CHAPTER 8
SOLVING MINIMUM SPANNING TREE PROBLEM USING PRIM'S ALGORITHM

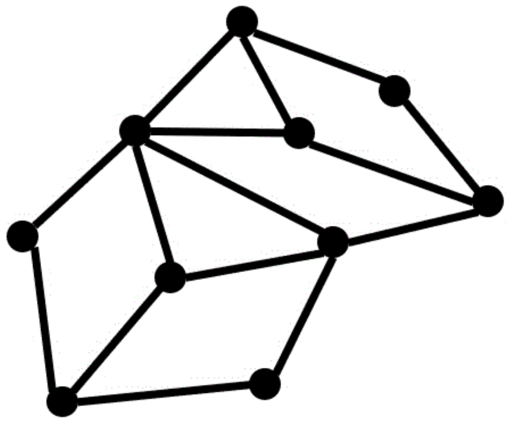
8.1 Learning Objectives

1. Students are able to implement Prim's algorithm to solve minimum spanning tree problem

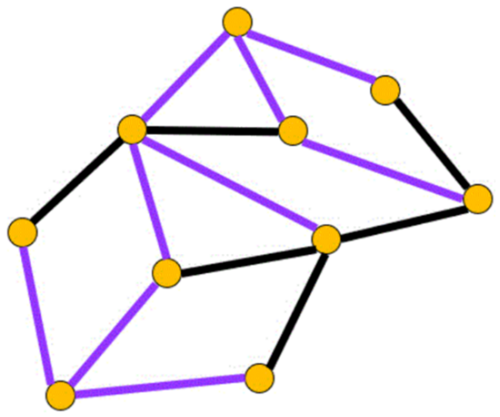
8.2 Material

8.2.1 Minimum Spanning Tree Problem

A spanning tree is a graph that contains all the vertices of the original graph and has selected edges as a tree. Note that, different from a graph, a tree cannot have a cycle.



For example, you can create a tree that passes through all the vertices by choosing the edges represented in purple below. This set of purple edges is one of the spanning trees.



Among the spanning trees of a weighted graph, the spanning tree with the smallest total weight of the selected edges is called the minimum spanning tree (MST). This problem can be solved using two famous algorithms, namely Kruskal and Prim algorithms. In this lab work, we will focus on solving MST using Prim's algorithm.

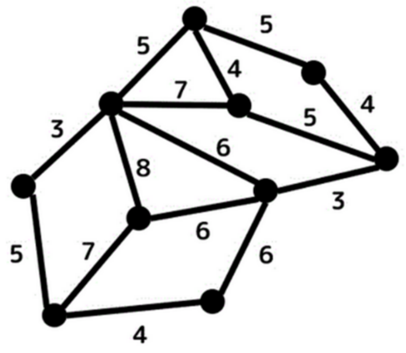
8.2.2 Prim's Algorithm

Prim's algorithm works as follows

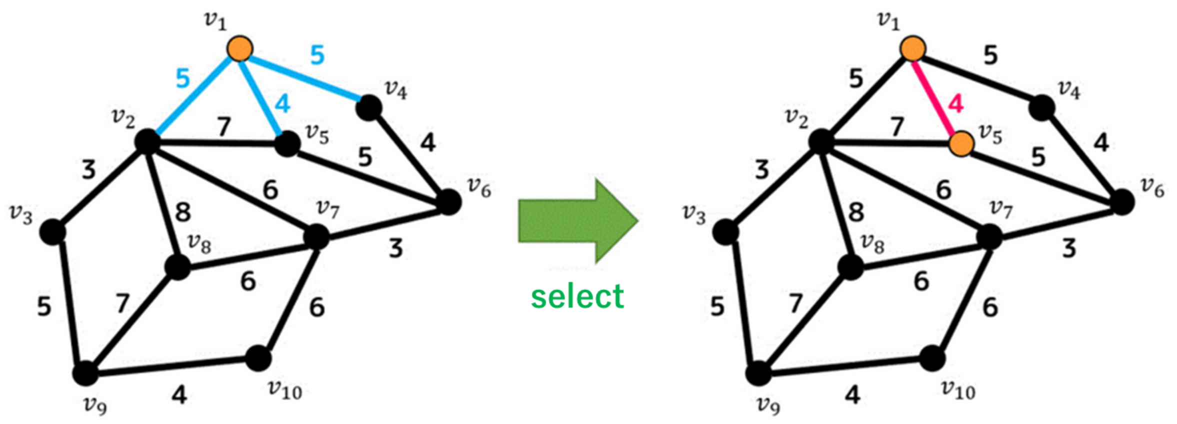
1. Decide one starting vertex (anywhere is OK)
2. Select the edge with the smallest weight among the edges that are connected to the starting vertex (if there are multiple edges with the smallest weight, you can choose any of them.)
3. Select a new edge with the smallest weight from the edges that are connected to the vertices of the previously selected edges AND this new edge must not create a cycle when selected.
4. Repeat step 3 until there is no more edge left to select

Let's apply this algorithm to the MST problem.

