W, 10/09/19

Fall'19 CSCE 629

Analysis of Algorithms

Fang Song

Texas A&M U

Lecture 16

- Interval partitioning
- Minimum spanning tree

Credit: based on slides by A. Smith & K. Wayne

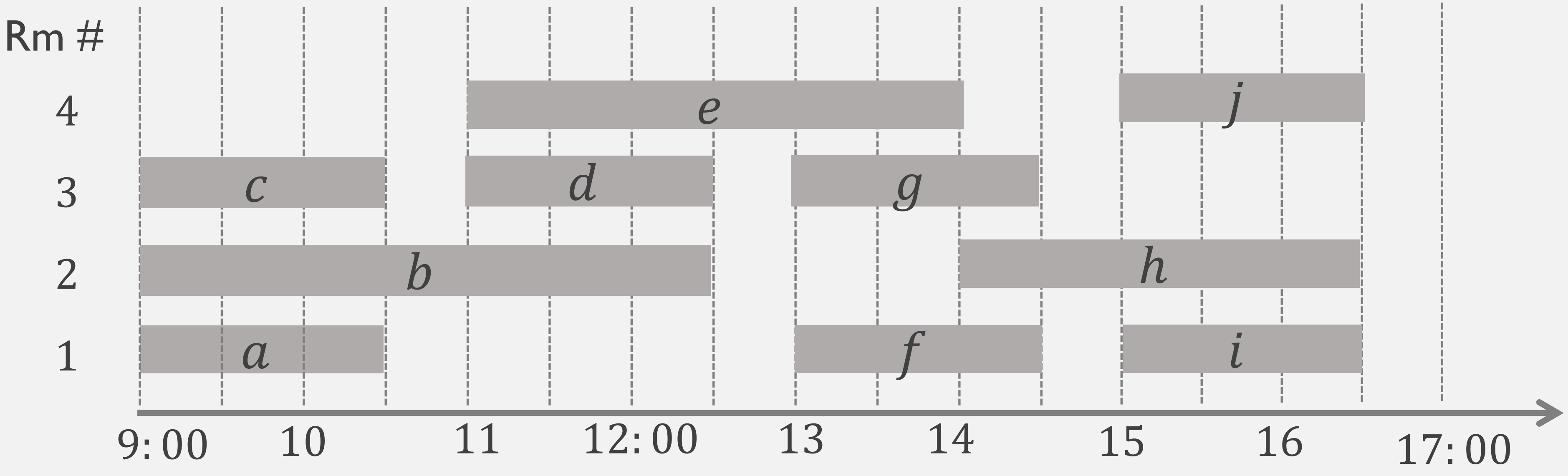
Interval Partitioning Problem

Scheduling classes

- Input. Lectures $\{s_j, f_j\}$
- Output. Minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.

Can you do better?

10 lectures scheduled in 4 classrooms

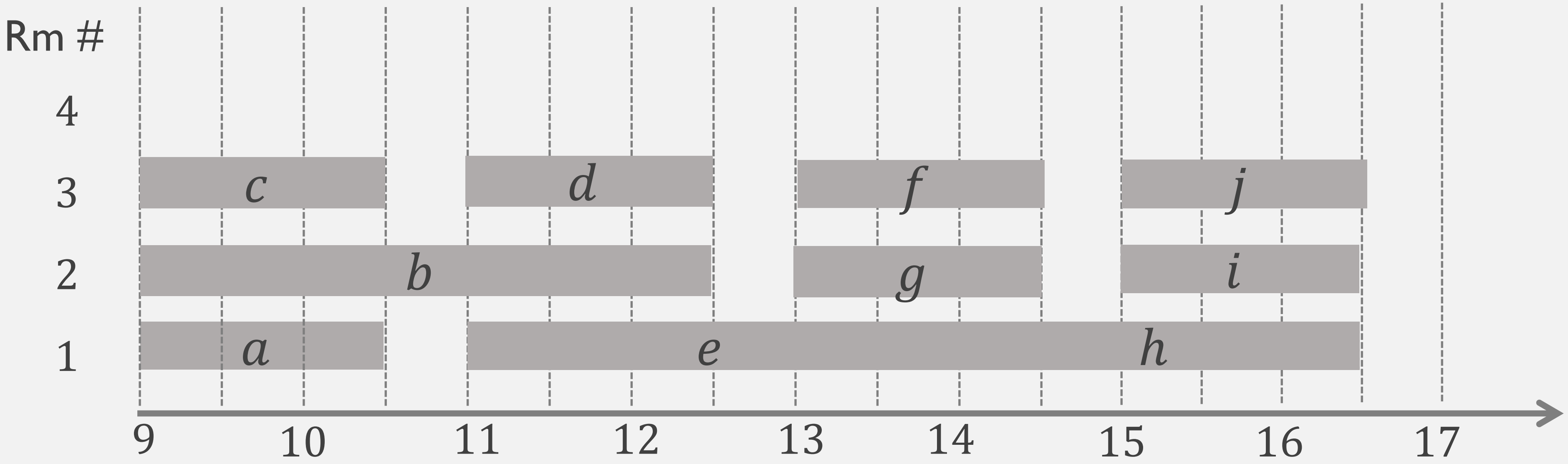


Interval Partitioning Problem

Scheduling classes

- Input. Lectures $\{s_j, f_j\}$
- Output. Minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.

YES! 10 lectures scheduled in 3 classrooms



Greedy algorithm

- Idea. Sort lectures in increasing order of start time: assign lecture to any compatible classroom.

IntPartition$(\{s_j, f_j\})$ // $r \leftarrow 0$ # of allocated rooms
1. Sort by starting time so that $s_1 \leq s_2 \leq \cdots \leq s_n$
2. For $j = 1, \ldots, n$
   - If j compatible with some classroom k
     - Schedule j in room k
   - Else allocate new classroom $r + 1$
     - Schedule j in room $r + 1$
     - $r \leftarrow r + 1$

How to do it in $O(\log r)$

OBS. # rm needed $\geq$ depth of input intervals (i.e., Max. number of lectures that overlap)

- Running time. $O(n \log n)$
- Optimality. #Rm allocated = depth of input intervals

Minimum spanning tree (MST)

- Input. A connected undirected graph $G = (V, E)$
  - Weight function $w: E \to \mathbb{R}$
  - For now, assume all edge weights are distinct

A tree that connects all vertices

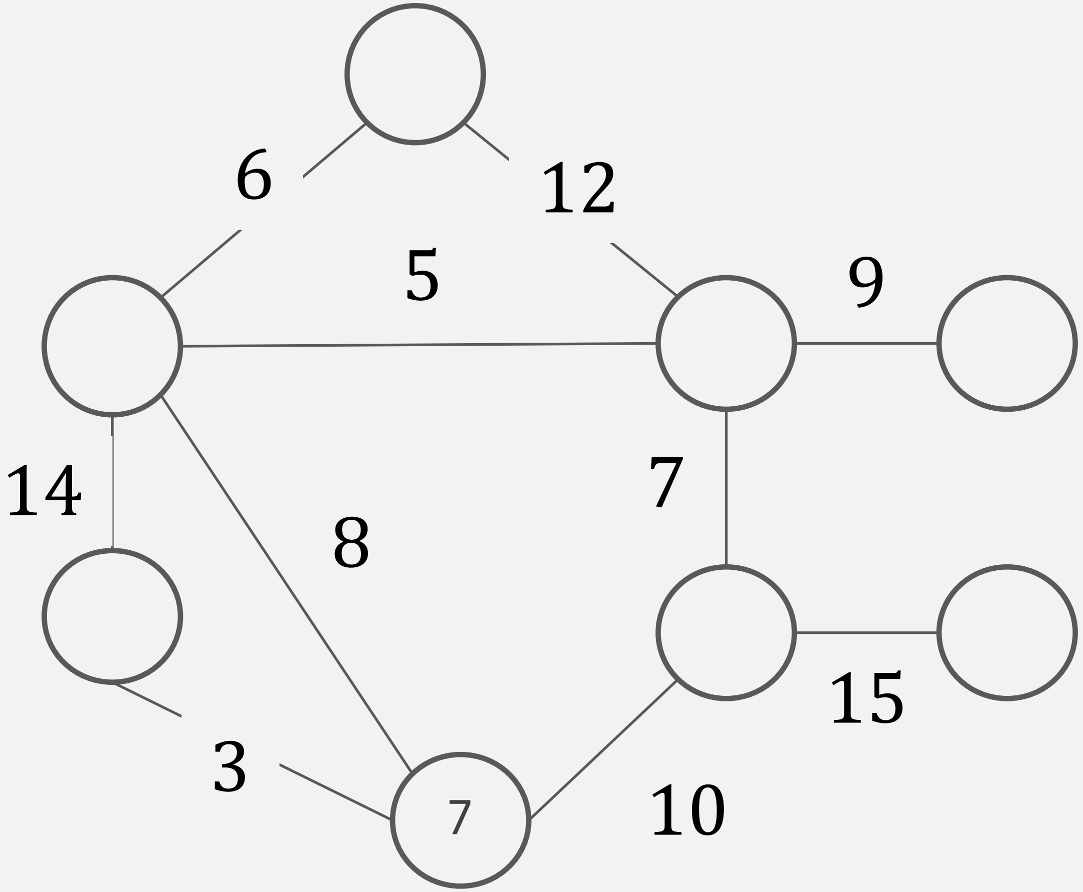
- Output. A spanning tree T of minimum weight

