# Minimum Spanning Tree

Terence Munro

# Contents

## 1. Problem Statement

The goal of Question 1 was to implement an algorithm for finding the minimum spanning tree of a given graph. Some previously written code is provided along with the pseudocode for the optimised Prim's algorithm. Implementing this pseudocode in C++ is one of the core requirements of the problem.

The MinimumSpanningTree class also needs to be able to:

- Generate a connected weighted graph from either a given input text file;
- Generate a random graph given the number of nodes;
- Be able to produce a hardcoded example figure 12-16 pg 708 from the text book Data Structures Using C++ by D.S. Malik.

For comparison an already implemented minimumSpanning function is provided.

## 2. User Requirements

The following outlines the user requirements for the program:

The user should be able to:

- Create a spanning graph using a text file with a specific format
- Create a randomized spanning graph given a number of nodes
- Create the graph from the text book figure 12-16 pg. 708 - Data Structures Using C++
- See a print out of the graph
- See a print out of the edges and their corresponding weights in the minimum spanning tree solution
- Compare the result of minimumSpanning function and prim2 function

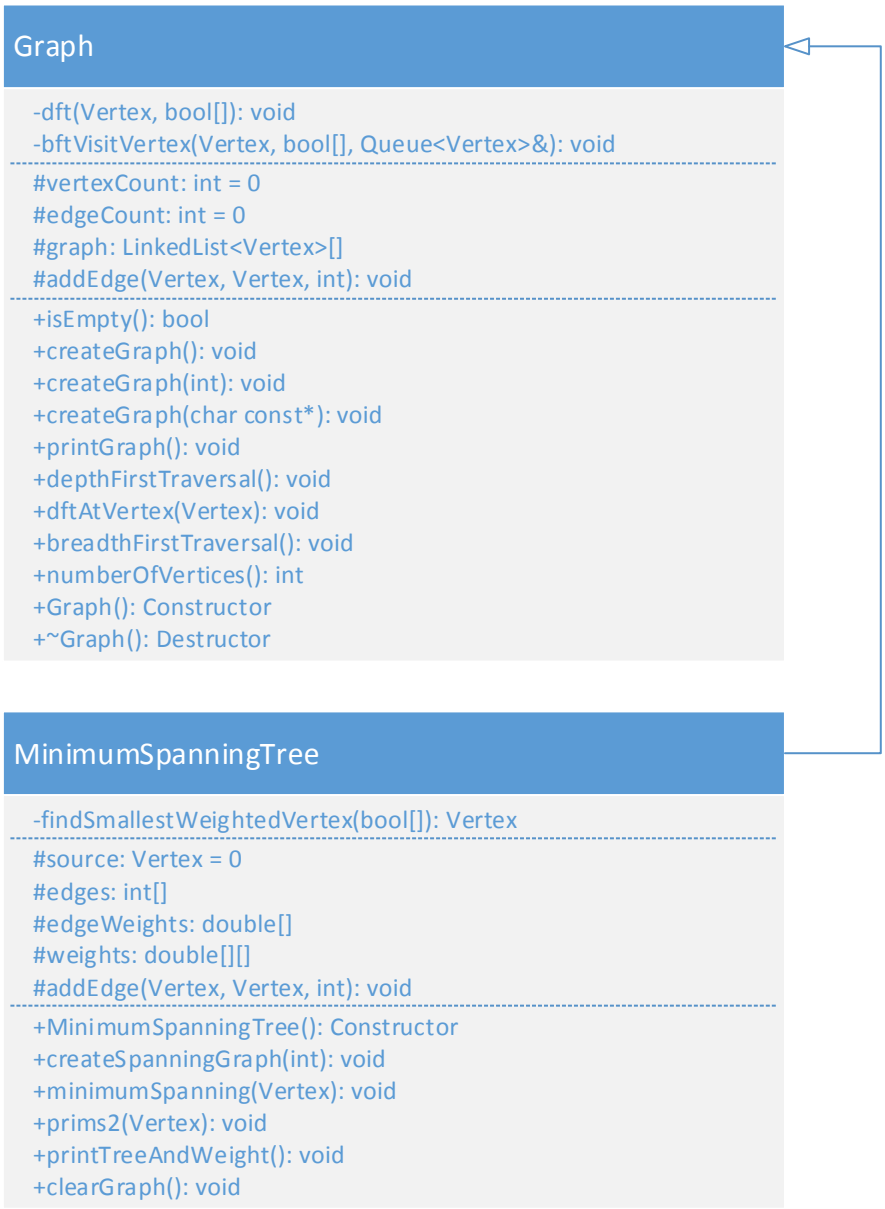
## 3. Software Requirements

The following outlines the software requirements for the program:

