Software Requirements Specifications: Minimum Spanning Tree

D Jayme Green

April 11, 2016

Contents

1 Introduction **3**
- 1.1 Purpose . . . 3
- 1.2 Scope . . . 3
- 1.3 Definitions, Acronyms, and Abbreviations . . . 3
- 1.4 References . . . 5
- 1.5 Overview . . . 5

2 Overall Description **5**
- 2.1 Product Perspective . . . 5
  - 2.1.1 System Interfaces . . . 5
  - 2.1.2 User Interfaces . . . 6
  - 2.1.3 Hardware Interfaces . . . 6
  - 2.1.4 Software Interfaces . . . 6
  - 2.1.5 Communications Interfaces . . . 7
  - 2.1.6 Memory Constraints . . . 7
  - 2.1.7 Operations . . . 7
  - 2.1.8 Site Adaptation Requirements . . . 7
- 2.2 Product Functions . . . 7
- 2.3 User Characteristics . . . 8
- 2.4 Constraints . . . 8
- 2.5 Assumptions and Dependencies . . . 8
- 2.6 Apportioning of Requirements . . . 8

3 Specific Requirements **9**
- 3.1 External Interfaces . . . 9
- 3.2 Functions . . . 11
- 3.3 Performance Requirements . . . 12
- 3.4 Logical Database Requirements . . . 12
- 3.5 Design Constraints . . . 12
  - 3.5.1 Standards Compliance . . . 12
- 3.6 Software System Attributes . . . 13
  - 3.6.1 Reliability . . . 13
  - 3.6.2 Availability . . . 13
  - 3.6.3 Security . . . 13
  - 3.6.4 Maintainability . . . 13
  - 3.6.5 Portability . . . 13

1 Introduction

1.1 Purpose

This software requirement specification is for the minimum spanning tree project introduced in Ms. Canosa's Analysis of Algorithms class in the spring 2016 section. The document is meant for the designer and programmer (and author of this document) which is D Jayme Green. Also, the document is meant for the reviewer and grader Professor Childs.

1.2 Scope

The minimum spanning tree analysis creates a randomized, undirected, connected graph which will be utilized to produce a minimum spanning tree. The minimum spanning tree will be produced using Krusal's algorithm and Prim's algorithm in both an adjacency list and an adjacency matrix. Kruskal's algorithm will also be examine with different algorithms of sorting including insertion, count, and quick sort.

The software's goal will be to execute each of the algorithms and report the time of each execution. This will enable the user to examine and determine the most efficient way to implement a minimum spanning tree under different circumstances including the amount of nodes in the tree and whether the graph was sparse or dense.

The software will run on any computer with any specifications. The program does not have any upper bounds for the amount of nodes and, therefore, the user can make the graph infinity large. Depending on the user's computer specification, the graph will have a limited amount of nodes (also depending on what the probability of edges the user has specified) that it can use without crashing.

1.3 Definitions, Acronyms, and Abbreviations

Graph

A computing data structure that has data in the form of nodes. These nodes can be connected to other nodes by edges which have weights associated with them.

Node

A node holds data together and is used in a multitude of computing data structures. In graphs, there are multiple nodes which can or cannot be connected together via edges.

Edge

An edge connects nodes (and, therefore, data) together. These edges can be weighted or unweighted.

Weight
An edge can have an optional attribute associated with them called an weight. This weight can be any type of value, but most commonly an integer. Weights can be considered the "cost" to get from the beginning node to the other connected node. In this minimum spanning tree project, the edges have an integer weight associated with them.

Connected Graph
Any node can be reached by any other node by traversing through edges.

Sparse Graph
A graph which has very little edges to node ratio.

Dense Graph
A graph which has a high edges to node ratio.

Cycle
When edges of nodes are connected in a way that if traversing in one direction through edges the starting node is reached again. Acyclic is a graph which does not have a cycle.