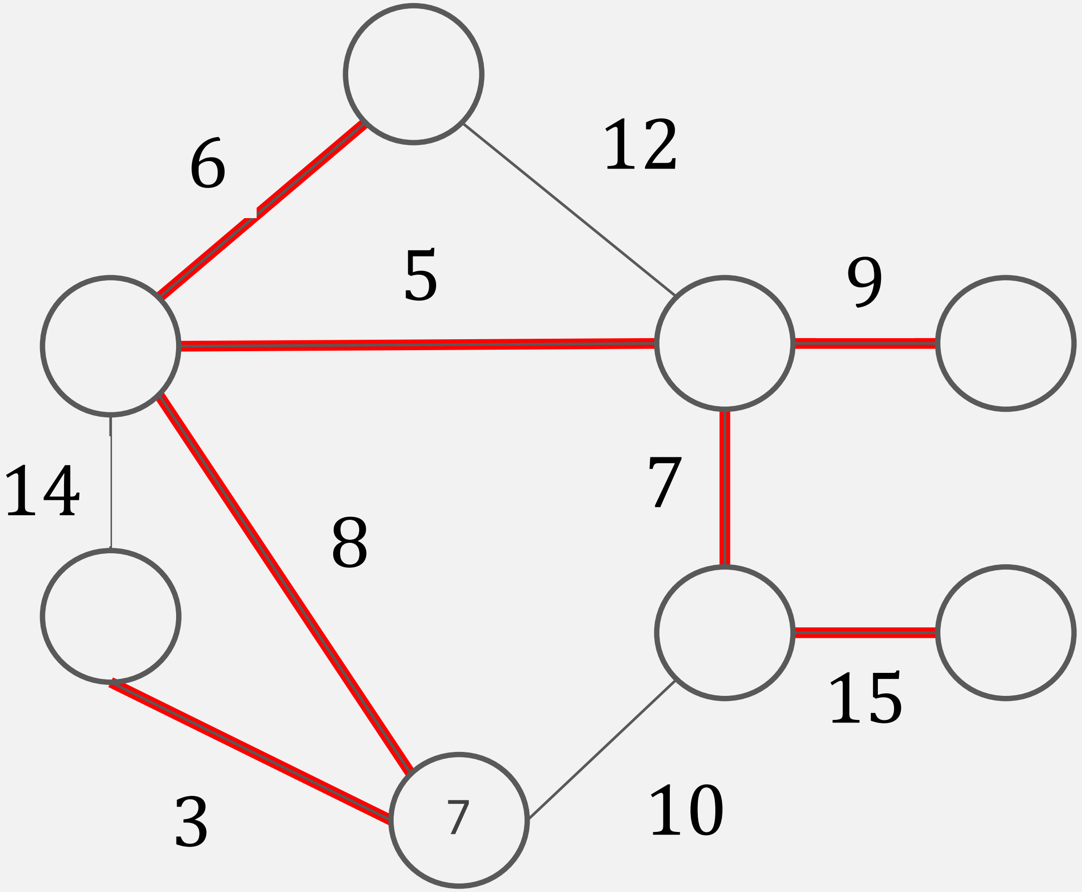
$$w(T) \coloneqq \sum_{(u,v)\in T} w(u, v)$$

Applications

- Cluster, Real-time face verification
- Network design (communication, electrical, computer, road)
- ....

Example of MST





Pop quiz 1

Which of the following are true for all spanning trees?

A. Contains exactly $|V| - 1$ edges
B. The removal of any edge disconnects it
C. The addition of any edge creates a cycle
D. All of the above

Cayley's theorem.

The complete graph on n nodes has n^{n-2} spanning trees.

[Brute-force forbidden]

Brainstorming

Greedy strategies for computing an MST?

Greedy algorithms for MST

- **Kruskal's.** Start with $T = \emptyset$. Insert edges in ascending order of weights, unless it creates a cycle.
- **Reverse-Delete.** Start with $T = E$. Remove edges in descending order of weights, unless it disconnects T.

Edge-driven

- **Prim's.** Start with some node s. Grow a tree T from s outward. Add v to T such that $w(u, v)$ cheapest and $u \in T$.

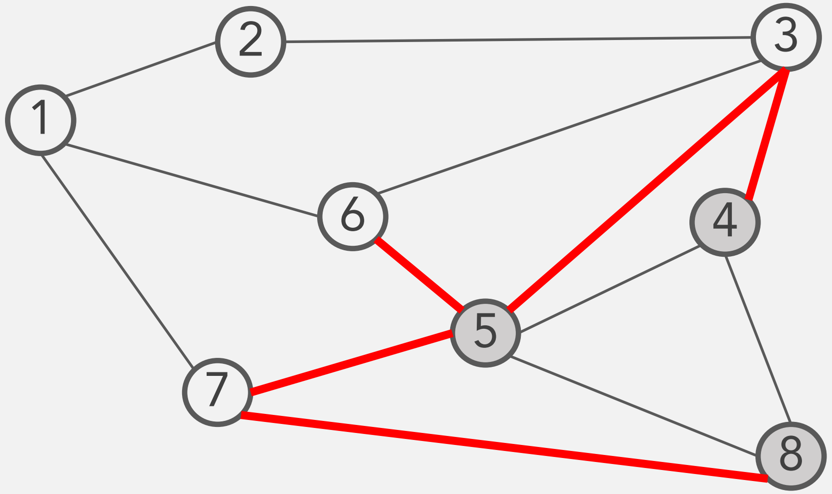
Node-driven

Sounds familiar? Dijkstra's?

☺ In this extremely lucky case, all of them work! But correctness proofs are non-trivial. We need the following tools to prove them.

Cycles and cuts

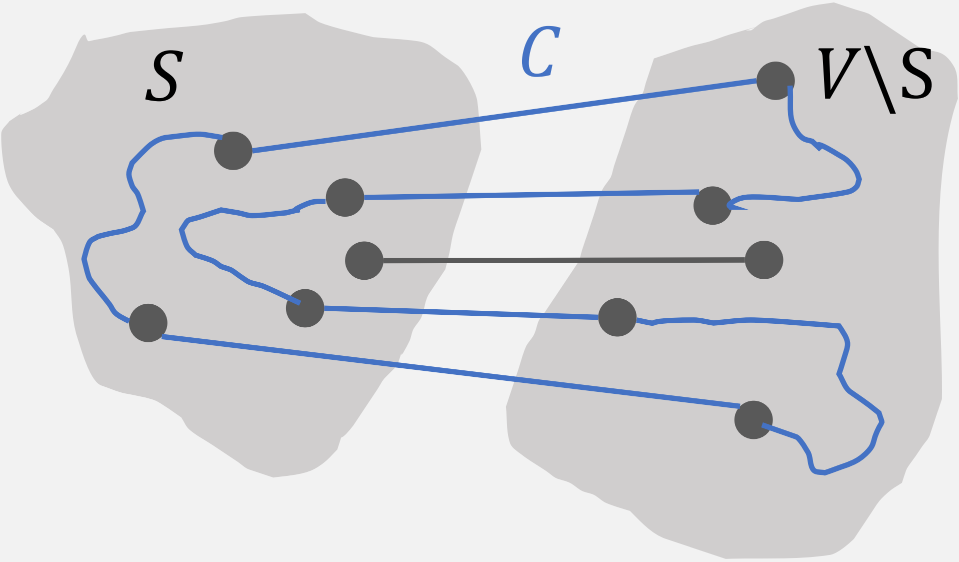
- Cycle: set of edges of form $(a, b), (b, c), \dots, (z, a)$
- Cut: a subset of nodes $S \subseteq V$
- Cutset $D(S)$: subset of edges with exactly one endpoint in S.



Ex. Cut $S = \{4,5,8\}$
Cutset $D(S) = \{(4,3), (5,7), (5,6), (7,8)\}$

Observation: cycle-cut intersection

Claim*. A *cycle* & a *cutset* intersect in an even number of edges.



