**HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY**

**School of Information and Communication Technology**

Software Requirement Specification – SRS

Version 1.0

Application for demonstrating Traveling Salesman Problem

Course: Software Engineering

Group: 4

Nguyễn Đăng Ninh

Phùng Hải Nguyên

Nguyễn Trường An

*Hanoi, May 9th, 2021*

Contents

[1. Introduction 3](#_Toc71481280)

[1.1. Purpose 3](#_Toc71481281)

[1.2. Scope 4](#_Toc71481282)

[1.3. Glossary 4](#_Toc71481283)

[1.4. Reference documents 5](#_Toc71481284)

[2. General description 5](#_Toc71481285)

[2.1. Objective 5](#_Toc71481286)

[2.2. General Use Case diagram 5](#_Toc71481287)

[2.3. Decompose Use Case diagram 7](#_Toc71481288)

[2.3.1. Decomposition of the “Input graph” usecase 7](#_Toc71481289)

[2.4. Procedure 7](#_Toc71481290)

[2.4.1. Procedure of using the application 7](#_Toc71481291)

[2.4.2. Procedure of inputting graph’s information 8](#_Toc71481292)

[2.4.3. Procedure of giving visual representation 8](#_Toc71481293)

[3. Functional Requirements 9](#_Toc71481294)

[3.1. Use Case UC001 “Input Graph” 9](#_Toc71481295)

[3.2. Use Case UC002 “Giving visual representation” 11](#_Toc71481296)

# Introduction

## Purpose

This document declares the Traveling Salesman Problem, modelizes the problem into a graph and attempts to give out three different solutions: using minimum spanning tree, using dynamic programming and using naïve programming. All the solutions are visually demonstrated using JavaFX. This document describes the purposes and the functionalities of the application, the interface and how the application reacts to interactions.

## Scope

The **travelling salesman problem**(also called the **traveling salesperson problem** or **TSP**) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?". TSP can be modelled as an undirected weighted graph, such that cities are the graph's vertices, paths are the graph's edges, and a path's distance is the edge's weight. It is a minimization problem starting and finishing at a specified vertex after having visited each other vertex exactly once. Often, the model is a complete graph (i.e., each pair of vertices is connected by an edge). If no path exists between two cities, adding a sufficiently long edge will complete the graph without affecting the optimal tour.

In this application, the user can create a graph with selected numbers of nodes and selected weighted edges. Then, they will be able to choose one of the three approaches to solve the problem and the chosen approach will be visually demonstrated to the users.

## Glossary

* In a graph, vertices are represented by points and edges are represented by curves such that any two edges intersect at most in a finite number of points.
* Weight: the number which is implemented to each branch (edge) of the graph.
* Undirected weighted graph: a set of objects (called vertices or nodes) that are connected together, where all the edges are bidirectional. An undirectedgraph is sometimes called an undirected network.
* MST (Minimum Spanning Tree): a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight.
* Dynamic Programming: Dynamic Programming is mainly an optimization over plain [recursion](https://www.geeksforgeeks.org/recursion/). The idea is to simply store the results of subproblems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial.
* Naïve programing: or called Naïve implementation, is an implementation that has taken shortcuts for the sake of simplicity or by lack of knowledge.

## Reference documents

* <https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/>
* <https://www.geeksforgeeks.org/travelling-salesman-problem-set-2-approximate-using-mst/>

