Greedy Algorithms

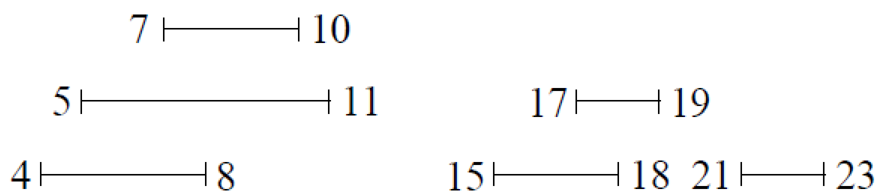
- Algorithm design technique
  - Usually more efficient than DP and DAC
  - Constructs optimal solution incrementally, by repeatedly choosing what looks promising *right now*
- Problem must satisfy the *greedy choice property*:

  a *locally optimal choice* is guaranteed to lead to some *globally optimal solution.*

Activity Selection

Input: set $S = \{a_1, \ldots, a_n\}$ of n activities requesting *exclusive* access to a common resource

$$a_i = [s_i, f_i)$$

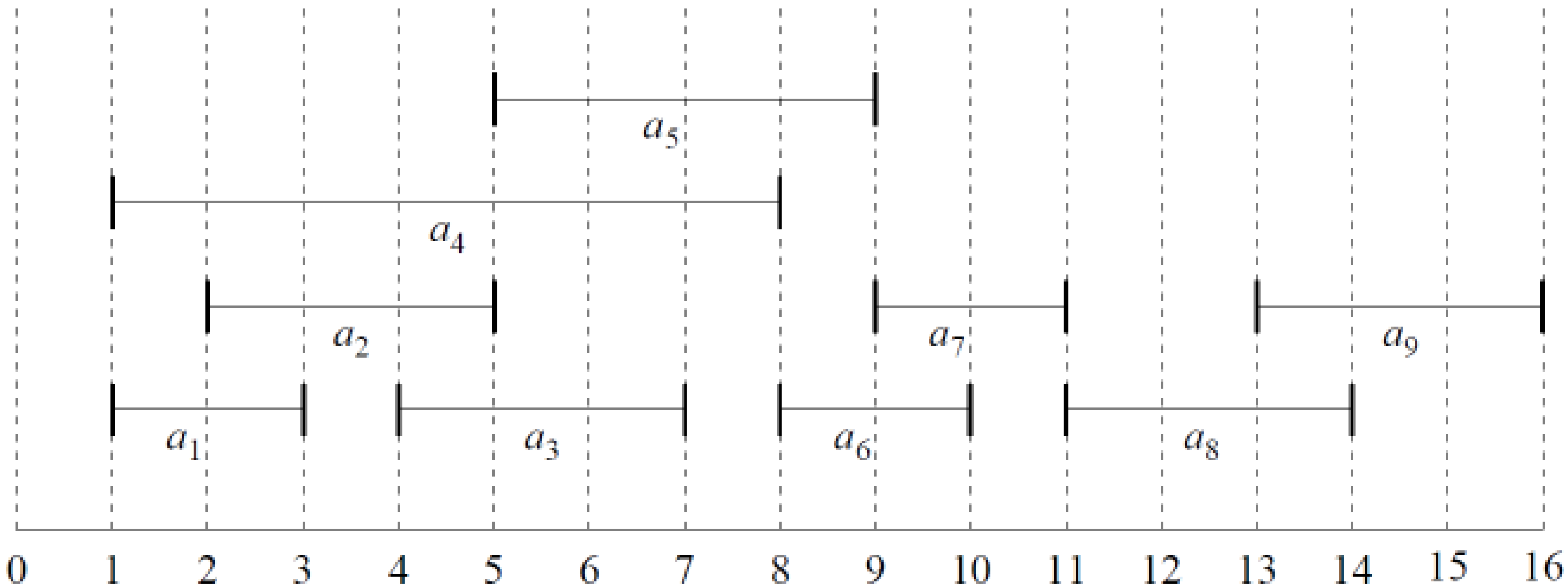


Output: a largest cardinality set A of non-overlapping activities

Example: schedule use of a room to maximize the number of events that use it

Example

i	1	2	3	4	5	6	7	8	9
s_i	1	2	4	1	5	8	9	11	13
f_i	3	5	7	8	9	10	11	14	16



- Assume S sorted by finish time: $f_1 < f_2 < \ldots < f_n$

Optimal Substructure

- Let $S_{ij} = \{a_k \in S : f_i \leq s_k \leq f_k \leq s_j\}$ and let A_{ij} denote an optimal solution for S_{ij}
- Activities in S_{ij} are compatible with
  - activities that finish no later than f_i
  - activities that start no earlier than s_j
- $a_k \in A_{ij}$ generates 2 subproblems: S_{ik} and S_{kj}
- Then, $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$ (cut-and-paste argument)
- For convenience add sentinels $a_0 = (-\infty, -\infty)$ and $a_{n+1} = (\infty, \infty)$. Then $A = A_{0,n+1}$

DAC

- Since optimal solution A_{ij} must contain optimal solutions to subproblems S_{ik} and S_{kj}
  $\Rightarrow$ can consider DAC (and DP)

  $c[i, j]$ = size of optimal solution to S_{ij}

  $c[i, j] = c[i, k] + c[k, j] + 1$, but what k ?

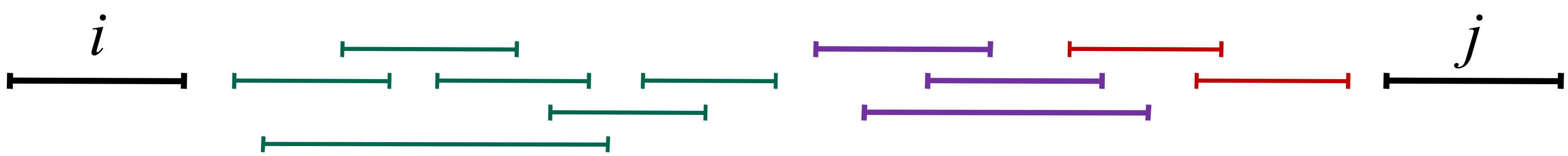
$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset \\ 1 + \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j]\} & \text{if } S_{ij} \neq \emptyset \end{cases}$$

- Can develop memoized DAC or bottom-up DP

Overlapping Subproblems

- Will generate many repeated problems when evaluating

$$c[i,j] = 1 + \max_{a_k \in S_{ij}} \{c[i,k] + c[k,j]\}$$



- The green intervals become the left subproblem of any of the purple intervals

Hallmark of Greedy Algorithms

Greedy Choice Property. A locally optimal choice leads to a globally optimal solution.

- Identify a simple heuristic to make the local choice.
- Prove that the choices made are part of some optimal solution.

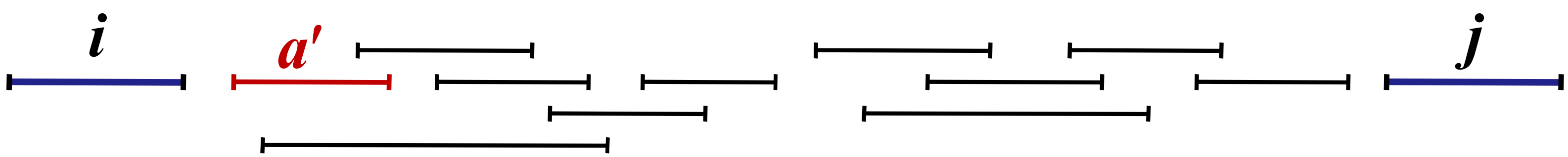
In Activity Selection, choose an activity a_k to add to $A = A_{0,n+1}$ *before* solving the ensuing subproblems.

Some greedy choices: Shortest activity, activity that ends first? activity that ends last, activity that overlaps the fewest number of other activities, etc.

The Greedy Choice

- Among all activities in subproblem S_{ij} choose the one (a') that ends first
- No activity of S_{ij} ends before a' starts

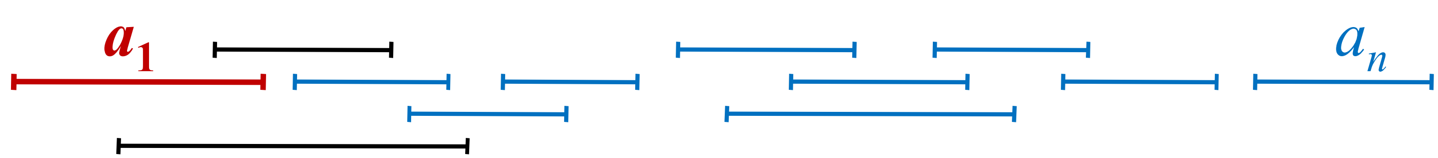
  $\Rightarrow$ Choosing a' eliminates subproblem S_{ik} and leaves S_{kj} only



- Making the greedy choice a_1 initially reduces the problem $S = S_{0,n+1}$ to problem $S_{1,n+1}$

The Greedy Choice…

- Since we only have one subproblem, we can simplify the notation: $S_k = \{a_i : s_i \geq f_k\}$
- Greedy choice $a_1 \Rightarrow \boldsymbol{S_1}$ is the only subproblem left
- By optimal substructure: if a_1 is in A then A consists of a_1 plus all activities in an optimal solution to $\boldsymbol{S_1}$



- Need to prove that a_1 is part of some optimal solution

Theorem. If S_k is non-empty and a_m ends earliest in S_k, then a_m is part of *some* optimal solution to S_k

Proof.

