ASSIGNMENT 4 – Graphs

Name: Vanaj Kamboj
Roll No: 21355100

Task 1

Header File(t1.h):

```
#ifndef T1_H_
#define T1_H_
#include<stdbool.h>

struct node {
  int vertex;
  struct node* next;
  int mark;
};



struct node* createNode(int v);

typedef struct Graph {
  int num_nodes;
  int* visited;
  int* visitedbfs;

  // We need int** to store a two dimensional array.
  // Similary, we need struct node** to store an array of Linked lists
  struct node** adjLists;
}Graph;

Graph* create_graph(int num_nodes); // creates a graph with num_nodes nodes, assuming nodes are stored in alphabetical order (A, B, C..)
void add_edge(struct Graph *g, int from, int to); // adds a directed edge
void bfs(struct Graph* g, int origin); //implements breath first search and prints the results
void dfs(struct Graph* g, int origin); //implements depth first search and prints the results
void delete_graph(struct Graph* g); // Deletes the graph

struct queue* createQueue();
void addToQueue(struct queue* q, int); // Adding elements to queue
```

```c
int removeFromQueue(struct queue* q); // Removing elements from queue
int isEmpty(struct queue* q); // Checking if the queue is empty


#endif
```

My Code(t1.c):

```c
#include <stdio.h>
#include <stdlib.h>
#include"t1.h"

// #define SIZE 40;
struct queue {
  int items[40];
  int front;
  int rear;
};

int DFScallonce  = 0; // Creating a flag variable to print "DFS: " once in the output(Recursive function making it print recursively)

// Create a node
struct node* createNode(int v) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->vertex = v;
  newNode->next = NULL;
  return newNode;
}

// Create graph
Graph* create_graph(int num_nodes) {
  Graph* graph = malloc(sizeof(Graph));
  graph->num_nodes = num_nodes;

  graph->adjLists = malloc(num_nodes * sizeof(struct node*));

  graph->visited = malloc(num_nodes * sizeof(int));

  graph->visitedbfs = malloc(num_nodes * sizeof(int));
```

```
  int i;
  for (i = 0; i < num_nodes; i++) {
    graph->adjLists[i] = NULL;
    graph->visited[i] = 0;
    graph->visitedbfs[i] = 0;
  }
  return graph;
}

// Add edge
void add_edge(Graph* graph, int from, int to) {
  struct node* newNode = createNode(to);
  newNode->next = graph->adjLists[from];
  graph->adjLists[from] = newNode;
  // free(newNode);
}

void delete_graph(struct Graph* g){
  free(g->visitedbfs);
  free(g->visited);
  for (int i = 0; i < g->num_nodes; i++)
    free(g->adjLists[i]);
  free(g->adjLists);
  free(g);
}

// DFS algorithm
void dfs(Graph* g, int origin) {
    if(DFScallonce == 0)
        printf("DFS:");
    struct node* adjList = g->adjLists[origin];
    struct node* temp = adjList;

    g->visited[origin] = 1;
    char charValue = origin + 65;
    printf("  %c", charValue);

    while (temp != NULL) {
        int connectedVertex = temp->vertex;
```

```
    if (g->visited[connectedVertex] == 0) {
      DFScallonce = 1;
      dfs(g, connectedVertex);
    }
    temp = temp->next;
    }

}




void bfs(struct Graph* graph, int startVertex) {
  printf("\n");
  printf("BFS ");
  struct queue* q = createQueue();

  graph->visitedbfs[startVertex] = 1;
  addToQueue(q, startVertex);

  while (!isEmpty(q)) {
    // printQueue(q);
    int currentVertex = removeFromQueue(q);
    // for (int i = 0; i < graph->num_nodes; i++) {
    //     graph->visited[i] = 0;
    //   }
    char charValue = currentVertex + 65;
    printf("  %c", charValue);
    // printf("Visited %d\n", charValue);

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

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

      if (graph->visitedbfs[adjVertex] == 0) {
        graph->visitedbfs[adjVertex] = 1;
        addToQueue(q, adjVertex);
      }
```