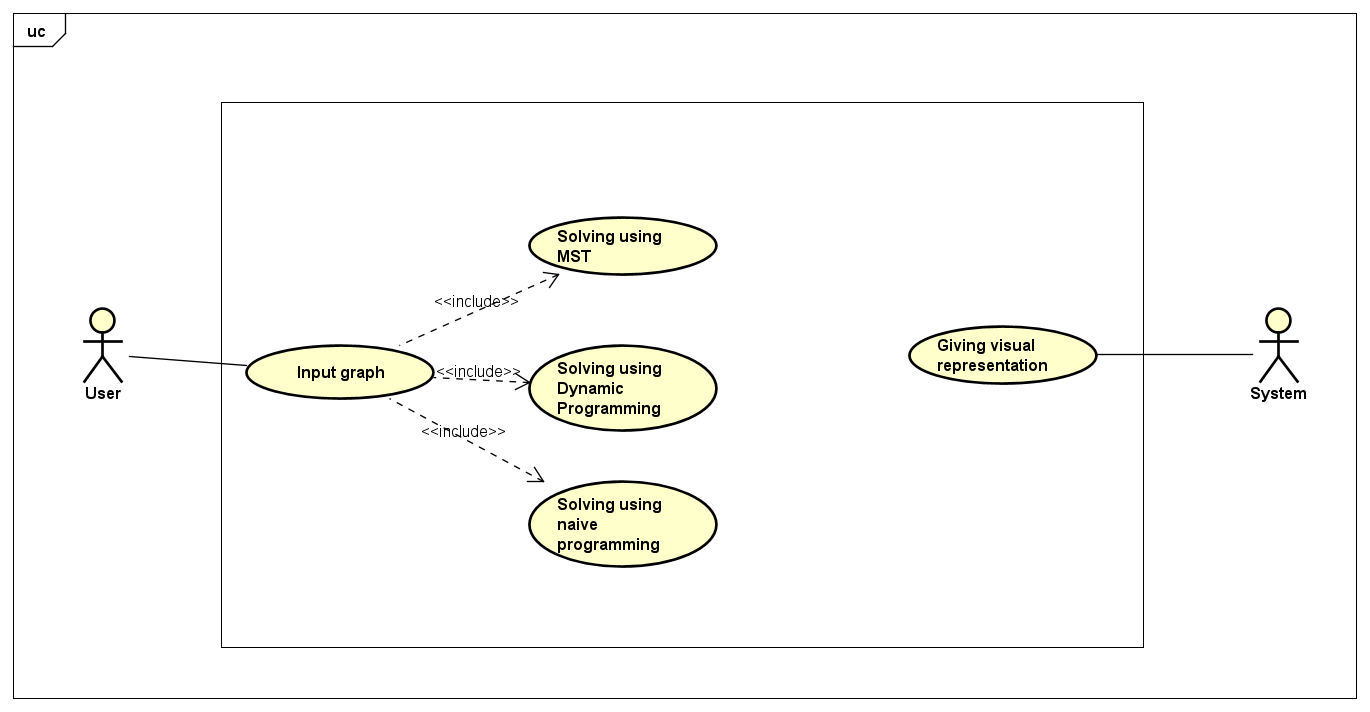
# General description

## Objective

The application includes 2 targets: User and System.

## General Use Case diagram

The user can input graph’s information and then select the way the TSP problem be solved. The system will take all of user’s input information to give the according visual representation of the selected approach.