Let A_k be an optimal solution to S_k and a_r have the earliest finish time of all activities in A_k

If $a_r = a_m$, we are done. A_k is the desired solution.

Otherwise, let $B_k = A_k - \{a_r\} \cup \{a_m\}$

The activities in B_k are disjoint

Since $|B_k| = |A_k|$, then B_k is optimal also, and includes a_m

Greedy DAC Solution

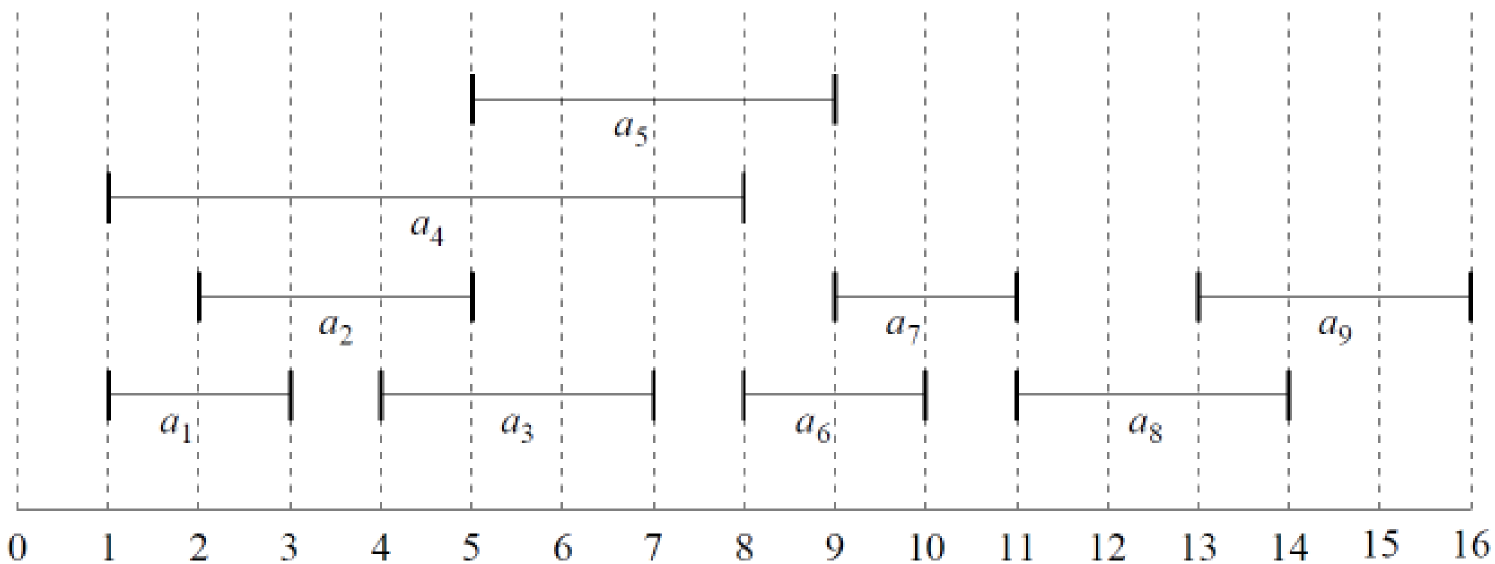
- Activities given by arrays $s[1{:}n]$, $f[1{:}n]$, sorted by f.
- Sentinel: activity a_0 has $f_0 = -\infty$, so that $S_0 = S$.
- Initial call: Activity-Selector(s, f, 0, n)

```
ACTIVITY-SELECTOR(s, f, k, n)
m ← k + 1
while m ≤ n and s[m] < f[k]
        do m ← m + 1
if m ≤ n
    then return {a_m} ∪ ACTIVITY-SELECTOR(s, f, m, n)
    else  return ∅
```

Example

i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
s		1	2	4	1	5	8	9	11	13
f	0	3	5	7	8	9	10	11	14	16

Subproblems: S_1 S_3 S_6 S_8



Greedy Incremental

$\text{ACTIVITY-SELECTOR}(s, f, n)$

$A \leftarrow \{a_1\}$
$k \leftarrow 1$
for $m \leftarrow 2$ **to** n
    do if $s[m] \geq f[k]$
        then $A \leftarrow A \cup \{a_m\}$
            $k \leftarrow m$
return A

- Time?

Greedy Strategy

- Go with the choice that seems best at the moment.
- What did we do for activity selection?
  1. Determine the optimal substructure
  2. Develop a recursive solution
  3. Show that if we make the greedy choice only one subproblem remains
  4. Prove that the greedy choice is always consistent with an optimal solution
  5. Develop a recursive or iterative algorithm

Which Method to Use?

- Dynamic Programming
  - Make a choice at each step
  - Choice depends on solution to subproblems
  - Solve subproblems first
  - Solve bottom up
- Greedy
  - Make a choice at each step
  - Make the choice *before* solving the subproblems (and then solve the remaining subproblems)
  - Solve top-down

Graphs (Review)

- Directed graph
  - Set V of n vertices
  - Set $E \subseteq V \times V$ of m edges
- Undirected graph
  - Set V of n vertices
  - Set $E = \{(i, j)\}$ of m edges
- Induced graphs (graph restricted to $U \subset V$)
- Graph representation
  - Adjacency matrix
  - Adjacency list

Minimum Spanning Tree

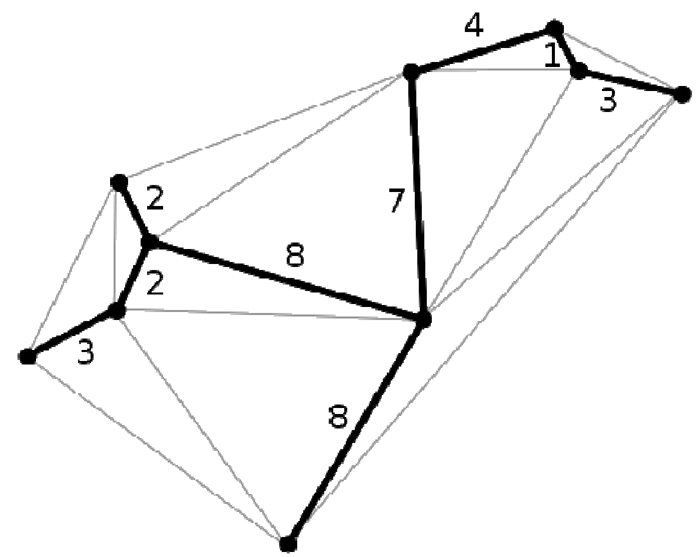
- A tree connecting all nodes of an undirected graph with minimal cost
- Many applications
  - Approximate solution of NP-hard problems
  - Basis of AT&T original billing system
  - Construction of LDPC codes
  - Image registration with Renyi entropy
  - Learning for real-time face verification
  - Reducing data storage for sequencing amino acids in proteins
  - Particle interactions in turbulent fluid flows
  - Autoconfig protocol to avoid cycles in a network

