Cut Property for MST's and Dijkstra's Shortest Paths algorithm

This writeup covers two somewhat unrelated topics that are not covered in the book.

Cut Property for MST's

The first topic is a remarkable property that Minimum Spanning Trees have, that is a key tool in proving the correctness of some of the algorithms we discussed. Here's the statement:

> **Cut Property for MST's**
>
> If we separate the vertices in two sets, set P and set Q, we call this a *cut*.
>
> When we have any cut, consider all the edges that have one end in P and the other in Q. If there is an edge in this set that has smaller weight/cost than all the others, then that edge must be included in any Minimum Spanning Tree.

Make sure to examine this property in some of the examples we have computed.

This property is the driving force behind Prim's algorithm: At any stage in that algorithm, we have the set of already selected vertices, and the remaining vertices, and we choose next the edge between those two sets with the smallest weight. The Cut Property guarantees for us that that edge would need to be part of any MST, so it's a good thing that we are including it.

Let us discuss a "proof" of the property, why is it true?

Suppose it were not true. Then we have some cut, so splitting the vertices in two sets P and Q, and if we look at the edges joining those sets there is an edge, call it e, that has the smallest cost. And that edge is not included in the MST.

Now let's see what happens if we include e to our Minimum Spanning Tree: So take the MST and add this extra edge e to it. This will create a cycle, because we already had a way in the MST of joining the vertices attached to e.

If we look at that cycle, then it must cross from P to Q at some point other than the edge e. Let's call this other edge f. f would have to have cost higher than e.

The key idea now is that if we remove f and add e to our MST, we get still a spanning tree, and it has a smaller overall cost than the original tree, which we thought was the MST. Take a moment to think this through. Why can we still reach all vertices, and have no cycles?

So that is not possible, as it violates the key property of the MST. We have arrived at a contradiction. So our initial assumption that e was not part of the spanning tree must be incorrect.

Shortest Paths and Dijkstra's Algorithm

Dijkstra's Algorithm solves an important problem called the *Shortest Paths Problem*:

> **Shortest Paths Problem**
>
> Given a graph with weights on the edges, and a starting vertex, find the shortest paths from that vertex to all other reachable vertices.

This is an important problem in the internet: When you request a web page halfway around the world, there are lots of different paths over the internet that would get you to that page. Your computer and router need to have a way to find the "shortest" such paths, and they need to be able to do that very efficiently, given the enormity of the internet.

We now describe Dijkstra's algorithm, which is one of the ways of obtaining these shortest paths:

> **Dijkstra's Algorithm**
>
> - Assign initial *costs* to all the vertices: 0 for the starting vertex, infinity or a very very large number for all the other vertices.
> - At each stage we pick from the "remaining" vertices the one with the smallest *cost* to be the "current vertex". At the first step this would be the starting vertex.
> - Look at all the edges that are adjacent to this current vertex. Use the edge weight, along with the cost of the current vertex, to see if you have just found a shorter route to their other endpoint. If so, update the cost on that other endpoint, and remember the edge we used to get to it.
> - Once we do this for all the adjacent edges of the current vertex, we are done with that vertex, and will not visit it again. We then repeat the second step to pick a new "current vertex".

Here is a fairly detailed demonstration of Dijkstra's algorithm:

- http://www.eoinbailey.com/content/dijkstras-algorithm-illustrated-explanation

Here are two visualizations of Dijkstra's algorithm:

- http://optlab-server.sce.carleton.ca/POAnimations2007/DijkstrasAlgo.html
- https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html