CSC 373 - Algorithm Design, Analysis, and Complexity **Summer 2016**

Lalla Mouatadid

Greedy Algorithms: Minimum Spanning Tree

Definitions :

$G(V, E, w)$ is an **edge weighted graph** if there exists a weight function $w : E \to \mathbb{R}$ that assigns a weight to every edge $e \in E$.

$G(V, E)$ is **connected** if there exists a path between any two vertices. $G(V, E)$ is a **tree** if it is connected and has has no cycles. It is easy to see (convince yourself otherwise) that if $G(V, E)$ is a tree and $|V| = n$, then $|E| = n - 1$.

Let $H(V', E')$ be a subgraph of $G(V, E)$. We say that H is a **spanning tree** of G if H is a tree and $V' = V$.

Let $G(V, E, w)$ be an edge-weighted graph where $w : E \to \mathbb{N}$. $H(V, E')$ is a **minimum spanning tree** of G if it is a spanning tree with weight less than or equal to the weight of any other spanning tree of G, i.e., $\sum\limits_{e \in E'} w(e) \leq \sum\limits_{e \in E''} w(e)$ for all other spanning trees $H'(V, E'')$ of G.

The MST problem asks for **a** minimum spanning tree of G. As we saw in class, a graph could have many MST's. It is quite amazing that many greedy algorithms for the MST problem are optimal, we covered two in class and tutorial: Prim's algorithm and Kruskal's algorithm. Try to come up with another greedy approach that gives you an optimal solution, for fun :) !

Before getting into the algorithms, let recall two **facts** about spanning trees:

1. Let T be a spanning tree of G. If you add any edge $e \notin T$ to T then $T' = T \cup \{e\}$ contains a cycle. This is easy to see. Let $e = (u, v)$ be an edge not in T. Since T is a spanning tree, then there is already a path between any two vertices of G; in particular there is a path between u and v. Adding e to T will thus close the cycle from u to v.
2. Consider $T' = T \cup \{e\}$ as defined above, and let $\mathcal{C}$ be the cycle created by adding the edge e. If you remove any edge $e' \in \mathcal{C}$ from the cycle, you will get a new spanning tree of G; since removing an edge from a cycle will not disconnect the graph.

Recall as well our discussion regarding **adaptive vs. non adaptive** greedy algorithms. Prim's Algorithm reorders its input in order to choose the cheapest edge. We say that Prim's Algorithm is an adaptive greedy algorithm; in the sense that, at every iteration, the algorithm tries to readjust the input to its own convenience. In contrast, Kruskal's Algorithm was non-adaptive, since the algorithm sorts the edges **once** at the beginning and blindly processes one edge at a time.

1 Prim's Algorithm

Proof of optimality:

Proof. Let T be the spanning tree returned by the algorithm, and suppose there doesn't exist any MST of G consistent with T. Consider an optimal MST O of G.

Algorithm 1 Prim's Algorithm

Input: A weighted graph $G(V, E, w)$

Output: A spanning tree T that minimizes $\sum_{e \in E'} w(e)$

1: $U \leftarrow V$
2: Pick an arbitrary start vertex $s \in V$
3: $T = \{s\}$
4: $U \leftarrow U \backslash \{s\}$
5: **while** $U \neq \emptyset$ **do**
6:     Choose $u \in U$ adjacent to a $v \in T$ such that $w(u, v)$ is smallest out of all such vertices.
7:     $T \leftarrow T \cup \{u\}$
8:     $U \leftarrow U \backslash \{u\}$
9: **end while**
10: **return** T

Let $e = (x, y)$ be the first edge chosen by the algorithm that is inconsistent with any MST of G, and let T' be the sub-tree of T created by the algorithm just before the edge e was chosen. Let P_{xy} be the path between x and y in O. Let $\mathcal{C}$ be the cycle created in O from adding e to P_{xy} (Fact 1).

P_{xy} must contain an edge $e' = (a, b)$ that has an end point in T' and end point outside of T'. Why? Since otherwise, P_{xy} would be fully contained in T', and choosing $e(x, y)$ next would create a cycle in $T' \cup e$, a contradiction to step 6 of the Algorithm.

Therefore both $e(x, y)$ and $e'(a, b)$ have an end point in T' and an point outside of T'. Since the algorithm chose e instead of e', it follows that $w(e) \leq w(e')$. By Fact (2), we can break $\mathcal{C}$ by removing $e'(a, b)$ and obtaining a new MST of G, call it O'. Notice that the total weight of O' is $w(O') = w(O) + w(e) - w(e')$. Since $w(e) \leq w(e')$, it follows that $w(O') \leq w(O)$. Thus there exists an optimal MST of G, O', that contains the edge $e(x, y)$. Therefore, in order to show that Prim's Algorithm does indeed produce an optimal MST fo G, it suffices to repeat this argument for every new edge $\tilde{e}$ chosen by the algorithm, such that $\tilde{e}$ doesn't appear in any optimal solution. $\square$

2 Kruskal's Algorithm

Algorithm 2 Kruskal's Algorithm

Input: A weighted graph $G(V, E, w)$

Output: A spanning tree T that minimizes $\sum_{e \in E'} w(e)$

Sort the edges of G in non-decreasing order of their weight: $w(e_1) \leq w(e_2) \leq ... \leq w(e_m)$.
$T \leftarrow \emptyset$
for $i \leftarrow 1...m$ **do**
    Remove e_i from E
    if Adding e_i to T does not create a cycle **then**
        $T \leftarrow T \cup \{e_i\}$
    end if
end for
return T

Proof of optimality: We will prove optimality by contradiction.

Proof. Let T be the spanning tree returned by the algorithm, and suppose there doesn't exist any MST of G consisten with T.

Let e be the first edge chosen by the algorithm that is inconsistent with tany MST and let F be the forest constructed by the algorithm just before adding e.

By the choice of e, there exists an optimal MST M of G consistent with F. By Fact (1), adding e to M creates a cycle $\mathcal{C}$. $\mathcal{C}$ must contain an edge e' connecting two trees in F (why?).

Since the algorithm chose e instead of e', $w(e) \leq w(e')$. Therefore, exchanging e for e' can only decrease the total weight of M. And by Fact (2), this exchange would also create a new spanning tree of G that contains e. A contradiction to our assumption. $\square$