Minimum Spanning Trees

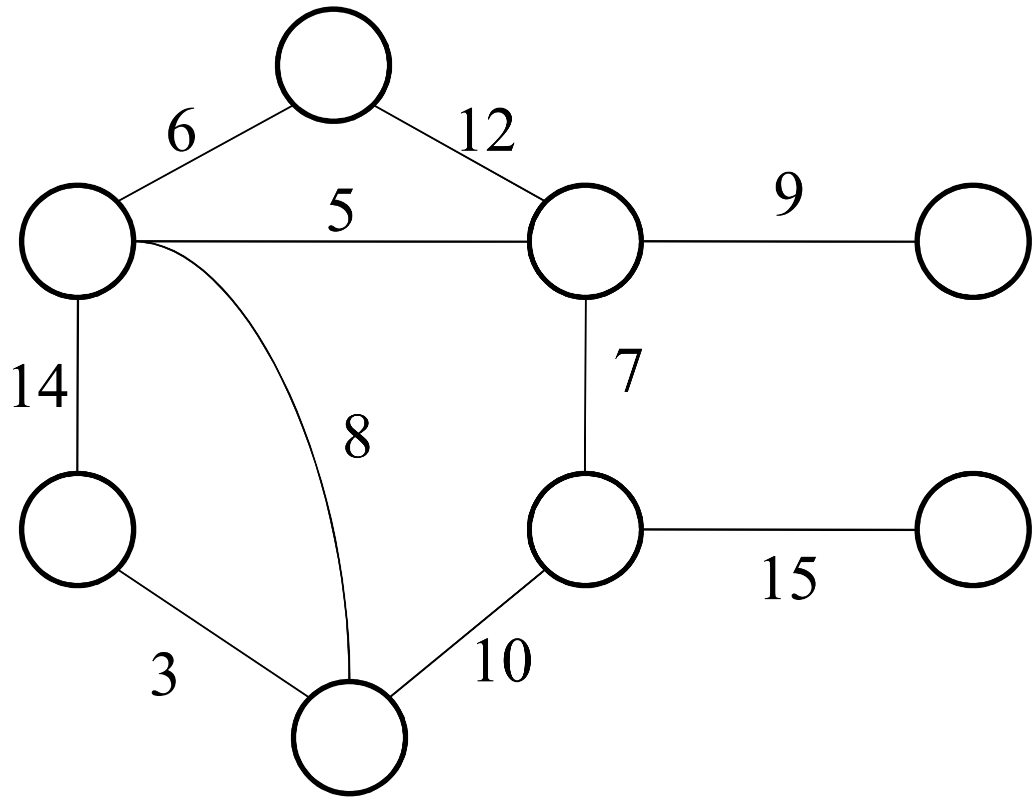
Input: undirected connected graph $G(V,E)$ with weights $w: E \rightarrow R^{+}$

Output: a tree T of minimum weight that spans V

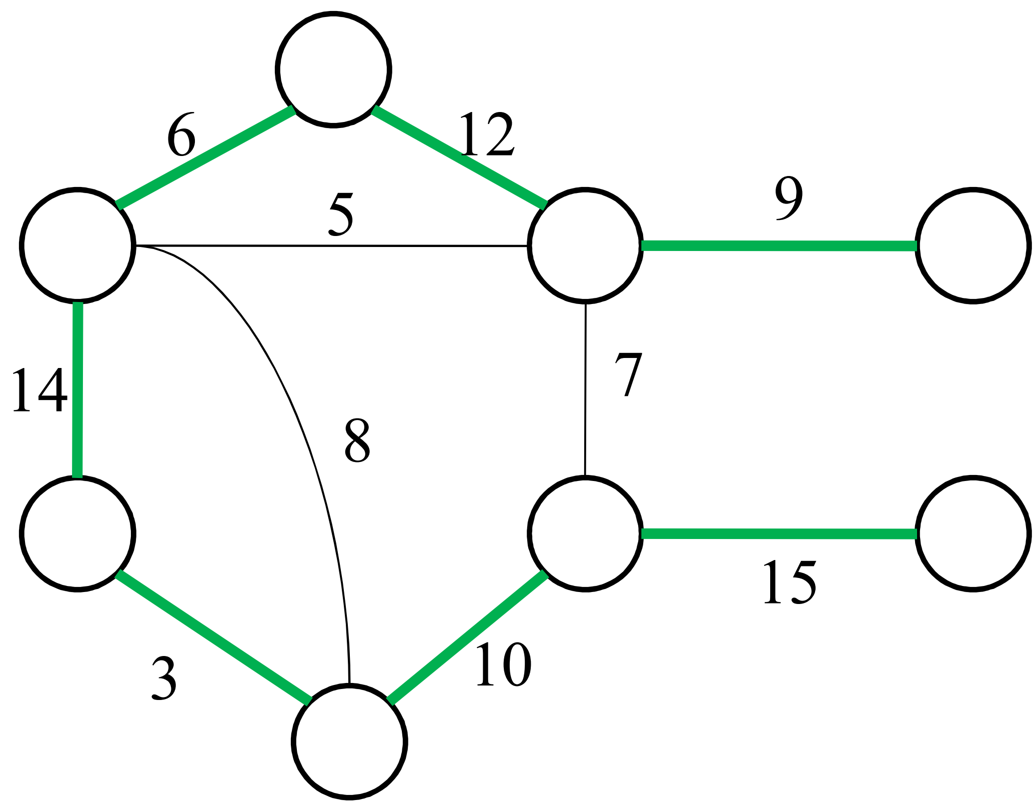
$$w(T) = \sum_{e \in T} w(e)$$



Example

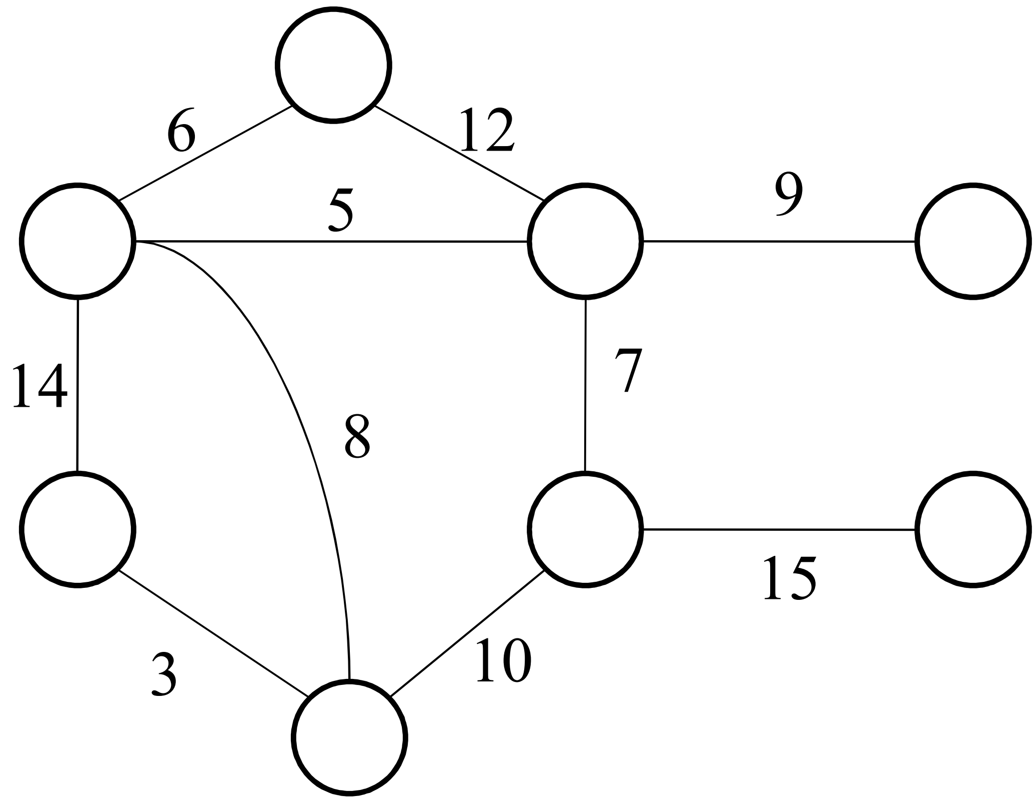


A Spanning Tree



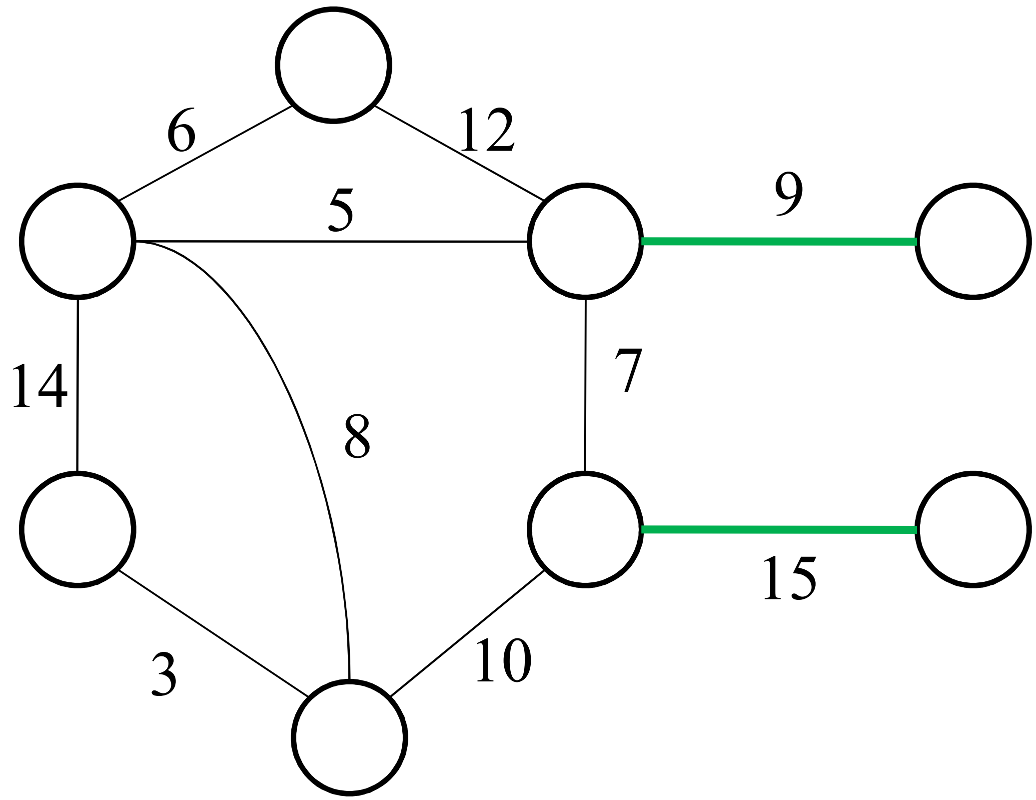
But is this minimal?