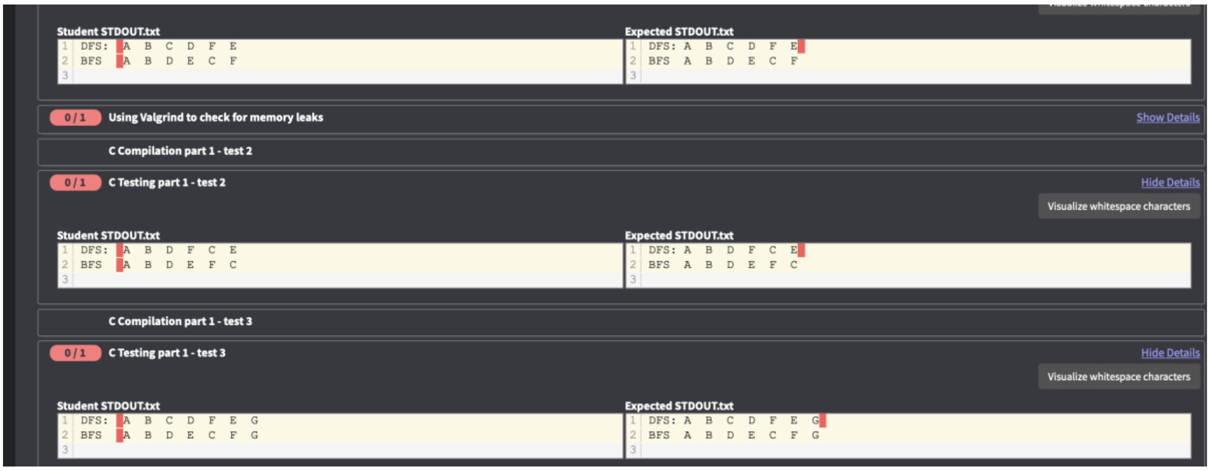
```
      temp = temp->next;
    }
  }
  printf(" \n");
}

// Create a queue
struct queue* createQueue() {
  struct queue* q = malloc(sizeof(struct queue));
  q->front = -1;
  q->rear = -1;
  return q;
}

// Check if the queue is empty
int isEmpty(struct queue* q) {
  if (q->rear == -1)
    return 1;
  else
    return 0;
}

// Adding elements into queue
void addToQueue(struct queue* q, int value) {
  if (q->rear == 40 - 1)
    printf("\nQueue is Full!!");
  else {
    if (q->front == -1)
      q->front = 0;
    q->rear++;
    q->items[q->rear] = value;
  }
}

// Removing elements from queue
int removeFromQueue(struct queue* q) {
  int item;
  if (isEmpty(q)) {
    // printf("Queue is empty");
    item = -1;
```

```c
  } 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[q->rear-1]);
    // }
  }
}
```

Output:

```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

vanajkamboj@Vanajs-MacBook-Pro A4 % cd "/Users/vanajkamboj/My_files/Study/DSA/A4/" && gcc t1.c -o t1 && "/Users/vanajkamboj/My_files/Study/DS
A/A4/"t1
DFS:  A  B  C  D  F  E
BFS   A  B  D  E  C  F
vanajkamboj@Vanajs-MacBook-Pro A4 % 
```

Output on submitty:



Task 2
Logic:
Reference for Dijkstra Algorithm - https://www.educative.io/edpresso/how-to-implement-dijkstras-algorithm-in-cpp