Minimum Spanning Tree (MST)
A sub-graph in the graph given which has the minimum amount of weights on edges that connects all the nodes together. Can be resolved by using one of two algorithms: Kruskal's algorithm and Prim's algorithm.

Kruskal's Algorithm
One of the algorithms that creates a minimum spanning tree. It involves sorting all edges from least to greatest by weight and then selecting the smallest edge that does not create a cycle until every node is reached.

Prim's Algorithm
One of the algorithms that creates a minimum spanning tree. It involves selecting an arbitrary node and putting it into the minimum spanning tree. Then, select the lowest edge from the minimum spanning tree to a node outside of the MST and add that node. This process continues until no more nodes are outside the MST.

Time Complexity
How much time it takes to run the algorithms. A high time complexity means the algorithm runs slower than another algorithm with a low time complexity. The purpose of the program is to compare the time complexity of different ways to implement getting a MST.

Adjacency Matrix
A 2-D matrix with the rows and column numbers representing the node number and the meeting point of the row, column being the edge weight between the two nodes. If the weight is represented as 0 in the matrix, there is no edge between the two nodes. This is one way to represent a graph's connections.

Adjacency List

A list within a list which can represent a graph's connections. The first list's indexes correspond with each node in the graph. The list within each index represents the connections that the node has with the index node. The weights of these connections are further described inside the edges which can be obtained using the two nodes.

1.4 References

CSCI-261 Programming Project: Overview
(https://www.cs.rit.edu/ rlc/Courses/Algorithms/
Projects/20152/Overview/overview.html)
CSCI-261 Programming Project: Graph Generation
(https://www.cs.rit.edu/ rlc/Courses/Algorithms/
Projects/20152/Proj1/proj1.html)
CSCI-261 Programming Project: Sorting Edges
(https://www.cs.rit.edu/ rlc/Courses/Algorithms/
Projects/20152/Proj2/proj2.html)
CSCI-261 Programming Project: Kruskal's Algorithm
(https://www.cs.rit.edu/ rlc/Courses/Algorithms/
Projects/20152/Proj3/proj3.html)
CSCI-261 Programming Project: Prim's Algorithm
(https://www.cs.rit.edu/ rlc/Courses/Algorithms/
Projects/20152/Proj4/proj4.html)
CSCI-261 Programming Project: Project Report
(https://www.cs.rit.edu/ rlc/Courses/Algorithms
/Projects/20152/Report/report.html)
Software Requirements Specifications IEEE Std-830
(https://mycourses.rit.edu/d2l/le/content/587211/
viewContent/3657980/View?ou=587211)

1.5 Overview

This software requirements specification contains all requirements, user specifications, and expectations of the analysis of algorithms minimum spanning tree project given by Professor Canosa during the Spring 2016 Rochester Institute of Technology semester.

The following SRS is organized using IEEE Std-850 and further specified by Professor Childs in his guide written and put on his myCourses website.

2 Overall Description

2.1 Product Perspective

The product is written in Java and the computer that runs the software must have Java of at least version 7.0. Using versions greater than 7.0 (including version 8.0) is acceptable. With most computers having Java version 7.0+ on it already, the program is self-contained and independent.

2.1.1 System Interfaces

The minimum spanning tree project has been tested and supports Windows 10, Windows 8.1, and Linux operating system. While untested, the project should be supported by Windows 7, Windows XP, and recent versions of OSX devices as long as they have Java 7.0 (or greater) installed and working on the machine.

The program also requires support of either a file (for Linux, OSX, or Windows) or of .txt file (Windows) depending on the operating system. The user must specify what format of file they anticipate using in order for the programming to give the correct, supported version of the program.

The software does not require any Internet connection.

2.1.2 User Interfaces

The user has few interfaces in the minimum spanning tree software. The user must be comfortable with using files or .txt files and must specify three parameters in exactly a certain format. The parameters include the amount of nodes, the seed of the random, and the probability of edges. The file follows the exact format below without the parenthesis which indicate the range of values for that particular item:

Node (2-infinity)

Random Seed (1-infinity)

Probability of Edges (.01-1)

The final result given at the end of software depends on the node parameter given above. If the node parameter above is below 10, the program will print the parameter values given above, the time it took to generate the graph, the adjacency matrix, the adjacency list, the times to generate both, the sorts on the edges using insertion, count, and quick sort with the time it takes to do each on both the adjacency list and adjacency matrix, Kruskal's algorithm and time, and Prim's algorithm and time. If the node value is 10 or greater, the program prints out the values given above and the times that each of the above algorithms have taken.

2.1.3 Hardware Interfaces

The program supports running by the command line as well as running as an executable. And since the program and the output relies on the terminal or an executable, the software can be put in full-screen if the user wants.

The program does not support nor need any additional ports or other hardware components.

2.1.4 Software Interfaces

The software utilizes Java 7.0 which its source can be opened via any Java editor like Eclipse, Intellij, vi, etc. Since the program is written in Java, the computer running the program must also have Java as specified in the CSCI-261 Programming Project: Overview (can be accessed under the reference section).

The software also requires an Operating System of either Windows XP, Windows 7, Windows 8.1, Windows 10, Linux, and OSX. These source files can be opened by in a notepad or file-opening program. Java requires an Operating System which is specified in the CSCI-261 Programming Project: Overview reference file.

2.1.5 Communications Interfaces

The program does not require any Internet or any other communication protocols.

2.1.6 Memory Constraints

The program has no upper limits restraints for the amount of nodes. With a high probability of edge connections, the chances of a dense graph is extremely high. A dense graph greatly uses memory over a sparse graph. A dense graph with a greater amount of nodes can easily use a significant amount of memory and could possibly crash the computer using the software. The computer using the program significantly impacts how many nodes in the graph.

The software should not produce any memory leaks nor secondary memory as the program is created in Java and the Java garbage collector should remove any memory leak possibilities.

2.1.7 Operations

The user must supply a file or .txt file with the amount of nodes, the seed for the random, and the probability of there being an edge connection. The file/.txt file must be in the following format:

Node (2-infinity)

Random Seed (1-infinity)

Probability of Edges (.01-1)

The program does not require any additional user operations.

2.1.8 Site Adaptation Requirements

The inputted file must be in the following format with the ranges specified next to the titles:

Node (2-infinity)

Random Seed (1-infinity)

Probability of Edges (.01-1)

The program does not require any additional changes to revert back after finishing. The user just must specify what parameters they want within the fields in the file.

2.2 Product Functions

The minimum spanning tree program requires the user to supply it with either a file or .txt file with the amount of nodes, the random seed, and the percent of probability of an edge the user wants as formatted above. The graph then takes these values and produces a random graph with the same attributes. The graph's information is stored in an adjacency matrix as well as an adjacency list.

With the completed graph, the program then sorts using insertion sort, count sort, and quick sort on the adjacency matrix and adjacency list recording the time to do each of the sorts. The graph then does Kruskal's algorithm storing how much time used. The graph afterward does Prim's algorithm and stores the amount of time it took.

Afterward, the software will print out the values given in the file. If the amount of nodes is greater than or equal to 10, the software will print out each of the times it took to complete each algorithm. If the program had less than 10 nodes, the program will print how the adjacency list, adjacency matrix, and other data structures look before and after each step of the algorithm processes stated above with times adjacent to them.

2.3 User Characteristics

The user should have basic knowledge of files or .txt files as modifying them are a necessity of the program. Additionally, while the user does not need to know

computer science concepts to run the program, the context of the program was aimed toward programming-knowledgeable user who does know the difference between an adjacency list and an adjacency matrix as well as all the algorithms the program states.

2.4 Constraints

The program has more running options (with more nodes) with the more advanced, higher-spec computer. Yet, even a slow computer can run the program in limited parameter range.

The software, while an independent product of the programmer, is specified, reviewed, and graded by Professor Canosa. While some freedoms can be established with certain coding and design, some algorithms must be present with formats exactly stated in her requirements (as seen in the references).

