CS340 Analysis of Algorithms

Handout:	**3**	**Professor:**	**Dianna Xu**
Title:	**Kruskal's MST**	**E-mail:**	dxu@cs.brynmawr.edu
Date:		**URL:**	**http://cs.brynmawr.edu/cs340**

Sample description, pseudo code and proof of correctness for Kruskal's minimum spanning tree algorithm. Please refer to lecture notes for details of the algorithm design and time analysis. Recall that given a connected, undirected graph $G=(V,E)$, with associated edge weights $w(u,v)$ for each edge $(u,v)\in E$, we want to compute a spanning tree (a tree that connects all vertices $v\in V$) of minimum total edge weight.

1 Description

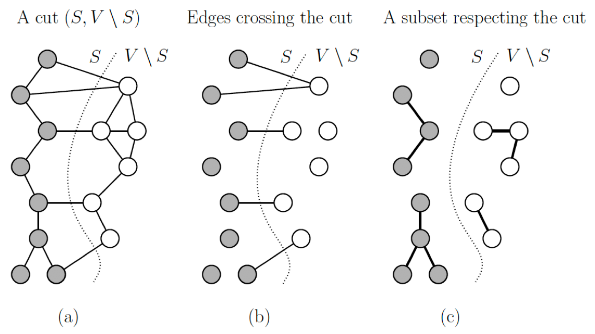
Kruskal's sorts all edges by weight and builds the MST by repeatedly inserting the next min-weight edge unless it creates a cycle. A Union-Find (UF) data structure of vertices is used to detect cycles by keeping the vertices of all currently connected trees (as the algorithm adds the MST edges one by one) in separate sets. The vertices start disconnected and each in its own set. Any time we encounter an edge that connects two vertices in the same set, a cycle is found. Therefore we only insert those edges with two end vertices in different UF sets. Every time we add an edge to the MST, we merge the sets of the two end vertices.

2 Pseudocode

```
Function kMST($G=(V,E)$)
    $A=\emptyset$
    sort $E$ in increasing order by weight
    //put each vertex in its own set
    for each $v\in V$ do
        create(v)
    end
    for each $(u,v)\in E$ do
        // u and v are in different sets
        if (find(u) != find(v)) then
            add (u, v) to A
            union(u, v)
        end
    end
```

3 Correctness Proof

Termination is easily argued because the set E is finite and we process at least one edge per iteration (whether we add it to the MST or not).
We prove the optimality of `kMST` by showing that the lightest edge which connects two disconnected subtrees in G is always a correct (safe) edge for the MST. For simplicity, we will also assume that all edge weights are distinct.



We will start with some definitions that will help us formalize the problem. Given a graph $G=(V,E)$, let S be a subset of vertices $S \subseteq V$.

Definition 3.1. A *cut* $(S,V \setminus S)$ is a partition of the vertices V into two disjoint subsets. See (a)

Definition 3.2. An edge (u,v) *crosses* a cut if $u \in S$ and $v \in V \setminus S$. See (b)

Definition 3.3. Given a subset of edges $A \subset E$, we say that a cut *respects* A if no edges in A crosses the cut. See (c)

Note that the notion of respecting cuts is important to this problem because Kruskal's maintains a collection of subtrees as it builds the MST. It is important for the algorithm to know which edges can be added that do not induce a cycle. Based on the definitions above, it should be clear that any edge that crossses a respecting cut does not induce a cycle and therefore is a possible MST candidate.

Lemma 3.1. *Let $G=(V,E)$ be a connected, undirected graph with associated edge weights $w(u,v)$ for each edge $(u,v) \in E$. Let A be a MST-viable subset of E that denote edges already in the MST. Let $(S,V \setminus S)$ be any cut that respects A. Then the lightest edge (u,v) crossing the cut is safe for A (or is an edge to add to the MST).*

Proof: by contradiction. Let T be any MST of G. If $(u,v) \in T$ then we are done. Otherwise, add (u,v) to T, thus necessarily creating a cycle. Since u and v are on opposite sides of a cut and since any cycle must cross the cut an even number of times, there must be at least one other edge (x,y) in T that crossses the same cut. See Fig 1(b). The edge (x,y) is not in A (because the cut respects A). By removing (x,y) we restore a spanning tree T' and we have

$$w(T') = w(T) - w(x,y) + w(u,v)$$

Since (u,v) is the lightest edge crossing the cut, $w(T') < w(T)$. Contradiction. ■

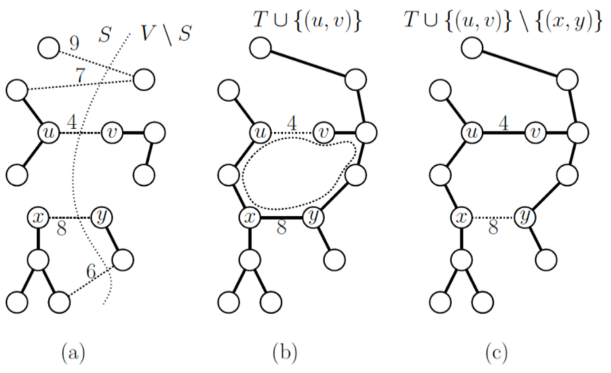


Figure 1: (u,v) is the lightest edge crossing the cut $(S,V \setminus S)$

Lemma 3.2. `kMST` *correctly detects cycles when processing edges, using UF.*

Proof: by contradiction. Suppose not. Then exists an edge (u,v) with two end points in different UF sets, but adding it to the MST created a cycle. Note that all sets maintained by the UF are disjoint, because the vertices start each in its own set and every time we add an edge between two vertices we merge the two sets of the two endpoints. Two disjoint sets represent a cut in the graph and any cycle must cross a cut an even number of times, therefore adding a single edge with two end points across the cut will not create a cycle. Contradiction. ■

We will now put it together. Intuitively, all vertices start as singleton trees and `kMST` merges or splices two trees together (when it inserts an edge), until all vertices are in the same tree (same UF set). Note that as the algorithm runs, the edges of A will induce a forest on the vertices (which consists of subtrees and singleton vertex trees, represented by the disjoint sets in the UF). Consider any edge (u,v) that `kMST` processes, and suppose that we already know this edge does not induce a cycle (Lemma 3.2 guarantees that we can detect cycles correctly). Let A' denote the particular tree in the forest that contains vertex u. Consider the cut $(A',V\setminus A')$. Every edge crossing this cut is not in A, so this cut respects A. In addition, (u,v) is the lightest edge crossing the cut (because any lighter edge would have been considered earlier). By Lemma 3.1, it is safe to add (u,v) to A, which is exactly what `kMST` does. Therefore, every edge `kMST` adds is a correct edge in the MST.
Now we argue that the MST is spanning. Suppose to the contrary that it is not. Then there must be at least two disconnected trees X and Y in A when `kMST` finishes, and in addition, there exists a lightest edge $(u,v)\in E$, where $u\in X$ and $v\in Y$ (if not, then the input G was disconnected and a spanning tree is not possible). At some time during the run, `kMST` must have processed edge (u,v) and would have added it since it's the lightest and it doesn't create a cycle (its two end points are in two disconnect trees/disjoint sets). Contradiction. ■