Example



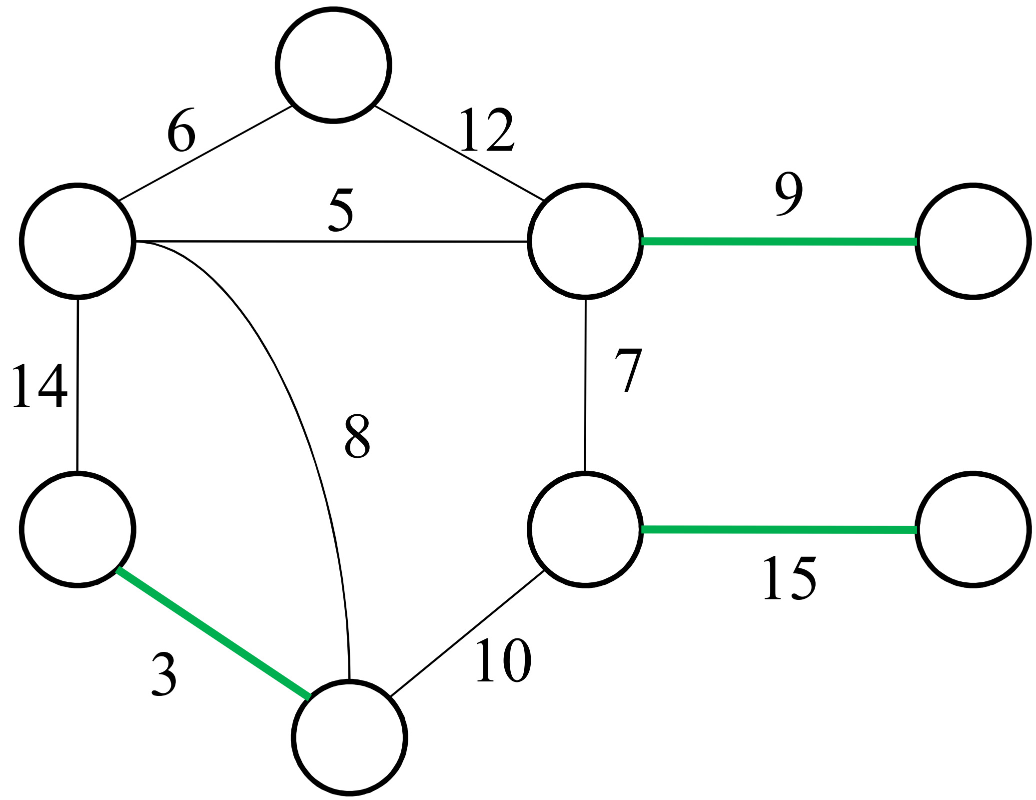
Can you argue that $9 \in T$?

Example



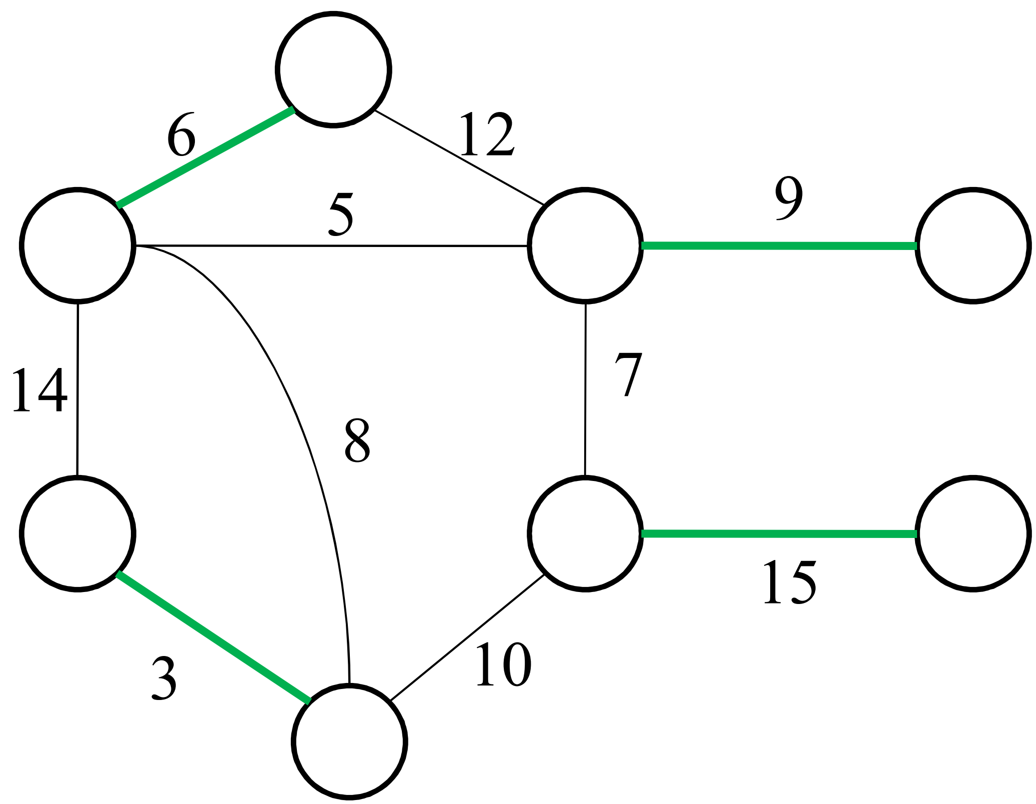
Can you argue that $3 \in T$?

Example



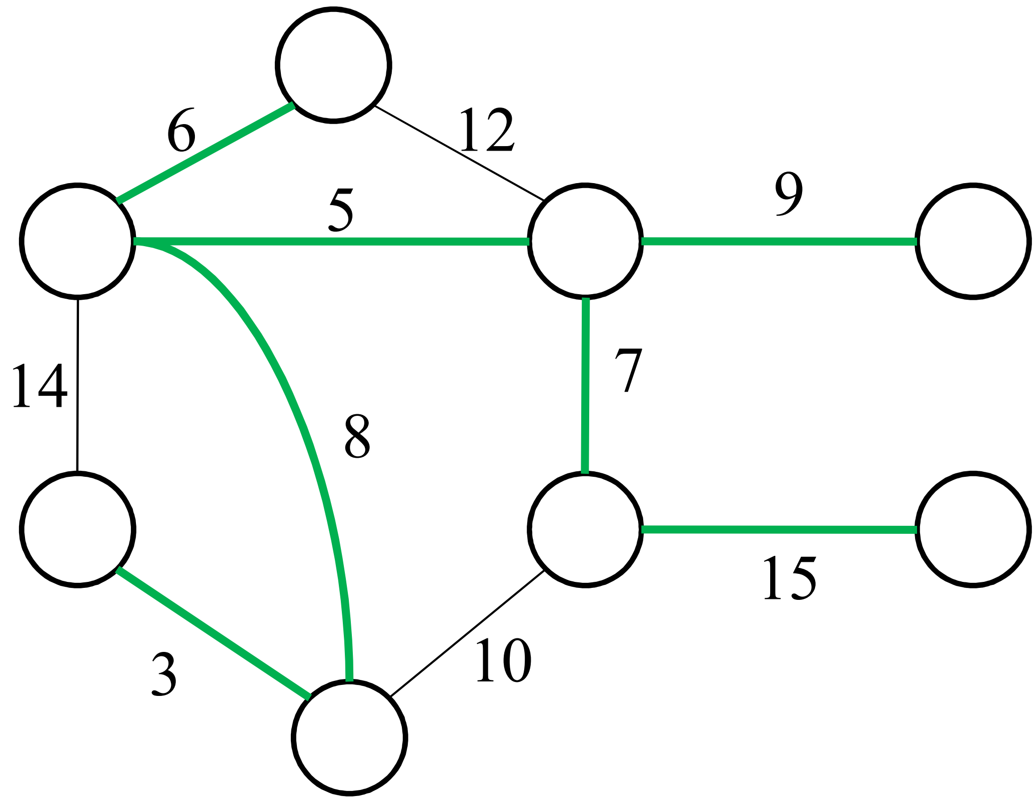
Can you argue that $6 \in T$?

