6.046	Lecture 12	Mar. 19, 2015

TODAY: Greedy algorithms
& Minimum Spanning Tree (MST)
- MST problem
- optimal substructure
- greedy-choice property
- Prim's algorithm
- Kruskal's algorithm

Recall: [Lecture 1]

Greedy algorithm: repeatedly make locally best choice/decision, ignoring effect on future
- saw greedy algorithm for scheduling problem
- Dijkstra's algorithm also ≈ greedy (cf. Bellman-Ford: incremental improvement)
- today: greedy algorithm for graph problem

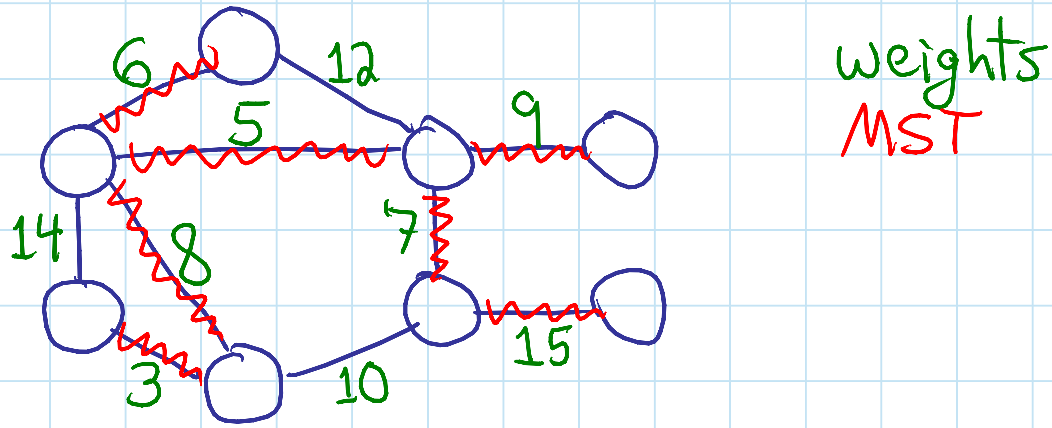
Tree = connected graph with no cycles
Spanning tree of graph
= subset of graph's edges that form a tree spanning (containing) all vertices

Minimum spanning tree (MST) problem:

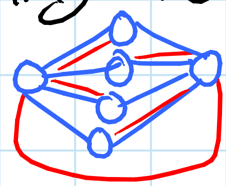
given a graph $G=(V,E)$ & edge weights $w: E \to \mathbb{R}$, find spanning tree $T \subseteq E$ of minimum weight:

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

Example:



Naïve algorithm: check all spanning trees
- exponential time ¨



Greedy properties: problems amenable to greedy algorithms usually satisfy:

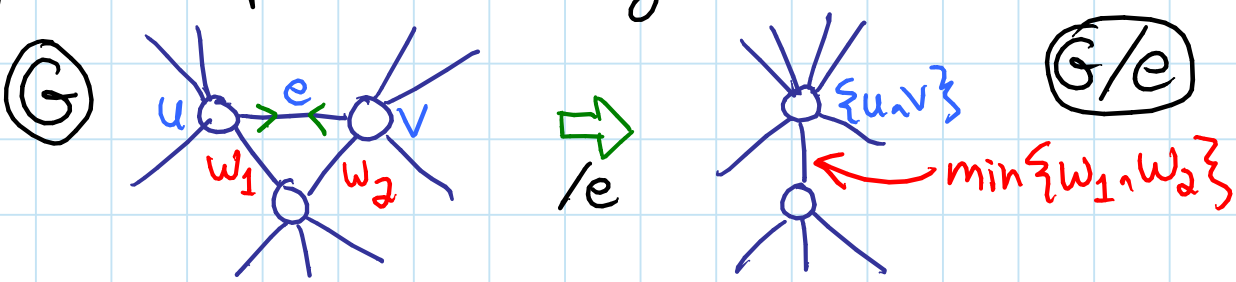
1. optimal substructure: optimal solution to problem incorporate optimal solution(s) to subproblem(s)
   - essentially dynamic programming

2. greedy-choice property: locally optimal choices lead to globally optimal solution

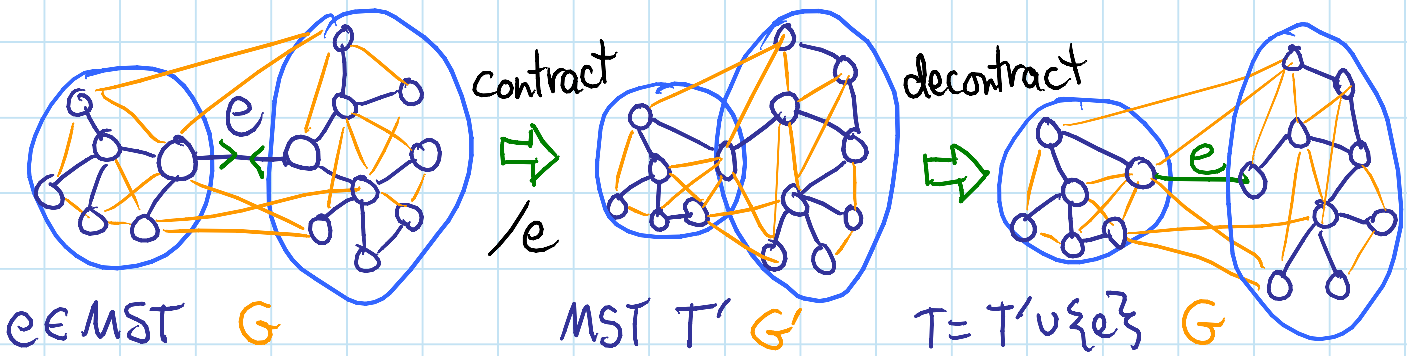
Optimal substructure for MST:

if $e=(u,v)$ is an edge of some MST of $G=(V,E,w)$:

- contract edge e: merge vertices u & v
- if we get multiple copies of an edge, just keep lowest weight:



- if T' is an MST of $G' = G/e$ then $T = \underbrace{T'}_{\text{remap edges to decontract } \{u,v\} \to u \,\&\, v} \cup \{e\}$ is an MST of G



Proof:

- let T^* be an MST of G containing edge e

$\Rightarrow T^*/e$ is a spanning tree of G'

- T' is an MST of G'

$\Rightarrow w(T') \le w(T^*/e)$

$\Rightarrow w(T) = w(T') + w(e) \le w(T^*/e) + w(e) = w(T^*)$. $\square$

Dynamic program attempt:

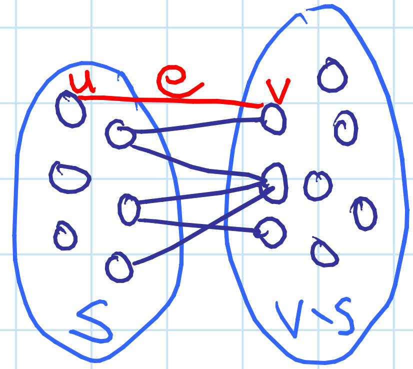
- guess an edge to put in MST
- contract to get new subproblem
- recurse
- decontract & add e

but # subproblems is exponential

greedy technique will make this polynomial!

Greedy-choice property for MST:

for any cut $(S, V \setminus S)$
in graph $G=(V, E, w)$,
any least-weight crossing edge
$e=\{u,v\}$ with $u \in S$ & $v \notin S$
is in some MST of G



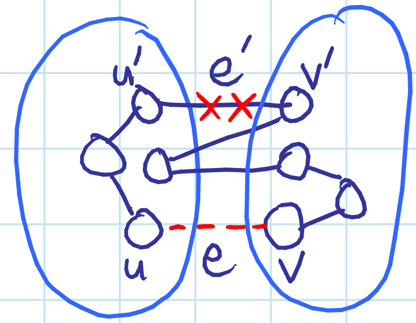
Proof: cut & paste argument ← typical for greedy proofs

- Consider an MST T of G
- T has a path from u to v
- $u \in S$ & $v \notin S$, so the path has some edge $e'=\{u',v'\}$ with $u' \in S$ & $v' \notin S$
- then $T' = T \setminus \{e'\} \cup \{e\}$ is a spanning tree of G & $w(T') = w(T) - w(e') + w(e)$
- but e is a least-weight edge crossing $(S, V \setminus S)$

$\Rightarrow w(e) \leq w(e')$

$\Rightarrow w(T') \leq w(T)$

$\Rightarrow T'$ is a MST too. $\square$



* modification only touches edge(s) crossing $(S, V \setminus S)$

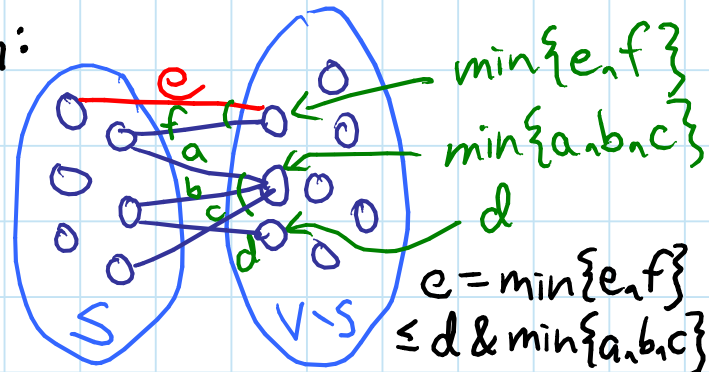
Two algorithms based on different choices of cut $(S, V \setminus S)$.

Prim's algorithm: start with $|S| = 1$ & grow from there

- maintain priority queue Q on $V \setminus S$, where $v.key = \min\{w(u,v) \mid u \in S\}$
- initially Q stores V ($S = \emptyset$)
  - $s.key = \emptyset$ for arbitrary start vertex $s \in V$
  - for $v \in V \setminus \{s\}$: $v.key = \infty$
- until Q empty:
  - $u = \text{Extract-Min}(Q)$ (add u to S)
  - for $v \in Adj[u]$:
    - if $v \in Q$ ($v \notin S$) & $w(u,v) < v.key$:
      - $v.key = w(u,v)$ ← Decrease-Key
      - $v.parent = u$
- return $\{\{v, v.parent\} \mid v \in V \setminus \{s\}\}$

Correctness:

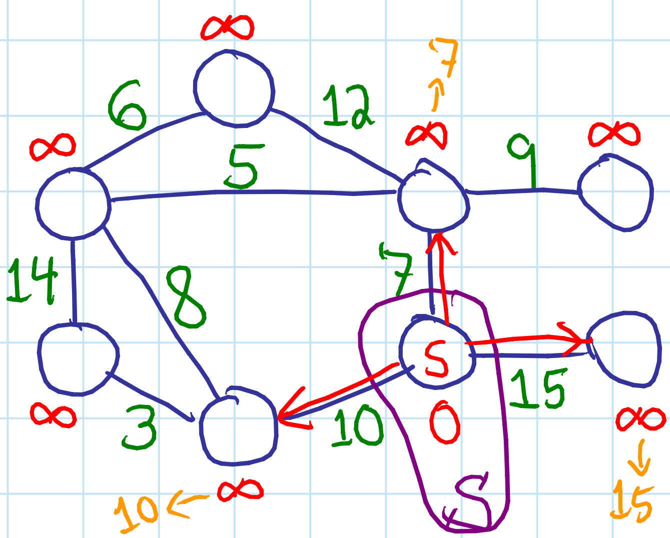
- invariant: $v \notin S \Rightarrow v.key = \min\{w(u,v) \mid u \in S\}$
- invariant: tree T_S within S $\subseteq$ MST of G
  - assume by induction: MST $T^* \supseteq T_S$
  - $S \to S' = S \cup \{e\}$ where e is a least-weight edge crossing cut $(S, V \setminus S)$



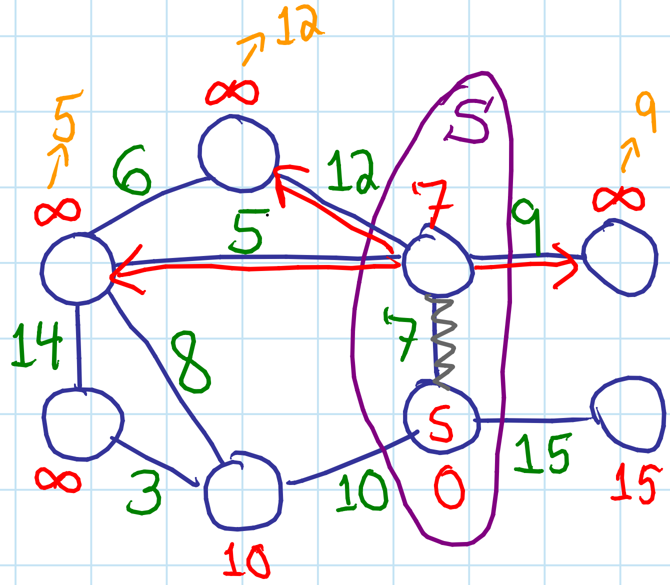
  - greedy cut & paste $\Rightarrow$ can modify T^* to include e without removing T_S ← don't cross cut

  $\Rightarrow$ new $T^* \supseteq T_{S'} = T_S \cup \{e\}$

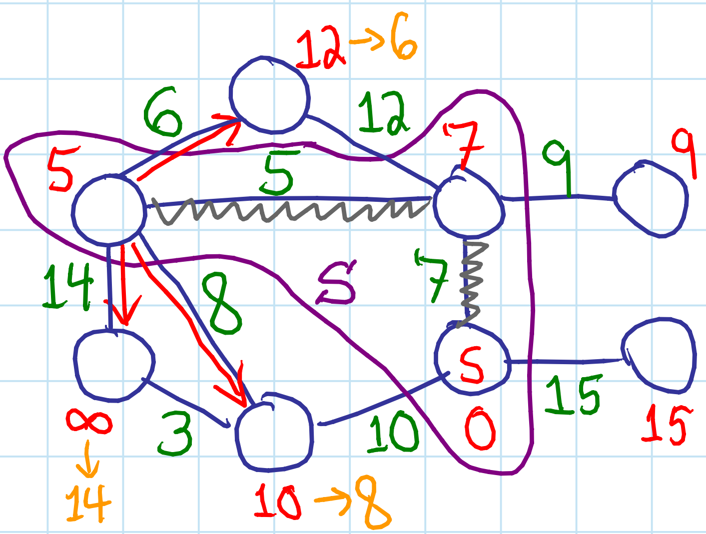
Example:



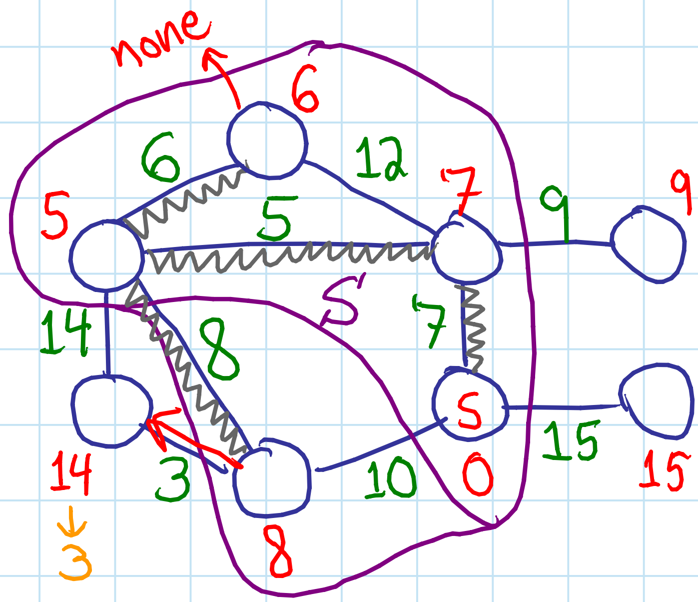
∞
6
12
∞ → 7
9
∞
∞
5
14
8
7
s
15
∞
3
10
0
∞ → 15
10 ← ∞
S



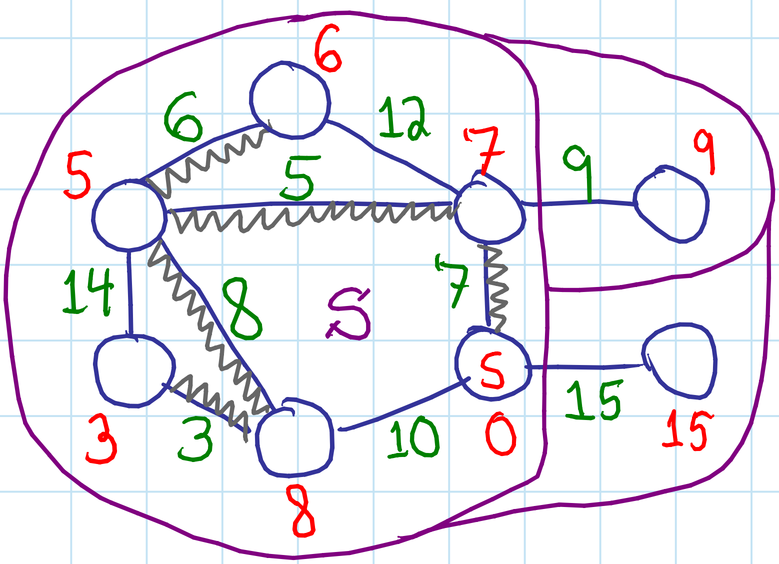
∞ → 12
∞ → 5
6
12
S
7
9
∞ → 9
5
14
7
8
s
15
∞
3
10
0
15
10



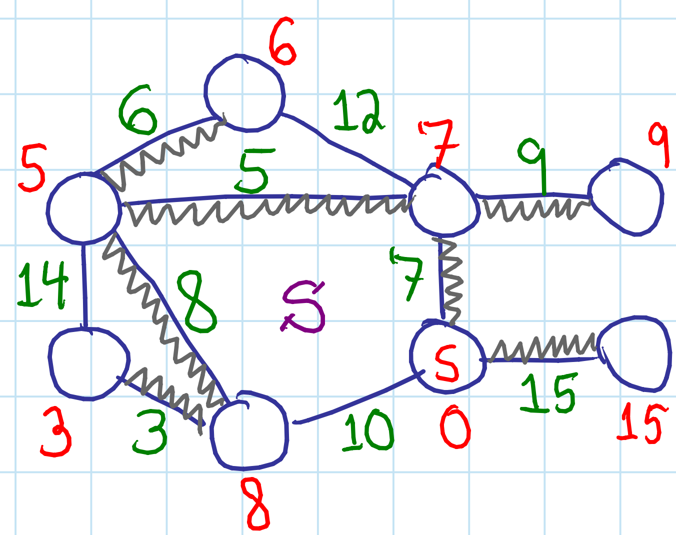
12 → 6
6
12
7
9
9
5
5
14
8
S
7
s
15
15
∞ → 14
3
10
0
10 → 8



none
6
6
12
7
9
9
5
5
S
14
8
7
s
15
15
14 → 3
3
10
0
8



6
6
12
7
9
9
5
5
14
8
S
7
s
15
15
3
3
10
0
8



6
6
12
7
9
9
5
5
14
8
S
7
s
15
15
3
3
10
0
8

Time: $\Theta(V) \cdot T_{\text{Extract-Min}} + \underbrace{\Theta(E)}_{} \cdot T_{\text{Decrease-Key}}$

$\sum_v |Adj[v]| = \sum_v \deg(v) = 2 \cdot |E|$ (Handshaking Lemma)

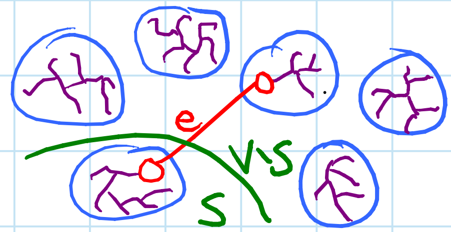
priority_queue	$T_{\text{Extract-Min}}$	$T_{\text{Decr.-Key}}$	total
- array (nothing)	$O(V)$	$O(1)$	$O(V^2)$
- binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
- Fibonacci heap [CLRS ch. 19]	$O(\lg V)$ amortized	$O(1)$ amortized	$O(E + V \lg V)$

Kruskal's algorithm: take globally lowest-weight edge & contract

- maintain connected components in MST-so-far T in Union-Find structure [Recitation 3]
- $T = \emptyset$ ← will become MST
- for $v \in V$: Make-Set(v) ← initially, 1 vertex/comp.
- sort E by w
- for $e = (u, v) \in E$ (in increasing weight order):
  - if Find-Set(u) $\neq$ Find-Set(v): ← different components ⇒ e won't make a cycle
    - add e to T
    - Union(u, v)

Correctness: invariant: tree T so far $\subseteq$ MST T^*

- assume by induction $T \subseteq T^*$
- when adding e between components C_1 & C_2: use greedy-choice property on cut $(C_1, V \setminus C_2)$



Time: $\underbrace{T_{sort}(E)}_{\substack{O(E \lg E) \\ O(E) \text{ e.g. if weights} \\ \text{are integers} \in [0, E^{O(1)}]}} + \underbrace{\Theta(V) \cdot T_{makeset}}_{\text{tiny}} + \Theta(E) \cdot \underbrace{(T_{find} + T_{union})}_{O(\alpha(V)) \text{ am.}}$

~ then can beat Prim

Best MST algorithm: [Karger, Klein, Tarjan 1993]
$O(V+E)$ expected time (randomized)