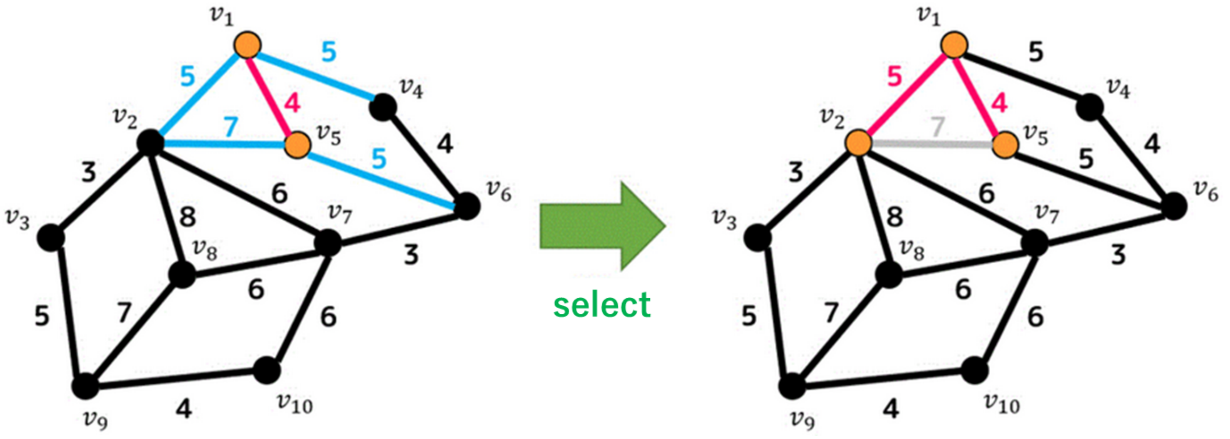
First, consider the following weighted graph.



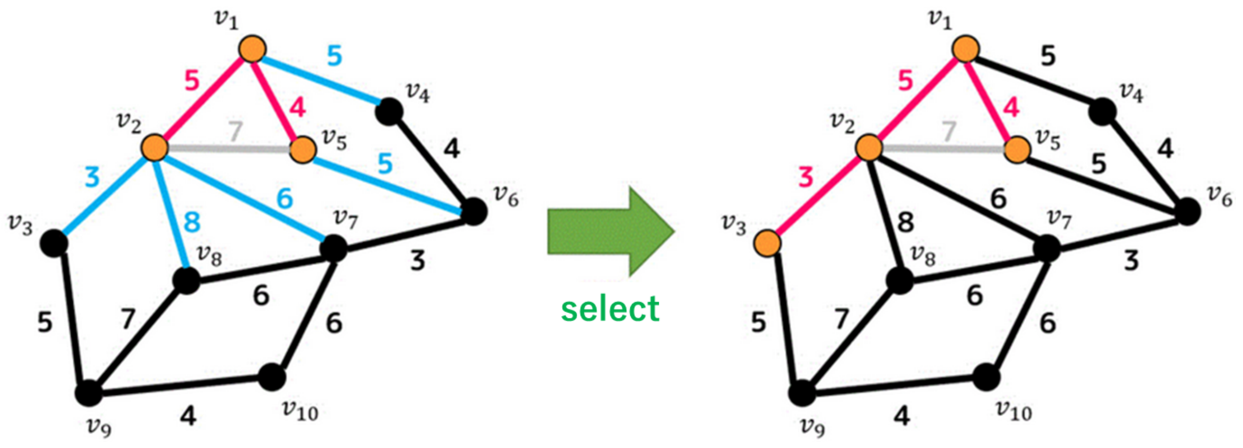
To reduce mistakes when applying Prim's algorithm, let's keep in mind from the beginning, "since the number of vertices is ten, we need to select nine edges!" Let's name the starting vertex as v1, which is the top most vertex, and the other vertices are named according to the figure below. Next, select the edge with the smallest weight among the edges connected to v1 (edges with blue color in the figure below). Since there is only one edge with weight 4, select this edge.



