.h>
     #include <stdlib.h>
     #define MAX 100
     void DFS (int graph[MAX][MAX], int vertex, int visited[],
               int numVertices){
         visited[vertex] = 1;
         printf("%d", vertex);
         for(int i=0; i<numVertices; i++){
             if(graph[vertex][i] == 1 && !visited[i]){
                 DFS(graph, i, visited, numVertices);
             }
         }
     }
```



```
(9b) Output:
Enter the no. of vertices: 4
Enter the adjacency matrix:
0 1 1 1
1 1 1 0
1 0 1 0
0 0 0 1
Enter the starting vertex: 0
DFS Traversal: 0 1 2 3
```

Code:

```c
 #include <stdio.h>
#include <stdlib.h>

#define MAX 100


void DFS(int graph[MAX][MAX], int vertex, int visited[], int numVertices) {
    visited[vertex] = 1;
    printf("%d ", vertex);

    for (int i = 0; i < numVertices; i++) {
        if (graph[vertex][i] == 1 && !visited[i]) {
            DFS(graph, i, visited, numVertices);
        }
    }
}


int main() {
    int graph[MAX][MAX], numVertices, startVertex;
    int visited[MAX] = {0};

    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);

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

    printf("Enter the starting vertex: ");
    scanf("%d", &startVertex);

    printf("DFS Traversal: ");
    DFS(graph, startVertex, visited, numVertices);
    printf("\n");

    return 0;
```

Output:

```
Enter the number of vertices: 4
Enter the adjacency matrix:
0 1 1 1
1 1 1 0
1 0 1 0
0 0 0 1
Enter the starting vertex: 0
DFS Traversal: 0 1 2 3
```