

Today:

- Theory behind Min. Spanning Tree
- Max Flow (as time allows)

COSC 581, Algorithms

February 27, 2014

Reading Assignments

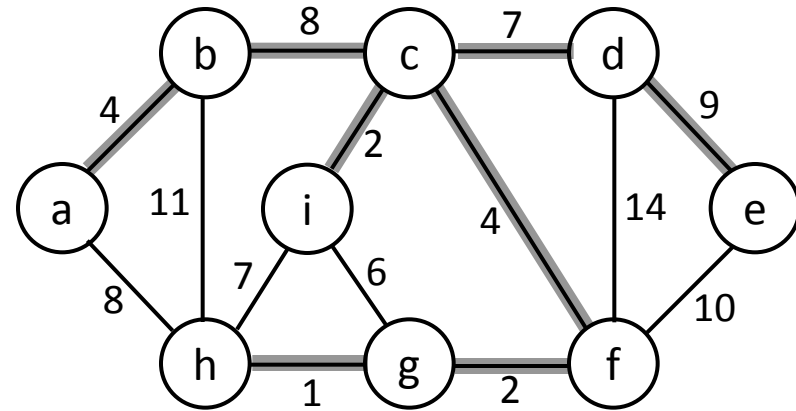
- Today's class:
 - Chapter 23, (if time) Ch. 26
- Reading assignment for next class:
 - Chapter 26

Growing a MST – Generic Approach

- Grow a set A of edges (initially empty)
- Incrementally add edges to A such that they would belong to a MST

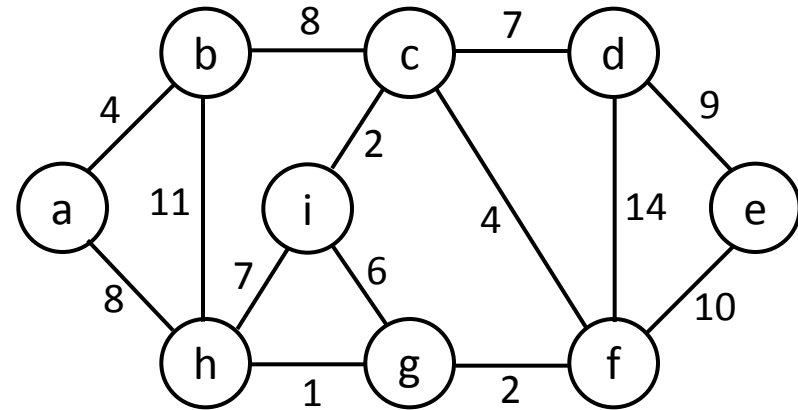
Idea: add only “safe” edges

- An edge (u, v) is **safe** for A if and only if $A \cup \{(u, v)\}$ is also a subset of **some** MST



Generic MST algorithm

1. $A \leftarrow \emptyset$
2. **while** A is not a spanning tree
3. **do** find an edge (u, v) that is **safe** for A
4. $A \leftarrow A \cup \{(u, v)\}$
5. **return** A



How do we find safe edges?

Finding Safe Edges

- Let's look at edge (h, g)

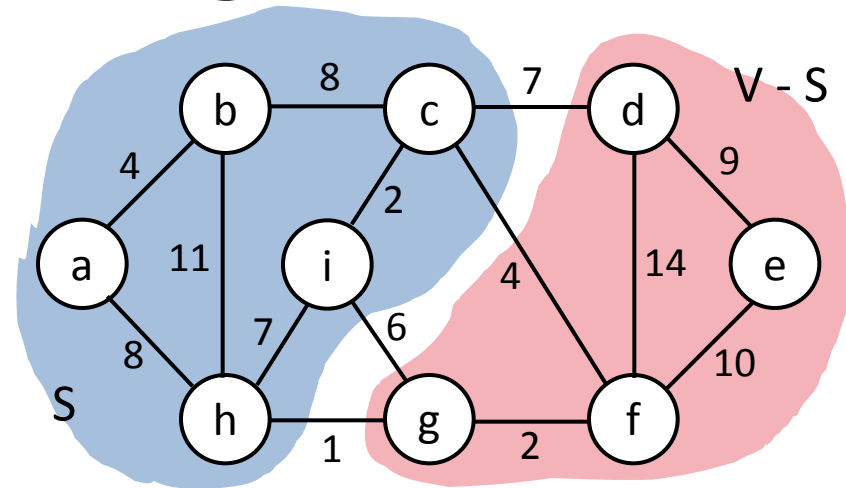
- Is it safe for A initially?

- Later on:

- Let $S \subset V$ be any set of vertices that includes h but not g (so that g is in $V - S$)

- In any MST, there has to be one edge (at least) that connects S with $V - S$

- Why not choose the edge with **minimum weight** (h,g)?