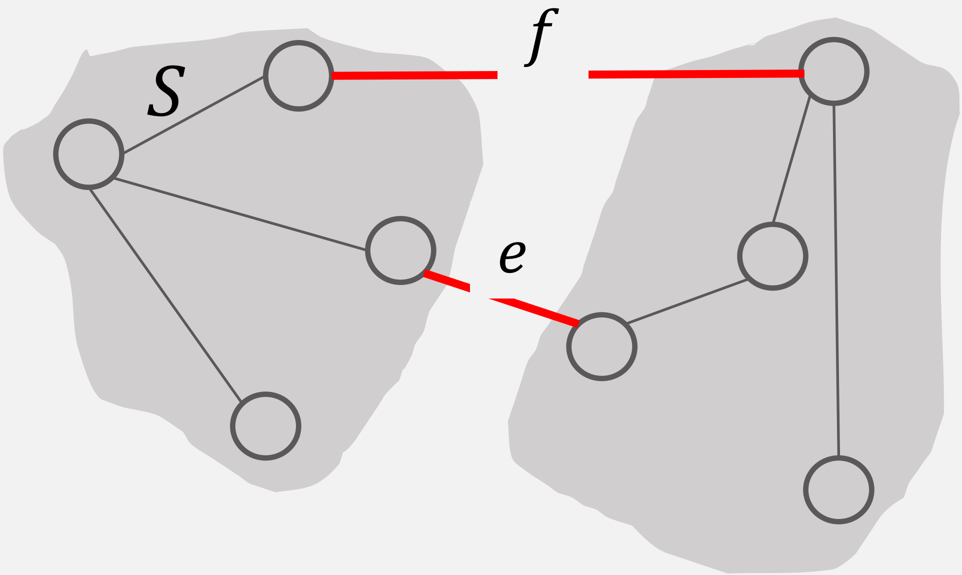
- Proof. A cycle has to leave & enter the cut the same number of times.

Cut Property

Cut property. Let S be a subset of nodes. Let e be the min weight edge with exactly one endpoint in S. Then any MST T contains e.

- Proof. (exchange argument)
  - Suppose e does not belong to T
  - Adding e to T creates a cycle C
  - Edge e is both in C and in the cutset $D(S)$

  ➔ there exists another edge, say f, that is in both C and D. [Claim*]
  - $T' \coloneqq T \cup \{e\} - \{f\}$ is also a spanning tree
  - $w_e < w_f$ ➔ $w(T') < w(T)$. Contradiction!

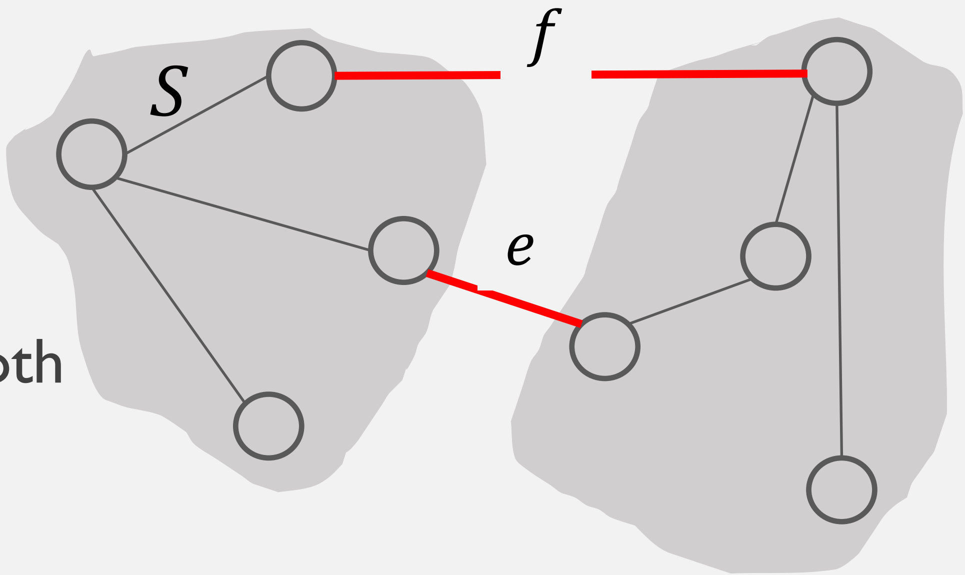


Cycle property

Cycle property. Let C be a cycle, and let f be the max weight edge in C. Then any MST T does not contain f.

- Proof. (exchange argument)
  - Suppose f belongs to T
  - Deleting f creates a cut S
  - Edge f is both in C and in the cutset $D(S)$

  ➔ there exists another edge, say e, that is in both C and D.
  - $T' := T \cup \{e\} - \{f\}$ is also a spanning tree
  - $w_e < w_f$ ➔ $w(T') < w(T)$. Contradiction!



F, 10/11/19

Fall'19 CSCE 629

Analysis of Algorithms

Fang Song

Texas A&M U

Lecture 17

- Minimum spanning tree

Credit: based on slides by A. Smith & K. Wayne

Pop quiz 2

Let G be a connected undirected graph w. distinct edge weights.

