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 $0$.

Minimum Spanning Tree: tag is weight of smallest edge connecting vertex to partially built tree $T$.

Update rule: $\min(v \text{'}s \text{ tag, weight})$. Also want min tag.
**Maximum Bottleneck problem**

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

1. tag for S

2. min or max tag? max

3. update

\[
\max \left( V's \ tag, \ \min \left( \text{parent edge}, \ \min \left( \text{tag of } V, \ \text{weight of } e \right) \right) \right)
\]
**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 $v \not\in U$, then $v$ is placed into $Q$ after setting the edge from its parent to $e$ and initializing $v_{cost}$ 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 $e$. (Did we find a better connection for $v$?)
Function to consider edge e

```java
void consider(Edge e, double parentCost, TaggedPriorityQueue<Double, Vertex> pq) {
    double newCost = getCost(e, parentCost);
    if (newCost < loc.get().getTag()) {
        edgeFromParent = e;
        cost = newCost;
        pq.updateTag(cost, loc);
    }
}
```

dijkstra's alg: \( \text{getCost}(e, \text{parentCost}) = e.\text{weight}() + \text{parentCost} \)

Prim's MST alg: \( \text{getCost}(e, \text{parentCost}) = e.\text{weight}() \)
void greedyTreeBuilder(tree, seedCost, comp)

init

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()

get vertex (associated data)

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, changes

in Q

{ if (vData.loc.inCollection(v))

vData.consider(v, e, vData.getCost(e, u's cost))

else

add v to tree with parent e + cost

vData.loc = pq.putTracked(vData.cost, v)