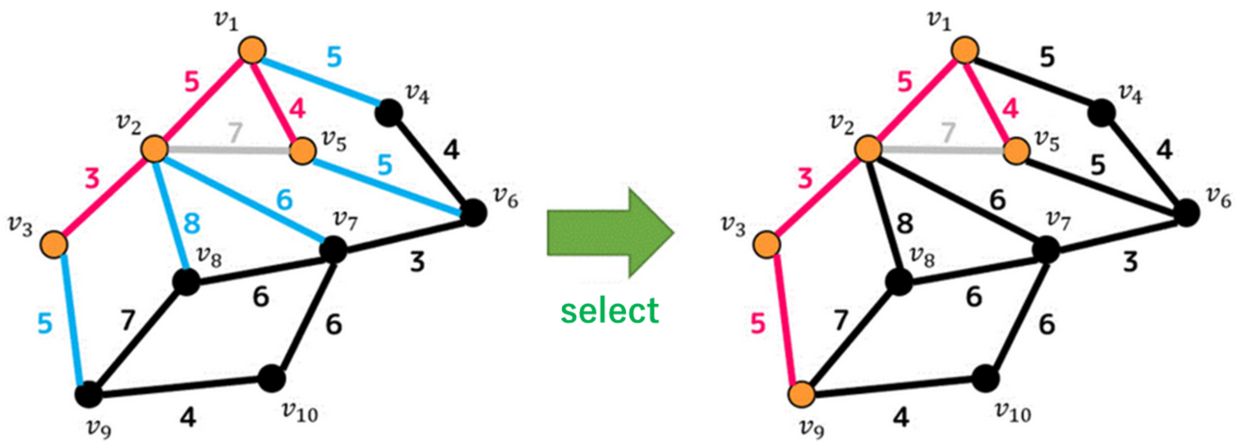
The selected edge is shown in red. The selected edge connects two vertices, v1 and v5, and therefore, the edges that are attached to these two vertices will be the candidates for the next edge selection. There are three edges with the smallest weight of 5 among all the candidate edges, so pick anything you like among these three edges. This time, we select the edge connecting v1 and v2, i.e., the one on the far left. Then, if we select the edge that connects v2 and v5, a cycle will be created, hence we must eliminate this edge from the candidate edges (grey edge).



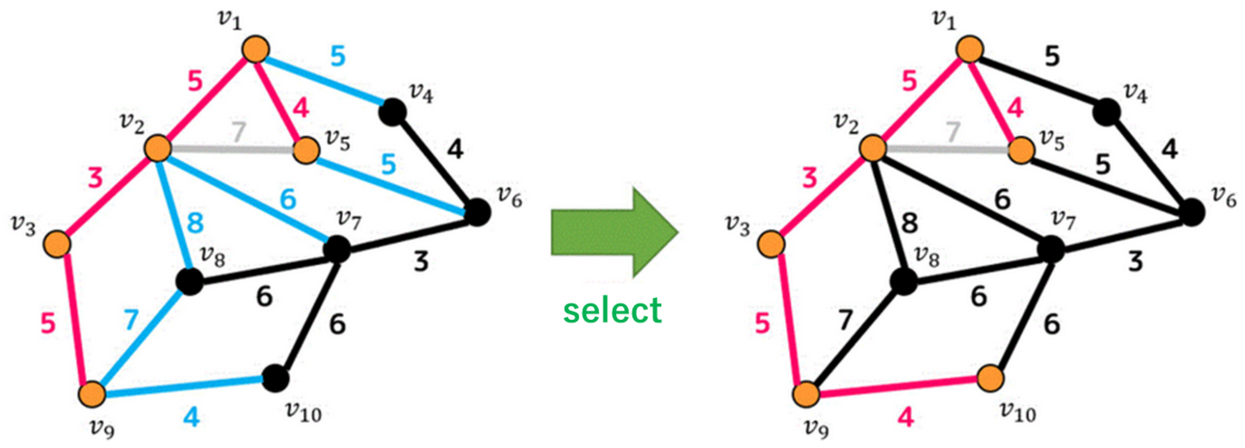
There are three vertices connected by the selected edges, namely v1, v2, and v5. Therefore, the edges that are attached to these three vertices will be the candidates for the next edge selection. Among the candidates, the edge with the smallest weight is the edge with weight 3. So we select this edge.