Example



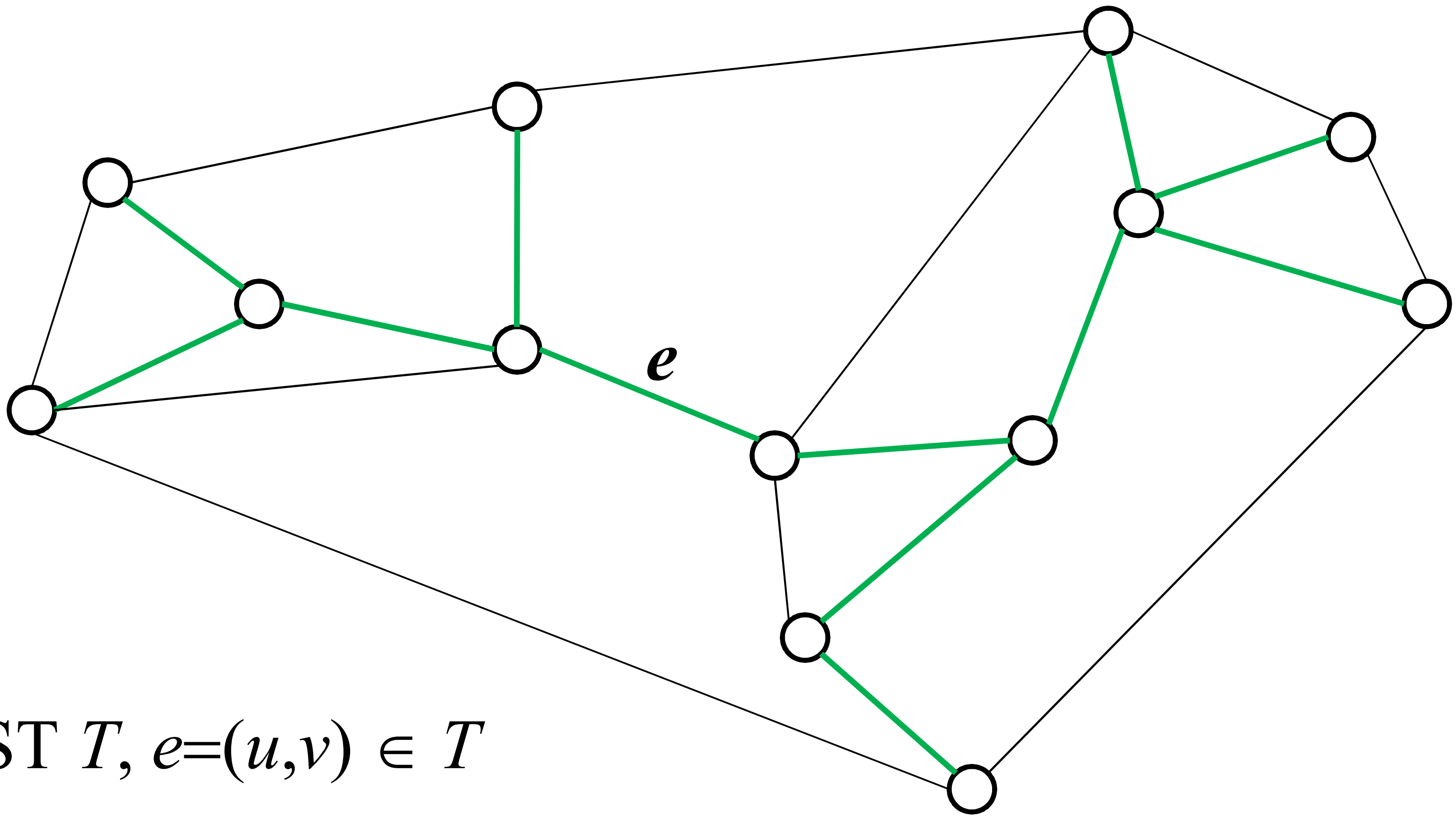
Any other edges ?

Example



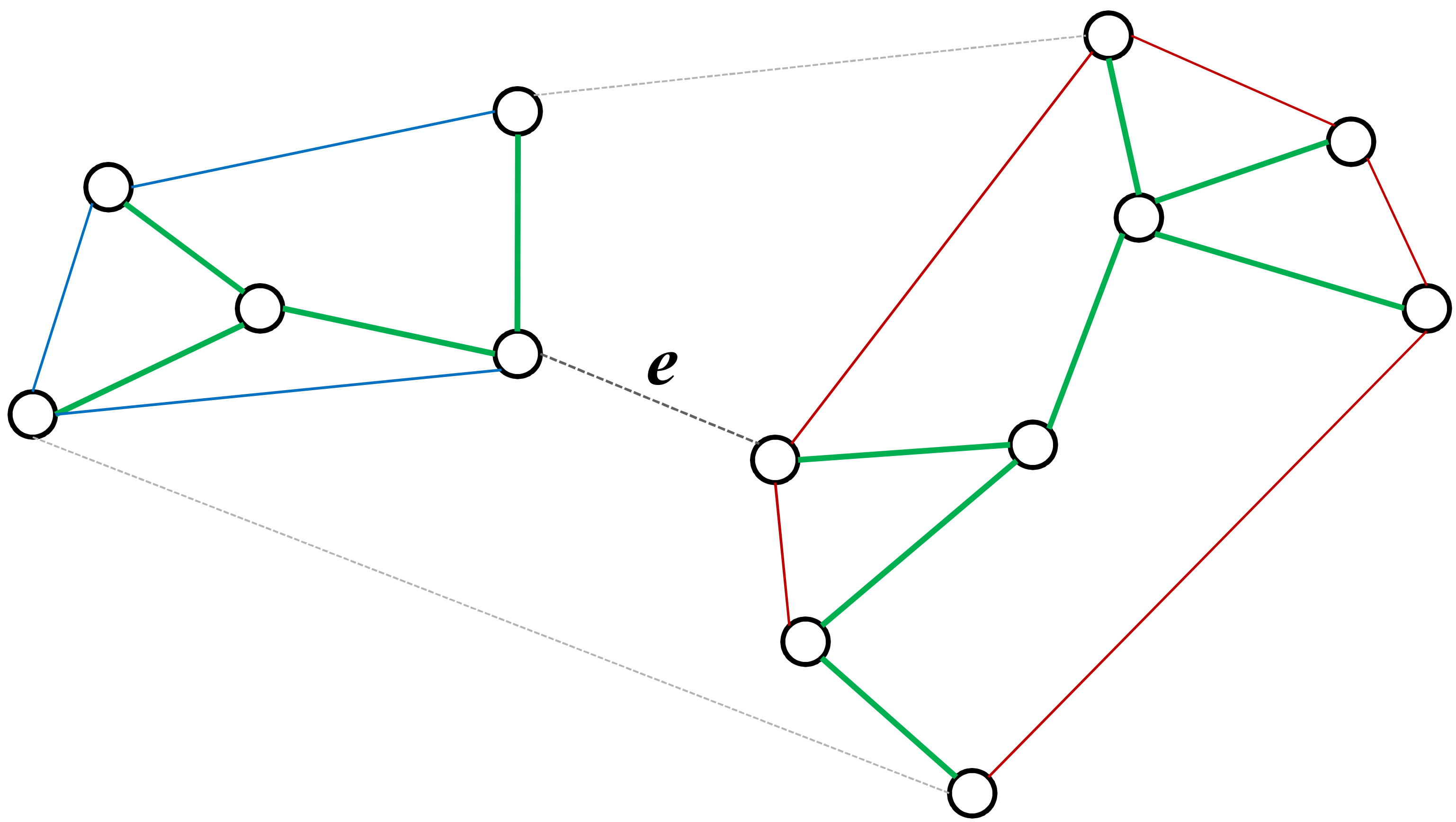
Any other edges ?

