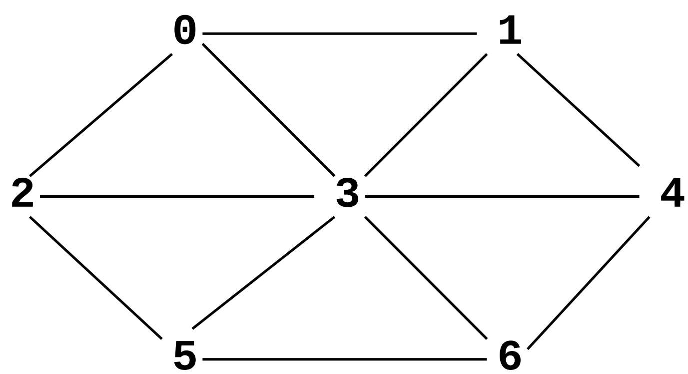
16-0: **Spanning Trees**

- Given a connected, undirected graph $G$
  - A *subgraph* of $G$ contains a subset of the vertices and edges in $G$
  - A *Spanning Tree* $T$ of $G$ is:
    - subgraph of $G$
    - contains all vertices in $G$
    - connected
    - acyclic

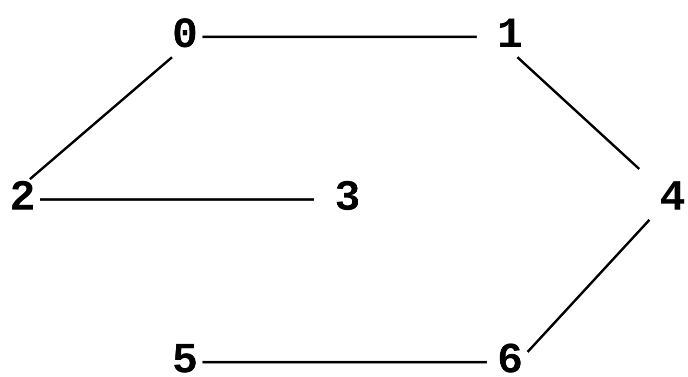
16-1: **Spanning Tree Examples**

- Graph



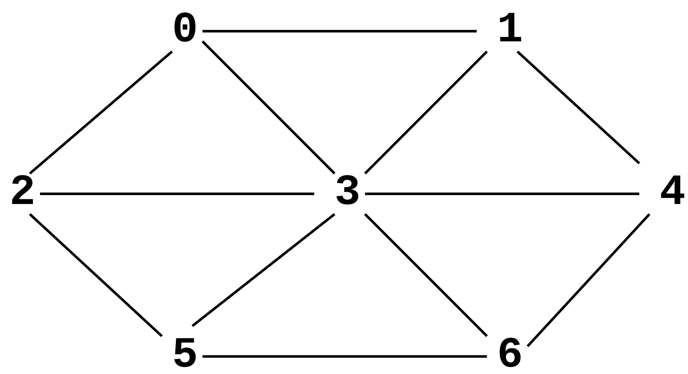

16-2: **Spanning Tree Examples**

- Spanning Tree



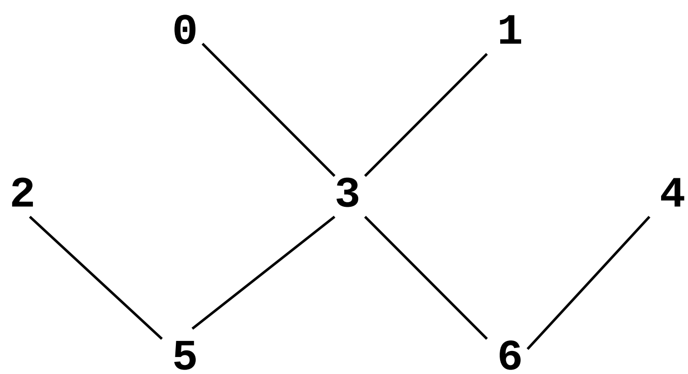

16-3: **Spanning Tree Examples**

- Graph




16-4: **Spanning Tree Examples**

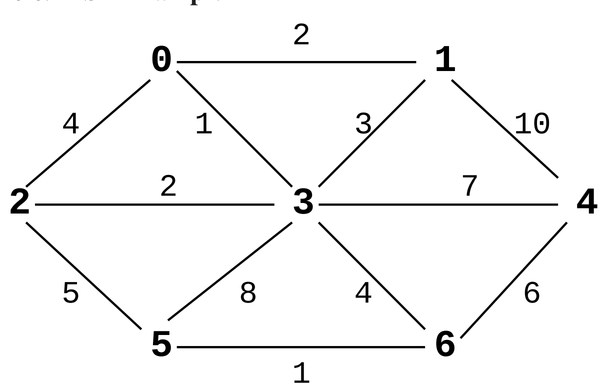
- Spanning Tree



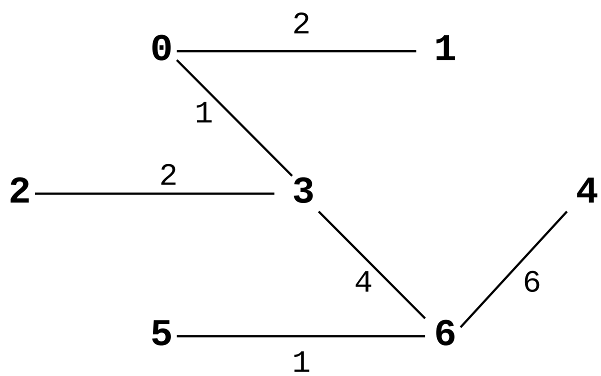

16-5: **Minimal Cost Spanning Tree**

- Minimal Cost Spanning Tree
  - Given a weighted, undirected graph $G$
