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

min (v's tag, weight e + u's tag)

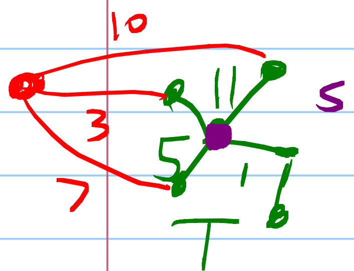
Shortest Path

tag is weight of shortest path found so far from s, for s tag is 0, pick vertex with min tag

Minimum Spanning Tree

Also want min tag

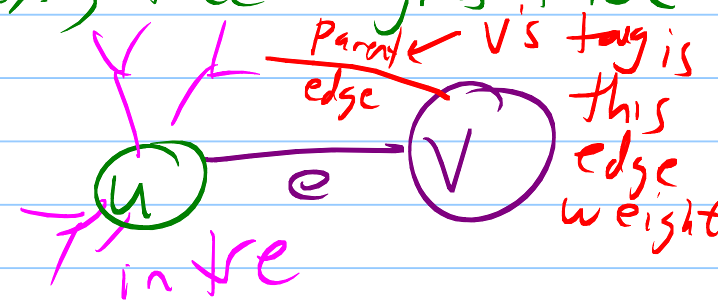
tag is weight of smallest edge connecting vertex to partially built tree T



seed s as any vertex, tree begins there

for s tag is 0

update rule: min (v's tag, weight e)



Maximum Bottleneck problem

semantics: tag of V is bottleneck of best path found so far from s to V

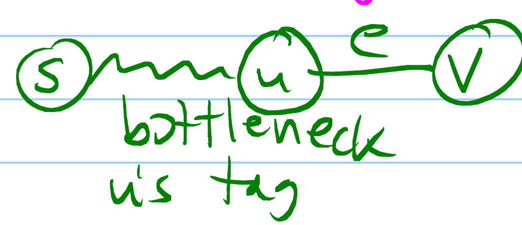
1. tag for s $\underline{\infty}$

2. min or max tag? $\underline{\text{max}}$

update

3. max(v's tag, min(u's tag, weight of e))

path of parent edge



bottleneck

- tag of V
- min of u's tag, weight of e

Greedy Tree Builder

S source/seed
T tree
Q queue holding discovered vertices

initialization

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

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 $v \notin T$.

did we find a better connection for v?

3. Consider all "outgoing" edges $e = (u, v)$ from u.

   undiscovered
   not in T
   or Q

   (a) If $v \in U$, 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. use update rule

   (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.

Function to consider edge e

$u \xrightarrow{e} v$

parent

tag for parent

```
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);
    }
  }
}
```

tracker

cost for best sol so far (v's tag)

getCost implements what I had been calling update

dijkstra's alg: getCost(e, parentCost) = e.weight() + parentCost

Prim's MST alg getCost(e, parentCost) = e.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
in Q
            if (vData.loc.inCollection())
                vData.consider(v, e, vData.getCost(e, u's cost)) changes
            else
undiscovered until now
                add v to tree with parent e & cost
                vData.loc = pq.putTracked(vData.cost, v)