Lecture 16: April 8, 2004

Today:

- Graph Representations & Algorithms
- Greedy Algorithms
  - Minimum Spanning Tree
    * Kruskal's Algorithm
    * Prim's Algorithm

Graph Representation

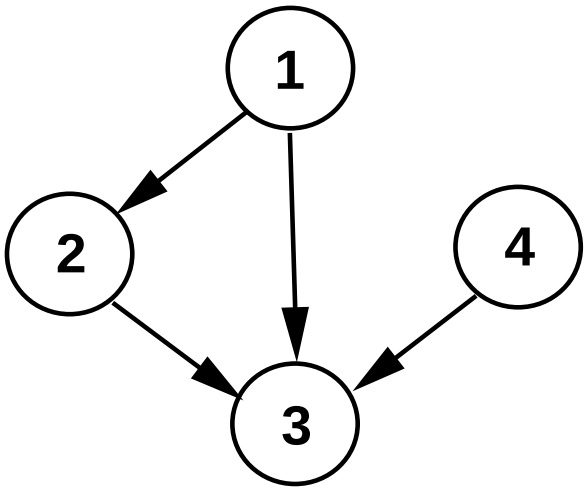
Graph $G = (V, E)$

V = set of vertices

E = set of edges = subset of $V \times V$

If G is connected, then

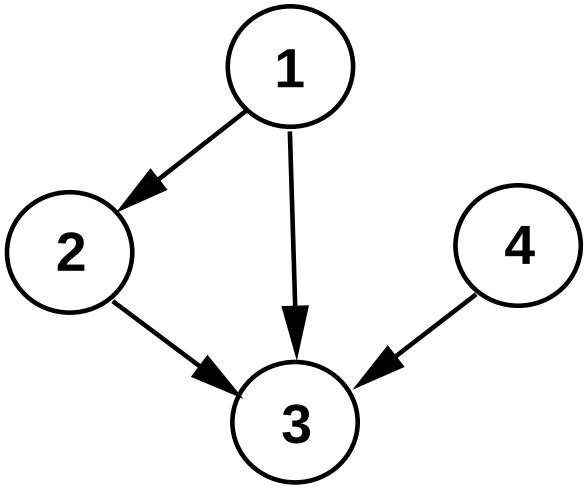
$|E| \geq |V| - 1 \Rightarrow \log |E| = \Theta(\log V)$



(Note: drop | | inside asymptotic notation)

Graph Representation

Graph can be *directed* or *undirected*



Assume vertices $V = \{1, 2, \ldots, n\}$

Define "Adjacency matrix" A[1 .. n, 1..n]

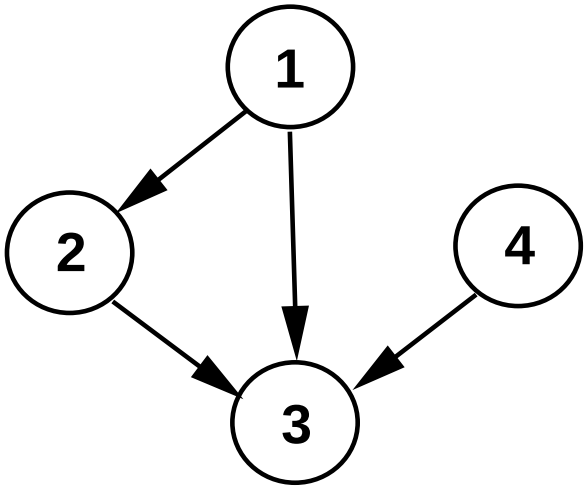
$$A[i,j] = \begin{cases} 1 \text{ if } (i,j) \in E \\ 0 \text{ if } (i,j) \notin E. \end{cases}$$

```
    |  j=1 2 3 4
  --|-----------
i=1 |    0 1 1 0
  2 |    0 0 1 0
  3 |    0 0 0 0
  4 |    0 0 1 0
```

Adjacency matrix:

$\Theta(V^2)$ storage $\rightarrow$ **dense** representation

Sparse Graph Representation



Adjacency **list**: $\Theta(V + E)$ storage
$\rightarrow$ **sparse** representation