- Let e be the cheapest edge in G. Some MST of G contains e?

True. By cut property

- Let e be the most expensive edge in G. No MST of G contains e?

False. Counterexample: if G is a tree, all its edges are in the MST

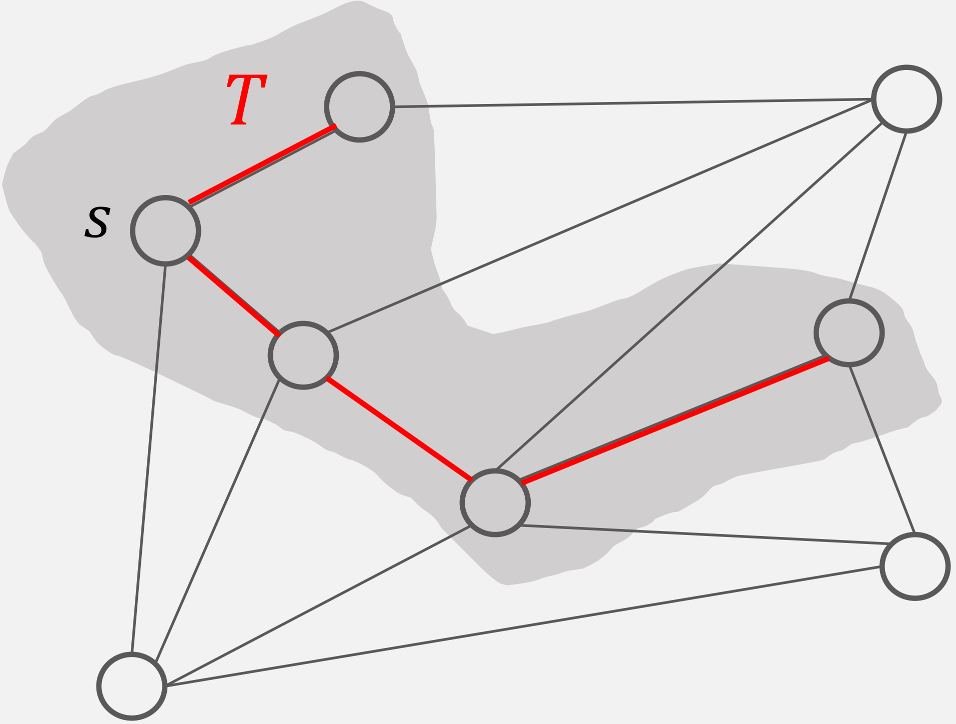
Prim's algorithm: correctness

Prim's algorithm [Janik 1930, Prim 1959]

Start with some node s. Grow a tree T from s outward. Add v to T such that $w(u, v)$ cheapest and $u \in T$.

- Correctness
  - Apply cut property to T
  - When edge weights are distinct, every edge that is added must be in the MST

  ➔ Prim's algorithm outputs the MST



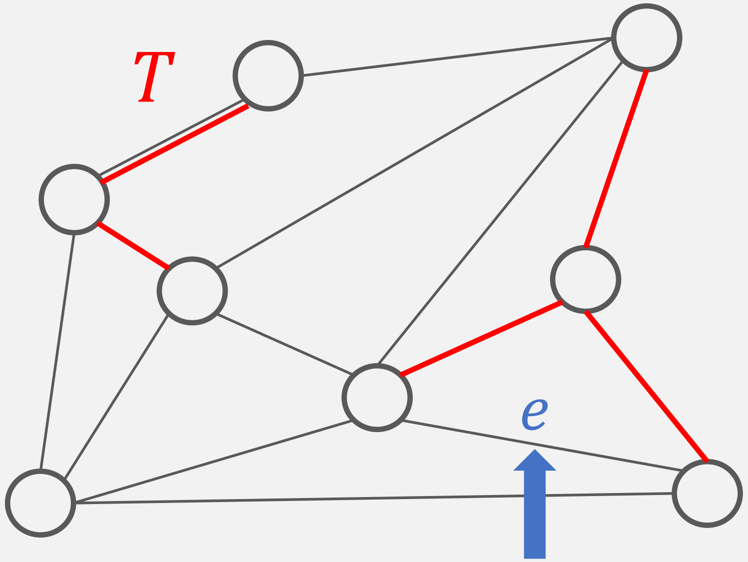
Kruskal's algorithm: correctness

Kruskal's algorithm [Kruskal 1956]

