Task 1:

Student STDOUT.txt

```
DFS: A B C D F E
BFS A B D E C F

```

Task 2:

Student STDOUT.txt

```
A B C G E D F
The length of the shortest path between A and A is 0
The length of the shortest path between A and B is 1
The length of the shortest path between A and C is 2
The length of the shortest path between A and D is 7
The length of the shortest path between A and E is 5
The length of the shortest path between A and F is 7
The length of the shortest path between A and G is 3

```

Task 3:

Student STDOUT.txt

```
Loaded 4806 vertices
Loaded 6179 edges
Please enter stating bus stop >            Please enter destination bus stop >     300 Eden Quay
497 Amiens Street
515 Amiens Street
516 North Strand Rd
4384 North Strand Rd
519 North Strand Rd
521 Annesley Bridge
522 Marino Mart
523 Marino Mart
669 Malahide Road
670 Malahide Road
671 Malahide Road
672 Malahide Road
4382 Malahide Road
1185 Collins Ave
1186 Collins Ave
1187 Collins Ave
1188 Collins Ave
1189 Collins Ave
216 Beaumont Road
217 Beaumont Road
242 Beaumont Road
243 Beaumont Road
253 Beaumont Hospital

```

Approach:

Using the same parser used in previous assignments data was read from the csv file.

Vertices:
I opted to store the weights between vertices in both nodes. If I ever wanted to expand upon this assignment, this would allow for the two directions to have different weights. I also opted to use an array initialised to the max number of bus stops. Vertices were then inserted into the index corresponding to their bus stop. This meant I did not have to linearly search for a bus stop number every time I wanted to find a stop. Instead I could just search array[bus_stop_number] to retrieve a node. This system assumes there are no duplicate bus stop numbers

1. Vertices were created (not inserted into graph yet) using the following function:

```
Node* createNode(int STOPID, char* NAME, float LAT, float LONG){
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->stopID = STOPID;
    strcpy(newNode->name, NAME);
    newNode->latitude = LAT;
    newNode->longitude = LONG;
    newNode->isPermanent = 0;
    newNode->numNeighbours = -1;
    newNode->distToSource = INF;
    newNode->neighbours = (Node**)malloc(maxNeighbours * sizeof(Node*));
    newNode->prevNode = NULL;
    newNode->weights = (int*)malloc(maxNeighbours * sizeof(int));
    newNode->neighbours[0] = NULL;
    return newNode;
}
```

2. A function was made to both create the graph and insert only the first vertice:

```
Graph* createGraph(Node* nodeToInsert){
    Graph* newG = (Graph*)malloc(sizeof(Graph));
    newG->numNodes = maxNumVertices;
    newG->nodes = (Node**)malloc(maxNumVertices * sizeof(Node*));

    //initiate array
    for(int i=0; i < maxNumVertices; i++){
        newG->nodes[i] = NULL;
    }

    newG->nodes[nodeToInsert->stopID] = nodeToInsert;
    return newG;
}
```

3. Then once the graph was created, all following vertices were inserted using:

```
void insertNode(Graph* g, Node* nodeToInsert){
    g->nodes[nodeToInsert->stopID] = nodeToInsert;
}
```

Edges:

1. Edges were inserted between vertices using the following function:

```
void insertEdge(int vertex1, int vertex2, int weight){
    Node* v1 = g->nodes[vertex1];
    Node* v2 = g->nodes[vertex2];

    if(v1 == NULL){
        printf("ERROR V1 DOESNT EXIST");
    }
    if(v2 == NULL){
        printf("ERROR V2 DOESNT EXIST %d", vertex2);
    }

    //create link from and to
    int i = 0;
    while(v1->neighbours[i] != NULL){
        i++;
    }

    v1->neighbours[i] = v2;
    v1->weights[i] = weight;
    v1->numNeighbours++;
    v1->neighbours[i+1] = NULL;


    i = 0;
    while(v2->neighbours[i] != NULL){
        i++;
    }

    v2->neighbours[i] = v1;
    v2->weights[i] = weight;
    v2->numNeighbours++;
    v2->neighbours[i+1] = NULL;
}
```