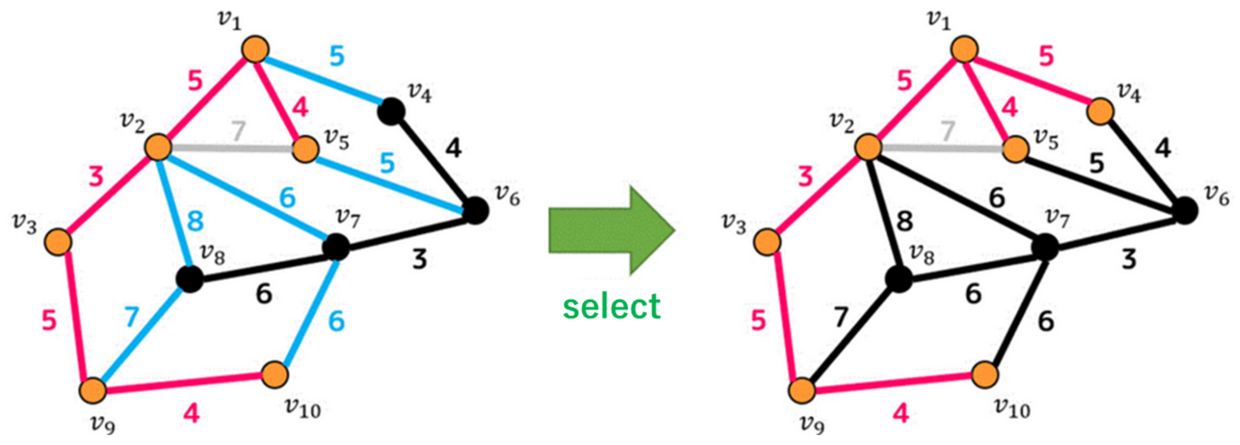
There are now four vertices connected by the selected edges, namely v1, v2, v3, and v5. Therefore, the edges that are attached to these four vertices will be the candidates for the next edge selection. Among the candidates, there are three edges with the smallest weight, which is 5. So, we select any edge among these three edges.



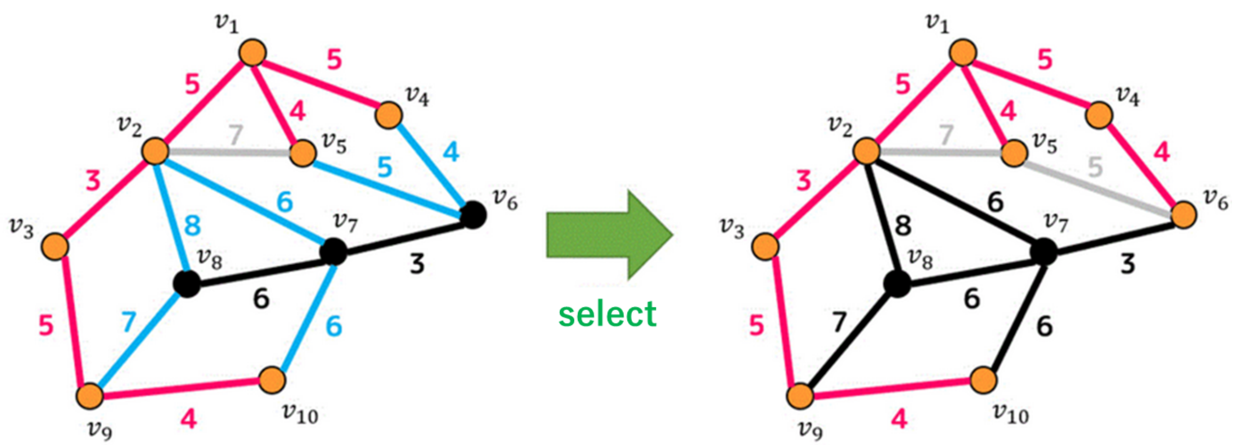
There are now five vertices connected by the selected edges, namely v1, v2, v3, v5, and v9. Therefore, the edges that are attached to these five vertices will be the candidates for the next edge selection. Among the candidates, the edge with the smallest weight is the edge with weight 4. So we select this edge.



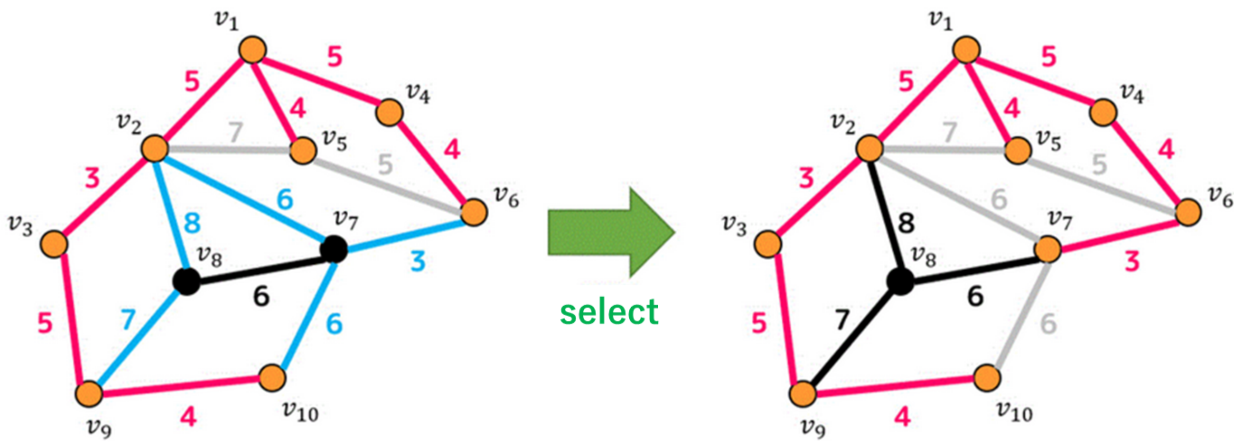
There are now six vertices connected by the selected edges. Among the candidates, there are two edges with the smallest weight, which is 5. So, we select one of these two edges.