2.5 Assumptions and Dependencies

The only assumption would be that the programmer follows the requirements of the project exactly without any further involvement. Optionally to the programmer, the software could have additional experimentations and features not included in this SRS.

2.6 Apportioning of Requirements

The program currently works without of few key features. Since this is a project for a computer science class, the program is broken into four main components (and currently is in the third stage). These all have due dates before the project is due. But these requirements are only the minimum with any additional experimentation optional for the programmer which could result in later patches after the final release (and graded) product.

3 Specific Requirements

3.1 External Interfaces

The input required by the program includes a file or .txt which has the amount of nodes in the first line, the seed of randomness in the second line, and the percent of probability of an edge connection in the third line. The ranges of the values are 1-infinity in nodes, 1-infinity in randomness, and .01-1 in percentage of an edge connections. These parameters into the program control the generation of the random graph. The nodes indicate the number of nodes in the graph. The seed makes it so the random becomes able to be repeated. Finally,

the percent of edges uses the random from the seed to calculate whether there should be an edge or not between two nodes.

The output depends on the input due to what graph is produced given the inputs. The output of the program if the the nodes is greater than 10 is:

TEST n=XX, seed=XX, p=XX The XX represents the parameters given in the input file

Time to generate the graph: XX milliseconds The XX represents the amount of milliseconds it took for the graph to generate depending on the user's input for the parameters

KRUSKAL WITH MATRIX USING INSERTION SORT

Total weight of MST using Kruskal: XX The XX represents the total amount of weights on all the of the edges in the graph.

Runtime: XX milliseconds The XX represents the amount of time in milliseconds it took to do Kruskal's algorithm using the adjacency matrix and insertion sort.

KRUSKAL WITH MATRIX USING COUNT SORT

Total weight of MST using Kruskal: XX The XX represents the total amount of weights on all the of the edges in the graph.

Runtime: XX milliseconds The XX represents the amount of time in milliseconds it took to do Kruskal's algorithm using the adjacency matrix and count sort.

KRUSKAL WITH MATRIX USING QUICKSORT

Total weight of MST using Kruskal: XX The XX represents the total amount of weights on all the of the edges in the graph.

Runtime: XX milliseconds The XX represents the amount of time in milliseconds it took to do Kruskal's algorithm using the adjacency matrix and quick sort.

KRUSKAL WITH LIST USING INSERTION SORT

Total weight of MST using Kruskal: XX The XX represents the total amount of weights on all the of the edges in the graph.

Runtime: XX milliseconds The XX represents the amount of time in milliseconds it took to do Kruskal's algorithm using the adjacency list and insertion sort.

KRUSKAL WITH LIST USING COUNT SORT

Total weight of MST using Kruskal: XX The XX represents the total amount of weights on all the of the edges in the graph.

Runtime: XX milliseconds The XX represents the amount of time in milliseconds it took to do Kruskal's algorithm using the adjacency list and count sort.

KRUSKAL WITH LIST USING QUICK SORT

Total weight of MST using Kruskal: XX The XX represents the total amount of weights on all the of the edges in the graph.

Runtime: XX milliseconds The XX represents the amount of time in milliseconds it took to do Kruskal's algorithm using the adjacency list and quick sort.

PRIM WITH ADJACENCY MATRIX

Total weight of MST using Prim: XX The XX represents the total amount of weights on all the of the edges in the graph.

Runtime: XX milliseconds The XX represents the amount of time in milliseconds it took to do Prim's algorithm using the adjacency matrix.

PRIM WITH ADJACENCY LIST

Total weight of MST using Prim: XX The XX represents the total amount of weights on all the of the edges in the graph.

Runtime: XX milliseconds The XX represents the amount of time in milliseconds it took to do Prim's algorithm using the adjacency list.

