Dijkstra’s Alg & Prim’s MST Alg

Last time we saw Dijkstra’s Single-source Shortest Path algorithm.

Today

• Generalize Dijkstra’s alg (Greedy Tree Builder)
• Analyze time complexity of Greedy Tree Builder
• Prove correctness of Dijkstra’s Alg
• Present Prim’s Minimum Spanning Tree Alg
Dijkstra Single Source Shortest Path (Vertex S)

S.tracker = pq.addTracked(0, S)  \textcolor{red}{\text{tag: semantics}}

For all \( v \in V - \{ S \} \)

\textcolor{green}{V.tracker = pq.addTracked(\infty, S)} \textcolor{green}{\text{length of shortest path from S Found so far}}

for all \( v \in V \)

\textcolor{green}{V.parent = null}

While (!pq.isEmpty())

\textcolor{red}{tag = pq.minTag()}  \textcolor{red}{\text{remain vertices in pq are not reachable from S}}

if (tag == \infty) return

\textcolor{red}{u = pq.extractMin()}

\textcolor{red}{u.dist = tag}

for each edge from \( u \)

\textcolor{green}{v = e.dest}

if V.tracker.inCollection() \&\& \textcolor{green}{u.dist + e.weight < v's current tag}

\textcolor{red}{v.tracker.update(u.dist + e.weight)} \textcolor{red}{\text{new tag}}

\textcolor{green}{v.parent = e}  \textcolor{green}{\text{found better path}}

\textcolor{green}{v's current tag is cost of this path}
Observe that

1. Each vertex in tagged priority queue has a tag which is cost of shortest path from $S$ found so far.

$$\text{Shortest path: } S \rightarrow u \rightarrow v$$

- Length is $u$'s tag.
- Path has cost $u$'s tag + weight of $e$. 
Minimum Spanning Tree Problem

Undirected weight graph
Find set $T$ of edges where there is a path between any pair vertices where $\sum_{e \in T}$ weight of $e$ is minimized

Observe: in opt sol $T$ can always be a tree
Maximum Bottleneck Problem

specified start/source $s$, end $t$

bottleneck path ($P$) has min weight edge on $P$

Goal: Find path from $s$ to $t$ with maximum bottleneck

\begin{align*}
100 & \rightarrow 100 \\
100 & \rightarrow \\
3 & \rightarrow 3 \\
3 & \rightarrow 3 \\
3 & \rightarrow 3
\end{align*}
Greedy Tree Builder

- Change the semantics associated with the tag (for each vertex)
- Change tag given to source/seed
- Decide if min or max value tag is the best one to pick next
Shortest Path tag is weight of shortest path found so far from $S$. For $S$ tag is 0. Pick vertex with min tag.

Minimum Spanning Tree tag is weight of smallest edge connecting vertex to partially built tree $T$. Seeds $S$ as any vertex, tree begins there. For $S$ tag is 0.

Update rule: \[
\min(v's \text{ tag}, \text{weight})
\]
Maximum Bottleneck problem

**Semantics:** tag of $V$ is bottleneck of best path found so far from $S$ to $V$

1. tag for $S$ - $D$

2. min or max tag? - $\max$

3. $\max\left(V \cdot \text{tag}, \min\left(U \cdot \text{tag}, \text{weight of } e\right)\right)$

path of parent edge

$\min\left(U \cdot \text{tag}, \text{weight of } e\right)$

$\max$
Greedy Tree Builder

Initially $s$ is placed in $T$. Then the following steps are repeated until all discovered vertices have been placed in $T$.

1. Select the vertex $u \in Q$ with the highest priority over all vertices in $Q$. (For each algorithm, a proof that this greedy choice is part of an optimal solution is required to prove that the final solution is optimal.)

2. Remove $u$ from $Q$, which implicitly places $u$ in $T$. Since the cost for each vertex $v \in Q$ represents its best connection to some vertex in $T$, the addition of $u$ to $T$ provides a new possible connection for each vertex $v \notin T$.

3. Consider all outgoing edges $e = (u, v)$ from $u$.
   
   (a) If $v \in U$, then $v$ is placed into $Q$ after setting the edge from its parent to $e$ and initializing $v_{\text{cost}}$ to the cost associated for parent edge $e$.

   (b) If $v \in Q$, the cost associated with $v$, for parent edge $e$, is computed. If this cost $c$ is better than $v$’s current cost, then the cost for $v$ is set to $c$ and its parent edge is set to $e$. 

---

$S$ source/seed

$T$ tree

$Q$ queue holding discovered vertices

Continue until $Q$ is empty
```java
void consider(E e, double parentCost, TaggedPriorityQueue<Double, V> pq) {
    double newCost = getCost(e, parentCost);
    if (newCost < loc.get().getTag()) {
        edgeFromParent = e;
        cost = newCost;
        pq.updateTag(cost, loc);
    }
}
```

Dijkstra's algorithm:

getCost(e, parentCost) = e.weight() + parentCost

Prim's algorithm:

getCost(e, parentCost) = e.weight()
void greedyTreeBuilder(tree, seedCost, comp)

Create a TaggedPriorityQueue<Double, V> pq that uses comp
add source/seed as root of tree with cost seedCost
source.loc = pq.putTracked(seedCost, source)

while (!pq.isEmpty())
    V u = pq.extractMax().getElement();

    for each outgoing edge e leaving u
        if (e.weight < 0) throw new NegativeWeightEdgeException
        V v = other endpoint of e (other than u)
        vData is an object holding data associated
        with v For this algorithm
        vData.loc in Collection()
        vData.consider(v, e, vData, getCost(e, u's cost))

    in Q if (vData.loc in Collection())
        vData.consider(v, e, vData, getCost(e, u's cost))

    uncovered
    v.add(v to tree with parent e + cost
    vData.loc = pq.putTracked(vData, cost, v)
Time Complexity

initialization $O(1) \text{ or } O(n)$

Each vertex

placed in $Q$ at most once

$+ \text{ removed at most once}$

$\left\{ \begin{align*}
O(n(T_E(n) + T_Q(n))) \\
\text{time to insert in } Q \\
\text{time to extract max in } Q \\
\end{align*} \right.$

For each edge

all constant time except

update (increase Priority)

$\left\{ \begin{align*}
O(mT_U(n)) \\
\text{time to update} \\
\end{align*} \right.$