Minimum Spanning Tree

Given

Undirected Graph G with positive edge weights (connected)

Definition A spanning tree of G is a subgraph T that is both a tree (connected and acyclic) and spanning (includes all vertices)

Goal = find a min weight spanning tree

Brute force : try all spanning trees?

Greedy Algorithm

Simplifying Assumptions:

- edge weights are distinct
- graph is connected

Consequence : MST exists and it is unique

Cut Property

A cut in a graph is a partition of its

vertices into two (nonempty) sets.

Crossing Edge : connects a vertex in one set with a vertex in the other

$\boxed{\text{Cut Property}}$ Given any cut, the crossing edge of min weight is in the MST

Greedy MST Algorithm

- start with all edges colored gray
- find cut with no black crossing edges
- color its min-weight edge black
- repeat until V-1 edges are colored black

* Consider a cut whose vertices are one connected component

- Can we remove the two simplifying assumptions
  1) edge weights not distinct
     => multiple MSTs

2) What if graph is not connected?

=> we compute a minimum spanning forest = MST of each connected component

Edge-Weighted Graph API

I) Weighted edge API

class Edge

- Edge (int v, int w, double weight)

- int eighter () - either endpoint
- int other (int o)
- int compareTo (Edge that)
- double weight()

II) Edge-weighted graph API

class EdgeWeightedGraph

- void addEdge (Edge e)
- Iterable <Edge> adj (int v)
- Iterable <Edge> edges ()

- int V()
- int E()

Conventions = allow self-loops and parallel edges

III) Minimum Spanning Tree API

class MST

- MST (EdgeWeightedGraph G)
- Iterable<Edge> edges()
- double weight()

Kruskal's Algorithm

- consider edges in ascending order of weight
- add next edge to tree T unless doing so would create a cycle

Proof: Kruskal's algorithm is a special case of the greedy MST algorithm

- Suppose Kruskal's algorithm colors the edge e (v – w) black
- cut = set of vertices connected to v in tree T
- no crossing edge is black
- no crossing edge has lower weight

<u>Challenge</u>

- Would adding edge $v \to w$ to tree T create a cycle? If not, add it!

1) - run DFS from v and check if w is reachable $\to O(V)$

2) - use $\boxed{\text{union-find}}$ – to see if v and w are connected $\longrightarrow O(\log V)$

* maintain a set of each connected comp. in T

* if v and w are in the same set $\Rightarrow$

adding v – w would create a cycle

- otherwise add v – w to T and merge the sets containing v and w

$\boxed{\text{Kruskal}} \longrightarrow O(E \log E)$

Prim's Algorithm

- start with vertex 0 and greedily grow tree T

- add to T the min weight edge with exactly one endpoint in T

- repeat until V-1 edges

Proof: Prim's algorithm is a special case of the greedy MST algorithm

⇒ suppose edge e = min weight edge connecting a vertex on the tree to a vertex not on the tree.

- cut = set of vertices connected on tree
- no crossing edge is black
- no crossing edge has lower weight

=> use a priority queue!

Lazy implementation => find the min weight edge with exactly one endpoint in T

=> Maintain a PQ of edges with (at least) one endpoint in T

- $\boxed{\text{key}}$ = edge, $\boxed{\text{priority}}$ = weight of edge
- deleteMin to determine the next edge e = v-w to add to T
- disregard edge that has both endpoints v and w in tree T
- otherwise < v is on the tree / w not on the tree
  - add to PQ any edge incident to w

(assuming the other endpoint not in tree)

- add w to T
- Running Time $O(E \log E)$

<u>Eager Implementation</u>

- maintain a PQ of vertices connected by an edge to T $\Rightarrow$ priority of vertex v = weight of the shortest edge connecting v to T
  - delete the min vertex v and add its associated edge $e = v \to w$ to T
  - update PQ by considering all edges $e = v - x$ incident to v
    - ignore if x is already in T
    - add x to PQ if not already on it
    - <u>decrease priority</u> of x if $v - x$ becomes shortest edge connecting x to T

Indexed Priority Queue

- associate an index between 0 and N-1 for each key in the priority queue
  - client can insert and delete the minimum
  - client can change the key by specifying the index

IndexMinPQ

- void insert (int i, key key)
- void decreasekey (int i, key key)
- boolean contains (int k)
- int delMin()
- boolean isEmpty ()
- int size ()

- ~~start~~ start with the same code for $\boxed{\text{MinPQ}}$
- maintain parallel arrays keys[ ],

pq[] and qp[] so that:

- keys[i] = priority of i
- pq[i] = index of the key in heap position i
- qp[i] = heap position of the key with index i

- use swim(qp[k]) implement <u>decrease key</u>

<u>MST Context</u>

- is there a linear time MST algorithm?
- Euclidean MST
- Clustering
- single-link clustering algorithm