  - Spanning tree of $G$ which minimizes the sum of all weights on edges of spanning tree

16-6: **MST Example**




16-7: **MST Example**




16-8: **Minimal Cost Spanning Trees**

- Can there be more than one minimal cost spanning tree for a particular graph?

16-9: **Minimal Cost Spanning Trees**

- Can there be more than one minimal cost spanning tree for a particular graph?
- YES!
  - What happens when all edges have unit cost?

16-10: **Minimal Cost Spanning Trees**

- Can there be more than one minimal cost spanning tree for a particular graph?
- YES!
  - What happens when all edges have unit cost?
  - All spanning trees are MSTs

16-11: **Calculating MST**

- Generic MST algorithm:

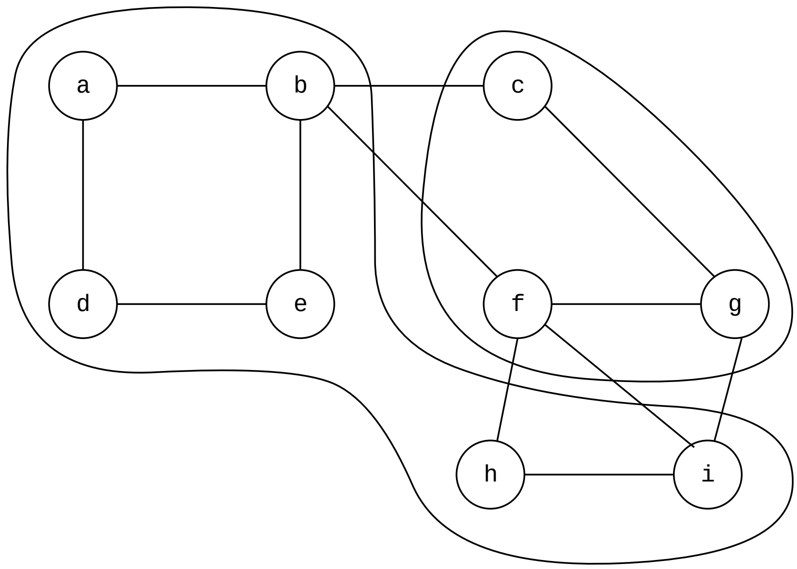
$A \leftarrow \{\}$
while $A$ does not form a spanning tree
  find an edge $(u, v)$ that is safe for $A$
  $A \leftarrow A \cup \{(u, v)\}$