Header File(t2.h):

```
#ifndef T2_H_
#define T2_H_

#define MAX_VERTICES 10 // you can assume that the graph has at most 10 vertex
typedef struct Graph{
   int numNodes;
   int **matrix;
} Graph;

Graph* create_graph(int num_nodes); // creates a graph with num_nodes nodes, assuming nodes are stored in alphabetical order (A, B, C..)
void add_edge(Graph *g, int from, int to, int weight); // adds an undirected weighted edge between from and to

void dijkstra(Graph* g, int origin); // implements the dijkstra algorithm and prints the order in which the nodes are made permament, and the length of the shortest path between the origin node and all the other nodes
void delete_graph(Graph* g);

#endif
```

My Code(t2.c):

```
#include "t2.h"
```

```c
#include <limits.h>

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

Graph* create_graph(int num_nodes) {
    // Allocate memory for graph's adj matrix
    Graph* graph = (Graph*) malloc(sizeof(Graph));
    graph->numNodes = num_nodes;

    graph->matrix = malloc(num_nodes * sizeof(graph->matrix));
    for (int i = 0; i < num_nodes; i++) {
        graph->matrix[i] =malloc(num_nodes * sizeof(graph->matrix[i]));
    }

    // Fill matrix with initial values as -1 (path does not exist)
    for (int i = 0; i < num_nodes; i++)
        for (int j = 0; j < num_nodes; j++)
            graph->matrix[i][j] = -1;

    // Fill matrix diagonal with zeros as path from node to itself is zero
    for (int i = 0; i < num_nodes; i++)
        graph->matrix[i][i] = 0;

    return graph;
}

void add_edge(Graph *g, int from, int to, int weight) {
    g->matrix[from][to] = weight;
    g->matrix[to][from] = weight;
}

void delete_graph(Graph* g){
    for (int i = 0; i < g->numNodes; i++) {
        free(g->matrix[i]);
    }
    free(g->matrix);
    free(g);
    // printf("Graph memory cleared\n");
}
```

```
// Utility functions
int minDistance(int *d, int *v, int len) {   // returns min distance node which hasn't been visited
    int min = INT_MAX, idx;
    for (int i = 0; i < len; i++) {
        if (v[i] == 0 && d[i] < min) {
            min = d[i];
            idx = i;
        }
    }

    return idx;
}

void printMatrix(Graph *g) {
    for (int i = 0; i < g->numNodes; i++) {
        for (int j = 0; j < g->numNodes; j++) {
            printf("%15d", g->matrix[i][j]);
        }
        printf("\n\n");
    }
}

void printArray(int *arr, int len) {
    for (int i = 0; i < len; i++)
        printf("%5d", arr[i]);
    printf("\n");
}

// Dijkstra algorithm
void dijkstra(Graph* g, int origin) {
    int nodes = g->numNodes;

    int *distance = (int*) malloc(nodes * sizeof(int));    // min distance for each node
    int *visited = (int*) malloc(nodes * sizeof(int));
    // int *selected = malloc(nodes * sizeof(int));

    // Set distance to all vertices as INFINITY and visited as -1
    for (int i = 0; i < nodes; i++) {
        if (i != origin)
```

```
        distance[i] = INT_MAX;
      visited[i] = 0;
    }

    distance[origin] = 0;

    // Iterate over all vertices
    for (int i = 0; i < nodes; i++) {
      // printArray(distance, nodes);
      // printArray(visited, nodes);

      int currNode = minDistance(distance, visited, nodes);
      visited[currNode] = 1;  // Set current node as visited
      printf("%c ", currNode + 65);

      // Check current node against all other unvisited vertices
      for (int i = 0; i < nodes; i++) {
        if (visited[i] == 0 && g->matrix[currNode][i] != -1 && distance[currNode] != INT_MAX &&
distance[currNode] + g->matrix[currNode][i] < distance[i])
          distance[i] = distance[currNode] + g->matrix[currNode][i];
      }
    }
    printf("\n");

    // Print final distances
    for (int i = 0; i < nodes; i++) {
      printf("The length of the shortest path between %c and %c is %d\n", origin + 65, i + 65, distance[i]);
    }
    free(distance);
    free(visited);
}
```

Output:

```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

vanajkamboj@Vanajs-MacBook-Pro A4 % cd "/Users/vanajkamboj/My_files/Study/DSA/A4/" && gcc t2.c -o t2 && "/Users/vanajkamboj/My_files/Study/DS
A/A4/"t2
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
vanajkamboj@Vanajs-MacBook-Pro A4 %
```

Task 3
Logic-
I have used my Dijkstra Algorithm from task 2 and created an array called LastNode which holds the position of the previous node and keeps updating itself.