## Decompose Use Case diagram

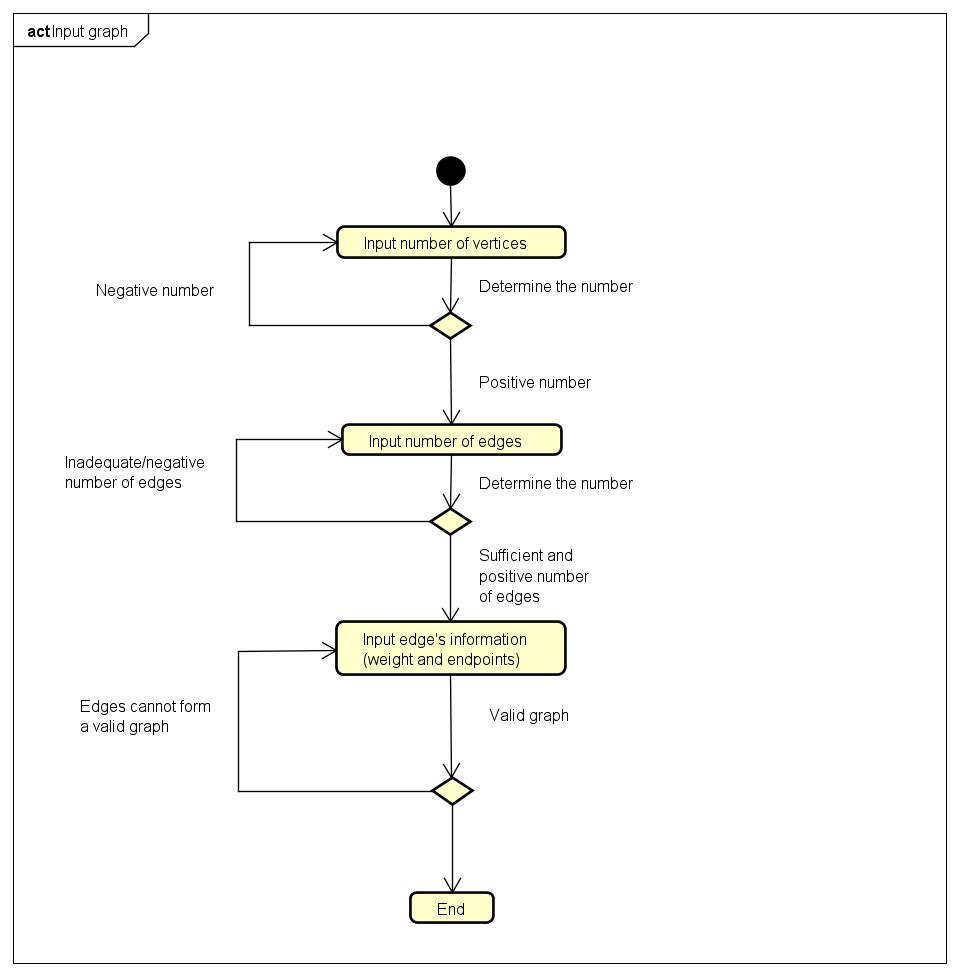
### Decomposition of the “Input graph” usecase

## Procedure

### Procedure of using the application

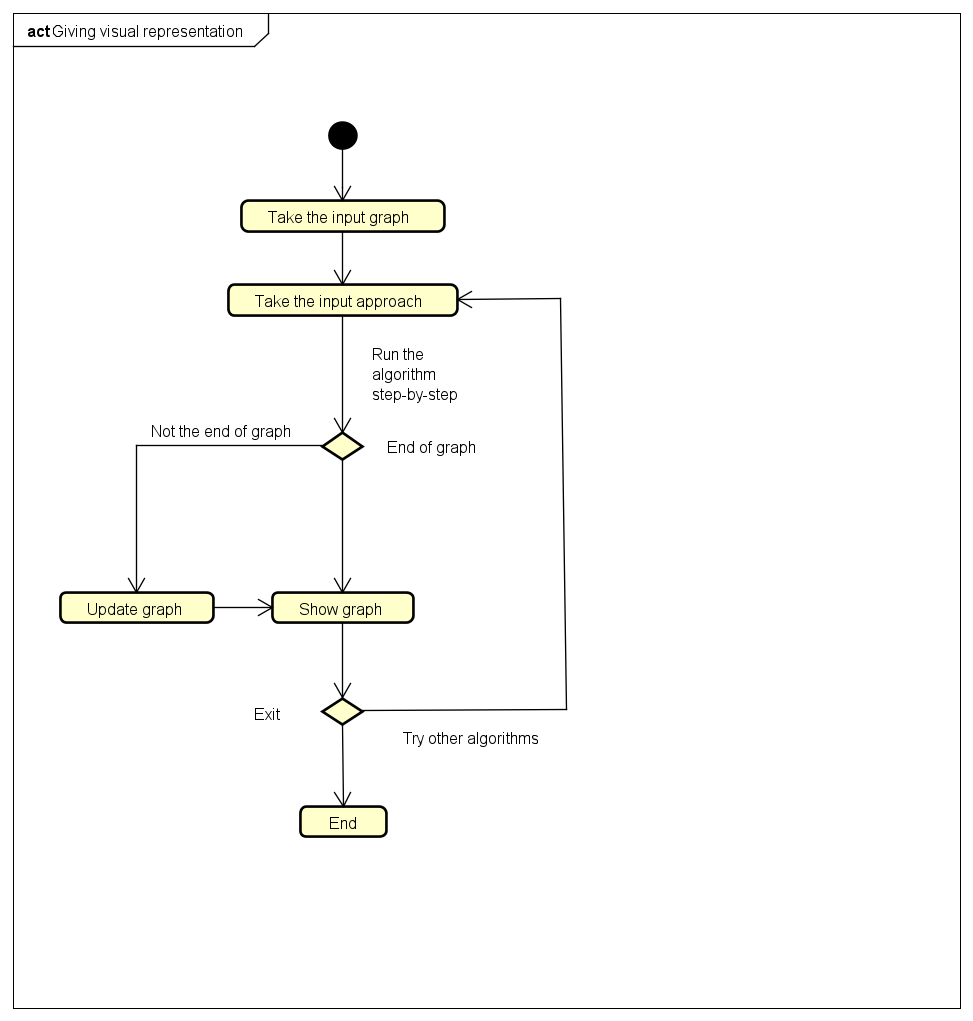
When opening the application, the system makes the user input necessary data for the graph of the problem. The user will then chooses what kind of algorithm to be solved. What the system will do is to give a visual demonstration to solve the problem using whatever algorithm is chosen.