Optimal Substructure



- MST T, e=(u,v) $\in$ T
- Remove e
- Get subtrees T_1 and T_2 with vertex sets V_1 and V_2

Optimal Substructure



- V_i = vertices in T_i and $E_i = \{(u,v) \in E,\ u,v \in V_i\}$
- T_1 and T_2 are MSTs of $G(V_1,E_1)$ and $G(V_2,E_2)$, the subgraphs of G induced by V_1 and V_2

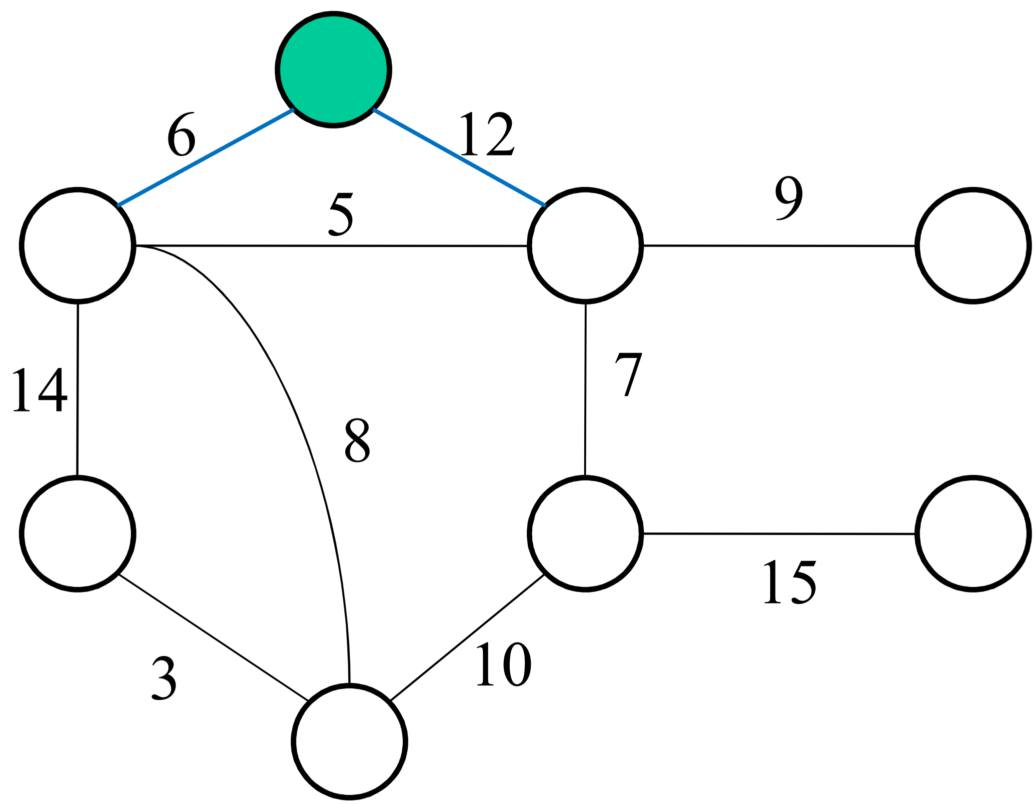
Hallmark of Greedy Algorithms

Greedy Choice Property. A locally optimal choice leads to a globally optimal solution.

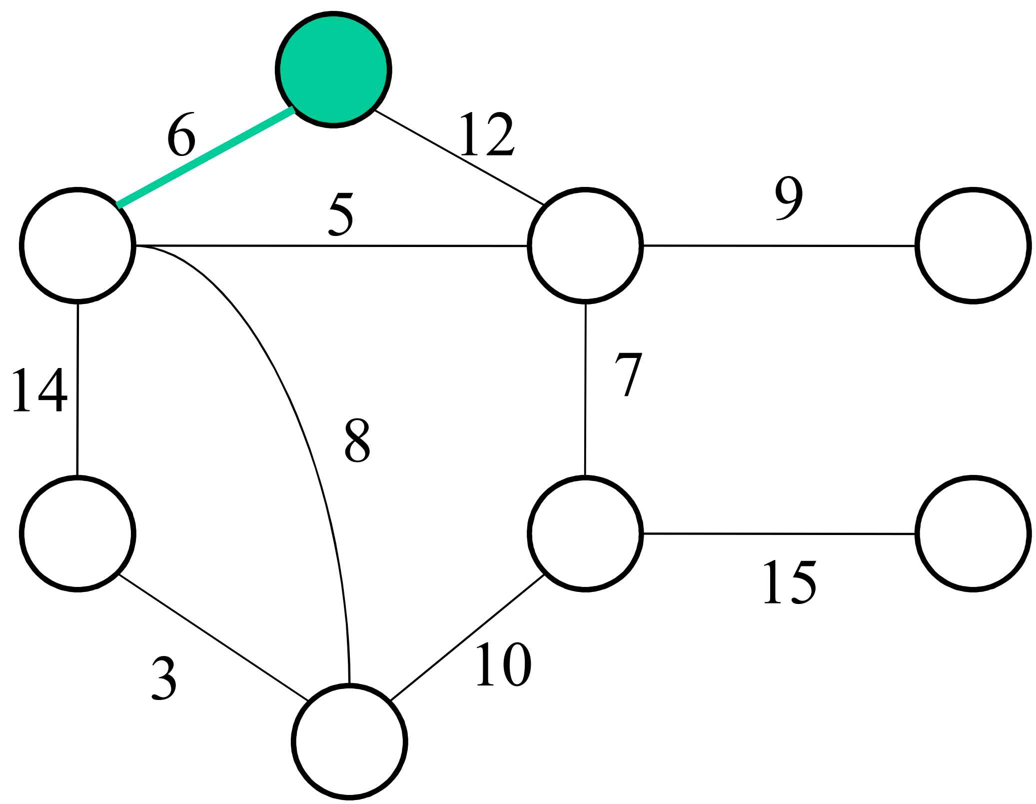
- Identify a simple to implement heuristic to make the local choices
- Prove that the choices made are part of some optimal solution.

Theorem. Let T be a MST of $G(V, E)$ and $A \subset V$. If $e = (u,v) \in E$ is the minimum weight edge connecting A to $V - A$ then $e \in T$.

