15-210
Parallel and Sequential Algorithms and Data Structures

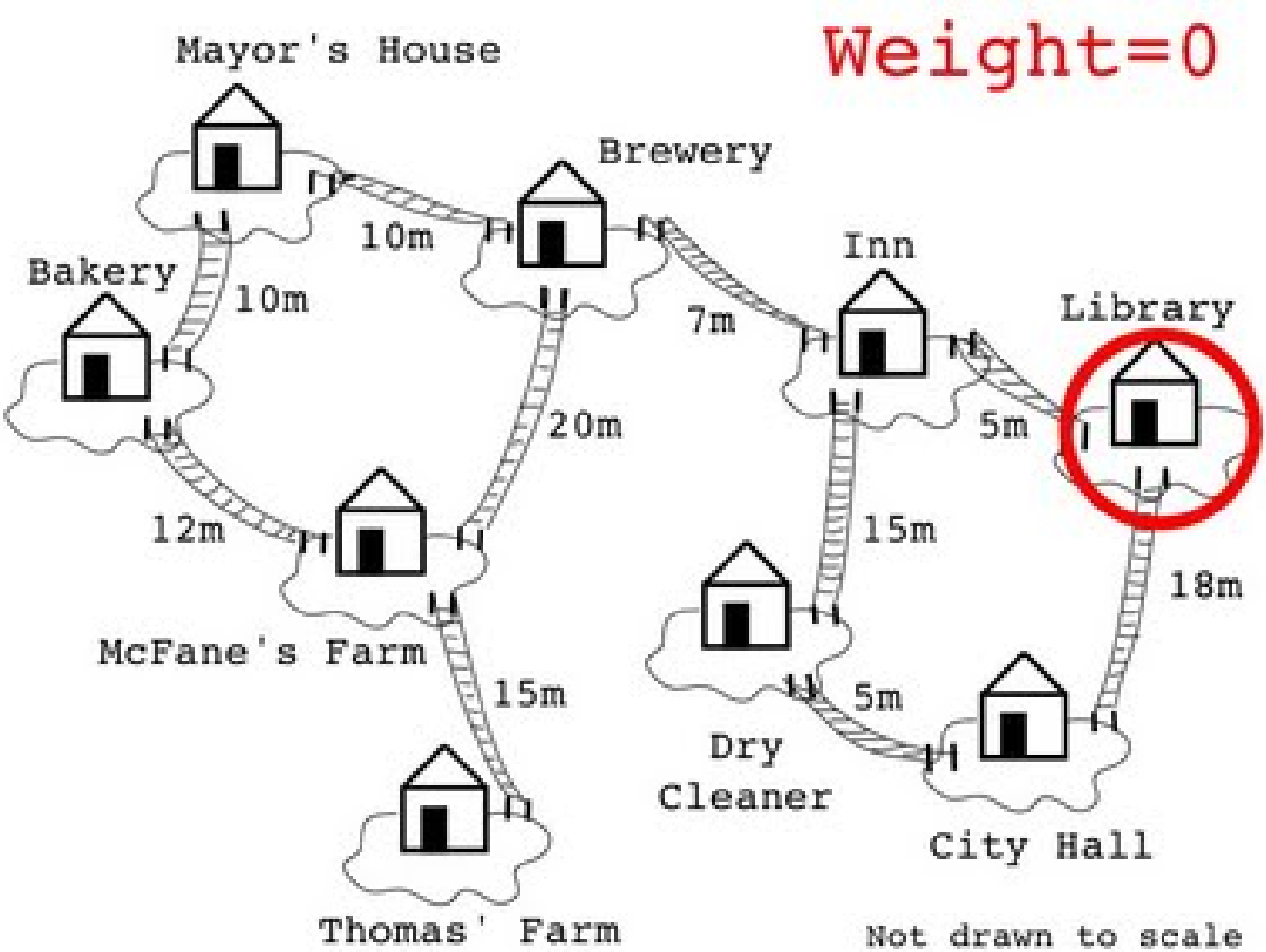
Lecture 17

Minimum Spanning Trees