### Procedure of inputting graph’s information

The application makes the user input the number of vertices, the number of edges, edges information (weight and endpoints). All will be checked whether satisfied the condition or not. If not, the user has to input again. All information inputted above will be used for visual representation.

### Procedure of giving visual representation

The procedure of the system outputting the visual representation to the screen to the user can be found in details in the graph below. The user will be able to see the changes in the graph with each step of the algorithm being executed (which leads to the change of the graph). When the algorithm ends, the program also ends, leaving on the screen the final graph with all nodes traversed.



# Functional Requirements

# Use Case UC001 “Input Graph”

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Code** | UC001 | **Use Case Name** | Input Graph |
| **Trigger** | User | | |
| **Precondition** | None | | |
| **Basic Path** | |  |  |  | | --- | --- | --- | | **No.** | **Performed by** | **Action** | | 1. | System | Display the interface for user to input data | | 2. | User | Input the number of vertices | | 3. | System | Check if the number sastisifies the condition | | 4. | User | Input number of edges | | 5. | System | Check if the number sastisifies  the condition | | 6. | User | Input edge’s weight and endpoints | | 7. | System | Check if the values can form a valid graph | | 8. | System | Call Use Case “Giving visual representation” | | | |
| **Alternative Path** | |  |  |  | | --- | --- | --- | | **No.** | **Performed by** | **Action** | | 3a. | System | Inform user about invalid input and let them input again | | 5a. | System | Inform user about invalid input and let them input again | | 7a. | System | Inform user about invalid input and let them input again | | | |
| **Postcondition** | None | | |

* Graph’s information includes:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No. | Data field | Description | Compulsory ? | Condition | Example |
| 1. | Number of vertices |  | Yes | The number must be possitive | 5 |
| 2. | Number of edges |  | Yes | The number must be possitive | 3 |
| 3. | Edge’s weight | Input an integer | Yes |  | -8 |
| 4. | Edge’s end points | Two positive integers separated by a space | Yes | The integers must be valid vertices | 1 3 |

## Use Case UC002 “Giving visual representation”

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Code** | UC002 | **Use Case name** | Giving visual reprsentation |
| **Trigger** | System | | |
| **Precondition** | Graph inputted successfully | | |
| **Basic Path** | |  |  |  | | --- | --- | --- | | **No.** | **Performed by** | **Action** | | 1. | System | Display algorithm menu | | 2. | User | Choose algorithm to visualize | | 3. | System | Run chosen algorithm | | 4. | System | Display graph visualization when graph is completed | | 5. | User | Choose to exit or try other algorithms | | | |
| **Alternative Path** | |  |  |  | | --- | --- | --- | | **No.** | **Performed by** | **Action** | | 3a. | System | If graph is not completed then update graph | | 5a1. | System | If user choose to exit then end use case | | 5a2. | System | If user choose to try other algorithms then call Use Case “Giving visual representation” again | | | |
| **Postcondition** | None | | |