KXC251 Algorithms and Metrics, 2011.

Assignment 2

Due: 3pm on Friday 16th December, 2011

This assignment is worth 15% of your total mark for the unit. It consists of 3 parts. Each part is worth 4 marks. An additional 3 marks are for overall Program Style. You need to complete all parts to get full marks.

Introduction

You are required to write a program to find the shortest driving distance from one town to all other towns in a given area. The figure below shows the island State of Tasmania in Australia. The circles on the map are the towns (nodes) which are to be travelled between.

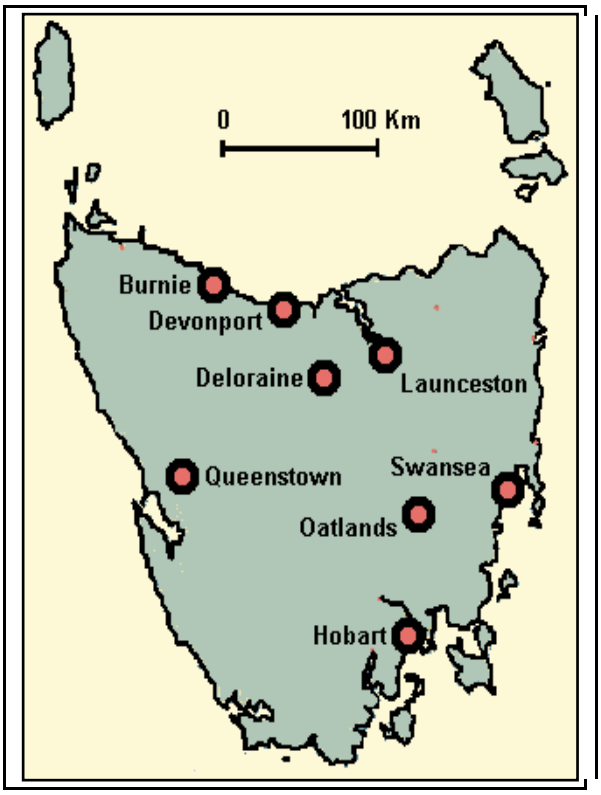

Figure 1: Network Nodes in Tasmania

Information must be input from a text file and stored in a graph. You will then use Dijkstra's Shortest Path Algorithm to solve the problem. The algorithm will return a list of shortest distances to each of the towns. The program will output a listing of the shortest paths to all towns.

Input Files

Input to your program is from a text file. The following is an example (*input1.txt*):

```
8
Burnie
Deloraine
Devonport
Hobart
Launceston
Oatlands
Queenstown
Swansea
 90  48   0   0   0 117   0
 49   0  42 103   0 137
  0  72   0 121   0
169  73 174 109
 98   0 114
154  59
  0
```

The 1st line contains an integer *x* which is the number of towns.

Each of the following *x* (eg. 8) lines contains the name of one town.

Each of the following *x-1* (eg. 7) lines contains the distances in Kilometres (Km) from the corresponding town to each of the towns below it in the list of towns. In the above example the distance numbers correspond to the following town pairs:

Bur-Del, Bur-Dev, Bur-Hob, Bur-Lau, Bur-Oat, Bur-Que, Bur-Swa
Del-Dev, Del-Hob, Del-Lau, Del-Oat, Del-Que, Del-Swa
Dev-Hob, Dev-Lau, Dev-Oat, Dev-Que, Dev-Swa
Hob-Lau, Hob-Oat, Hob-Que, Hob-Swa
Lau-Oat, Lau-Que, Lau-Swa
Oat-Que, Oat-Swa
Que-Swa

A distance of '**0**' indicates that no direct connection exists (eg. to drive from Queenstown to Swansea you must go via Hobart, Oatlands, or Launceston).

2 sample input files are provided with the assignment: *input1.txt* and *input2.txt*. These files are for you to test your program. Your program should work correctly for both input files. It should also work correctly for any other input file of the correct format. Marking may include testing with a different file.

Part 1 – Store Information

You must firstly read in the input file and store the input information in a graph. The nodes in the graph will each represent a town in the input file. The edges in the graph will represent the distances to towns that can be connected.

Implementation

Write a main method to run the program. This should be put in a file called "assignment2.cpp".