For each vertex v, keep a list Adj[v] of vertices adjacent to v

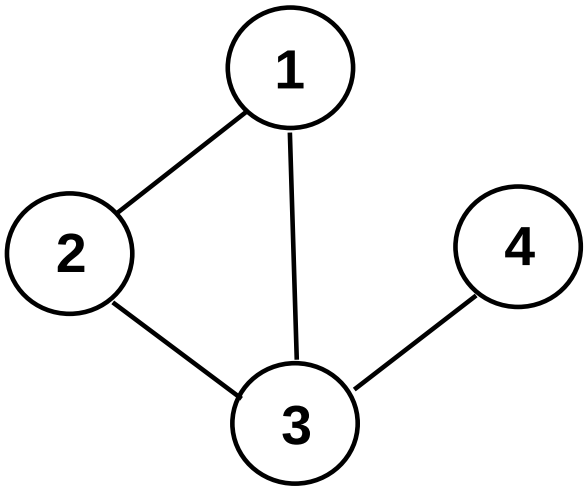
```
Adj[1] = {2,3}
Adj[2] = {3}
Adj[3] = {}
Adj[4] = {3}
```

Adj[v] = degree(v) [for undirected graphs]
Adj[v] = out-degree(v) [for digraphs]

Handshaking Lemma

For undirected graphs,

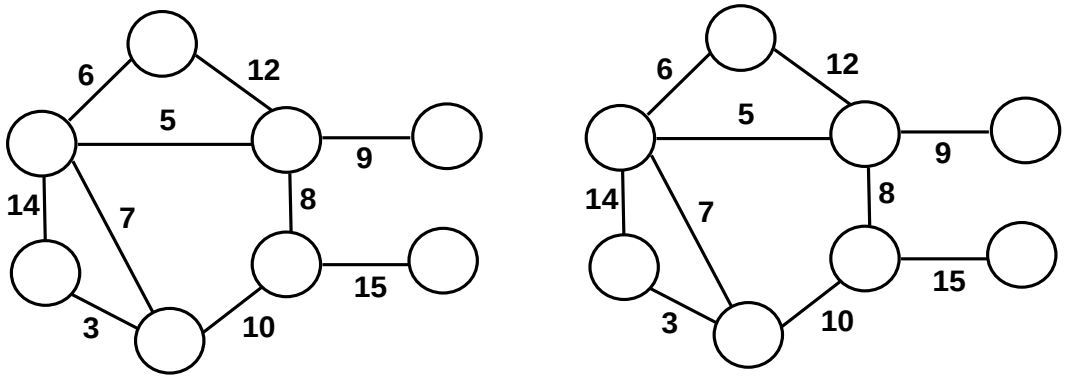
$$\Sigma \deg(v) = 2|E|$$



Thus adjacency list uses $\Theta(V + E)$ storage

Minimum Spanning Tree

Undirected, connected graph $G = (V, E)$
Weight function $W : E \to \Re$
(Here, assume edge weights distinct)



Spanning Tree: Tree that connects all vertices
What is ST of above graph?
Minimum Spanning Tree:
Tree that connects all vertices, and minimizes

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

What is MST of above graph?

Generic-MST

GENERIC-MST(V, E):

1 $A \leftarrow \emptyset$
2 **for** $i \leftarrow 1$ **to** $|V| - 1$:
3 Find a "safe" edge $e = (u, v) \in E$
4 $A \leftarrow A \cup \{e\}$

"Safe" means "maintains A is a subset of the MST".

Proof:

1. Loop Invariant: A is a subset of the MST T.
2. Initialization: $A = \emptyset \subset T$. (Trivially)
3. Maintenance: Only add "safe" edges. $A \cup \{e\} \subseteq T$.
4. Termination: After $|V| - 1$ loops, added all MST edges.

(Using "safe" edges seems like circular logic. We'll define it later.)