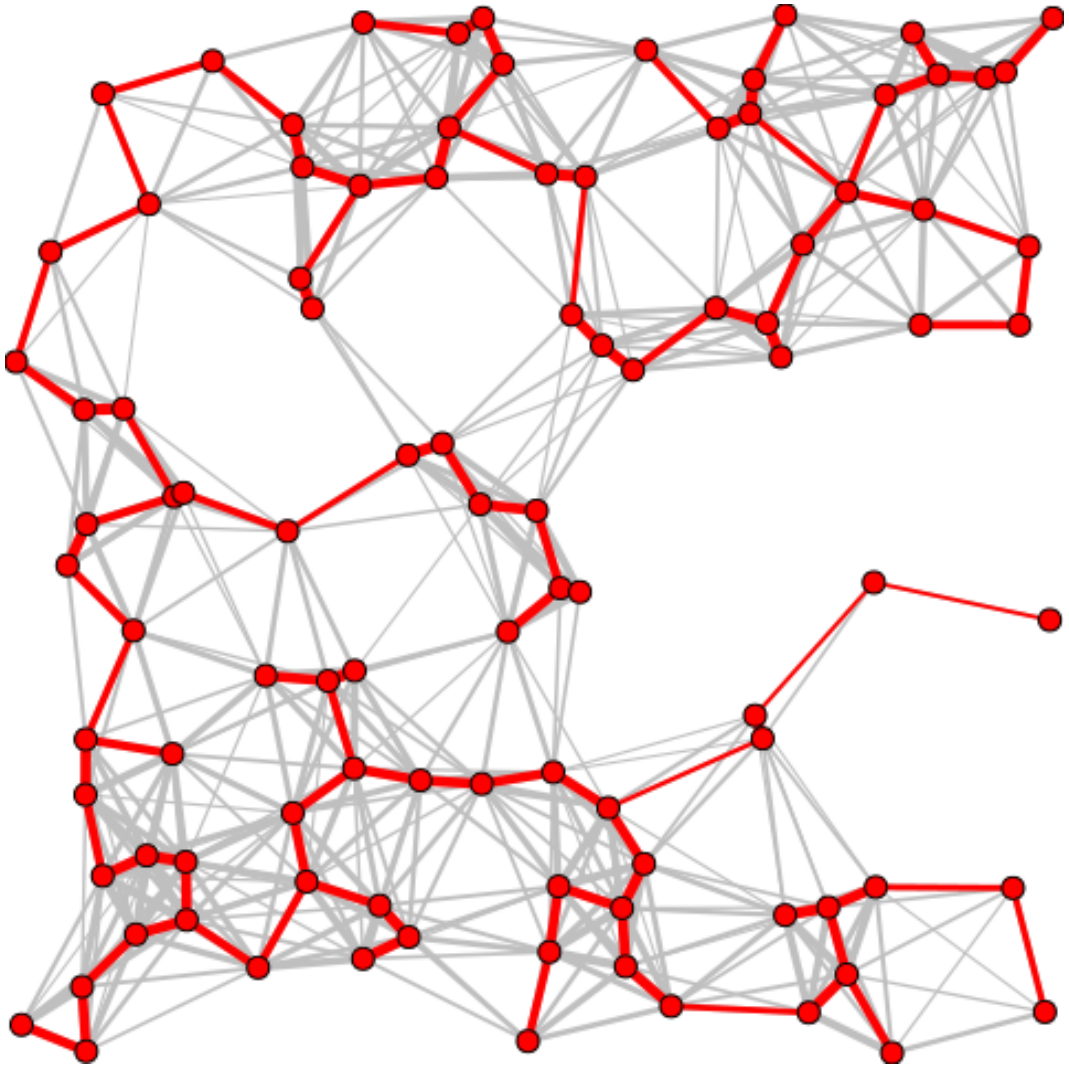
Synopsis

- Minimum Spanning Trees
- Kruskal's and Prim's Algorithms
- Using Star Contraction for MST

