Algorithms after Dijkstra and Kruskal for Big Data

User Manual

Ovak Technologies
2016

Contents

1. Introduction 3

1.1. Definition and Acronyms 3

1.2. Purpose 3

1.3. Overview 3

2. Algorithms 4

2.1. Description of Dijkstra's Algorithm 4

2.2. Description of Kruskal's Algorithm 5

3. Architecture 5

4. Examples 7

5. LabVIEW Features and Concepts Used 8

6. Support Information 9

1. Introduction

1.1. Definition and Acronyms

Graph – in mathematics, and more specifically in graph theory, a graph is a representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by mathematical abstractions called vertices (also called nodes or points), and the links that connect some pairs of vertices are called edges (also called arcs or lines).

Tree – in mathematics, and more specifically in graph theory, a tree is an undirected graph in which any two vertices are connected by exactly one path. In other words, any acyclic connected graph is a tree.

Spanning Tree – in the mathematical field of graph theory, a spanning tree T of an undirected graph G is a sub graph that includes all of the vertices of G that is a tree.

Minimum Spanning Tree (MST) – a minimum spanning tree is a spanning tree of a connected, undirected graph. It connects all the vertices together with the minimal total weighting for its edges. A single graph can have many different spanning trees.

Graph's Path – in graph theory, a path in a graph is a finite or infinite sequence of edges, which connect a sequence of vertices that, by most definitions, are all distinct from one another.

Source Node – here source node is a node, where the path begins.

Destination Node – here destination node is a node, where the path ends.

NP-hardness – non-deterministic polynomial-time hard.

1.2. Purpose

This document is designed to provide necessary information to LabVIEW developers of how to use *«Algorithms after Dijkstra and Kruskal for Big Data»* toolkit.

1.3. Overview

The word «graph» was first used in this sense by J. J. Sylvester in 1878. Graphs are the basic subject studied by graph theory.

In the most common sense of the term, a graph is an ordered pair $G = (V, E)$ comprising a set V of vertices, nodes or points together with a set E of edges, arcs or lines, which are 2-element subsets of V (i.e. an edge is related with two vertices, and the relation is represented as an unordered pair of the vertices with respect to the particular edge). To avoid ambiguity, this type of graph may be described precisely as undirected and simple.

There are many algorithms for graphs, and two of them are used in this toolkit. They are *Dijkstra's algorithm and Kruskal's algorithm.*

Dijkstra's algorithm is used to find the shortest path between two nodes in graph, which may represent, for example, road networks. For a given source node in the graph, the algorithm finds the shortest path between that node (source node) and any other (destination node).

Kruskal's algorithm is used to find the minimum spanning tree of graph. MST problem is to find connected graph *G* with positive edge weights, find a minimum weight set of edges that connects all of the vertices. It is a greedy algorithm in graph theory as it finds a minimum spanning tree for a connected weighted graph adding increasing cost arcs at each step.

The *«Algorithms after Dijkstra and Kruskal for Big Data»* enables to find the shortest path between two points. The shortest path problems form the foundation of an entire class of optimization problems that can be solved by a technique called *Column Generation*. Examples include vehicle routing problem, survivable network design problem, find the shortest path from machine *A* to machine *B* among others. This tool also enables to find MST of graph.

MST is a fundamental problem with diverse applications:

1. **Network design**
    - Telephone, electrical, hydraulic, TV cable, computer, road.
2. **Approximation algorithms for NP-hardness problems**
    - Traveling salesperson problem, Steiner tree
3. **Indirect applications**
    - Max bottleneck paths
    - LDPC codes for error correction
    - Image registration with Renyi entropy
    - Learning salient features for real-time face verification
    - Reducing data storage in sequencing amino acids in a protein
    - Model locality of particle interactions in turbulent fluid flows
    - Autoconfig protocol for Ethernet bridging to avoid cycles in a network
4. **Cluster analysis**

«K» clustering problem can be viewed as finding a MST and deleting the *«K-1»* most important edges.

This tool is mostly used for big data, enabling to find the shortest path between two nodes and the minimum spanning tree of graph very fast and easy.

2. Algorithms

2.1. Description of Dijkstra's Algorithm

1. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes called the unvisited set.

3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance to B (through A) will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.
4. When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

Dijkstra's algorithm enables to determine the shortest distance between two nodes, while *«Algorithms after Dijkstra and Kruskal for Big Data»* finds the edges (or the nodes), through which passes the shortest path and displays that path on the picture.

2.2. Description of Kruskal's Algorithm

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
3. Repeat step 2 until there are (V-1) edges in the spanning tree. *V* is the amount of nodes.

Kruskal's algorithm, enables to give the edges of minimum spanning tree, and shows the minimum spanning tree on picture.

3. Architecture

Graphs' Algorithms. Lvlib library contains two libraries: *Dijktra's Algorithm.lvlib* and *Kruskal's Algorithm.lvlib*.

1. *Dijkstra's Algorithm.lvlib* library contains six VIs:
    - **Shortest path.vi** – contains the main code.
    - **Choose minimum.vi** – returns the shortest distance for all destination nodes, and finds the next node that should be visited.
    - **Get picture.vi** – returns the picture of graph and shows the shortest path between source node and destination node.
    - **Initialize arrays.vi** – creates arrays of graph's edges and the lengths of edges. *Initialize arrays.vi* returns true, if the lengths' values are correct (positive number). It returns false, if the lengths' value is negative number or zero.

- **Rebuild array.vi** – for the current node, considers all of its unvisited neighbors and calculates their tentative distances. It compares the new calculated tentative distance to the current assigned value and assigns the smaller one. For example, if the current node *A* is marked with a distance of 6, and the edge connecting it with a neighbor *B* has length 2, then the distance to *B* (through A) will be 6 + 2 = 8. If *B* has been previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.
- **The edges of the shortest path.vi** – finds the shortest distance between source node and destination node, and returns the shortest path.

2. *Kruskal's Algorithm.lvlib* library contains four VIs:
   - **Minimal spanned tree.vi** – contains the main code.
   - **Delete unnecessary items.vi** – deletes loops (e.g., *AA*) and similar items (e.g., *AB, BA*).
   - **Identify next edge.vi** – returns true if the edges generate loop, if not returns false (e.g., *AB-BC-AC* is a loop).
   - **Loop.vi** – determines whether the path is a loop for 3 edges or not.

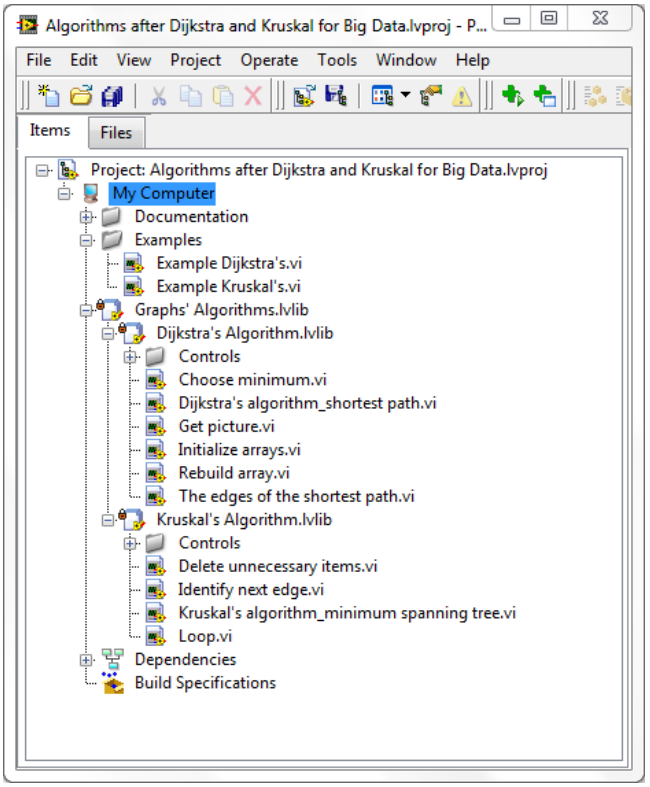

Figure 1 Graphs' Algorithms library

4. Examples

1. Dijktra's algorithm

Complete the following steps to run this example:

1. In the **Project Explorer** window, open **Example Dijktra's.vi**.
2. Enter **Array of Edges and their lengths, Array of Nodes, Node 1** and **Node 2** parameters (if these parameters are not entered, the VI will work with default parameters).
3. Click **RUN** button.

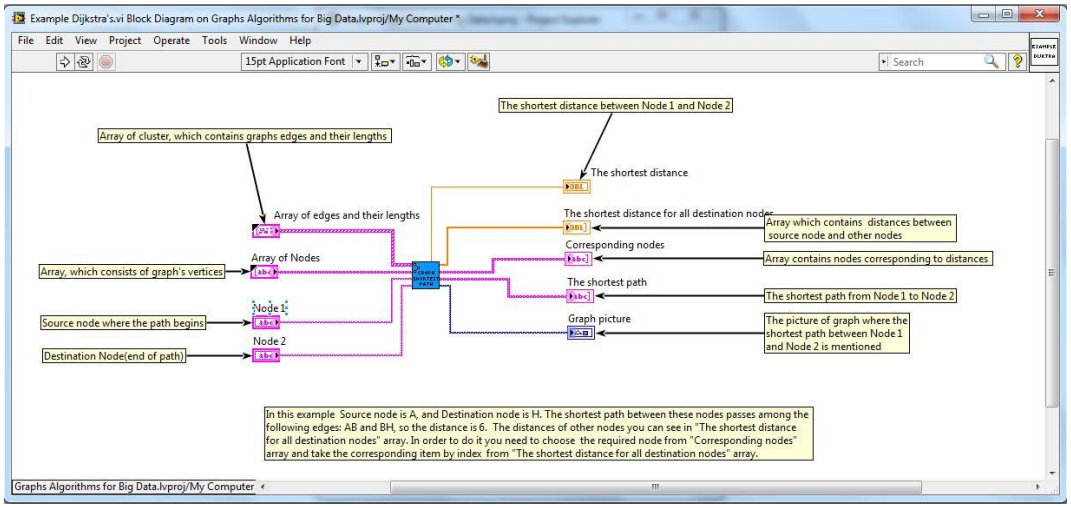

Figure 2 Dijkstra's Algorithm Block Diagram

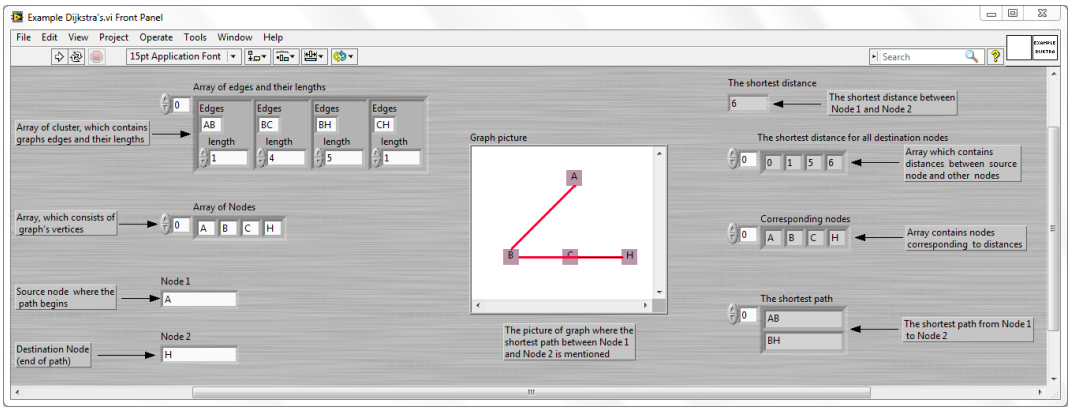

Figure 3 Dijkstra's Algorithm Front Panel

2. Kruskal's algorithm

Complete the following steps to run this example:

1. In the **Project Explorer** window, open **Example Kruskal's.vi**.

2. Enter **Array of Edges and their lengths** and **Array of Nodes** parameters (if these parameters are not entered, the VI will work with default parameters).
3. Click **RUN** button.

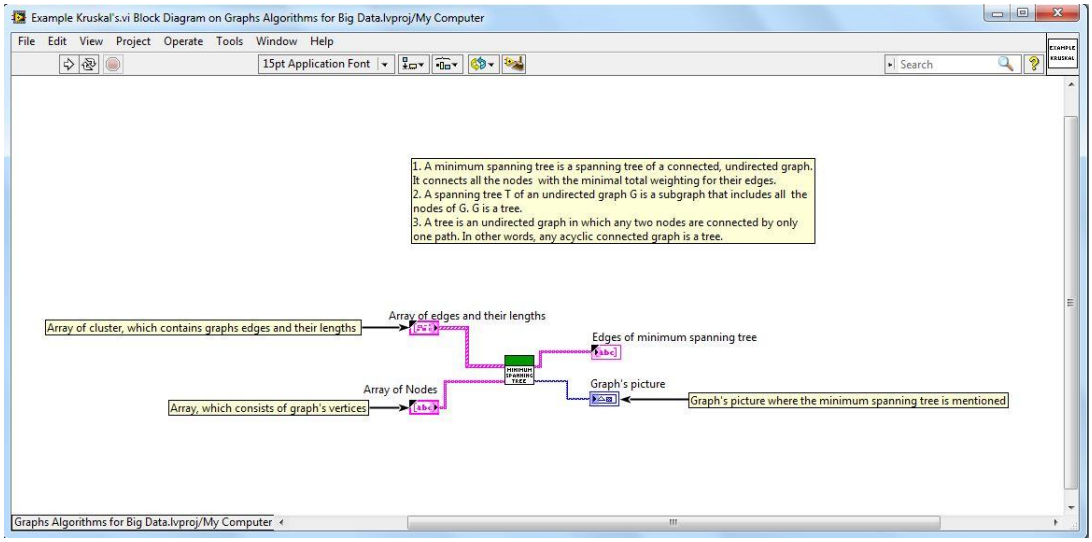

Figure 4 Kruskal's Algorithm Block Diagram

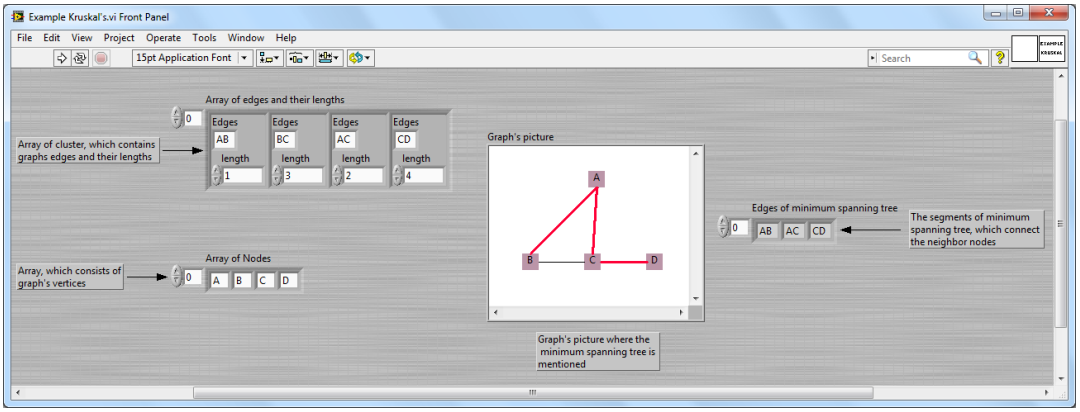

Figure 5 Kruskal's Algorithm Front Panel

5. LabVIEW Features and Concepts Used

- Case structures
- Clusters
- Arrays
- Enums
- Shift registers
- While Loops
- For Loops
- State machines
- String Functions
- Picture Functions

6. Support Information

For technical support, please, contact Ovak Technologies at:

Phone: + 374 10 21-97-68
Email: support@ovaktechnologies.com
Web: www.ovaktechnologies.com