Greedy Algorithms

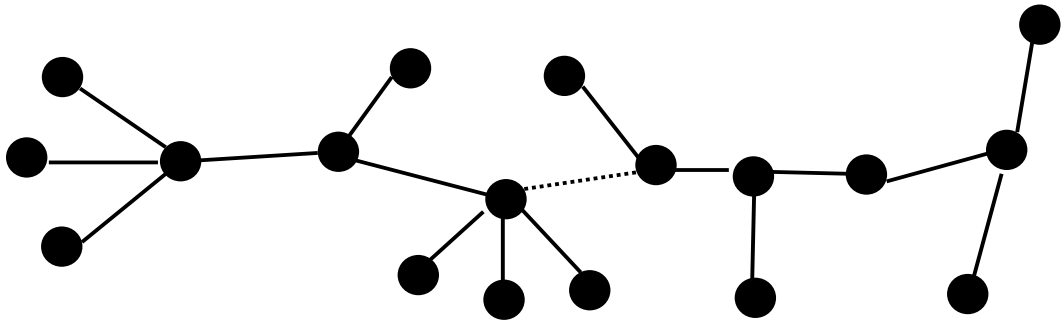
1. Optimal Substructure - Optimal solution to problem is also optimal for subproblems.
2. Locally Optimal (Greedy) Choices - Pick the choice that "looks" best on current problem.
3. Need to prove that there is an optimal solution that makes the greedy choice in the original problem.
4. Show that we can combine greedy choice with optimal subproblem solution to get the global optimal.

In Dynamic Programming, local choice could depend on subproblem (i.e. future) choices - optimal comes from "bottom up".

In Greedy Algorithms, local choice depends only on past choices - solution comes from the "top down".

Optimal Substructure

T (Note: some other edges omitted):



Removing (u, v) partitions T into T_1 and T_2.

Claim: T_1 is MST of $G_1 = (V_1, E_1)$, the subgraph of G **induced** by vertices in T_1.

$V_1 =$ vertices in T_1

$E_1 = \{(x, y) \in E : x, y \in V_1\}$

$T2$ is MST of $G2$.

$w(T) = w(u, v) + w(T1) + w(T2)$

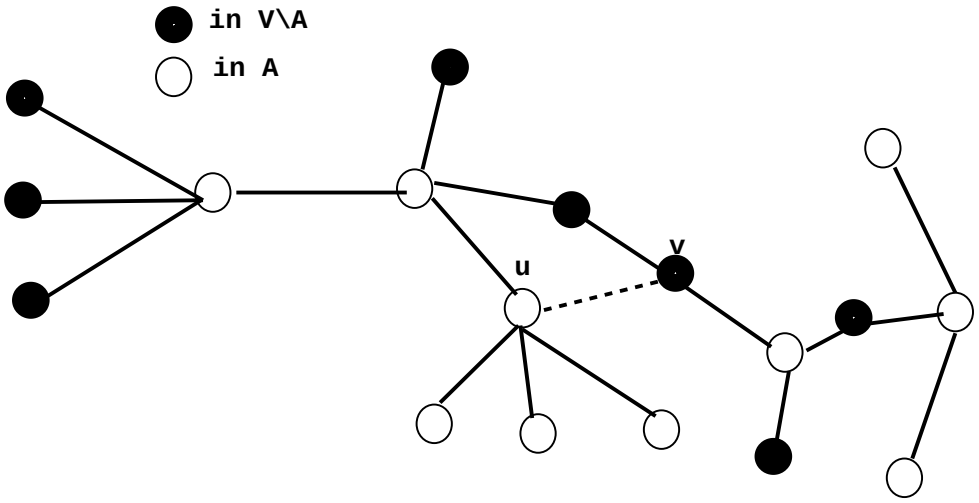
Can't be a tree better than T_1 or T_2, or T would be suboptimal!

(Overlapping subproblems? Dynamic prog? Yes, but...)

Greedy Choice

Greedy choice property:
Locally optimal (greedy) choice
yields a globally optimal solution!

Theorem: Let T be MST of G, and let $A \subseteq V$.
Let (u, v) be min weight edge in G connecting A to $V - A$.
Then, $(u, v) \in T$. **Proof:** "cut and paste"



Suppose $(u, v) \notin T$
Look at path from u to v in T
Swap (u, v) with first edge on path from u
to v in T that crosses from A to $V - A$.
This improves T!