Header File(t3.h):

```
#ifndef T3_H_
#define T3_H_

typedef struct Node {
    int stopId;
    char *name;
    char *latitude;
    char *longitude;
} Node;

typedef struct Graph{
    int numNodes;
    int **matrix;
} Graph;

typedef struct Edge{
    int weight;
    int startId;
    int endId;
} Edge;


int load_edges ( char *fname ); //loads the edges from the CSV file of name fname
int load_vertices ( char *fname );  //loads the vertices from the CSV file of name fname
void shortest_path(int startNode, int endNode); // prints the shortest path between startNode and endNode, if
there is any
```

```
void free_memory ( void ) ; // frees any memory that was used

#endif
```

My Code(t3.c):

```
#include "t3.h"
#include <limits.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#define MAX_BUFFER_SIZE 1024
#define MAX_NODES 8000
#define elementSize 256

// Global graph variable and list of nodes
Graph* graph;
Node *nodes[MAX_NODES];

Node *getNode(char buffer[]) {
    // printf("\n%s\n", buffer);
    int column = 0;
    int foundQuote = 0;
    char element[elementSize];
    memset(element, 0, elementSize); // Clear variable to build next field

    // Result Game struct
    Node *node = malloc(sizeof(Node));

    // Iterate over the buffer and print column elements
    for (int i = 0; i < strlen(buffer); i++)
    {
        switch (buffer[i])
        {
        case ',':
            if (!foundQuote)
            {
                // Column data present in element
                // printf("Column data -> %s, Found Quote %d\n", element, foundQuote);
```

```
                // Store this element in struct!
                // TODO: This part can be abstracted into a neat function...
                switch (column)
                {
                case 0:
                    node->stopId = atoi(element);
                    break;
                case 1:
                    node->name = strdup(element);
                    break;
                case 2:
                    node->latitude = strdup(element);
                    break;
                case 3:
                    node->longitude = strdup(element);
                    break;
                default:
                    break;
                }
                column++;
                // Store this in our struct.
                memset(element, 0, elementSize); // Clear variable to build next field
            }
            break;
        case '\"':
            foundQuote = !foundQuote;
            continue;
        default:
            strcat(element, (char[2]){buffer[i], '\0'});
            break;
        }
    }

    node->longitude = strdup(element);
    return node;
}

Edge *getEdge(char buffer[]) {
    // printf("\n%s\n", buffer);
```

```
int column = 0;
int foundQuote = 0;
char element[elementSize];
memset(element, 0, elementSize); // Clear variable to build next field

// Result Game struct
Edge *edge = malloc(sizeof(Edge));

for (int i = 0; i < strlen(buffer); i++)
{
    switch (buffer[i])
    {
    case ',':
        if (!foundQuote)
        {
            // Column data present in element
            // printf("Column data -> %s, Found Quote %d\n", element, foundQuote);

            // Store this element in struct!
            // TODO: This part can be abstracted into a neat function...
            switch (column)
            {
            case 0:
                // graph->stopId = atoi(element);
                edge->startId = atoi(element);
                break;
            case 1:
                // node->name = strdup(element);
                edge->endId = atoi(element);
                break;
            case 2:
                // node->latitude = strdup(element);
                edge->weight = atoi(element);
                break;
            default:
                break;
            }
            column++;
            // Store this in our struct.
            memset(element, 0, elementSize); // Clear variable to build next field
```

```c
        }
        break;
      case '\"':
        foundQuote = !foundQuote;
        continue;
      default:
        strcat(element, (char[2]){buffer[i], '\0'});
        break;
      }
   }

   edge->weight = atoi(element);
   return edge;
}

void printMatrix(Graph *g) {
   for (int i = 0; i < 25; i++) {
      for (int j = 0; j < 25; j++) {
         printf("%5d", g->matrix[i][j]);
      }
      printf("\n\n");
   }
}

//loads the vertices from the CSV file of name fname
int load_vertices ( char *fname ) {
   FILE *file = fopen(fname, "r");

   int numNodes = 0;
   if (!file)
      printf("Can not open the File\n");
   else {
      char buffer[MAX_BUFFER_SIZE];