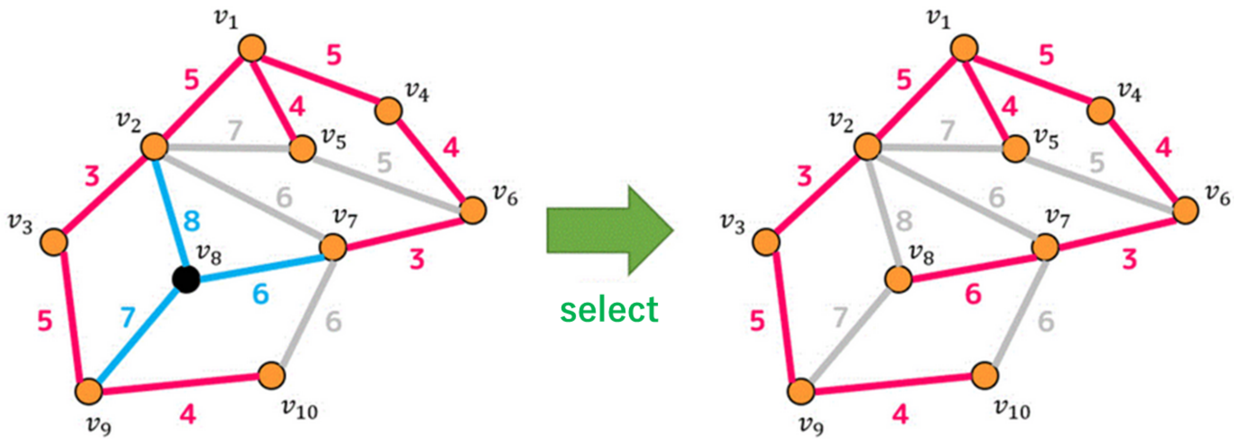
There are now seven vertices connected by the selected edges. Among the candidates, the edge with the smallest weight is the edge with weight 4. So, we select this edge. By doing so, the edge that connects v5 and v6 will form a cycle if get selected in the future, hence we must eliminate this edge from the candidate edges.



There are now eight vertices connected by the selected edges. Among the candidates, the edge with the smallest weight is the edge with weight 3. So, we select this edge. Furthermore, two edges (the ones that connect v2-v7 and v7-v10) will form a cycle if selected, hence these edges are eliminated from the candidate edges.



From the number of red edges, it is clear that so far we have selected a total of eight edges, and recall that we only need to select one more edge so that the total selected edges becomes nine. From the remining candidates, we select the one with the lowest weight, which is 6. Thus, the final form for minimum spanning tree of this graph is as shown below.



Selecting the remaining edges will form a cycle, hence we cannot select any more edge and this marks the end of Prim's algorithm. We obtained an MST with total cost of 39.