Kruskal's algorithm for MST

Disjoint-set data structure
Sets $S = \{S_i\}$, S_i intersects S_j = empty set
Operations:

- Insert(x): $S \leftarrow S \cup \{\{x\}\}$
- Union(S_i, S_j): $S \leftarrow S - \{S_i, S_j\} \cup \{S_i \cup S_j\}$
- FindSet(x): returns unique $S_i \in S$ where $x \in S_i$

(Sounds familiar if you went to Friday's recitation.)

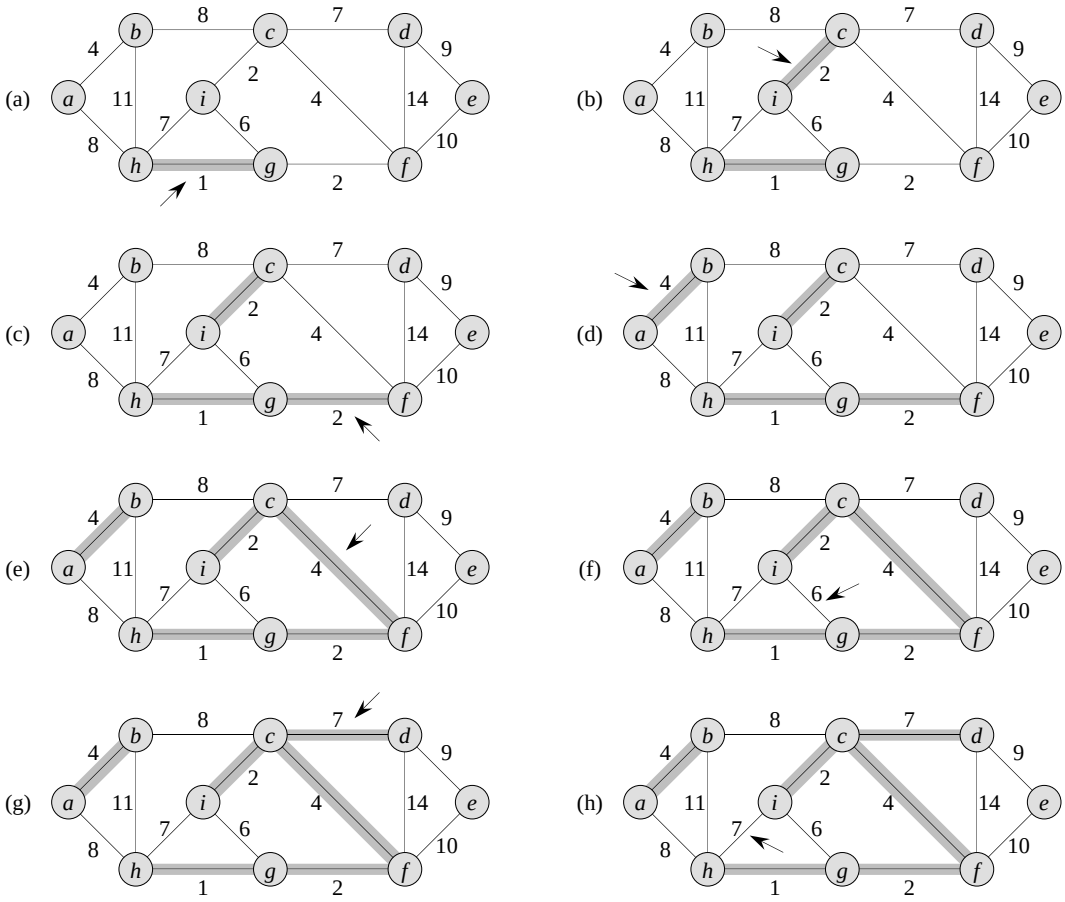
Kruskal's algorithm for MST

Main Idea: Each node is starts as an island. Keep adding lightest edge that doesn't create a cycle. That is, add cheapest edge that connects two trees in the "forest".

