Lecture 12: Greedy Algorithms and Minimum Spanning Tree

Introduction

- Optimal Substructure
- Greedy Choice Property
- Prim's algorithm
- Kruskal's algorithm

Definitions

Recall that a *greedy algorithm* repeatedly makes a locally best choice or decision, but ignores the effects of the future.

A *tree* is a connected, acyclic graph.

A *spanning tree* of a graph G is a subset of the edges of G that form a tree and include all vertices of G.

Finally, the *Minimum Spanning Tree* problem: Given an undirected graph $G = (V, E)$ and edge weights $W : E \to \mathbb{R}$, find a spanning tree T of minimum weight $\sum_{e \in T} w(e)$.

A naive algorithm

The obvious MST algorithm is to compute the weight of every tree, and return the tree of minimum weight. Unfortunately, this can take exponential time in the worst case. Consider the following example:

If we take the top two edges of the graph, the minimum spanning tree can consist of any combination of the left and right edges that connect the middle vertices to the left and right vertices. Thus in the worst case, there can be an exponential number of spanning trees.

Instead, we consider greedy algorithms and dynamic programming algorithms to solve MST. We will see that greedy algorithms can solve MST in nearly linear time.

Properties of Greedy Algorithms

Problems that can be solved by greedy algorithms have two main properties:

- Optimal Substructure: the optimal solution to a problem incorporates the optimal solution to subproblem(s)
- Greedy choice property: locally optimal choices lead to a globally optimal solution

We can see how these properties can be applied to the MST problem

Optimal substructure for MST

Consider an edge $e = \{u, v\}$, which is an edge of some MST. Then we can contract e by merging the vertices u and v to create a new vertex. Then any edge adjacent to vertex u or v is adjacent to the newly created vertex, and the process could result in a multiedge if u and v have a mutual neighbor. We resolve the multiedge problem by creating a single edge with the minimum weight between the two edges.

This leads us to the following lemma:

Lemma 1. *If T' is a minimum spanning tree of G/e, then $T' \cup \{e\}$ is an MST of G.*

Proof. Let T^* be an MST of G containing the edge e. Then T^*/e is a spanning tree of $G' = \mathrm{G}/\{e\}$. By definition, T' is an MST of G'. Thus the total weight of T' is less than or equal to that of T^*/e, or $\mathrm{w}(T') \leq w(T^*/\mathrm{e})$. Then

$$w(T) = w(T') + w(e) \leq w(T^*/e) + w(e) = w(T')$$

. □

The statement can be used as the basis for a dynamic programming algorithm, in which we guess an edge that belongs to the MST, retract the edge, and recurse. At the end, we decontract the edge and add e to the MST.

The lemma guarantees that this algorithm is correct. However, this algorithm is requires exponential time, because there are an exponential number of edges that we can guess to form our MST.

We make the algorithm polynomial time by removing the guessing process.

Greedy Choice Property

The MST problem can be solved by a greedy algorithm because the the locally optimal solution is also the globally optimal solution. This fact is described by the Greedy-Choice Property for MSTs, and its proof of correctness is given via a "cut and paste" argument common for greedy proofs.

Lemma 2 (Greedy-Choice Property for MST)**.** *For any cut $(S, V \setminus S)$ in a graph $G = (V, E, w)$, any least-weight crossing edge $e = \{u, v\}$ with $u \in S$ and $v \notin S$ is in some MST of G.*

Proof. First, consider an MST T of G. Then T has a path from u to v. Because $u \in S$ and $v \notin S$, the path has some edge $e' = \{u', v'\}$ which also crosses the cut. Then $T' = T \setminus \{e'\} \cup \{e\}$ is a spanning tree of G and $w(T') = w(T) - w(e') + w(e)$, but e is a least-weight edge crossing $(S, V \setminus S)$. Thus $w(e) \leq w(e')$, so $w(T') \leq w(T)$. Therefore T' is an MST too. □

Prim's Algorithm

Now, we can apply the insights from the optimal structure and greedy choice property to build a polynomial-time, greedy algorithm to solve the minimum spanning tree problem.