      int isFirst = 1;

      while (fgets(buffer, 1024, file))
      {
         if (isFirst)
         {
            isFirst = 0;
```

```
            continue;
        }

        Node *node = getNode(buffer);
        nodes[node->stopId] = node;
        numNodes++;
    }
  }
  printf("Loaded %d vertices\n", numNodes);

  return numNodes;
}

int load_edges ( char *fname ) {
  // Before we load edges, build graph
  // Allocate memory for graph's adj matrix
  graph = (Graph*) malloc(sizeof(Graph));
  graph->numNodes = MAX_NODES;

  graph->matrix = malloc(MAX_NODES * sizeof(graph->matrix));
  for (int i = 0; i < MAX_NODES; i++) {
    graph->matrix[i] =malloc(MAX_NODES * sizeof(graph->matrix[i]));
  }

  // TODO: Fill everything with -1 except actual nodes
  // Fill matrix with initial values as -1 (path does not exist)
  for (int i = 0; i < MAX_NODES; i++)
    for (int j = 0; j < MAX_NODES; j++)
      graph->matrix[i][j] = -1;

  // Fill matrix diagonal with zeros as path from node to itself is zero
  for (int i = 0; i < MAX_NODES; i++)
    graph->matrix[i][i] = 0;

  // Graph is built. Add edges!
  // printf("INFO: Base graph built. Loading edges...\n");

  FILE *file = fopen(fname, "r");
  int numEdges = 0;
  if (!file)
```

```
        printf("Can not open the File\n");
    else {
        char buffer[MAX_BUFFER_SIZE];
        int isFirst = 1;

        while (fgets(buffer, 1024, file))
        {
            if (isFirst)
            {
                isFirst = 0;
                continue;
            }

            Edge *edge = getEdge(buffer);
            // Edge data parsed. Add this to graph's adj matrix
            graph->matrix[edge->startId][edge->endId] = edge->weight;
            graph->matrix[edge->endId][edge->startId] = edge->weight;

            numEdges++;
        }
    }
    printf("Loaded %d edges\n", numEdges);

    return numEdges;
}

int minDistance(int *d, int *v, int len) {   // returns min distance node which hasn't been visited
    int min = INT_MAX, idx;
    for (int i = 0; i < len; i++) {
        if (v[i] == 0 && d[i] < min) {
            min = d[i];
            idx = i;
        }
    }
    // printf("Returning idx %d\n", idx);
    return idx;
}

void printArray(int *arr, int len) {
    for (int i = 0; i < len; i++)
```

```
        printf("%d ", arr[i]);
    printf("\n");
}

void printPath(int *lastNodes, int start, int end) {
    int pathIdx = 0;
    int reversePath[MAX_NODES] = {0};

    int currNode = end; // Start from the back

    while (currNode != start) {
        reversePath[pathIdx] = currNode;
        currNode = lastNodes[currNode];
        pathIdx++;
    }

    // printf("Total In Path: %d\n", pathIdx);

    printf("%5d %20s %15s %15s\n", nodes[start]->stopId, nodes[start]->name, nodes[start]->latitude,
nodes[start]->longitude);
    for (int i = pathIdx - 1; i >= 0; i--) {
        int id = reversePath[i];
        printf("%5d %20s %15s %15s\n", nodes[id]->stopId, nodes[id]->name, nodes[id]->latitude, nodes[id]-
>longitude);
    }
    printf("\n");
}

void free_memory() {
    for (int i = 0; i < MAX_NODES; i++) {
        free(graph->matrix[i]);
    }
    free(graph->matrix);
    free(graph);
    // printf("Graph memory cleared\n");
}

// Shortest path using Dijkstra's algorithm
void shortest_path(int startNode, int endNode) {
    int *distance = (int*) malloc(MAX_NODES * sizeof(int));    // min distance for each node
```

```c
    int *visited = (int*) malloc(MAX_NODES * sizeof(int));
    int *lastNode = (int*) malloc(MAX_NODES * sizeof(int));

    // Set distance to all vertices as INFINITY and visited as -1
    for (int i = 0; i < MAX_NODES; i++) {
        if (i != startNode)
            distance[i] = INT_MAX;
        visited[i] = 0;
        lastNode[i] = -1;
    }

    distance[startNode] = 0;

    // Iterate over all vertices
    for (int i = 0; i < MAX_NODES; i++) {
        int currNode = minDistance(distance, visited, MAX_NODES);
        visited[currNode] = 1;  // Set current node as visited

        // Check current node against all other unvisited vertices
        for (int i = 0; i < MAX_NODES; i++) {
            if (visited[i] == 0 && graph->matrix[currNode][i] != -1 && distance[currNode] != INT_MAX &&
distance[currNode] + graph->matrix[currNode][i] < distance[i]) {
                distance[i] = distance[currNode] + graph->matrix[currNode][i];
                lastNode[i] = currNode;
            }
        }
    }

    // printf("Last node for 497 is %d\n", lastNode[497]);
    printPath(lastNode, startNode, endNode);

    // Free memory
    free(distance);
    free(visited);
    free(lastNode);
}
```