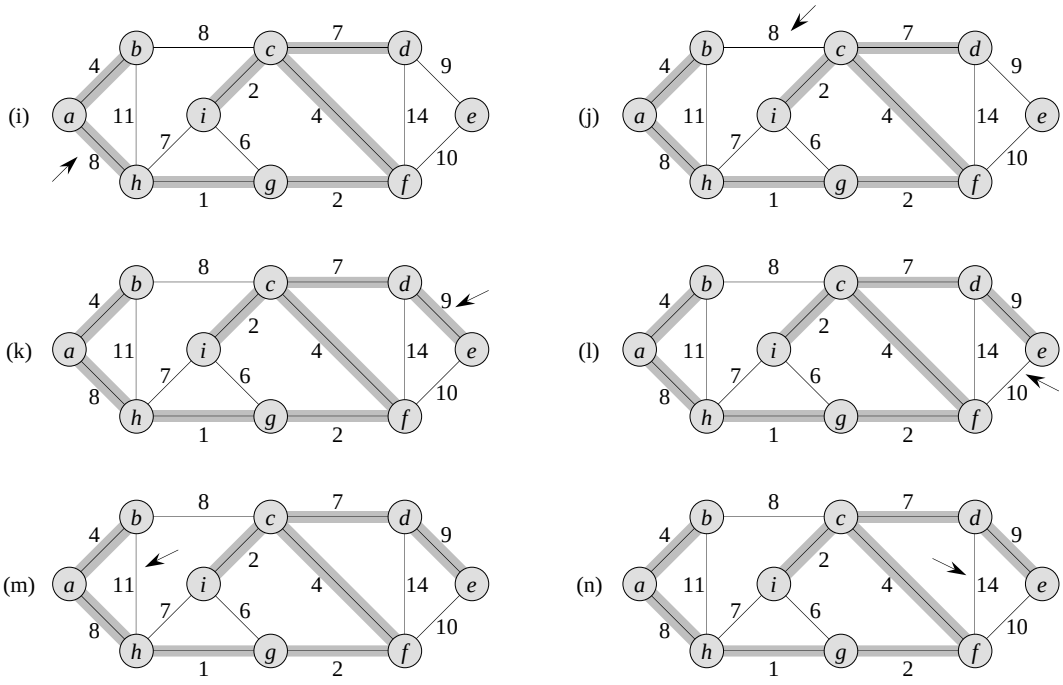
```
    KRUSKAL-MST(V, E):
1   T ← empty set
2   for each v ∈ V:
3         INSERT(v)
4   Sort E by edge weight
5   for each edge e = (u, v) ∈ E:
6         if (FINDSET(u) ≠ FINDSET(v)):
7            T ← T ∪ {e}
8            UNION(FINDSET(u), FINDSET(v))
```

Why is this algorithm correct?

Example



Example (cont.)



Kruskal's Running Time

- Sort: $\Theta(E \times \log E) = \Theta(E \times \log V))$, since $|E| \leq |V^2|$
- $\Theta(V)$ calls to Insert
- $\Theta(E)$ calls to FindSet
- $\Theta(V)$ calls to Union
- We can use the Union-Find data structure in CLR Chapter 22
- That will take $\Theta(E \times \alpha(E, V))$ time to perform E Union and FindSet operations on a set of size V.

$\alpha(m, n)$ is "functional inverse" of Ackermann's function. $\alpha(m, n) \leq 4$ even for $m, n = 10^{80}$ grows very slowly, but not constant!

Overall running time $\Theta(E \log E)$.
Best MST to date: 1993 Karger, Klein, Tarjan
$\Theta(E)$ time randomized (uses fast MST test)