- $(u, v)$ is safe to for $A$ when $A \cup \{(u, v)\}$ is a subset of some MST

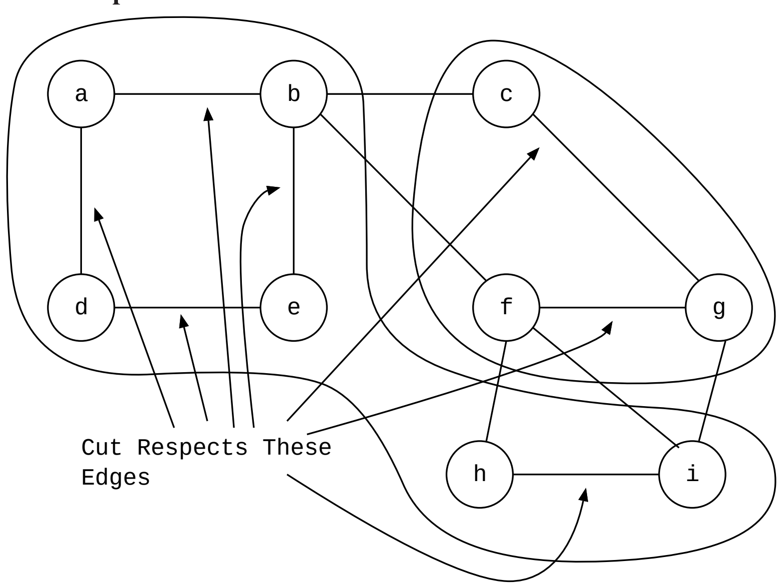
16-12: **Graph Cut**

- "Cut" of a undirected graph is a partition of the vertices in the graph
  - An edge crosses a cut if the vertices are in different sets of the partition
  - A cut respects a series of edges of no edge crosses the cut
  - light edge is an edge that crosses the cut that has minimum cost

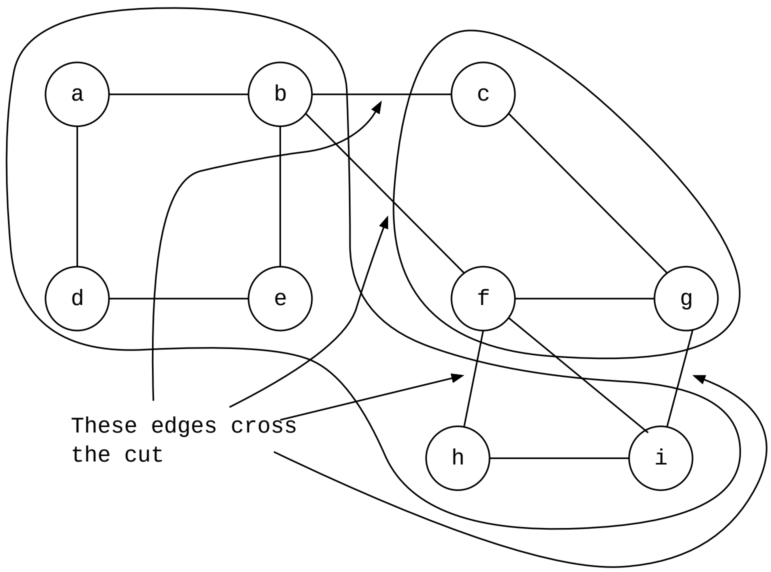
16-13: **Graph Cut**



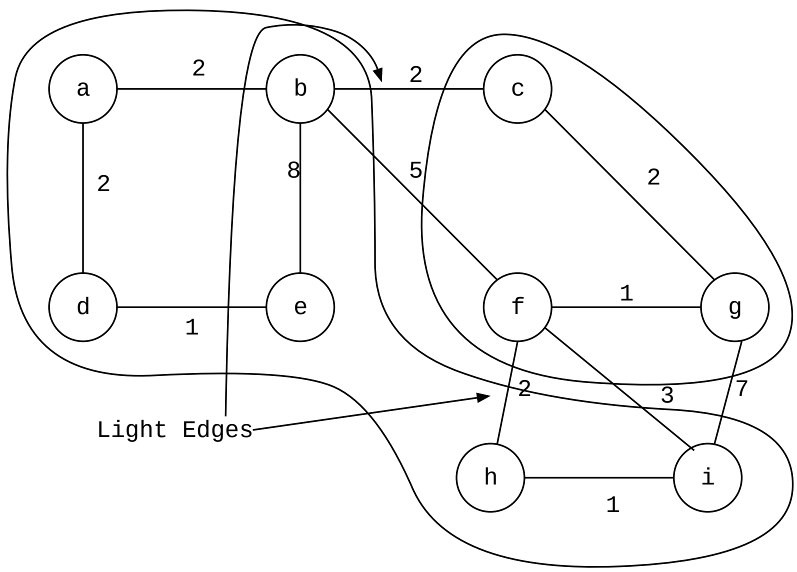

16-14: **Graph Cut**




16-15: **Graph Cut**




16-16: **Graph Cut**




16-17: **Safe Edges**

- $A$ is a set of edges, which is a subset of some MST
- Cut $\{S, V - S\}$ which respects $A$
- Any light edge (with respect to the cut $\{S, V - S\}$) is safe
  - That is, $A \cup \{(u, v)\}$ is a subset of some MST if $\{(u, v)\}$ is a light edge in a cut that respects $A$

