Software Design Document for Graph Algorithm

By

Tanmay Kanti Biswas

Date: 17/06/2017

Contents

[1. Approach 3](#_Toc485477991)

[2. Use cases 3](#_Toc485477992)

[2.1. Creating a process and scheduling multiple threads 3](#_Toc485477993)

[2.2. Reading neighbour information from a file during thread creation 3](#_Toc485477994)

[2.3. Building a network graph using all neighbours information 3](#_Toc485477995)

[2.4. Finding shortest path from any two nodes 4](#_Toc485477996)

[2.5. Finding minimum spanning tree 4](#_Toc485477997)

[2.5.1. Reconfiguration of graph when any change happens in the network 4](#_Toc485477998)

[3. Design 5](#_Toc485477999)

[3.1. High Level Design 5](#_Toc485478000)

[3.2. Detail Design 6](#_Toc485478001)

[3.2.1. Class NetworkProcess\_t 6](#_Toc485478002)

[3.2.2. Class ProcessInfo \_t 6](#_Toc485478003)

[3.2.3. Class NetworkGraph \_t 6](#_Toc485478004)

[4. Control Flow 7](#_Toc485478005)

[5. Development Environment 7](#_Toc485478006)

[6. Test cases 7](#_Toc485478007)

# Approach

To come up with the most practical solution for the Graph I approach in the following steps:

* Step 1: Requirement analysis and Use case identification
* Step 2: High level design for each use cases and detailed design for those
* Step 3: Control flow design
* Step 4: Development platform selection and start developing
* Step 5: Debugging using IDE and bug fixing
* Step 6: Testing

# Use cases

# Creating a process and scheduling multiple threads

This use case shall create a single process which will create multiple threads for each node which will run concurrently along with other threads

|  |  |  |
| --- | --- | --- |
| Sl no | Requirement Id | Description |
|  |  | Create a process |
|  |  | Create multiple threads from the process |
|  |  | Monitors threads if they are alive or not |

# Reading neighbour information from a file during thread creation

This use case shall read the neighbouring nodes information from a text file when it is created

|  |  |  |
| --- | --- | --- |
| Sl no | Requirement Id | Description |
|  |  | Read the text file and gather neighbour information |
|  |  | Create data structure with the information |
|  |  | Re-read information if recreated and repeat R2.2 |

# Building a network graph using all neighbours information

This use case shall build a network graph using all neighbours’ information available.

|  |  |  |
| --- | --- | --- |
| Sl no | Requirement Id | Description |
|  |  | Collect all nodes information |
|  |  | Create a network graph using a graph algorithm |

# Finding shortest path from any two nodes

This use case shall find the shortest path from any two nodes given in the network

|  |  |  |
| --- | --- | --- |
| Sl no | Requirement Id | Description |
|  |  | Take two nodes information as input |
|  |  | Check if both nodes are reachable and show messages if not |
|  |  | Find the shortest path between these two nodes and return result |

# Finding minimum spanning tree

This use case shall find the minimum spanning tree in the network

|  |  |  |
| --- | --- | --- |
| Sl no | Requirement Id | Description |
|  |  | Take nodes information as input |
|  |  | Check if both nodes are reachable and show messages if not |
|  |  | Find the minimum spanning tree in the network and return result |

# Reconfiguration of graph when any change happens in the network

This use case shall periodically monitor the network whether any discontinuity occurred in the network and shall reconfigure the network

|  |  |  |
| --- | --- | --- |
| Sl no | Requirement Id | Description |
|  |  | Periodically monitor each thread status in the main thread |
|  |  | If any thread dies update the graph |
|  |  | If any thread gets alive update the graph |
|  |  | Rebuild network graph after any changes |
|  |  | Check for use case 2.4 and 2.5 |

# Design

# High Level Design

Class **NetworkGraph\_t::**

Create\_graph()

Find\_shortest\_path()

Find\_minimum\_span\_tree()

Reconfigure\_graph()

Class **NetworkProcess\_t::**

Create\_process()

Create\_thread()[]

Restart\_process()

Restart\_thread()[]

Class **ProcessInfo\_t::**

Read\_file()

Update\_node()

# Detail Design

# Class NetworkProcess\_t

* Create\_process()
* This function shall create a process which will run continuously
* Create\_thread()[]
* This function shall create multiple threads based on the requirement
* Restart\_process()
* This function shall restart the process when it it is terminated.
* Restart\_thread()[]
* This function shall restart the thread if any thread is terminated.

# Class ProcessInfo \_t

* Read\_file()
* This function shall read the neighbour information from a file
* Update\_node()
* This function shall update the data structure after any changes in the network

# Class NetworkGraph \_t

* Create\_graph()
* This function shall create a graph using the nodes created by threads
* Find\_shortest\_path()
* This function shall implement shortest path algorithm between two nodes
* Find\_minimum\_span\_tree()
* This function shall implement minimum span tree in the network
* Reconfigure\_graph()
* This function shall reconfigure the graph after any changes happen in the network

# Control Flow

Create Process

Create Threads

yes

Read File

yes

If thread died

If process died

Create Graph

no

no

Compute SP/MST

# Development Environment

For the development following resources used

* Linux Operating System
* Eclipse IDE (for Editing, compilation & debugging)
* C/C++ programming Language

# Test cases

Perform following test cases in the application

|  |  |  |
| --- | --- | --- |
| Sl No | Test Case | Pass/Fail |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |