**Department of Computing**

**CS250: Data Structures and Algorithms**

**Class: BEE-6AB**

# Lab 12: Minimum Spanning Tree

**Date: 17th December, 2015**

**Time: 10am-1pm & 2pm-5pm**

# Instructor: Dr. Faisal Shafait

**Lab 12: Minimum Spanning Trees**

**Introduction**

In this lab, you will be introduced to what is necessary to implement a Minimum Spanning Tree

(Prim’s algorithm, 1957).

**Objectives**

Objective of this lab is to get familiar with Minimum Spanning Tree.

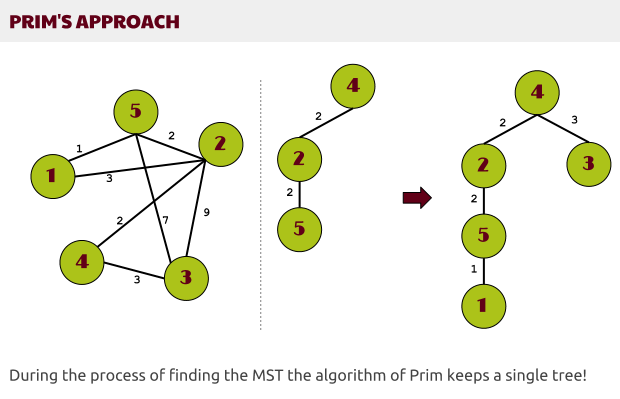
**Tools/Software Requirement**

Visual Studio C++

**Description**

The idea behind minimum spanning trees is simple: given a graph with weighted edges, find a tree of edges with the minimum total weight. What is a spanning tree? A graph that satisfies these three properties: connected, acyclic, and consisting of |V| - 1 edges. (In fact, any two of the three conditions imply the third condition.) What this means in practice is that a spanning tree is a collection of edges that connect any two nodes via a single path. It's probably easiest to understand a tree using the definition that a tree is both connected and acyclic; think about binary trees--every node in the tree is reachable, so it's connected, and there are no cycles because you can't go back up the tree.

During the process of building the MST this algorithm keeps a single tree, which is finally sub-tree of the final minimum weight spanning tree.

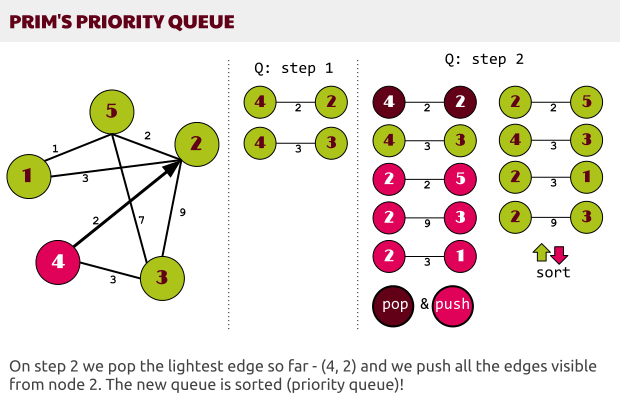
[](http://www.stoimen.com/blog/wp-content/uploads/2012/11/3.-Prims-approach.png)

On each step we chose an edge which we add to the growing tree that finally forms the MST. It is somehow unnatural approach! We start from a given vertex and initially we don’t choose the lightest edge. Thus during the whole process the tree grows, but outside the tree (T) there might be edges that are lighter than those in the tree (i.e. the edge (5, 1) from the tree above is lighter than (2, 5) but (2, 5) is added to the growing tree before the edge (5, 1)).

Compared to the Kruskal’s algorithm this time everything seems to be really unnatural. How we should be sure the final tree (T) will be a minimum spanning tree since we don’t get the lightest edge on each step?

Actually we are sure that the final tree is a MST because of another obvious feature of the minimum spanning trees. They should “connect” all the vertices of G, thus somehow at least one edge reaching each vertex will appear in the MST. Thus we shouldn’t care where do we start, the only important thing is to choose the lightest edge that’s visible so far.

This algorithm looks much like [Dijkstra’s shortest path in a graph](http://www.stoimen.com/blog/2012/10/15/computer-algorithms-dijkstra-shortest-path-in-a-graph/), because we start from a vertex, we push all the edges starting from this node to a priority queue and we chose the lightest edge. Going to the next node connected by this edge we append to the queue all the edges that aren’t in the queue. That way the queue grows and we get always the lightest edge – thus forming a priority queue. Now let’s summarize the algorithm of Prim

[](http://www.stoimen.com/blog/wp-content/uploads/2012/11/4.-Prims-Priority-Queue.png)

**Pseudo Code**

As an initial input we have the graph (G) and a starting vertex (s).

1. Make a queue (Q) with all the vertices of G (V);

2. For each member of Q set the priority to INFINITY;

3. Only for the starting vertex (s) set the priority to 0;

4. The parent of (s) should be NULL;

5. While Q isn’t empty

6. Get the minimum from Q – let’s say (u); (priority queue);

7. For each adjacent vertex to (v) to (u)

8. If (v) is in Q and weight of (u, v) < priority of (v) then

9. The parent of (v) is set to be (u)

10. The priority of (v) is the weight of (u, v)

**Sample PHP Code**

Here’s a [PHP](http://www.stoimen.com/blog/category/php/) implementation of the algorithm of Prim, which directly follows the pseudo code.

|  |
| --- |
| *// Prim's algorithm*  define('INFINITY', 100000000);  $G = array(  0 => array( 0, 4, 0, 0, 0, 0, 0, 0, 8),  1 => array( 4, 0, 8, 0, 0, 0, 0, 0, 11),  2 => array( 0, 8, 0, 7, 0, 4, 2, 0, 0),  3 => array( 0, 0, 7, 0, 9, 14, 0, 0, 0),  4 => array( 0, 0, 0, 9, 0, 10, 0, 0, 0),  5 => array( 0, 0, 4, 14, 10, 0, 0, 2, 0),  6 => array( 0, 0, 2, 0, 0, 0, 0, 6, 7),  7 => array( 0, 0, 0, 0, 0, 2, 6, 0, 1),  8 => array( 8, 11, 0, 0, 0, 0, 7, 1, 0),  );    **function** prim(&$graph, $start)  {  $q = array(); *// queue*  $p = array(); *// parent*  foreach (array\_keys($graph) as $k) {  $q[$k] = INFINITY;  }  $q[$start] = 0;  $p[$start] = **NULL**;  asort($q);    while ($q) {  *// get the minimum value*  $keys = array\_keys($q);  $u = $keys[0];  foreach ($graph[$u] as $v => $weight) {  if ($weight > 0 && in\_array($v, $keys) && $weight < $q[$v]) {  $p[$v] = $u;  $q[$v] = $weight;  }  }  unset($q[$u]);  asort($q);  }  return $p;  }    prim($G, 5); |

**Lab Tasks**

The goal of the lab is to gain familiarity with a Prim’s algorithm for computing MST. Implement MST algorithm in C++ based on a graph representation of the graph as shown in the pseudocode and the example PHP code.

**Deliverables**

Hand in the source code from this lab at the appropriate location on the blackboard system at LMS. You should hand in a single compressed/archived file that contains the following.

1. C++ source code files representing the work accomplished for this lab. All source code files should contain author in the comments at the top of the file.
2. A plain text file named OUTPUT.txt that includes
   1. Author information at the beginning,
   2. A brief explanation of the lab and
   3. List some possible applications of a Minimum Spanning Tree.