8.2.3 Java Implementation of Prim's Algorithm

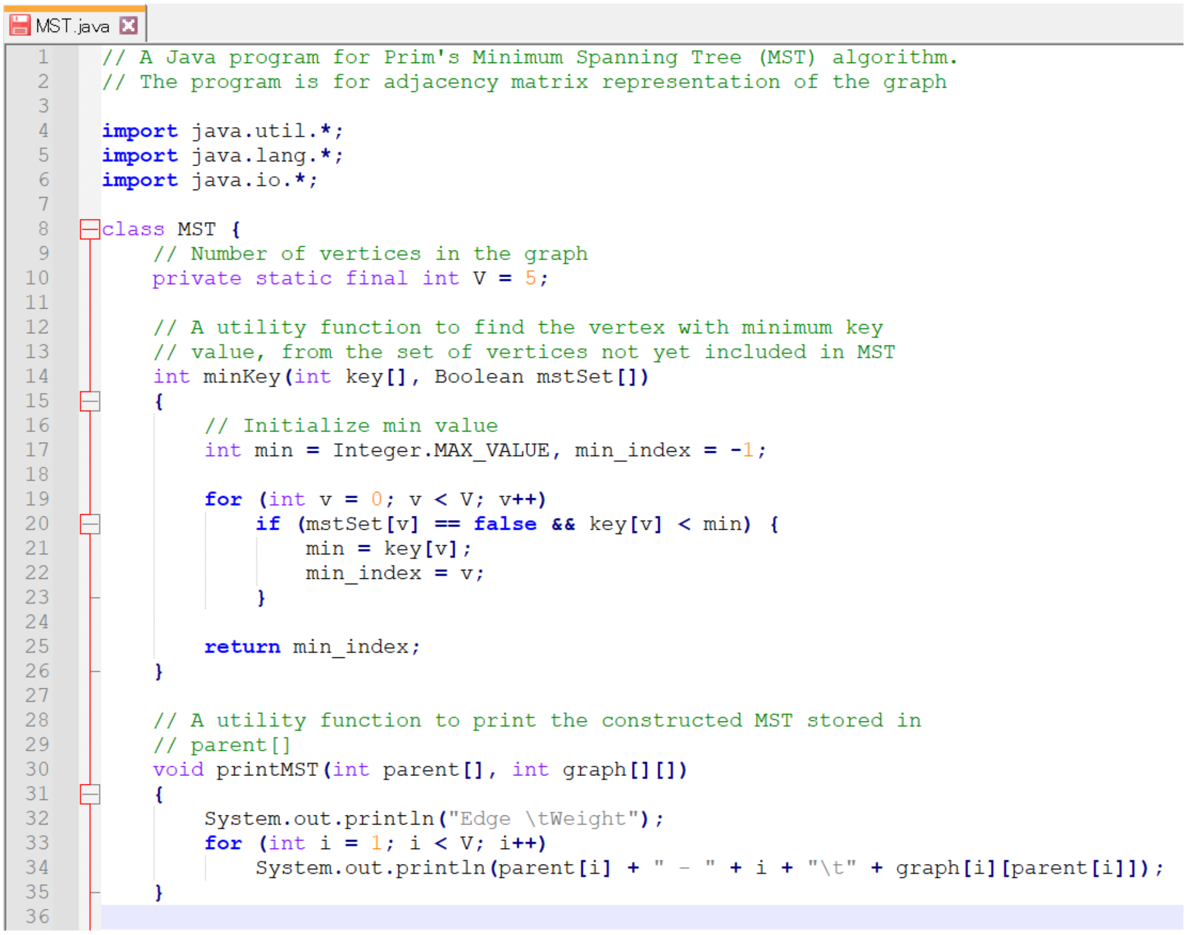
In this implementation, we assume that the graph is represented using adjacency matrix. We need to keep track of which vertices already included in the MST. For this purpose, we use an array called mstSet. We assign a key value to each vertex of the graph. The key value represents a vertex's minimum weighted edge whose one-end is connected to a vertex in mstSet. Note that, here the condition "one-end of the edge" is used to avoid creating a cycle. Since at the beginning mstSet is empty, we set the key values initially to INFINITE. A vertex with the lowest key value will be included in mstSet, thus key value 0 is assigned to the starting vertex so that it is included first to the MST. Then, the key values will be updated throughout the execution of the program and it program ends when all vertices have been included to mstSet.

The basic idea of the code is as follows: while mstSet does not include all vertices, do:

1. Pick a vertex *u* which is not in the mstSet and has minimum key value
2. Include *u* to mstSet
3. Update key values of all adjacent vertices of *u*

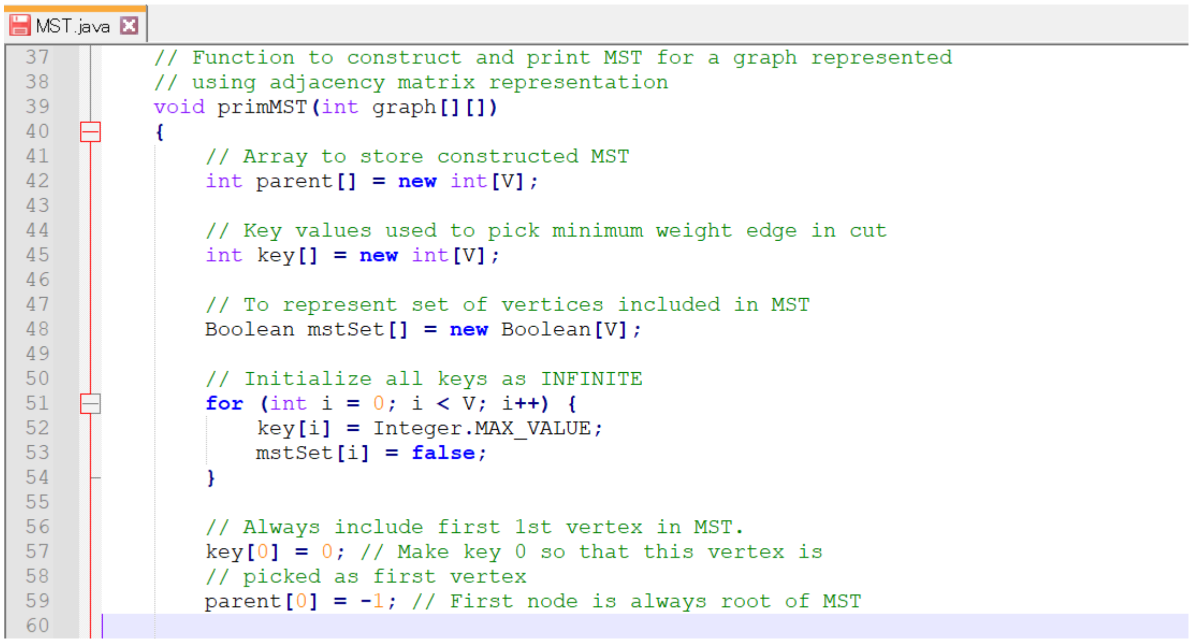
To update the key values, iterate through all adjacent vertices. For every adjacent vertex *v*, if weight of edge *u*-*v* is less than the previous key value of *v*, update the key value as weight of *u*-*v*.