Minimum Spanning Trees



Minimum Spanning Trees



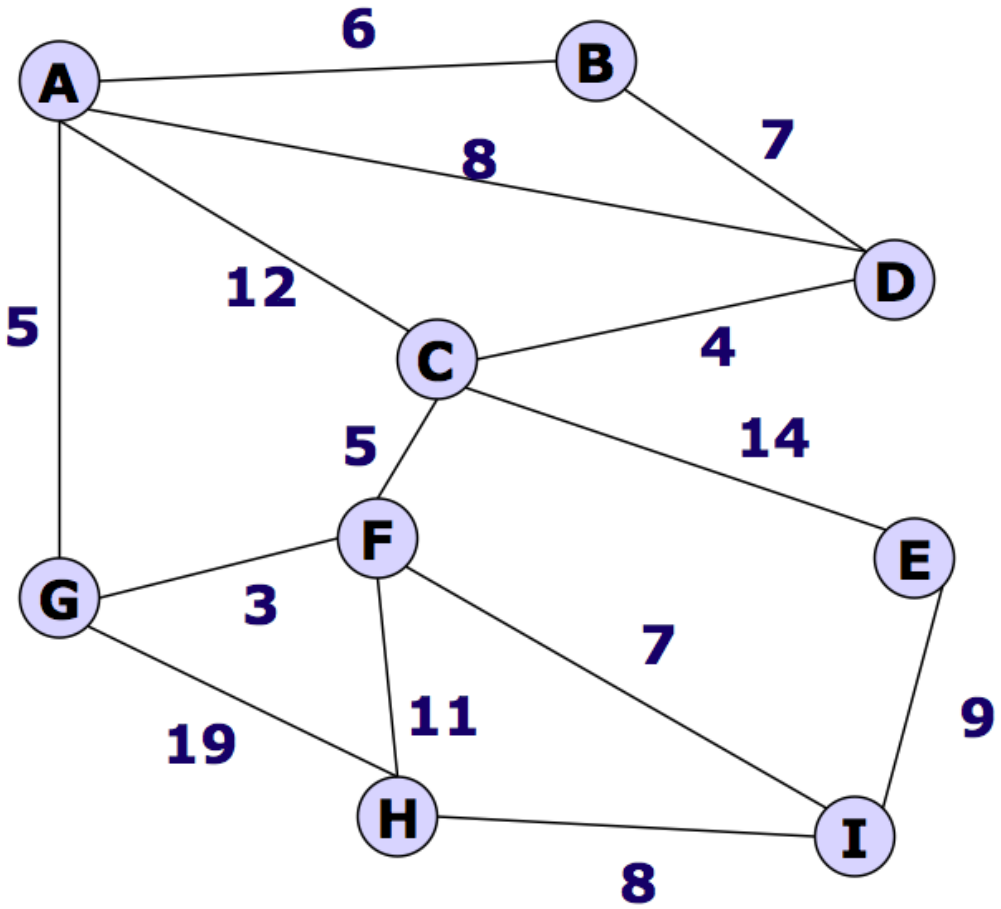
Minimum Spanning Trees

- Given a connected undirected graph $G = (V, E)$
  - Each edge e has $w_e \geq 0$
- Find a spanning tree, T that minimizes

$$w(T) = \sum_{e \in E(T)} w_e.$$

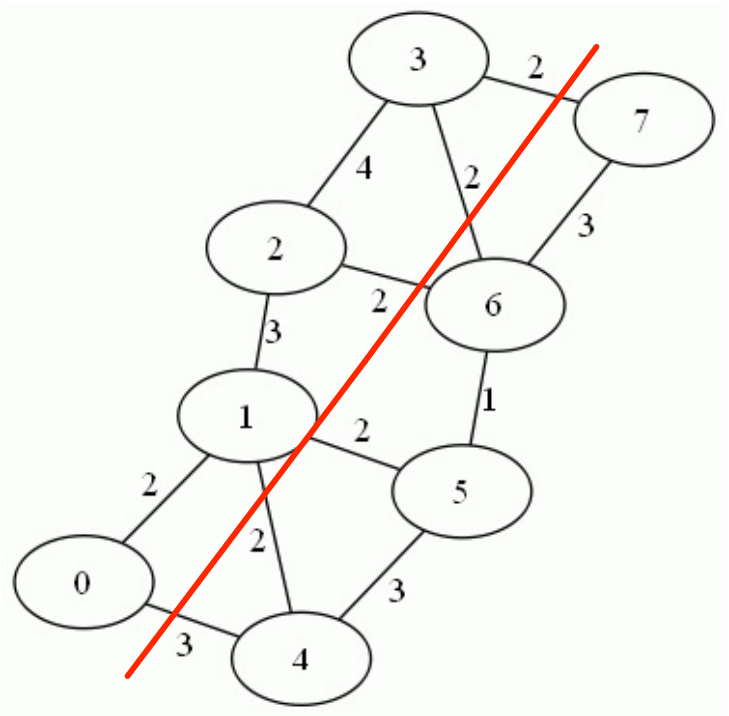
- Sequential algorithms:
  - Kruskal's Algorithm
  - Prim's Algorithm