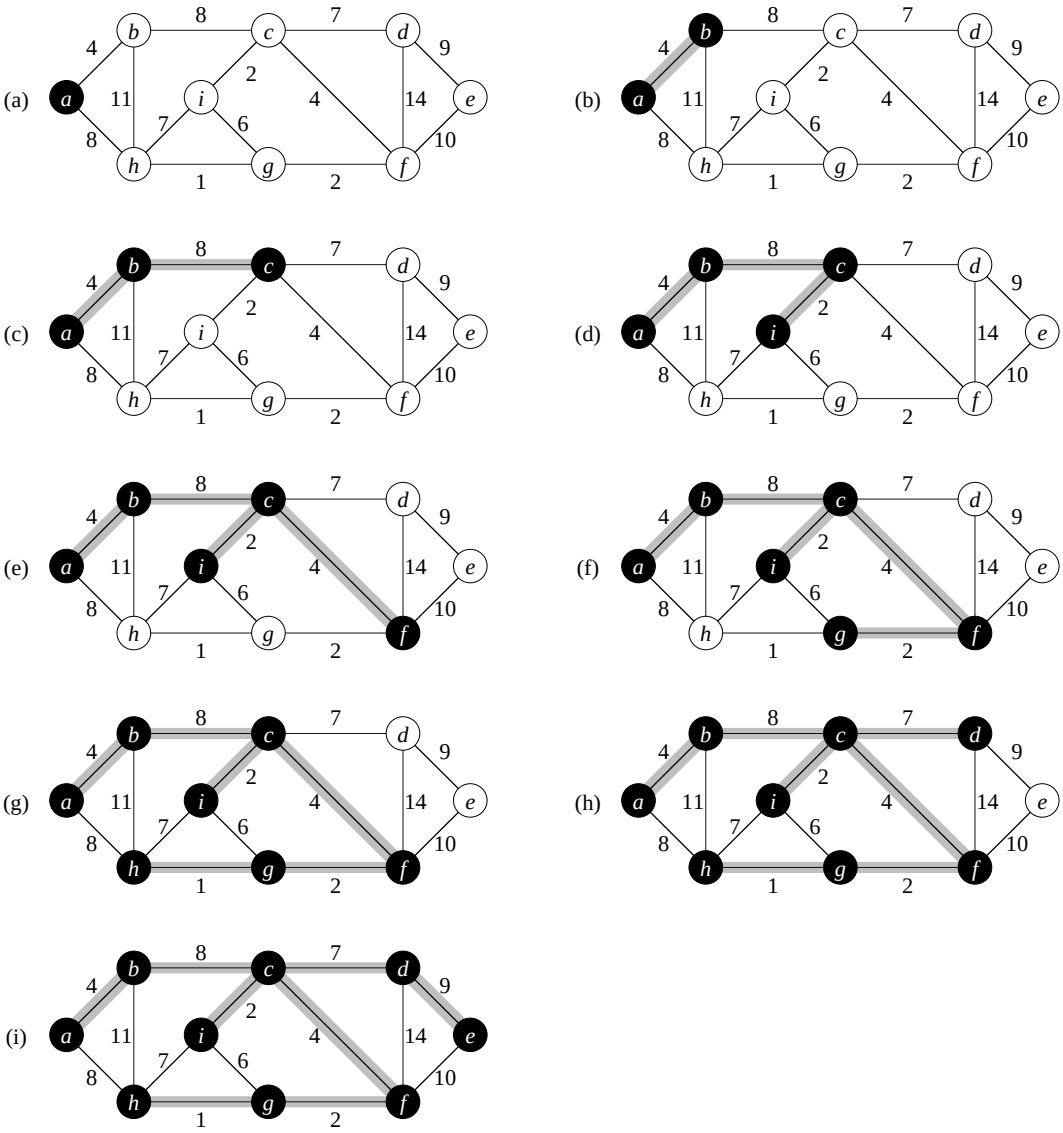
Prim's Algorithm

Main Idea: Start with a node. Grow a "frontier" by adding nearest node to frontier.
Keep $V - A$ in priority queue Q, sorted by weight of lightest edge connecting to A

```
    PRIM-MST(V, E):
1   Q ← V
2   key[v] ← ∞, ∀v ∈ V
3   key[s] ← 0, for arbitrary s ∈ V
4   while Q ≠ ∅:
5         u ← Extract − Min(Q):
6         for each v ∈ Adj[u]:
7            if v ∈ Q and w(u, v) ≤ key[v]:
8                  key[v] ← w(u, v)
9                  π[v] ← u
```

At end, $\{(v, \pi[v]\}$ forms MST.

Example



Prim's Running Time

Using a Min-Heap to store vertices, Prim's makes:

- 1 call to Build-Heap.
- $|V|$ calls to Extract-Min.
- $\Theta(E)$ calls to Decrease-Key.

Extract-Min and Decrease-Key runtimes depend on the implementation:

Q	T(Extract-Min)	T(Decrease-Key)	Total
Array	$\Theta(V)$	$\Theta(1)$	$O(V^2)$
Binary Heap	$\Theta(\log V)$	$\Theta(\log V)$	$O(E \log V)$
Fib. Heap	$\Theta(\log V)$	$\Theta(1)$	$O(V \log V + E)$