- It must not use any C++11 features that are not supported by C++98
- It must include a statically linked executable
- C++ source code in appropriately labelled files
- VS2010 (or 2008) C++ project file or a Makefile

# 4. Software Design

## UML Diagram




## Public Function List

### *Graph*

- isEmpty: Check if the graph is empty
- createGraph(*n*): Creates the graph from one of three different sources depending on the input *n*. If *n* = -1 the function calls createGraph("graph.txt"); for *n* = 0 function calls createGraph() and *n* > 0 creates a random graph with *n* nodes;
- createGraph: Creates the graph from the textbook figure 12-16 pg 708.
- createGraph(*s*): Creates the graph by reading in from file *s*.
- printGraph: Prints the graph to standard out
- depthFirstTraversal: Traverses the graph depth first
- dftAtVertex(*v*): Traverses the graph depth first starting at *v*

- breadthFirstTraversal: Traverses the graph breadth first
- numberOfVertices: Returns the number of nodes in the graph

### *MinimumSpanningTree*

- createSpanningGraph(*n*): Creates a spanning graph which calls createGraph from its parent.
- minimumSpanning(*v*): Provided algorithm for calculating minimum spanning $O(n^3)$
- prims2(*v*): Psuedocode provided algorithm for calculating minimum spanning $O(n^2)$
- printTreeAndWeight: Prints the tree, its edges and their corresponding weights plus the overall weight and other statistics.
- clearGraph: Clears the graph

## Data structures in the software

### *Array of LinkedLists*

- **Name:** graph
- **Type:** List
- **Purpose:** To keep a list of adjacencies of each Vertex

### *Queue*

- **Name:** queue
- **Type**: First in First out
- **Purpose**: Store vertices in the breadth first search algorithm

### *Array*

- **Name:** edges
- **Type:** int array
- **Purpose:** hold the minimal edges required for minimal spanning tree

### *Array*

- **Name:** edgeWeights
- **Type:** double array
- **Purpose:** hold the weights of the edges in the minimal spanning tree

### *Array*

- **Name:** weights
- **Type:** double 2d array
- **Purpose:** hold a matrix of weights for all edges in the spanning graph

**Detailed Design – Pseudocode for all non-standard and non-trivial algorithms that operate on datastructures**

```
Prim2(G,W,n,s)
        Let T = (V,E), where E = 0
        For every Vertex j
        Begin
                edgeWeightj = Ws,j
                edgesj = s
                visiteds = false
        End
        edgeWeights = 0
        visiteds = true

        While there are more nodes to visit
        Begin
                Select the node that has not been visited and
                has the smallest weight and call it k
                visitedk = true
                E = E U {(k, Edgesk)}
                V = V U {k}
                For each node j that is not visited
                If (Wk,j < edgeWeightk)
                Begin
                        edgeWeightk = Wk,j
                        edgesj = k
                End
        End
```

## 5. Requirement Acceptance Tests

| Acceptance Requirement No | Test | Implemented (Full/Partial/None) | Test Results (Pass/Fail) | Comments |
|---|---|---|---|---|
| 1 | Can print the graph from textbook fig 12-16 | Full | Pass | |
| 2 | Can read graph from text file | Full | Pass | |
| 3 | Can generate random graph | Full | Pass | |
| 4 | prims2 function produces same result as minimumSpanning function | Full | Pass | |
| 5 | printTreeAndWeight operates correctly | Full | Pass | |

## 6. Detailed Software Testing

| No | Test | Expected Results | Actual Results |
|---|---|---|---|
| **1.0** | **Standard Cases** | | |
| 1.1 | – Print textbook graph | List of vertices and their edges | Figure 1 in appendix |
| 1.1 | – Print graph from text file | List of vertices and their edges | Figure 2 in appendix |
| 1.2 | – Print randomized graph | List of vertices and their edges | Figure 3 in appendix |
| 1.3 | – User quits at menu | Application closes | User is asked to press any key before application is closed |
| **2.0** | **Error Cases** | | |
| 2.1 | – User tries to create randomized graph with > 100 nodes | Error message | Error message is displayed and graph is created with 100 nodes |

| No | Test | Expected Results | Actual Results |
|---|---|---|---|
| 2.2 | – User tries to enter negative number that is not -1 | Error message | Default “0” is selected and graph from textbook is printed |
| 2.3 | – User tries to enter invalid number for source vertex when prompted | Error message | Error message is displayed and vertex 0 is selected by default |
| 2.4 | – User tries to enter an invalid filename when prompted | Error message | Error message as expected |
| 2.5 | – User tries to enter non-digits that are not “q” at the menu | Error message | Default “0” is selected and graph from textbook is printed |
| 2.6 | – User tries to enter non-digits for source vertex | Error message | No error message and default 0 vertex is selected as source |