Prim's Algorithm Psuedocode

```
1  Maintain priority queue Q on V \ S, where v.key = min{w(u,v) | u ∈ S}
2  Q = V
3  Choose arbitrary start vertex s ∈ V, s.key = ∅
4  for v in V \ {s}
5      v.key = ∞
6  while Q is not empty
7      u = Extract-Min(Q), add u to S
8      for v ∈ Adj[u]
9          if v ∈ Q and v ∉ S and w(u,v) < v.key:
10             v.key = w(u,v) (via a Decrease-Key operation)
11             v.parent = u
12 return {{v, v.parent} | v ∈ V \ {s}}
```

In the above pseudocode, we choose an arbitrary start vertex, and attempt to sequentially reduce the distance to all vertices. After attempting to find the lowest

weight edge to connect all vertices, we return our MST

Correctness

We prove the correctness of Prim's Algorithm with the following invariants.

1. $v \notin S \implies v.key = \min\{w(u,v) \mid u \in S\}$
2. Tree T_S within $S \subseteq$ MST of G.

The first invariant is follows from Step 8 of the algorithm above. A proof of the second invariant follows:

Proof. Assume by induction that $T_S \subseteq$ MST T^*. Then $S \to S' \cup \{e\}$, where e is a least-weight edge crossing the cut $(S, V \setminus S)$. Then we can greedily cut and paste e, which implies that we can modify T^* to include e without removing T_S, since the edges of T_S do not cross the cut. Therefore $T_S \cup \{e\} = T'_S \subseteq T^*$. □

Thus Prim's Algorithm always adds edges that have the lowest weight and gradually builds a tree that is always a subset of some MST, and returns a correct answer.

Runtime

Prim's algorithm runs in

$$O(V) \cdot T_{\text{Extract-Min}} + O(E) \cdot T_{\text{Decrease-Key}}$$

The $O(E)$ term results from the fact that Step 8 is repeated a number of times equal to the sum of the number of adjacent vertices in the graph, which is equal to $2|E|$, by the handshaking lemma.

Then the effective runtime of the algorithm varies with the data structures used to implement the algorithm. The table below describes the runtime with the different implementations of the priority queue.

Priority Queue	$T_{\text{Extract-Min}}$	$T_{\text{Decrease-Key}}$	Total
Array	$O(V)$	$O(1)$	$O(V^2)$
Binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap [CLRS ch. 19]	$O(\lg V)$ (amortized)	$O(1)$ (amortized)	$O(E + V \lg V)$

Kruskal's Algorithm

Kruskal's Algorithm is another algorithm to solve the MST problem. It constructs an MST by taking the globally lowest-weight edge and contracting it.

Kruskal's Algorithm Pseudocode

```
1  Maintain connected components that have been added to the
   MST so far T, in a Union-Find structure
2  Initialize T = ∅
3  for v in V
4      Make-Set(v)
5  Sort E by weight
6  For e = (u, v) ∈ E (in increasing-weight order):
7      if Find-Set(u) ≠ Find-Set(v):
8          Add e to T
9          Union(u, v)
```

Correctness

We use the following invariant to prove the correctness of Kruskal's Algorithm.

Claim 3. *The tree-so-far $T \subseteq$ MST T^*.*

Proof. We give an induction proof. We begin by assuming that the tree-so-far $T \subseteq T^*$, via the inductive hypothesis. When we add an edge e between some components C_1 and C_2, we use the greedy-choice property on the cut $(C_1, V \setminus C_2)$. Thus we have added the edge without removing T, and our new tree-so-far remains a subset of the MST T^*. □

Runtime

Kruskal's algorithm has an overall runtime of

$$T_{sort}(E) + O(V) \cdot T_{\text{Make-Set}} + O(E)(T_{\text{Find}} + T_{\text{Union}}) = O(E \lg E + E\alpha(V))$$

We note that $T_{\text{Make-Set}}$ is $O(1)$ and $T_{\text{find}} + T_{\text{Union}}$ is amortized $O(\alpha(V))$ for Union-Find data structures.

Furthermore, if all weights are integer weights, or all weights are in the range $[0, E^{O(1)}]$, then the runtime of the sorting step is $O(E)$, using Counting Sort or a similar algorithm, and the runtime of Kruskal's Algorithm will be better than that of Prim's Algorithm.

Other MST Algorithms

Currently, the fastest MST algorithm is a randomized algorithm with an expected runtime of $O(V + E)$. The algorithm was proposed by Karger, Klein, and Tarjan in 1993.

6.046J / 18.410J Design and Analysis of Algorithms
Spring 2015