Minimum Spanning Trees



Light Edge Rule

- Given $G = (V, E)$, $U \subsetneq V$ partitions the graph into two parts with vertices U and $V \setminus U$.
- The edges between U and $V \setminus U$ are called the cut edges $E(U, \overline{U})$.

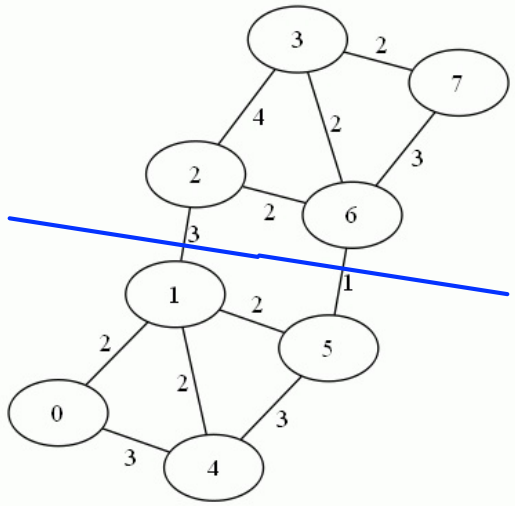


Light Edge Rule

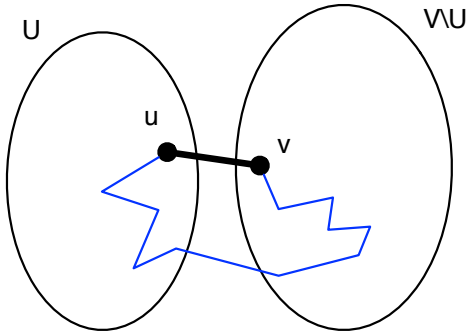
Theorem

Let $G = (V, E, w)$ be a connected undirected weighted graph with distinct edge weights.

- For any nonempty $U \subsetneq V$
- the minimum weight edge e between U and $V \setminus U$ is in the minimum spanning tree MST(G) of G.