## 7. User Instructions

- User can load up the project using the Visual Studio 2008 solution file in the same directory as this Documentation.
- Statically compiled executable is available in the Release folder

## 8. Appendix

```
c:\Users\terry\Dropbox\Uni\7505ICT - Programming Algorithms\Assignment 2\s2587341\VS2008-...
[0]# 0

V[0]: 1, 2, 3
V[1]: 0, 4, 6
V[2]: 0, 5, 6
V[3]: 0, 4
V[4]: 1, 3, 5
V[5]: 2, 4
V[6]: 1, 2

Select source vertex # 0

Printing minimumSpanning:
Source Vertex: 0

  Edges         Weight
  =====         ======
( 6,  1)         4.00
( 0,  2)         5.00
( 0,  3)         2.00
( 1,  4)         2.00
( 2,  5)         7.00
( 2,  6)         5.00

Tree Weight:            25.00
Number of vertices:     7
Number of edges:        18

Printing prim2:
Source Vertex: 0

  Edges         Weight
  =====         ======
( 6,  1)         4.00
( 0,  2)         5.00
( 0,  3)         2.00
( 1,  4)         2.00
( 2,  5)         7.00
( 2,  6)         5.00

Tree Weight:            25.00
Number of vertices:     7
Number of edges:        18

Please choose one of the following or press enter for default:
        [ 0] Graph from Textbook figure 12-16 pg. 708 (default)
        [-1] Read graph in from file
        [>0] Create random graph with number of vertices specified (max: 100)
        [ q] Quit the application

[0]# _
```

*Figure 1*

```
c:\Users\terry\Dropbox\Uni\7505ICT - Programming Algorithms\Assignment 2\s2587341\VS2008-...

[0]# -1
Enter the filename or press enter for 'graph.txt':

[graph.txt]#

V[0]: 1, 3, 4
V[1]: 0, 2, 3, 4, 5
V[2]: 1, 3, 5, 6
V[3]: 0, 1, 2, 6
V[4]: 0, 1, 5, 9
V[5]: 1, 2, 4, 6, 8, 9
V[6]: 2, 3, 5, 7, 8
V[7]: 6, 8, 9
V[8]: 5, 6, 7, 9
V[9]: 4, 5, 7, 8

Select source vertex # 0

Printing minimumSpanning:
Source Vertex: 0

  Edges         Weight
  =====         ======
( 0,  1)         3.00
( 1,  2)         2.00
( 2,  3)         2.00
( 5,  4)         8.00
( 2,  5)         8.00
( 5,  6)         7.00
( 6,  7)         4.00
( 7,  8)         1.00
( 8,  9)         3.00

Tree Weight:            38.00
Number of vertices:     10
Number of edges:        42

Printing prim2:
Source Vertex: 0


  Edges         Weight
  =====         ======
( 0,  1)         3.00
( 1,  2)         2.00
( 2,  3)         2.00
( 5,  4)         8.00
( 2,  5)         8.00
( 5,  6)         7.00
( 6,  7)         4.00
( 7,  8)         1.00
( 8,  9)         3.00

Tree Weight:            38.00
Number of vertices:     10
Number of edges:        42
```

*Figure 2*

c:\Users\terry\Dropbox\Uni\7505ICT - Programming Algorithms\Assignment 2\s2587341\VS2008-...

```
        [-1] Read graph in from file
        [>0] Create random graph with number of vertices specified (max: 100)
        [ q] Quit the application

[0]# 8

V[0]: 0, 2
V[1]: 1, 4
V[2]:
V[3]: 4, 6
V[4]: 0, 2
V[5]: 4, 5, 7
V[6]: 0, 1, 3, 6
V[7]: 2, 4

Select source vertex # 0

Printing minimumSpanning:
Source Vertex: 0

  Edges         Weight
  =====         ======
( 0,  1)         3.00
( 1,  2)         2.00
( 2,  3)         2.00
( 5,  4)         3.00
( 2,  5)         8.00
( 5,  6)         7.00
( 6,  7)         4.00

Tree Weight:            29.00
Number of vertices:     8
Number of edges:        17

Printing prim2:
Source Vertex: 0

  Edges         Weight
  =====         ======
( 0,  1)         3.00
( 1,  2)         2.00
( 2,  3)         2.00
( 5,  4)         3.00
( 2,  5)         8.00
( 5,  6)         7.00
( 6,  7)         4.00

Tree Weight:            29.00
Number of vertices:     8
Number of edges:        17
```

*Figure 3*