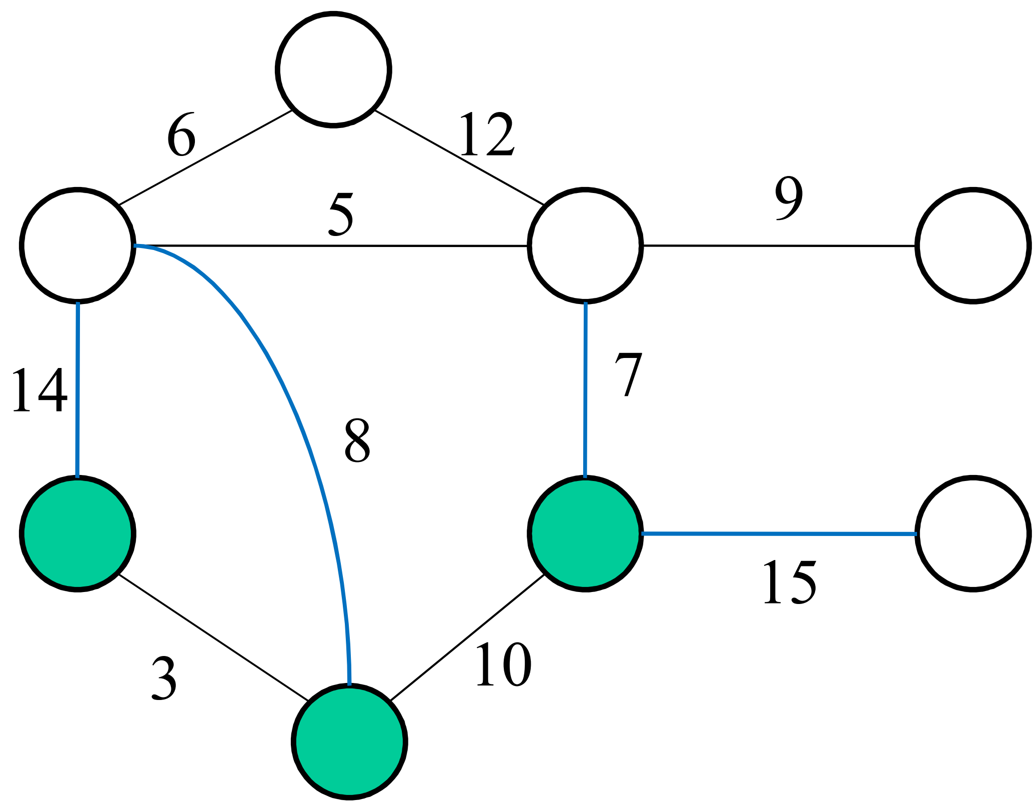
Example



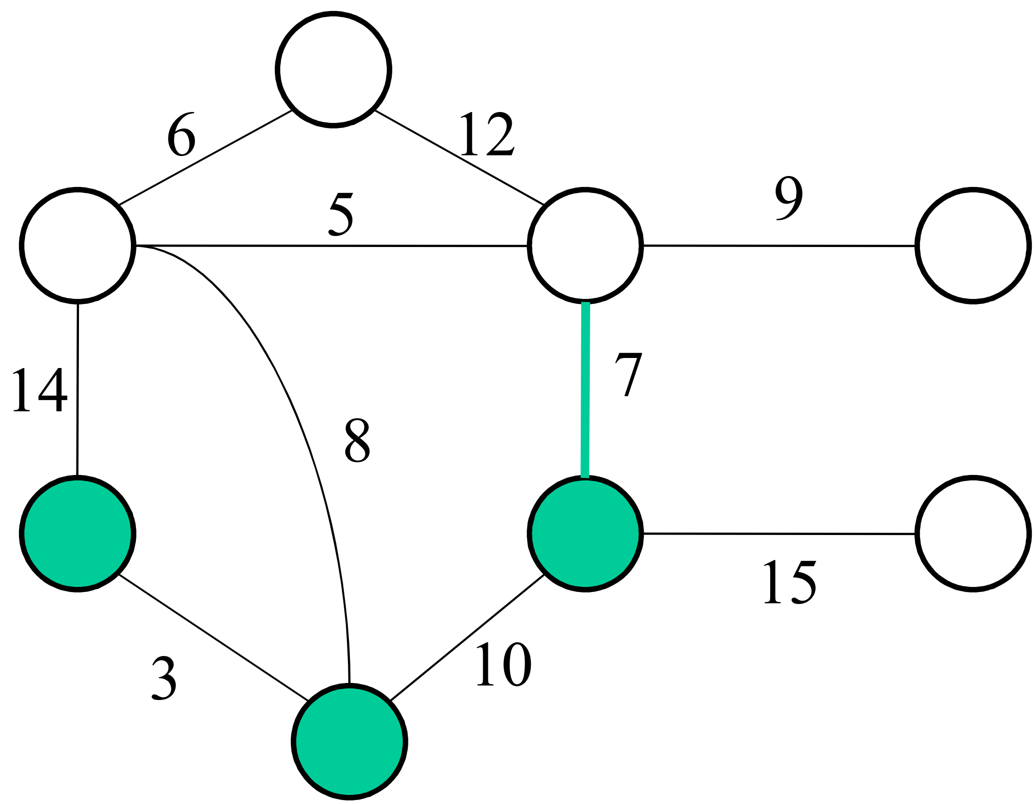
Example



Example



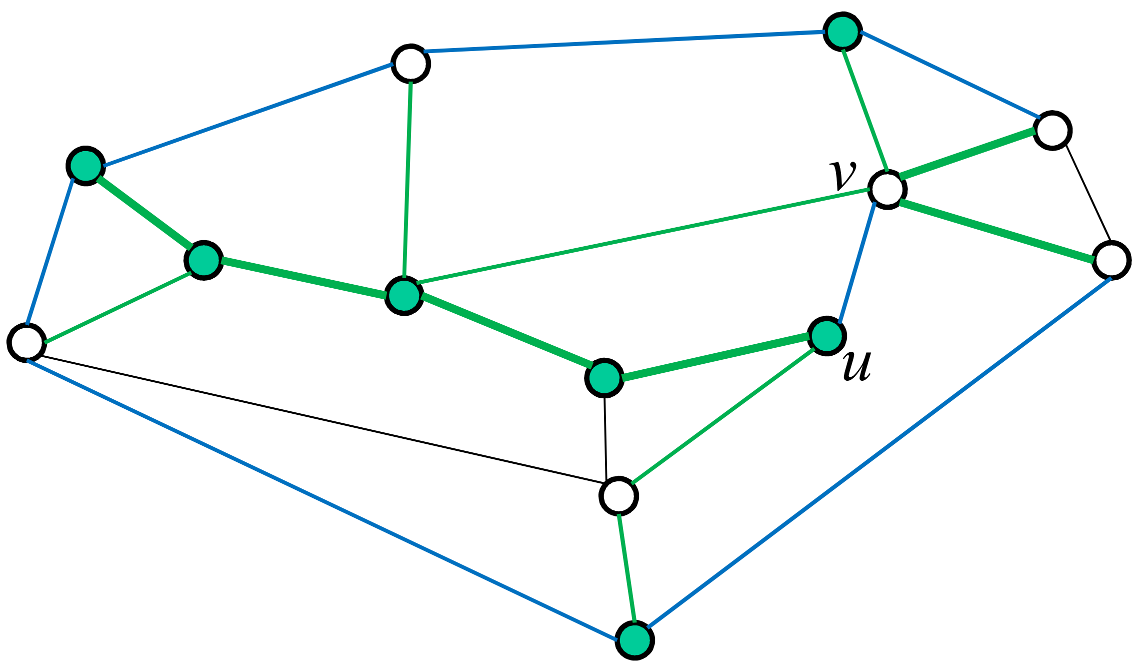
Example



Theorem. Let $G(V, E)$ be a weighted graph and $A \subset V$. If $e = (u,v) \in E$ is a minimum weight edge connecting A to $V - A$ then there is a MST T of G such that $e \in T$.

Proof (by contradiction). Suppose $e \notin T$.

What happens when you add e to T?



Proof (cut-and-paste)

- Consider the unique simple path ρ in T from u to v.
- Swap e with the first edge f in ρ in that connects a vertex in A to a vertex in $V - A$
- A lower weight MST T' results, a contradiction

Prim's Algorithm

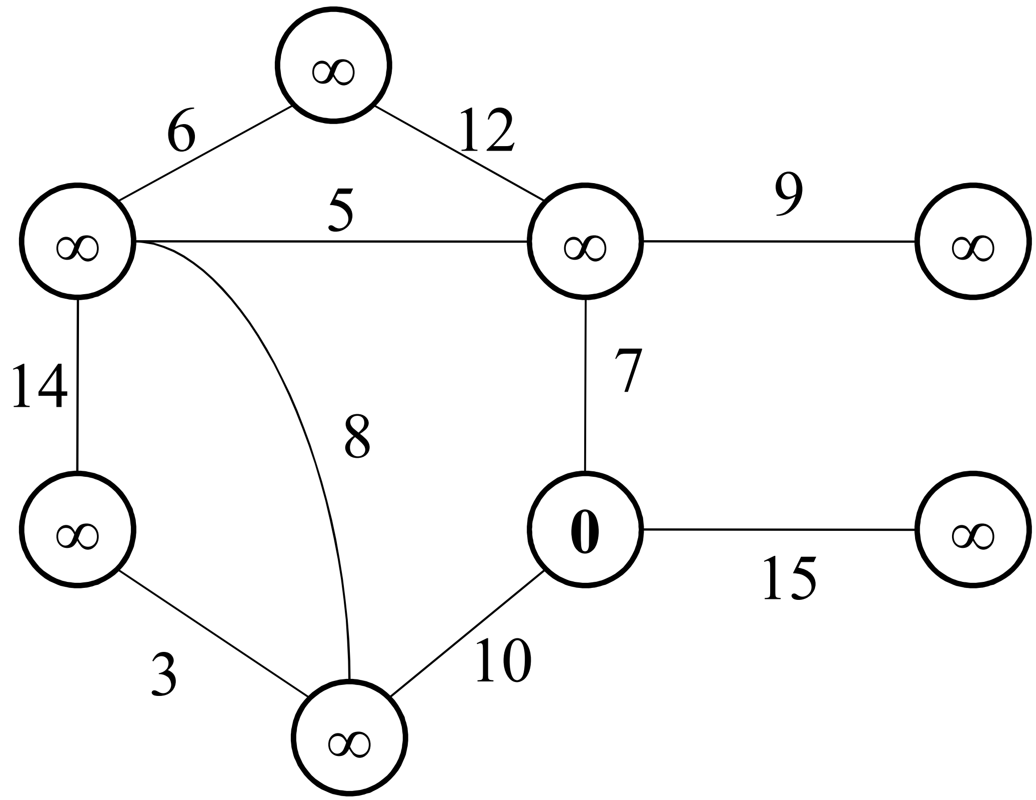
- Keep $V - A$ in a priority queue Q
- The priority of each vertex q in Q is the minimum cost required to connect q to a vertex in A.
- Repeatedly remove the minimum vertex u from Q (using ExtractMin) and add it to A
- Update Q by (possibly) updating the priorities of u's neighbors (using DecreaseKey)

