KIT205 Data Structures and Algorith
Assignment 3

Due 15th October, 11:55pm

Introduction

Ticket to Ride is a popular board game that involves connecting cities in a network. In this assignment you will prototype some potential approaches f player for this game (since the AI players for the computerised version are c

The basic gameplay of Ticket to Ride requires players to fulfill "tickets" that are randomly s consists of two cities that need to be connected. Adjacent cities are connected by placing train tokens on the track. While there are other complications in the full game, the basis o fulfill your tickets using the least number of train tokens - this in turn allows you to fulfill m number of train tokens available.

Data Structures and Input

We will represent the map using the following data structures, as used in tutorials.

```
typedef struct edge{
    int to_vertex;
    int weight;
} Edge;

typedef struct edgeNode{
    Edge edge;
    struct edgeNode *next;
} *EdgeNodePtr;

typedef struct edgeList{
    EdgeNodePtr head;
} EdgeList;

typedef struct graph{
    int V;
```

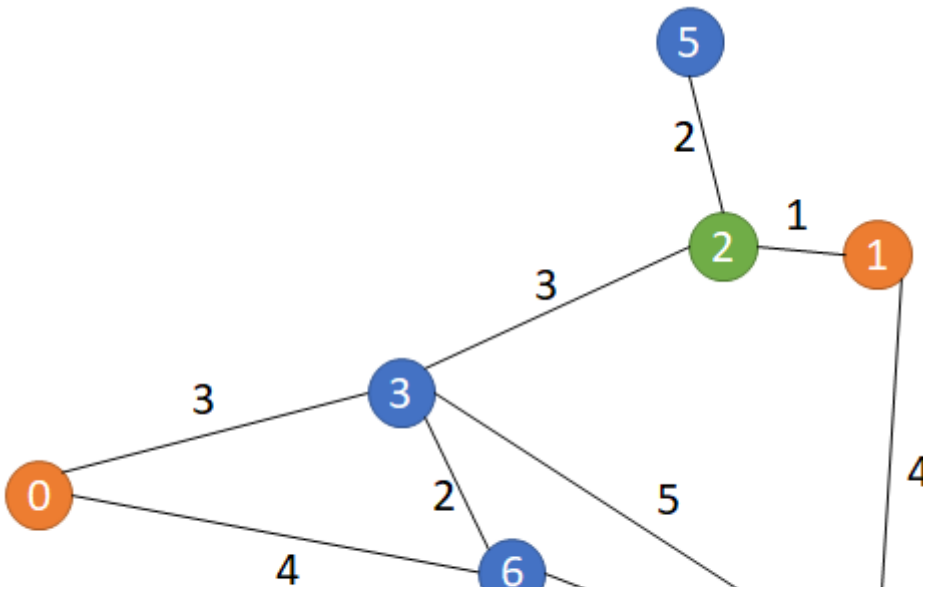
```
    int V;
    EdgeList *edges;
} Graph;
```

In this case, the vertices represent the cities on the map, and the edge weight will be the nu
required to connect adjacent cities.

We will also use the same redirected input method used in tutorials to create the graph. H
case is *undirected*, so **two** edges need to be added for each pair of adjacent cities. i.e. if citie
edge needs to be added from vertex 2 to 7 *and* from vertex 7 to 2. The graph input will incl
cities with a lower index to cities with a higher index, but you must also add the edges in th

The graph input will then be followed by input for a given number of tickets. For example,
create a graph of **7** cities, with **2** tickets from cities 2 to 4 and 0 to 1:

```
7
2
6,4 3,3
2
4,4 2,1
2
5,2 3,3
2
6,2 4,5
1
6,3
0
0
2
2,4 0,1
```



Part A

The problem of finding the cheapest way to fulfill tickets is obviously related to the minima
So part A is a warm-up exercise where you will implement Prim's MST algorithm. The follo
pseudocode should be followed:

Prim's Minimal Spanning Tree (from Wikipedia)

1. Associate with each vertex *v* of the graph a number *C[v]* (the cheap
   connection to *v*) and an edge *E[v]* (the edge providing that cheapes
   initialize these values, set all values of *C[v]* to +∞ (or to any numbe
   maximum edge weight) and set each *E[v]* to a special flag value indi
   is no edge connecting *v* to earlier vertices.

2. Initialize an empty forest *F* and a set *Q* of vertices that have not yet been includ
   vertices).

3. Repeat the following steps until *Q* is empty:
   a. Find and remove a vertex *v* from *Q* having the minimum possible value o
   b. Add *v* to *F* and, if *E[v]* is not the special flag value, also add *E[v]* to *F*
   c. Loop over the edges *vw* connecting *v* to other vertices *w*. For each such e
      belongs to Q and *vw* has smaller weight than *C[w]*, perform the following
      i. Set *C[w]* to the cost of edge *vw*
      ii. Set *E[w]* to point to edge *vw*

4. Return *F*

Your implementation must follow the above pseudocode and use the following C functio

```
Graph prims_mst(Graph *self);
```

The following assumptions and hints will be useful.