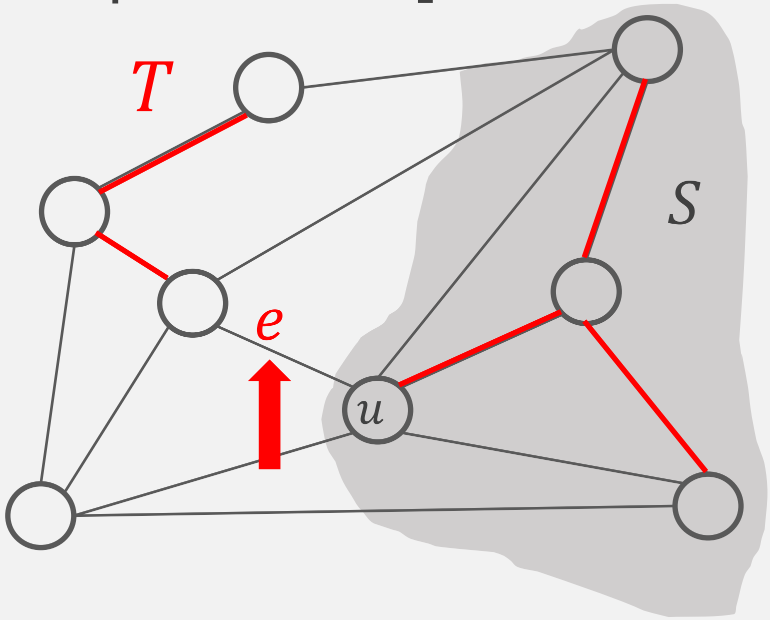
Start with $T = \emptyset$. Insert edges in ascending order of weights, unless it creates a cycle.

- Correctness

Case 1. If adding e to T creates a cycle, discard e according to cycle property.

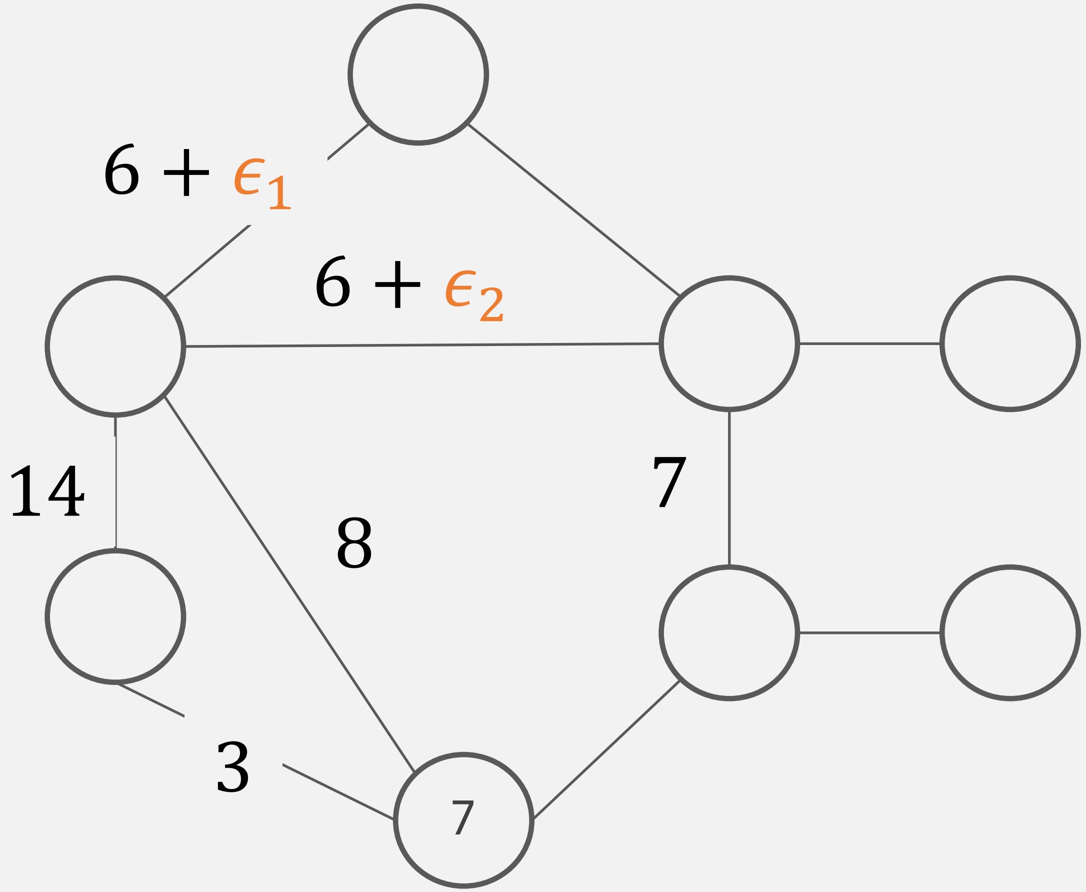


Case 2. Adding $e = (u, v)$ to T according to cut property. [S = connected component of u]