If the nodes is less than ten, the output is similar to above but includes more details about the graph. Below the time to generate the graph, the following is printed:

The graph as an adjacency matrix: In the following lines, the graph's adjacency matrix (as represented in the program) is printed as a 2-D array.

The graph as an adjacency list: The graph, once again, prints as it looks like in the adjacency list (ArrayList within a ArrayList). Every node number is printed vertical with an arrow (-¿) pointing. Following the arrow, all of the node numbers which connect to the that vertical node number is displayed with its edge weight between them adjacent with the weight in parenthesis.

Depth-First Search:

Vertices: Display all of the numbered nodes.

Predecessors: Upon doing a depth-first search, the predecessors are the nodes that came right before the node in the search. They are printed with the first predecessor printed being the predecessor of the 0th node (using base-0 counting).

The following format goes between the uppercased titles of the sorts and algorithm and the total weight of the graph after the algorithm.

X Y weight = ZZ The X represents one node on this edge with Y being the other node. The ZZ represents the weight of the edge between X and Y. This format continues with all the edges in the graph in sorted order of least to greatest of the weights, then least of the "left" node, and finally the least of the "right" node.

3.2 Functions

The input files contains three separate parameters including the amount of nodes in the graph, the seed randomness, and the percentage of whether there is an edge or not between two nodes. There are five separate error handling and catching the program does:

Input file not found When the user does not put in a file or the file cannot be found by the program.

N and Seed must be integers The N and Seed must be whole number integer values. This error message with show when the user puts an invalid type (like a String) or puts in a decimal place (like a Double or Float).

N must be greater than 1 Since N represents the amount of nodes in the graph, the graph must be a graph and have at least 2 nodes. The program will also detect and reject negative numbers.

P must be a real number The percentage of edges must be a number. If the user gives a String or another invalid number type, this error message will display.

P must be between 0 and 1 Since P is the percentage of getting an edge between two nodes, the value must be between 0 and 1.

These errors are checked in the ordered listed and will catch and display the error message for the first error it detects. After catching the error, the program will display the error message and gracefully end the program.

The output of the program is specified in the above section.

3.3 Performance Requirements

The software uses more or less hardware depending on the user's parameter input. If the user selects a large, dense graph, the program will require much more processing power than if the user selected a small, sparse graph. Also, since the user can put whatever parameters he/she wants, the software can try to execute a graph too big for the computer and either take a very large time or crash the computer.

3.4 Logical Database Requirements

The software does not utilize a database.

3.5 Design Constraints

3.5.1 Standards Compliance

The program uses basic Java stylization and naming mechanics. The names of all variables have the first word lowercase with the following uppercase and the comments follow the Javadoc formatting. Yet, these standards are lightly

enforced with good programming organization more important than style standard.

There are no additional constraints and standards compliance for the software.

3.6 Software System Attributes

3.6.1 Reliability

The software must run on an Linux OS and not crash due to any of the error catching specified above. Also, for no mater the parameters, the software must produce a correct minimum spanning tree within an efficient time frame.

3.6.2 Availability

The software can only be installed by sharing of the .java files (and executable if asked). Because this program was designed for a college class, the availability of the program is limited and meant for commercial release.

3.6.3 Security

The software does not store any sensitive information or access any part of the machine which contains any important, sensitive information. Because of this, there is no security associated with the software.

3.6.4 Maintainability

Since the program relies on already discovered algorithms, the software should require no maintenance in order to continue to work. Unless Java does not back-support (which it always have in the past), the program does not rely on any third-party software that may make it unable to run in the future.

3.6.5 Portability

In order to switch from OSX and Linux to Windows (and vice versa) for the software, 2 lines of code must be commented and uncommented out. These lines are labeled and at the beginning of the program involving the file input. The Windows version requires a concatenation of ".txt" at the end of the file in. The OSX and Linux will crash with the addition of the ".txt". These lines are both in the code an commented/uncommented depending on what operating system the software is going to run on.

Besides the above, the software should be able to run on Linux, OSX, and

Windows.