CSE421: Design and Analysis of Algorithms Jan 31, 2018

Lecture 12 Union Find, Edge Distinct Properties

Lecturer: Shayan Oveis Gharan Scribe:

1 In-class exercise

Recall the following two lemmas proved in the last lecture that we proved when all edge weights are distinct.

Lemma 1 (Cut property)**.** *If $(S, V - S)$ is a cut and e is the smallest edge that crosses from inside the cut to outside, then e must be in every MST.*

Lemma 2 (Cycle Property)**.** *If e is the largest edge of some cycle in the graph, then e cannot be in any MST.*

Lemma 3. *If the edge weights are distinct, then the MST is* unique*.*

There are in facts two proofs of this lemma: One using the cut property, and the other using the cycle property.

Suppose $T_1 \neq T_2$ are two minimum spanning trees. Then there must be some edge $e \in T_1, e \notin T_2$.

Cycle property proof: Adding e to T_2 creates a cycle. If e is the heaviest edge of this cycle, then $e \in T_1$ contradicts the cycle property. On the other hand, if some other edge in this cycle is the heaviest, then that contradicts the cycle property with T_2.

Cut property proof: Removing e from T_1 breaks T_1 into two connected components. Let S be the vertices of one of them and $V - S$ the other set. If e is the smallest edge in this cut, then T_2 cannot be an MST by the cut property. If some other edge is the smallest edge, then that edge is not included in T_1, so T_1 cannot be an MST.

2 The Union-Find Data Structure

To efficiently implement Kruskal's algorithm, we need to avoid looking for cycles every time. Instead, we shall think of the algorithm as maintaining a set of connected components. In each step, it tries to use an edge to merge two connected components. To store a connected component, we shall use nodes that represent each vertex.

```
Initialize: Set $P(v) \leftarrow v$ and $L(v) \leftarrow 0$ for all vertices $v$
Function Find(x)
    while $P(x) \neq x$ do
        $x \leftarrow P(x)$
    end
    Output x
Function Merge(x,y)
    $r_x \leftarrow Find(x)$
    $r_y \leftarrow Find(y)$
    if $L(r_x) \geq L(r_y)$ then
        $P(r_y) \leftarrow r_x$
        If $L(r_x) = L(r_y)$, set $L(r_x) \leftarrow L(r_x) + 1$
    end
    else
        $P(r_x) \leftarrow r_y$
    end
```

In order to run the algorithm efficiently, we need a data structure that can maintain the connected components of the graph. We do this using the following data structure. The set of connected components will be maintained using a directed rooted tree where every vertex has out-degree 1 except the root. The connected components will be represented by the root of each tree, so two vertices x, y are in the same component if and only if the root of their trees are the same (In the above algorithm this corresponds to $Find(x) = Find(y)$. To merge the components of x, y we simply call $Merge(x, y)$.

For every vertex u, we write $L(u)$ to denote the length of the longest path that ends at u. So, for a root r, $L(r)$ is the depth of the tree rooted at r. To merge two components with roots r_x, r_y, if $L(r_x) > L(r_y)$ we make r_y point to r_x. Similarly, if $L(r_y) > L(r_x)$ we make r_x point to r_y. If $L(r_x) = L(r_y)$ we make an arbitrary choice, say r_y points to r_x, and then increase $L(r_x)$ by 1. The following claim implies that any find operation runs in time $O(\log n)$.

Claim 4. *For every vertex v, $L(v) \leq \log n$.*

Proof We prove by induction that if the root of a component satisfies $L(r) = k$, then the corresponding component has at least 2^k nodes. Thus if $L(r) > \log n$ for some node r, then the corresponding component will have more than n nodes, which is a contradiction. We induct on the time.

Base Case: At the beginning all components satisfy $L(v) = 0$ and they all have a single node ($2^0 = 1$).

IH: Suppose the claim holds up to some time t.

IS: If at time $t+1$ we call $Find(x, y)$ it will not change any tree. So, suppose we call $Merge(x, y)$. So, we merge the trees of r_x, r_y. If $L(r_x) \neq L(r_y)$ then $L(.)$ of the new root does not increase, during the merge so the conclusion follows immediately. Otherwise, we have $L(r_x) = L(r_y)$. Say both of them are equal to k. In this case, by IH, we know that the tree of r_x has at least 2^k nodes and the tree of r_y has at least 2^k nodes. So, the new tree has at least $2^k + 2^k = 2^{k+1}$ nodes. So, we can rightfully increase $L(.)$ of the new root by 1. ■

3 Removing Weight Distinction Assumption

So far, in the description of Kruskal's algorithm, we assumed that edge weights are distinct.

In this section we see that even if the edge weights are not distinct still, the Kruskal's algorithm will give us a MST. Note that, unlike the distinct weights case, if the edge weights are not distinct there could be *multiple* MSTs, and we are only going to find one of them by Kruskal's algorithm.

Claim 5. *Suppose the edge weights are not distinct. Kruskal's algorithm still gives us an MST.*

Proof Note that in the proof we can *no longer* use the Cycle and Cut properties, because these properties only hold/proven when the edge weights are distinct.

Suppose the edge weights are not distinct. Kruskal's algorithm sorts the edges such that

$$c_{e_1} \leq c_{e_2} \leq \cdots \leq c_{e_m}$$

and then finds a tree based on this order. Suppose T is the output of the algorithm. For the sake of contradiction, assume that T is not an MST, then there is a tree T^*, such that $c(T^*) < c(T)$.

Perturb the weights of the edges such that all of the above inequalities are *strict*, i.e.,

$$c'_{e_1} < c'_{e_2} < \cdots < c'_{e_m}.$$

For example, it is enough to define $c'_{e_i} = c_{e_i} + i \cdot \epsilon$ for a very very small $\epsilon > 0$.

Note that since the ordering does not change, still T will be the output of the Kruskal's algorithm. Furthermore, if ϵ is very small, much smaller than $(c(T) - c(T^*))/m^2$, we have $c'(T^*) < c'(T)$ This is because

$$c'(T^*) \leq c(T^*) + \epsilon + 2\epsilon + \cdots + m\epsilon \leq m^2\epsilon.$$

Therefore, $c'(T^*) < c'(T)$, i.e., Kruskal's algorithm does not output the MST for the edge weights c' which are distinct. But this contradicts the correctness of Kruskal's algorithm that we proved in the last lecture. Recall, whenever the edge weights are distinct, Kruskal will output the MST. That is a contradiction! Therefore, T is an MST with respect to original weights c as well. ■