Dijkstra's + Prim's Algorithm

Greedy Tree Builder

Time Complexity

- initialization: $O(1)$ or $O(n)$
- Each vertex placed in $Q$ at most once
- removed at most once

For each edge:
- all constant time except update (increase Priority)

Time to initialize the PQ to have space for all $n$ vertices

$O(n(T_e(n) + T_{E}(n))$

- time to insert in $Q$
- time to extract Max in $Q$

$O(mT_U(n))$

- time to update
\( O(n T_\text{I}(n) + n T_\text{E}(n) + m T_\text{U}(n)) \) \( \{ \text{Adj. List rep.} \) 

- \( \text{insert} \)
- \( \text{extract Max} \)
- \( \text{update (increase Priority)} \)

Binary heap

\( T_\text{I}(n), T_\text{E}(n), T_\text{U}(n) \) \( O(\log n) \)

Time \( O(m \log n) \) assume \( m \geq n \)

Fibonacci heap

\( T_\text{I}(n) + T_\text{U}(n) \) when increase priority \( O(1) \) \( \text{amortized} \)

WORST-case cost of \( O(m + n \log n) \)
Correctness of Dijkstra's Alg

*Aside*

Greedy Algs aren't always optimal

Knapsack that can hold 100 lbs

\[80, 50, 40\]

Greedy alg: Take highest value that fits & repeat
Dijkstra's Alg Proof of Correctness

Prove Following invariants

1. For vertex \( u \in T \), \( u.d \) is length of shortest path from \( S \) to \( u \)

2. For vertex \( u \in Q \), tag for \( u \) is length of shortest path from \( S \) to \( u \) with intermediate vertices restricted to \( T \)
Inductive Proof

Base \( T = \emptyset, Q = \{S\} \) rest is in \( U \) undiscovered vertices

Prop 1 holds vacuously

Prop 2 holds since tag for \( s \) is 0

Inductive Step

Suppose it fails at some point.

Proof by contradiction.
Tree $T$

Priority Queue $Q$

Path with weight $v$ such that:
- $v \leq \frac{\text{tag}}{2}$
- Weight $p^*$ at least as large as weight of green path.

Better path from $s$ to $v$ extracted Max gives $v$ and this causes first failure of one of the properties.
Prim's Minimum Spanning Tree Alg

Greedy Tree Builder where semantics of tag is minimum weight edge connecting vertex to $T$
Correctness Proof

Key fact: Prove that the first edge placed in $T$ is part of some minimum spanning tree.

Let $T$ be a minimum spanning tree

Suppose not!

$T$ remove $e_2$ + add $e_1$, we have a minimum spanning tree
Graph contraction

Problem is reduced to finding a minimum spanning tree in this graph with 1 less vertex.
Prim's $O(m+n \log n)$

Kruskal's Minimum Spanning Tree Algorithm

Also greedy alg.

$O(m \log m)$

Sorts all edges from smallest to highest weight.

Nearly $O(m)$

Repeatedly add next to $T$ unless it creates a cycle

$O(m \cdot \text{inverse-ackerman}(n))$
Kruskal's algorithm is done when $n-1$ edges in the tree.

See soon how to determine if there is a cycle in $O(n+m)$ time.
Data Structure called Union-Find

Amortized cost for each union & find is inverse Ackerman function nearly constant
How does Prim's Alg avoid the sorting cost?

each vertex (using priority queue)

remembers it's min weight connection

doesn't sort all adjacent edges by weight
Next class

Depth-first search

See what kinds problems it's useful for

Next Thur

Garbage Collection