Light Edge Rule

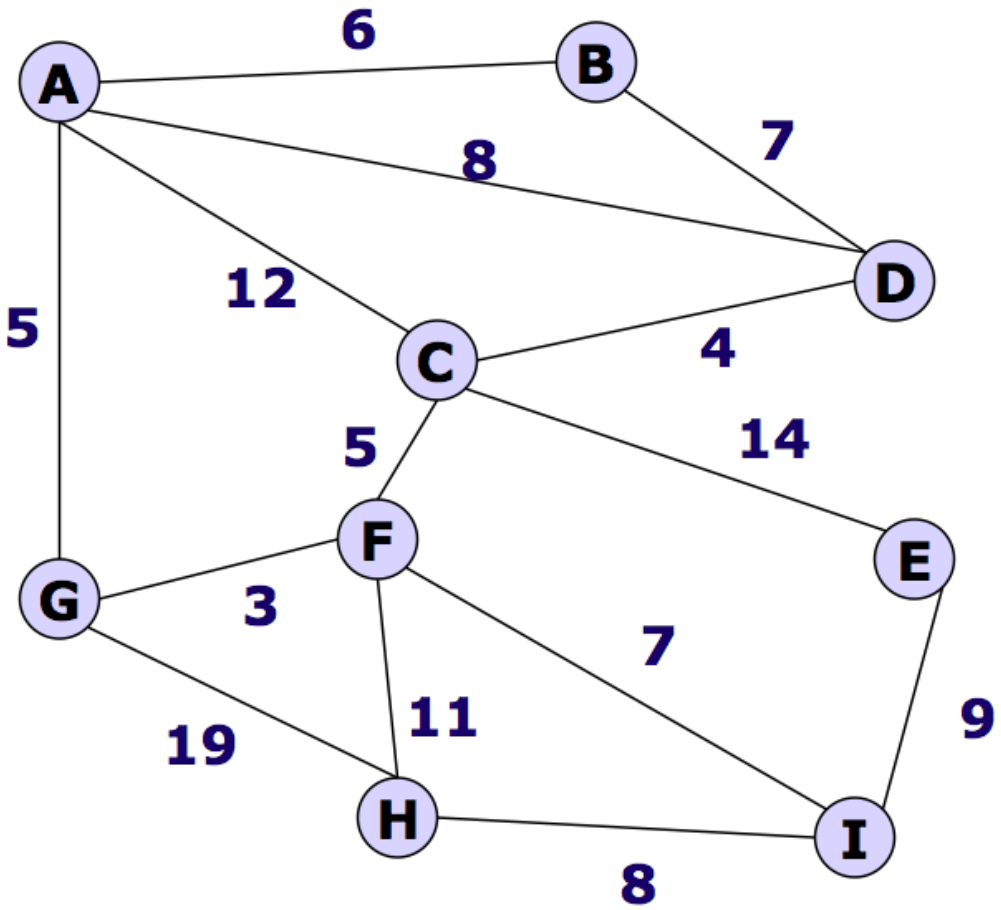


- Assume $e = (u, v)$ is the minimum edge in the cut but not in the MST.
- MST should have at least another edge in the cut.
- Adding e to the path between u and v creates a cycle.
- Removing the max edge from path (blue line) and adding e should give a ST with less weight.
- Original (claimed) MST (through blue line) can not be a MST!

Kruskal's Algorithm

- Greedy
- Each vertex is a subtree by itself initially
- Combine the two sub-trees on both sides of the next smallest edge (if they are different)
- Uses the union-find data structure.
- $O(m \log n)$ work and span!

Kruskal's Algorithm



Prim's Algorithm

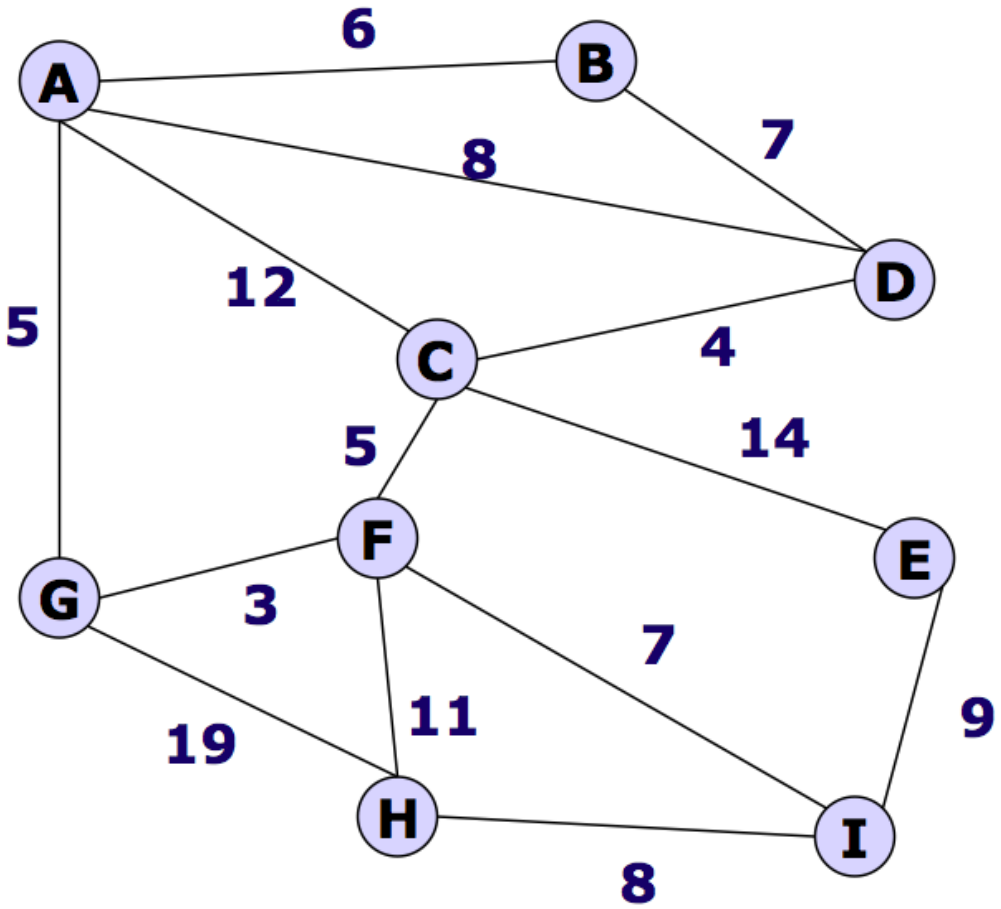
- Greedy
- Based on Priority-based Search – Variant of Dijsktra's Algorithm
- Maintain visited X and frontier F vertices.
- Visit the nearest unvisited vertex in the frontier.
- $O(m \log n)$ work and span!