OUTPUT:

The outputs weren't printing out on Submitty. Here is an output from my own PC-

```
→  A4 cd "/Users/vanajkamboj/My_files/Study/DSA/A4/" && gcc t3.c -o t3 && "/Users/vanajkamboj/My_files/Study/DSA/A4/"t3 vertices.csv edges.csv
Loaded 4806 vertices
Loaded 6179 edges
Please enter stating bus stop >         300
Please enter destination bus stop >     253
  300           Eden Quay     53.34826889   -6.255763056

  497       Amiens Street     53.35050306   -6.250701111

  515       Amiens Street     53.35350389   -6.248088889

  516      North Strand Rd       53.35568   -6.245661944

 4384      North Strand Rd    53.35767111   -6.242686111

  519      North Strand Rd    53.36030194   -6.239553056

  521     Annesley Bridge      53.361625   -6.237988889

  522         Marino Mart     53.36327194   -6.235341111

  523         Marino Mart     53.36428111   -6.231608056

  669       Malahide Road     53.36631111   -6.228656944

  670       Malahide Road        53.36895   -6.226008889

  671       Malahide Road     53.37071889   -6.224138056

  672       Malahide Road      53.373465   -6.221061111

 4382       Malahide Road         53.3749        -6.2196

 1185         Collins Ave     53.37637111   -6.221506111

 1186         Collins Ave     53.37764194   -6.226321944

 1187         Collins Ave     53.37860611       -6.23134

 1188         Collins Ave     53.38001389   -6.235576944

 1189         Collins Ave     53.38072194   -6.237976944

  216       Beaumont Road     53.38232889   -6.238176111

  217       Beaumont Road     53.38432389       -6.23678

  242       Beaumont Road        53.38565   -6.231991944

  243       Beaumont Road     53.38577889      -6.229525

  253   Beaumont Hospital     53.38994194   -6.224378889
```