You should create 3 classes:

- The *map* class should consist of an array of towns.
- The *town* class should store the town name, plus a list of edges which each represent a possible connection.
- The *connection* class should store the index of the connected town and the distance to it.

You should also include a *list* class:

- You MUST use the list code provided in lectures, but you will need to modify it to store connections.
  IMPORTANT: You are NOT allowed to use any other list class (for example, the STL list class)

Points to Note for all Parts

The program needs to correctly free ALL dynamically allocated memory and close after it runs.

All strings should be stored efficiently. This means that you will need to read strings into a buffer and then dynamically allocate the appropriate amount of memory for the string, as shown in lectures.

You may find that it is easier to test your program if you have some error checking code.

Try and minimise the amount of code in the main method. Where practical put methods in the other classes and make calls to these in the main method.

Part 2 – Dijkstra's Algorithm

For this part of the assignment, you must implement Dijkstra's Algorithm.

Implementation

Put your code for Dijkstra's Algorithm in a new file called *dijkstra.cpp*. Note the following outline of Dijkstra's Algorithm from the lectures:

- The method consists of finding the shortest path distances from the source vertex to the other vertices, in an order in which we find these distances in order of the distances, nearest vertex first.

In this assignment:

1. The source vertex will always be **the 1st town listed** in the input file.
2. Your function that implements Dijkstra's algorithm **must take the stored map (graph) as a parameter** and **return a list (or array) which specifies the minimum distances to each other node**.

Points to Note

The method by which the nearest vertex is found determines the time complexity. Use a method which is efficient.

The general form of Dijkstra's algorithm works for directional graphs – we are representing a problem with bi-directional roads, so each road must be represented as two directed edges.

Part 3 – Print the Distance to Each Town

For this part of the assignment, you must print out the **names and distances to towns** that are calculated **in order of their length** (shortest to longest).

Implementation

The format of the output should be as follows (output for *input1.txt* example):

```
Shortest path from Burnie to:
   Devonport          48Km
   Deloraine          90Km
   Queenstown         117Km
   Launceston         120Km
   Oatlands           193Km
   Swansea            227Km
   Hobart             266Km
```

After this is printed, memory should be freed and the program should close.

Points to Note

The following code shows how output can be formatted when using cout.

```
#include <iostream>
#include <iomanip>
using namespace std;

// NOTES:
// setw (10) makes column of width=10 characters
// left moves characters to the left of the column
// right moves characters to the right of the column

int main ()
{
  cout << setw (10) << 11 << setw (5) << 2222 << endl;
  cout << setw (10) << left << 33 << setw (5) << 4444 << setw (3) << right << 55 << endl;
  cout << 66 << endl;
  cout << setw (5) << 77 << endl;
  cout << setw (5) << left << 88 << setw (5) << 9999 << endl;
  return 0;
}
```

The output from the above code:

```
        11 2222
33        4444  55
66
   77
88   9999
```

Similar things can be achieved using the *printf()* function (which is a C function).

Submission

1. You should submit an electronic copy your program to your Chinese lecturer, including the following files:

   list.h

   list.cpp

   map.h

   map.cpp

   town.h

   town.cpp

   connection.h

   connection.cpp

   dijkstra.h

   dijkstra.cpp

   assignment2.cpp (containing the "main" method)

   Your files should be well documented. If you have used code from the lectures, or from anywhere else, this should be clearly stated at the top of the file.

2. You should submit a signed Assignment Cover Sheet to your Chinese Lecturer.

Marking

Your program will be executed and tested for correctness. Your code will also be examined to determine a mark for coding style (e.g. comments, suitable indentation, structure etc). The following marking sheet (which will be returned to you) shows the mark allocation for each part of the assignment.

Assignment 2

Student Name: **Student #:**

Item	Marks
Program functions correctly:	
Part 1:	
• Information read correctly from file	/1
• Information stored correctly	/1
• Required classes are implemented	/1
• Data structures function correctly	/1
Part 2:	
• Dijkstra code functions correctly	/5
Part 3:	
• Print out formatted correctly	/1
• Correct distances printed out	/1
• Order of towns correct	/1
Program style:	
• Program contains appropriate classes and methods	/1
• Dynamically allocated memory is handled correctly	/1
• Code is well documented and formatted	/1
Total:	**/15**

Comments:

_____

_____

_____

_____

_____

_____