Prim's Algorithm

```
1  fun prim(G) =
2  let
3     fun enqueue v (Q,(u,w)) = PQ.insert (w,(v,u)) Q
4     fun prim'(X, Q, T) =
5        case PQ.deleteMin(Q) of
6           (NONE, _) ⇒ T
7         | (SOME(d,(u,v)),Q') ⇒
8           if (v ∈? X) then prim'(X, Q', T)
9           else let
10             X' = X ∪ {v}
11             T' = T ∪ {(u,v)}
12             Q'' = iter (enqueue v) Q' N_G(v)
13          in prim'(X', Q'', T') end

14       s = an arbitrary vertex from G
15       Q = iter (enqueue s) {} N_G(s)
16 in
17    prim'({s}, Q, {})
18 end
```

Prim's Algorithm

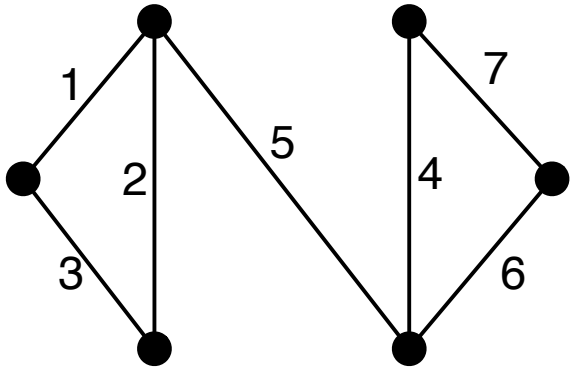


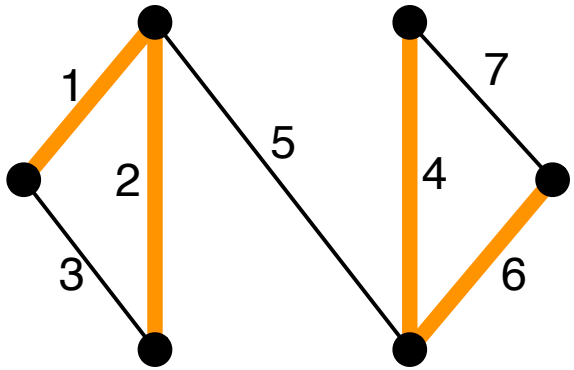
Parallel MST Algorithms

Observation

- The minimum weight edge out of every vertex of a weighted graph G belongs to its MST.

- Why should this be the case?