minKey and printMST methods:



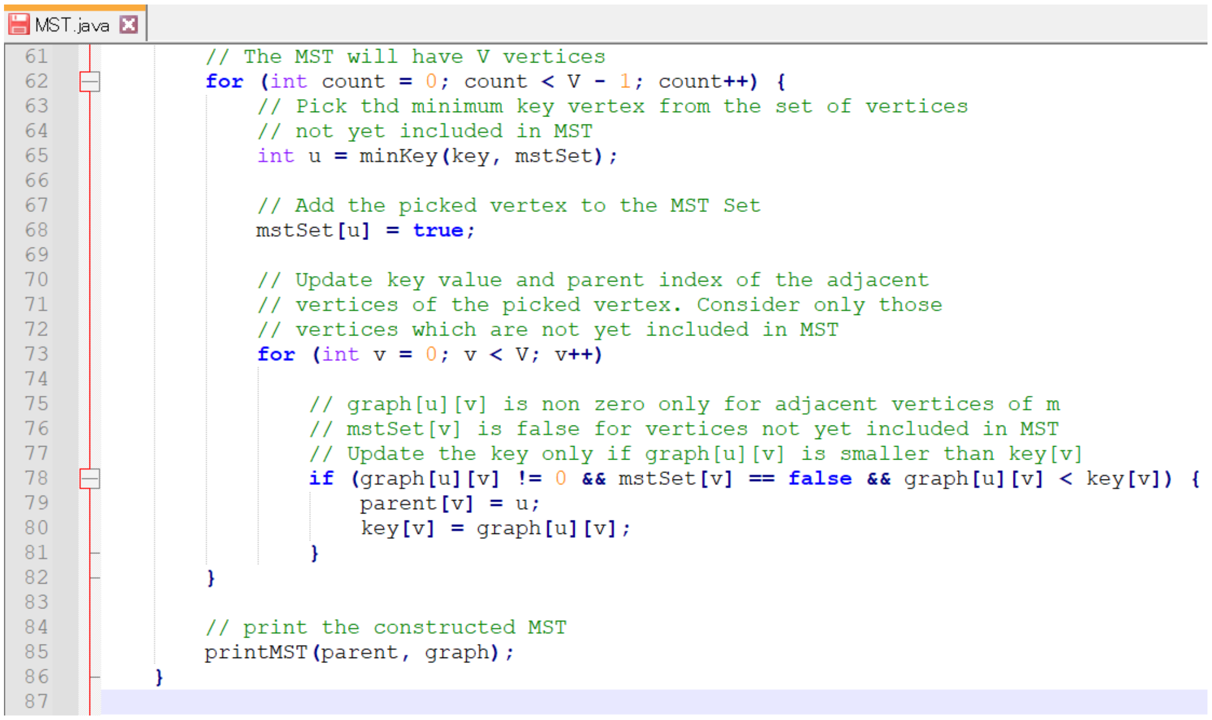
```
// A Java program for Prim's Minimum Spanning Tree (MST) algorithm.
// The program is for adjacency matrix representation of the graph

import java.util.*;
import java.lang.*;
import java.io.*;

class MST {
    // Number of vertices in the graph
    private static final int V = 5;

    // A utility function to find the vertex with minimum key
    // value, from the set of vertices not yet included in MST
    int minKey(int key[], Boolean mstSet[])
    {
        // Initialize min value
        int min = Integer.MAX_VALUE, min_index = -1;

        for (int v = 0; v < V; v++)
            if (mstSet[v] == false && key[v] < min) {
                min = key[v];
                min_index = v;
            }

        return min_index;
    }

    // A utility function to print the constructed MST stored in
    // parent[]
    void printMST(int parent[], int graph[][])
    {
        System.out.println("Edge \tWeight");
        for (int i = 1; i < V; i++)
            System.out.println(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);
    }