16-18: **Safe Edges**

- Proof by contradiction:
  - Assume there is:

        - a subset of a MST $A$
        - a Cut $\{S, V - S\}$ that respects $A$
        - a light edge $(u, v)$
    - such that $A \cup \{(u, v)\}$ is not a subset of any MST

- We will show that this leads to a contradiction

16-19: **Safe Edges**

- Let $A'$ be a MST that is a superset of $A$
- Add $(u, v)$ to $A'$ to get $A''$ – now have a cycle
- This cycle must cross the cut at least twice
    - $(u, v)$ is one crossing
    - Must be another crossing $(u', v')$ back across the cut
- remove $(u', v')$ from $A''$ to get $A'''$
- $A'''$ is a spanning tree
- cost($A'''$) = cost($A'$) - cost($(u', v')$) + cost($(u, v)$)
- cost($(u, v)$) $\leq$ cost($(u', v')$ $\Rightarrow$ cost($A'''$) $\leq$ cost($A'$)

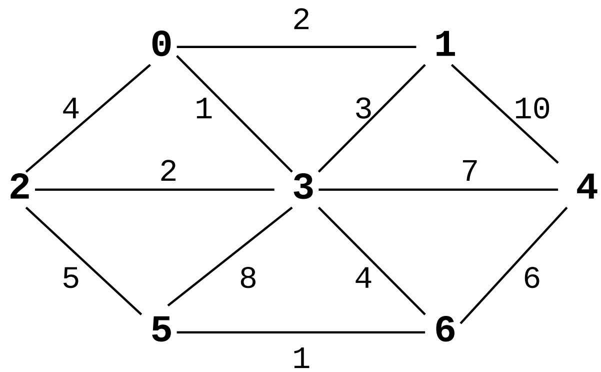
16-20: **Safe Edges**

- Let $A'$ be a MST that is a superset of $A$
- Add $(u, v)$ to $A'$ to get $A''$ – now have a cycle
- This cycle must cross the cut at least twice
- Must be another crossing $(u', v')$ back across the cut
- remove $(u', v')$ from $A''$ to get $A'''$
- $A'''$ is a spanning tree
- cost($A'''$) = cost($A'$) - cost($(u', v')$) + cost($(u, v)$)
- cost($(u, v)$) $\leq$ cost($(u', v')$ $\Rightarrow$ cost($A'''$) $\leq$ cost($A'$)
- Thus $A'''$ must be a MST that contains $A$ and $\{(u, v)\}$, a contradiction

16-21: **Kruskal's Algorithm**

- Start with an empty graph (no edges)
- Sort the edges by cost
- For each edge $e$ (in increasing order of cost)
    - Add $e$ to $G$ if it would not cause a cycle

16-22: **Kruskal's Algorithm Examples**




16-23: **Kruskal's Algorithm**

- Correctness proof:
  - Kruskal's algorithm always selects a light edge, with according to some cut that respects all edges added so far.
    - Let $(u, v)$ be the cheapest edge that does not cause a cycle
    - Let $S$ be the connected component that contains $u$.
    - $\{S, V - S\}$ respects edges chosen so far
    - $(u, v)$ crosses the cut, and is the edge with the smallest cost that crosses the cut $\Rightarrow (u, v)$ is a light edge
  - Thus, Kruskal's algorithm always selects a safe edge, and produces a MST

16-24: **Kruskal's Algorithm**

- Coding Kruskal's Algorithm:
  - Place all edges into a list
  - Sort list of edges by cost
  - For each edge in the list
    - Select the edge if it does not form a cycle with previously selected edges
    - How can we do this?

16-25: **Kruskal's Algorithm**

- Determining of adding an edge will cause a cycle
  - Start with a forest of $V$ trees (each containing one node)
  - Each added edge merges two trees into one tree
  - An edge causes a cycle if both vertices are in the same tree
    - (examples)

16-26: **Kruskal's Algorithm**

- We need to:
  - Put each vertex in its own tree

    - Given any two vertices $v_1$ and $v_2$, determine if they are in the same tree
    - Given any two vertices $v_1$ and $v_2$, merge the tree containing $v_1$ and the tree containing $v_2$

- ... sound familiar?

16-27: **Kruskal's Algorithm**

- Disjoint sets!
- Create a list of all edges
- Sort list of edges
- For each edge $e = (v_1, v_2)$ in the list
    - if FIND($v_1$) != FIND($v_2$)
        - Add $e$ to spanning tree
        - UNION($v_1, v_2$)

16-28: **Kruskal's Algorithm**

- Running time?