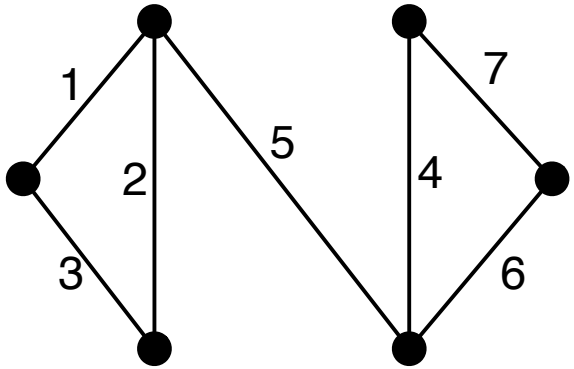


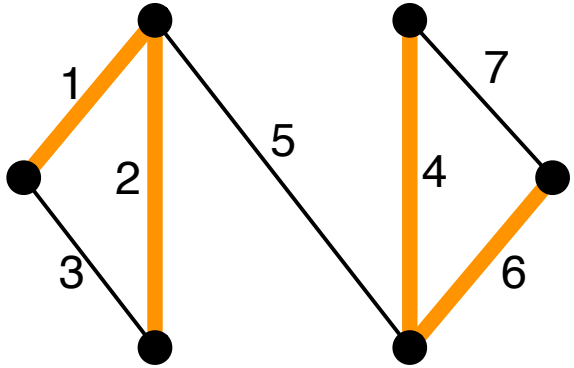


- MST can contain other edges!

Parallel MST - Idea #1

- Throw all minimum weight edges into MST
- Tree contract the vertices for all these edges
- Repeat until no edges remain!





- Each rounds removes at least 1/2 of the vertices (Why?)

Parallel MST - Idea #2

- Let *minE* be the set of minimum weight edges.
- Let $H = (V, minE)$ be a subgraph of G
- We apply (modified) star contraction to H
  - The tails hook up through the minimum weight edge!

```
1  fun minStarContract(G = (V, E), i) =
2  let
3      minE = minEdges(G)
4      P = {u ↦ (v, w) ∈ minE | ¬heads(u, i) ∧ heads(v, i)}
5      V' = V \ domain(P)
6  in (V', P) end
```

Parallel MST - Idea #2

- Even though we are working with a subgraph, the star contract lemma still applies.

Lemma

For a graph G with n non-isolated vertices, let X_n be the random variable indicating the number of vertices removed by `minStarContract`(G, r). Then, $\mathbf{E}(X_n) \geq n/4$.

- MST will take expected $O(\log n)$ rounds.

Bookkeeping

- As the graph contracts, the end point of each edge changes!
- At the end, the edges may not have the original end points.
- Associate a unique label to each edge initially:
  - $(\mathit{vertex} \times \mathit{vertex} \times \mathit{weight} \times \mathit{label})$
  - The end points change but the label does not!

Modified Star Contract

```
1  fun minStarContract(G = (V, E), i) =
2  let
3      minE = minEdges(G)
4      P = {(u ↦ (v, w, ℓ)) ∈ minE | ¬heads(u, i) ∧ heads(v, i)}
5      V′ = V \ domain(P)
6  in (V′, P) end
```

- Line 3: Finds min edge for each vertex.
  - All these belong to the MST
- Line 4: Picks tails and heads, and the creates mapping from tails to heads.
- Line 5: Removes all tail vertices from the vertex set.

The MST Algorithm

```
1   fun MST((V, E), T, i) =
2   if |E| = 0 then T
3   else let
4        (V', PT) = minStarContract((V, E), i)
5        P = {u ↦ v : u ↦ (v, w, ℓ) ∈ PT} ∪ {v ↦ v : v ∈ V'}
6        T' = {ℓ : u ↦ (v, w, ℓ) ∈ PT}
7        E' = {(P[u], P[v], w, l) : (u, v, w, l) ∈ E | P[u] ≠ P[v]}
8   in
9      MST((V', E'), T ∪ T', i + 1)
10  end
```

- Invoked by $\mathtt{MST}(G, \{\}, 1)$.

Implementing minEdges(G)

fun $\mathit{joinEdges}((v_1, w_1, l_1), (v_2, w_2, l_2)) =$
  if $(w_1 \leq w_2)$ **then** (v_1, w_1, l_1) **else** (v_2, w_2, l_2)

fun $\mathit{minEdges}(E) =$
let
  $ET = \{u \mapsto (v, w, l) : (u, v, w, l) \in E\}$
in
  $(\mathit{merge}\ \mathit{joinEdges})\quad \{\}\quad ET$
end