Removing distinct weight assumption

- Perturbation argument



$$\sum \epsilon_i \ll |w(T') - w(T)|$$

Implementing Prim's

- Maintain $V - T$ as a priority queue. [as in Dijkstra's]
- $Key(v)$: weight of the least-weight edge connecting it to a vertex in T

$\boldsymbol{Prim}(G, \{w_e\})$
1. $Q \leftarrow MakeQueue(V)$
2. $key[s] \leftarrow 0$ for an $s \in V$; $key[v] \leftarrow \infty$ otherwise
3. While Q not empty
   - $u \leftarrow$ Delete−min(Q) // add u to T
   - For $v \in Adj[u]$ // consider neighbors of u
     - If $v \in Q$ and $w(u, v) < key[v]$
       - $key[v] \leftarrow w(u, v)$
       - Change−key(v)
       - $parent(v) \leftarrow u$
4. Return $T \leftarrow \{(v, parent(v))\}$

$O(n)$

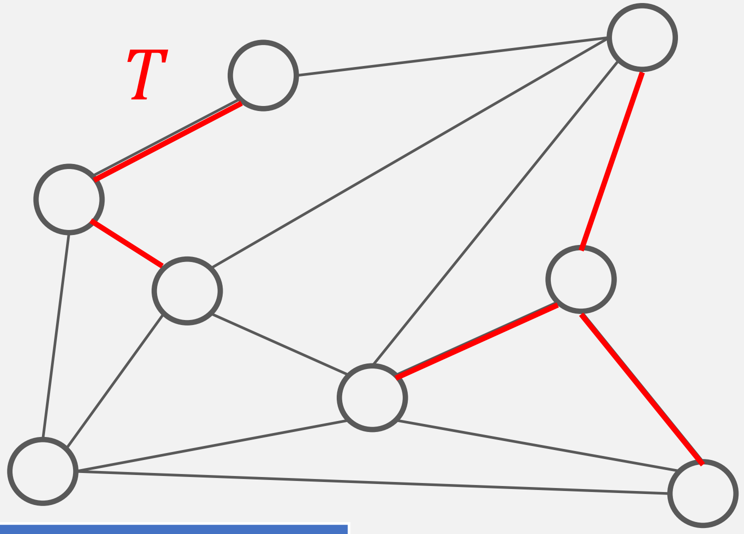
n Delete-min
m Change-key

Time: $O((m+n)\log n)$

Same as Dijkstra's

Implementing Kruskal's

- Disjoint-set (aka Union-Find) data structure
  - $\text{Make-Set}(x)$: create a singleton set containing x
  - $\text{Find-Set}(x)$: return the "name" of the unique set containing x
  - $\text{Union}(x, y)$: merge the sets containing x and y respectively



	Linked list	Balanced tree
Find (worst-case)	$\Theta(1)$	$\Theta(\log n)$
Union (worst-case)	$\Theta(n)$	$\Theta(\log n)$
Amortized analysis: k unions and k finds, starting from singleton	$\Theta(k \log k)$	$\Theta(k \log k)$

Implementing Kruskal's

Kruskal$(G, \{w_e\})$
// $T \leftarrow \emptyset$; sort m edges so that $w(e_1) \leq w(e_2) \leq \cdots$ $\}$ $O(m \log m)$
1. For $v \in V$, MakeSet(v)
2. For $i = 1, \ldots, m$
   $(u, v) \leftarrow e_i$ // ith cheapest edge
   If Find−Set$(u) \neq$ Find−Set(v) // same component?
   $T \leftarrow T \cup \{e_i\}$
   Union−Set(u, v)
3. Return T

$2m$ Find-Set
n Union-Set

Running time: $O(m \log m + n \log n) = O(m \log n)$

Warning on Greedy algorithms

Correctness

Greedy algorithms are tempting but rarely work!
Only with care (as sanity check or last resort)

"You will not receive any credit for any greedy algorithm, on any homework or exam, even if the algorithm is correct, without a formal proof of correctness." –Erickson

I second, and we adopt this policy in this class too!