16-29: **Kruskal's Algorithm**

- Running time?
    - Sort edges: $\Theta(|E| \lg |E|)$
    - Build tree: $O(E)$
- Total: $\Theta(|E| \lg |E|)$

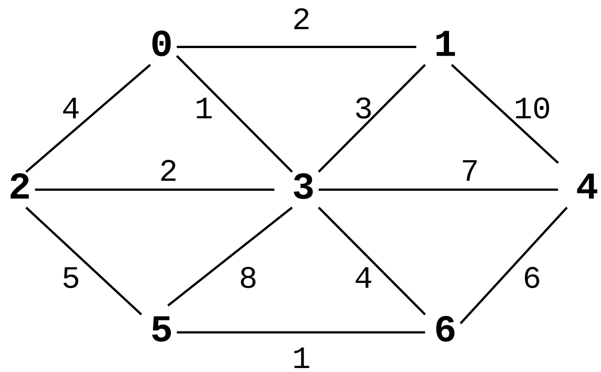
16-30: **Prim's Algorithm**

- Grow that spanning tree out from an initial vertex
- Divide the graph into two sets of vertices
    - vertices in the spanning tree
    - vertices *not* in the spanning tree
- Initially, Start vertex is in the spanning tree, all other vertices are not in the tree
    - Pick the initial vertex arbitrarily

16-31: **Prim's Algorithm**

- While there are vertices not in the spanning tree
    - Add the cheapest vertex to the spanning tree

16-32: **Prims's Algorithm Examples**




16-33: **Prim's Algorithm**

- Maintain a table, which keeps track of:
  - Whether or not the vertex has been added to the MST (Known)
  - Current cheapest cost to add the vertex to the MST (Cost)
  - Neighber to connect to, to get the cheapest cost (Path)

16-34: **Prim Code**

```
void Prim(Edge G[], int s, tableEntry T[]) {
  int i, v;
  Edge e;
  for(i=0; i<G.length; i++) {
    T[i].distance = Integer.MAX_VALUE;
    T[i].path = -1;
    T[i].known = false;
  }
  T[s].distance = 0;
  for (i=0; i < G.length; i++) {
    v = minUnknownVertex(T);
    T[v].known = true;
    for (e = G[v]; e != null; e = e.next) {
      if (T[e.neighbor].distance > e.cost) {
        T[e.neighbor].distance = e.cost;
        T[e.neighbor].path = v;
      }
    }
  }
}
```

16-35: **Prim Running Time**

- If minUnknownVertex(T) is calculated by doing a linear search through the table:
  - Each minUnknownVertex call takes time $\Theta(|V|)$
    - Called $|V|$ times – total time for all calls to minUnkownVertex: $\Theta(|V|^2)$
  - If statement is executed $|E|$ times, each time takes time $O(1)$
  - Total time: $O(|V|^2 + |E|) = O(|V|^2)$.

16-36: **Prim Running Time**

- If minUnknownVertex(T) is calculated by inserting all vertices into a min-heap (using distances as key) updating the heap as the distances are changed
  - Each minUnknownVertex call tatkes time $\Theta(\lg |V|)$
    - Called $|V|$ times – total time for all calls to minUnknownVertex: $\Theta(|V| \lg |V|)$

  - If statement is executed $|E|$ times – each time takes time $O(\lg |V|)$, since we need to update (decrement) keys in heap
  - Total time: $O(|V| \lg |V| + |E| \lg |V|) \in O(|E| \lg |V|)$

- Is this better or wose than the previous method? Explain!

16-37: **Prim Running Time**

- If minUnknownVertex(T) is calculated by inserting all vertices into a Fibonacci heap (using distances as key) updating the heap as the distances are changed
  - Each minUnknownVertex call takes amortized time $\Theta(\lg |V|)$
    - Called $|V|$ times – total amortized time for all calls to minUnknownVertex: $\Theta(|V| \lg |V|)$
  - If statement is executed $|E|$ times – each time takes amortized time $O(1)$, since decrementing keys takes time $O(1)$.
  - Total time: $O(|V| \lg |V| + |E|)$
- Is this better or wose than the previous methods? Explain!

16-38: **Prim Correctness**

- Every time we select a vertex as known, pick an edge to add to MST
- If the set of known vertices are $K$:
  - Create a partition $\{K, V - K\}$
  - Next vertex that we select will be connected to the known vertices by the cheapest possible edge
  - Thus, we're always picking a light edge, according to some partition that repsects all edges we've previously chosen