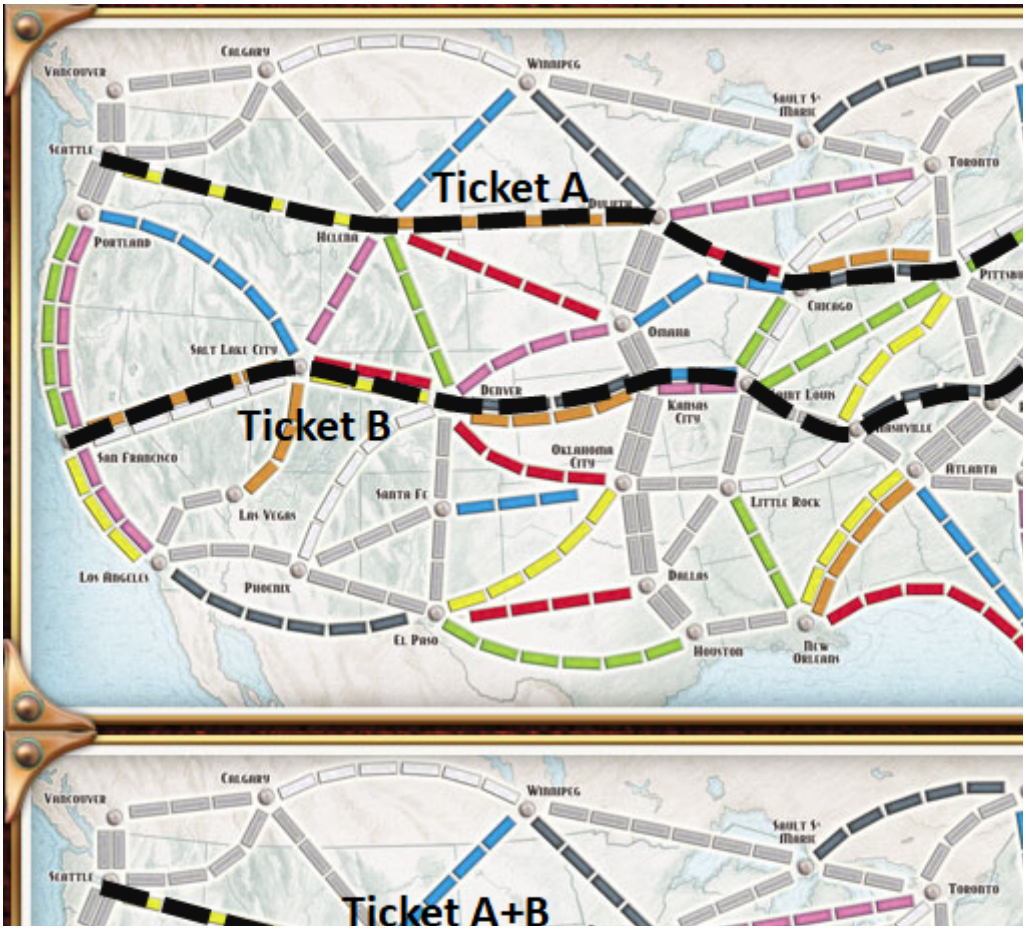
Hints

- In the pseudocode *F* will be a `Graph` data structure
- Since you know that the graph is connected, you may start by initialising *F* in st containing all vertices (but no edges). This means that you can skip the part "A you will only need to add the edge *E[v]* to *F*
- You can combine the *C[v]* and *E[v]* lists by using an array of `Edge`s
- *Q* can also be implemented as a membership array (i.e. array representation of
- You can use a simple search for step 3a. You do not need to use any of the mor heap-based approaches.

For part A, you can ignore the tickets. You just need to return the MST for the given fully c returned the graph, you should print all of the edges that have been added and the total co

Part B

You will now write an algorithm to find the cheapest way to fulfill your tickets. This is not a example, a naive solution might be to find the shortest path for each ticket and then add al However, for certain tickets, this may be very wasteful, as illustrated below. In this case th shortest paths for tickets A and B. The bottom map shows one way that these tickets can b efficiently.



So, the ticket fulfillment problem is related to the MST problem and also to the Steiner tre an MST for a subset of the vertices in the graph. While there are many polynomial time alg are suprisingly no known efficient algorithms for the Steiner tree problem - it is an NP-har

Like the Steiner tree problem, I suspect that the problem of finding the most efficient way NP-hard. Luckily, I do not expect you to find an exact solution. Your task is to find a good a Some of the approaches for finding an approximate solution to the Steiner tree probler approaches that you can use here.

Any solution that finds a set of edges to fulfill all your tickets (including a correct combinat receive an HD grade for this part of the assessment. However, to get full marks for this pa good approximation of the *optimal* set of edges. One example of a solution that would get

1. Find an approximate solution to the Steiner tree problem of finding the minimal spa of vertices that are destinations in any of your tickets

2. Remove any edges that are not required to fulfill any ticket.

This solution would get full marks, but I am sure that you can do better!

The output for Part B should be the edges that were added and the total cost.

Assignment Submission

Assignments will be submitted via MyLO (an Assignment 2 dropbox will be created). You s procedure to prepare your submission:

- Make sure that your project has been thoroughly tested using the School's lab comp
- Choose "Clean Solution" from the "Build" menu in Visual Studio. This step is very im the version that the marker runs will be the same as the version that you believe the
- Quit Visual Studio and zip your entire project folder
- Upload a copy of the zip file to the MyLO dropbox

History tells us that mistakes frequently happen when following this process, so you shoul

- Unzip the folder to a new location
- Open the project and confirm that it still compiles and runs as expected
    - If not, repeat the process from the start

Learning Outcomes and Assessment

This assignment is worth 15% of your overall mark for KIT205. Your submission will be as code and by running additional tests in Visual Studio. If your submission does not run in Vi Windows, it cannot be assessed.

Grades will be assigned in accordance with the Assignment 3 Rubric

The assignment contributes to the assessment of learning outcomes:

> **LO1 Transform a real-world problem into a simple abstract form that is suitable for**
> **LO2 Implement common data structures and algorithms using a common programm**
> **LO4 Use common algorithm design strategies to develop new algorithms when ther**
> **solutions**