Prim's Algorithm

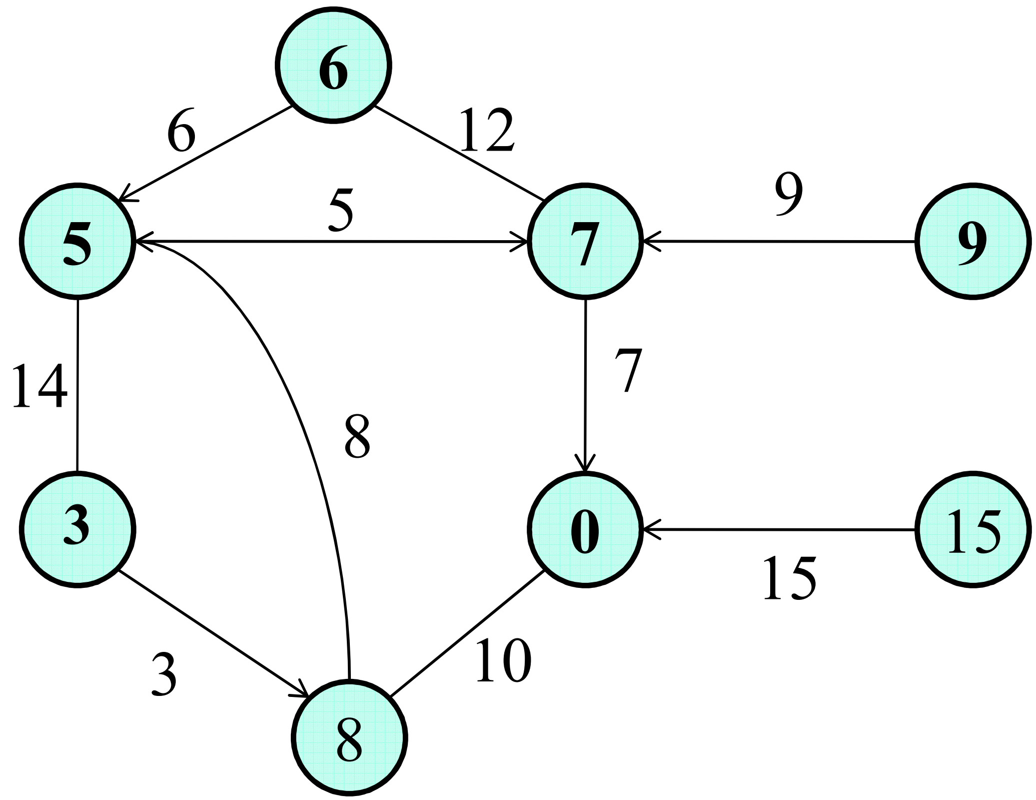
PRIM-MST(V, E, w, s)

```
foreach v ∈ V
        do key[v] ← ∞
           P[v] ← NIL
key[s] ← 0
Q ← V
while Q ≠ ∅
        do u ← EXTRACT-MIN(Q)
           foreach v ∈ Adj[u]
                do if v ∈ Q and w(u, v) < key[v]
                      then key[v] ← w(u, v)
                           P[v] ← u
return P
```

Example



Example



Analysis

<u>Frequency count:</u>

1-3 n times

4-5: once

6-7: n times

8-11: $2m$ times

12: once

```
PRIM-MST(V, E, w, s)
1   foreach v ∈ V
2         do key[v] ← ∞
3            P[v] ← NIL
4   key[s] ← 0
5   Q ← V
6   while Q ≠ ∅
7         do u ← EXTRACT-MIN(Q)
8            foreach v ∈ Adj[u]
9                do if v ∈ Q and w(u, v) < key[v]
10                     then key[v] ← w(u, v)
11                          P[v] ← u
12  return P
```

$$T(n,m) = n + \mathrm{T}_{\text{build}} + n\,\mathrm{T}_{\text{extract}} + m\,\mathrm{T}_{\text{decrease}}$$

Analysis

- Actual time depends on implementation of Q

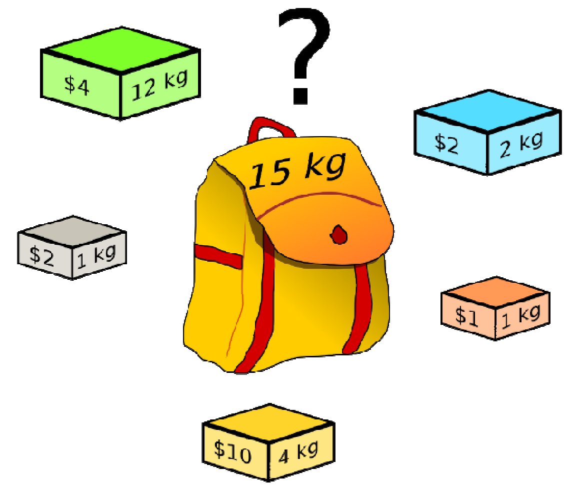
$$T(n,m) = \mathrm{T}_{\mathrm{build}} + n\,\mathrm{T}_{\mathrm{extract}} + m\,\mathrm{T}_{\mathrm{decrease}}$$

Q	Build	Extract	DecreaseKey	Total
Unsorted array	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(n^2)$
Heap	$\Theta(n)$	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta((n+m)\log n)$
Fibonacci heap	$\Theta(n)$	$\Theta(\log n)$	$\Theta(1)$	$\Theta(m+n\log n)$

- Known results
  - Best deterministic: $O(m\,\alpha(m,n))$, Chazelle 2000
  - Best randomized: $O(n + m)$ expected, Karger et al 1995
  - Holy grail: $O(n + m)$ worst case, open

Packing a Knapsack

- You are given a container with a limited weight capacity W, and a list of items, each with a *weight* and a *value*. Choose which items to take so that the weight limit is not exceeded and the total value of the packed items is as large as possible



- A fund manager is considering 100 potential investments and has estimated the expected return from each one. Choose which investments to buy to maximize the return without exceeding the budget
- Choose how to load shipping containers

More Formally…

- Given a set $T = \{(v_1, w_1), \dots, (v_n, w_n)\}$ of items and a target M, choose a subset $S \subseteq \{1, \dots, n\}$ that maximizes $\sum_{i \in S} v_i$ subject to $\sum_{i \in S} w_i \leq M$

	Value	Weight
Clock	175	10
Painting	90	9
Radio	20	4
Vase	50	2
Book	10	1
Computer	200	20

Which items should the thief take if he can carry up to $W = 20$ pounds?

Greedy Choices

- There are several reasonable greedy strategies:
  1. *Best value first.* Add items in decreasing order by value until capacity M is exceeded
  2. *Lowest weight first.* Add items in increasing order of weight until capacity M is exceeded
  3. *Best bang for the buck.* Add items in decreasing order of density (profit per unit weight) until capacity M is exceeded
- Various greedy and DP algorithms guarantee nearly optimal or optimal results for variants of this problem

Variant: Fractional Knapsack

- Suppose you are allowed to take an arbitrary *fraction* of each item (e.g., 2.5 lbs. from a 10 lb. wheel of expensive French cheese)
- Greedy is optimal, by greedy by what?

```
GreedyKnapsack(A, v[1 : n], w[1 : n], M)
Sort v and w, concurrently, by density // So v[i]/w[i] ≥ v[i + 1]/w[i + 1]
Initialize X[1 : n] with zeroes
rem = M // Unused capacity
for i = 1 to n
    if w[i] > rem
        break
    X[i] = 1
    rem = rem − w[i]
if i ≤ n
    X[i] = rem/w[i]
return X
```

Claim. X is an optimal solution

Proof

1. After sorting $v_1/w_1 \geq v_2/w_2 \geq \cdots \geq v_n/w_n$
2. If $x_1 = x_2 = \cdots = x_n = 1$, X is optimal
3. Else, let j be the smallest index such that $X_j < 1$

   $x_i = 1$ for $1 \leq i < j, x_j < 1, x_k = 0$ for $j < k \leq n$
4. Let $Y = (y_1, \ldots, y_n)$ be optimal. Then $\sum w_i y_i = M$
5. Let k be the smallest index such that $x_k \neq y_k$
6. We must have $y_k < x_k$, why?
7. Increase y_k to x_k and decrease $y_{k+1}, \ldots y_n$ as needed to keep capacity at M
8. This results in a solution $Z = (z_1, \ldots z_n)$ with $z_i = x_i$ for $1 \leq i \leq k$ and $\sum_{k<i\leq n} w_i(y_i - z_i) = w_k(z_k - y_k)$
9. $\sum_{1\leq i\leq n} v_i z_i = \sum_{1\leq i\leq n} v_i y_i + (z_k - y_k) w_k \frac{v_k}{w_k} - \sum_{k<i\leq n}(y_i - z_i) w_i \frac{v_i}{w_i} \geq \sum_{1\leq i\leq n} v_i y_i$
10. Since Y is optimal, we must have $\sum_{1\leq i\leq n} v_i z_i = \sum_{1\leq i\leq n} v_i y_i$
11. If $Z = X$, then X is optimal; else repeat the argument

Exercise

- Derive greedy or DP solutions for the following variants. You may assume that weights and values are integers.

1. All items have the same weight
2. All items have the same value
3. For all items $v_i = w_i$

- Can you guarantee optimality? Can you get close?