```

primMST method:



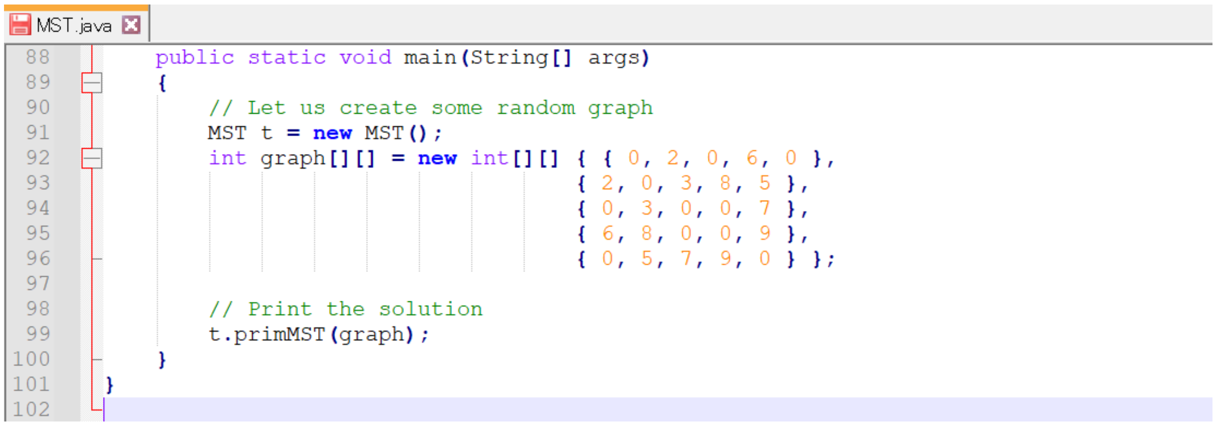
```
    // Function to construct and print MST for a graph represented
    // using adjacency matrix representation
    void primMST(int graph[][])
    {
        // Array to store constructed MST
        int parent[] = new int[V];

        // Key values used to pick minimum weight edge in cut
        int key[] = new int[V];

        // To represent set of vertices included in MST
        Boolean mstSet[] = new Boolean[V];

        // Initialize all keys as INFINITE
        for (int i = 0; i < V; i++) {
            key[i] = Integer.MAX_VALUE;
            mstSet[i] = false;
        }

        // Always include first 1st vertex in MST.
        key[0] = 0; // Make key 0 so that this vertex is
        // picked as first vertex
        parent[0] = -1; // First node is always root of MST

```

primMST method (cont'd):



```java
        // The MST will have V vertices
        for (int count = 0; count < V - 1; count++) {
            // Pick thd minimum key vertex from the set of vertices
            // not yet included in MST
            int u = minKey(key, mstSet);

            // Add the picked vertex to the MST Set
            mstSet[u] = true;

            // Update key value and parent index of the adjacent
            // vertices of the picked vertex. Consider only those
            // vertices which are not yet included in MST
            for (int v = 0; v < V; v++)

                // graph[u][v] is non zero only for adjacent vertices of m
                // mstSet[v] is false for vertices not yet included in MST
                // Update the key only if graph[u][v] is smaller than key[v]
                if (graph[u][v] != 0 && mstSet[v] == false && graph[u][v] < key[v]) {
                    parent[v] = u;
                    key[v] = graph[u][v];
                }
        }

        // print the constructed MST
        printMST(parent, graph);
    }

```

main method:



```java
    public static void main(String[] args)
    {
        // Let us create some random graph
        MST t = new MST();
        int graph[][] = new int[][] { { 0, 2, 0, 6, 0 },
                                      { 2, 0, 3, 8, 5 },
                                      { 0, 3, 0, 0, 7 },
                                      { 6, 8, 0, 0, 9 },
                                      { 0, 5, 7, 9, 0 } };

        // Print the solution
        t.primMST(graph);
    }
}

```

8.3 Assignments

1. Test the code with the graph from 8.2.2 and modify the code such that it also calculate the total cost of the MST. Did you get the same result compared to the theoretical one?
2. Another spanning tree (non-minimum) can be created from an MST by changing the edge connections. Write a program that output the number of spanning trees that can be created by changing the position of one edge only. Then, among these new spanning trees, output the

minimum cost. Test your code on the graph from 8.2.2. Check the correctness by comparing the result with the one performed through manual calculation.