2) Start at 747 and Destination at 3663(Test case from Submitty)

```
→  A4 cd "/Users/vanajkamboj/My_files/Study/DSA/A4/" && gcc t3.c -o t3 && "/Users/vanajkamboj/My_files/Study/DSA/A4/"t3 vertices.csv edges.csv
Loaded 4806 vertices
Loaded 6179 edges
Please enter stating bus stop >         747
Please enter destination bus stop >     3663
  747       Kildare Street     53.33992806   -6.255696944

  748          Merrion Row     53.33851306   -6.255003889

 2905       Merrion Sq West     53.34006694       -6.25192

  494         Clare Street     53.34141694   -6.251671111

  495         Westland Row     53.34358611   -6.249726111

  496       Beresford Place     53.34945694      -6.252165

  497        Amiens Street     53.35050306   -6.250701111

  515        Amiens Street     53.35350389   -6.248088889

  516       North Strand Rd        53.35568   -6.245661944

 4384       North Strand Rd     53.35767111   -6.242686111

  519       North Strand Rd     53.36030194   -6.239553056

  521      Annesley Bridge       53.361625   -6.237988889

  522          Marino Mart     53.36327194   -6.235341111

  523          Marino Mart     53.36428111   -6.231608056

  669         Malahide Road     53.36631111   -6.228656944

  670         Malahide Road        53.36895   -6.226008889

  671         Malahide Road     53.37071889   -6.224138056

  672         Malahide Road       53.373465   -6.221061111

 4382         Malahide Road         53.3749        -6.2196

 1185          Collins Ave     53.37637111   -6.221506111

 1186          Collins Ave     53.37764194   -6.226321944

 1187          Collins Ave     53.37860611       -6.23134

 1188          Collins Ave     53.38001389   -6.235576944

 1189          Collins Ave     53.38072194   -6.237976944

  216        Beaumont Road     53.38232889   -6.238176111

  217        Beaumont Road     53.38432389       -6.23678

  218       Shantalla Road     53.38556611        -6.2336

  219       Shantalla Road     53.38785806   -6.238386111

  220          Swords Road       53.387935   -6.244966944

 1622          Swords Road     53.39003694   -6.246393889

 1623          Swords Road     53.39193306   -6.246233889

 1624          Swords Road       53.396075   -6.245461111

 1625          Swords Road        53.40109       -6.24328

 1626          Swords Road     53.40416194   -6.240321944

 1627          Swords Road       53.408245   -6.237873056

 1628          Swords Road     53.41371111   -6.239061944

 1629          Swords Road     53.41662889   -6.239163056

 1630          Swords Road        53.42204   -6.231693056

 7348       Dublin Airport     53.42774806   -6.241613889

 3663       Dublin Airport     53.42459611   -6.234888889

→  A4 █
```

3) Start at 3235 and Destination at 7652

```
→  A4 cd "/Users/vanajkamboj/My_files/Study/DSA/A4/" && gcc t3.c -o t3 && "/Users/vanajkamboj/My_files/Study/DSA/A4/"t3 vertices.csv edges.csv
Loaded 4806 vertices
Loaded 6179 edges
Please enter stating bus stop >         3235
Please enter destination bus stop >     7652
 3235        Pearse Street     53.27627194   -6.139121944

 3236      Sallynoggin Road     53.27797889   -6.140298889

 3343      Sallynoggin Road     53.27619194      -6.143965

 3248      Sallynoggin Road     53.27474611      -6.148335

 7056        Rochestown Ave     53.27359611   -6.147656111

 4731        Rochestown Ave     53.27580306   -6.151338056

 3249            Pottery Rd     53.27304389   -6.153161944

 7667           Barnhill Rd     53.27689389    -6.119151944
 7652       Killiney Hill Rd     53.26482889      -6.114855

→  A4 █
```