Definitions

- A **cut** $(S, V - S)$

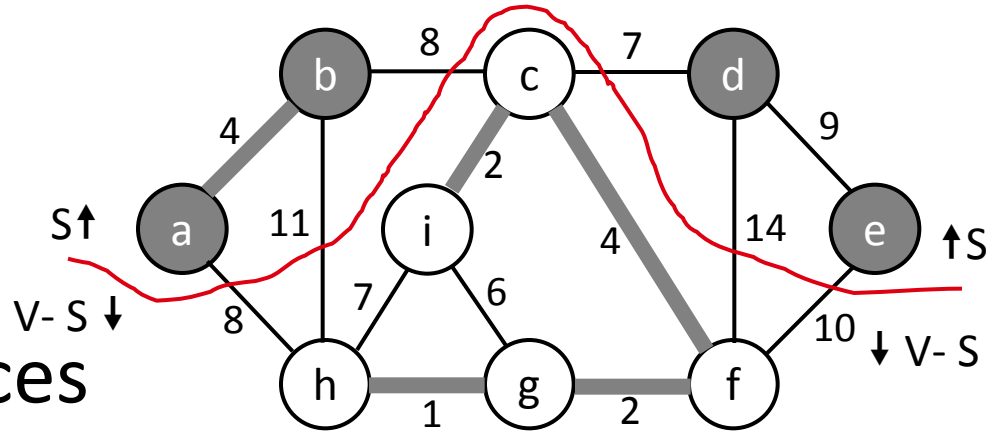
is a partition of vertices

into disjoint sets S and $V - S$

- An edge **crosses** the cut

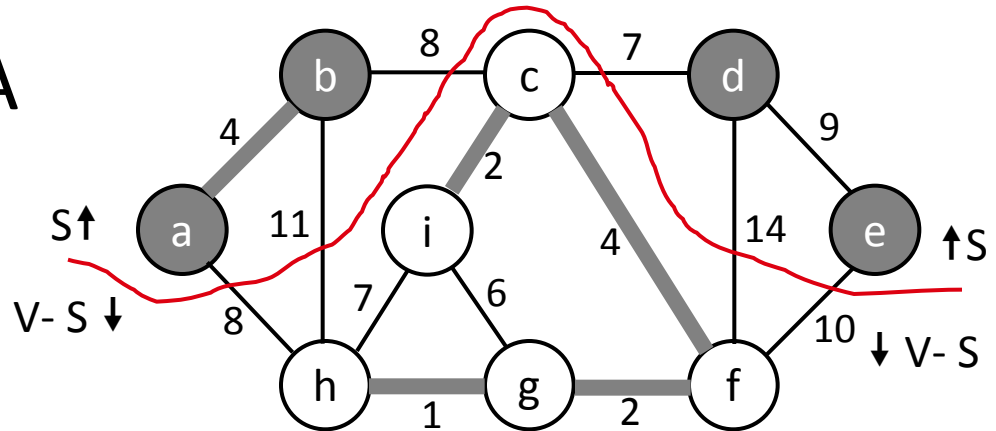
$(S, V - S)$ if one endpoint is in S

and the other in $V - S$



Definitions (cont'd)

- A cut **respects** a set A of edges \Leftrightarrow no edge in A crosses the cut



- An edge is a **light edge**

crossing a cut \Leftrightarrow its weight is minimum over all edges crossing the cut

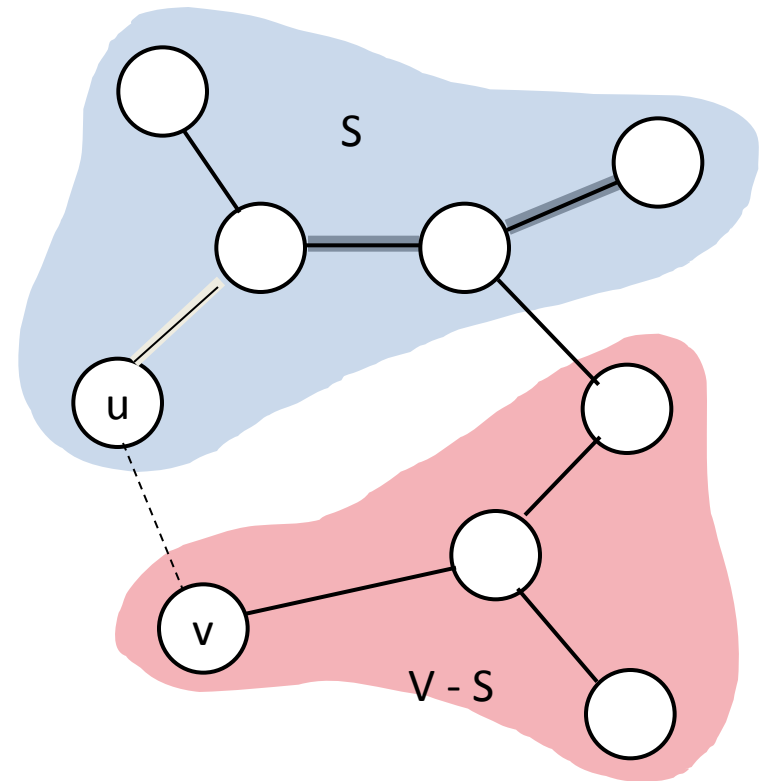
- Note that for a given cut, there can be > 1 light edges crossing it

Theorem

- Let A be a subset of some MST (i.e., T), $(S, V - S)$ be a **cut** that respects A , and (u, v) be a **light edge** crossing $(S, V - S)$. Then (u, v) is safe for A .

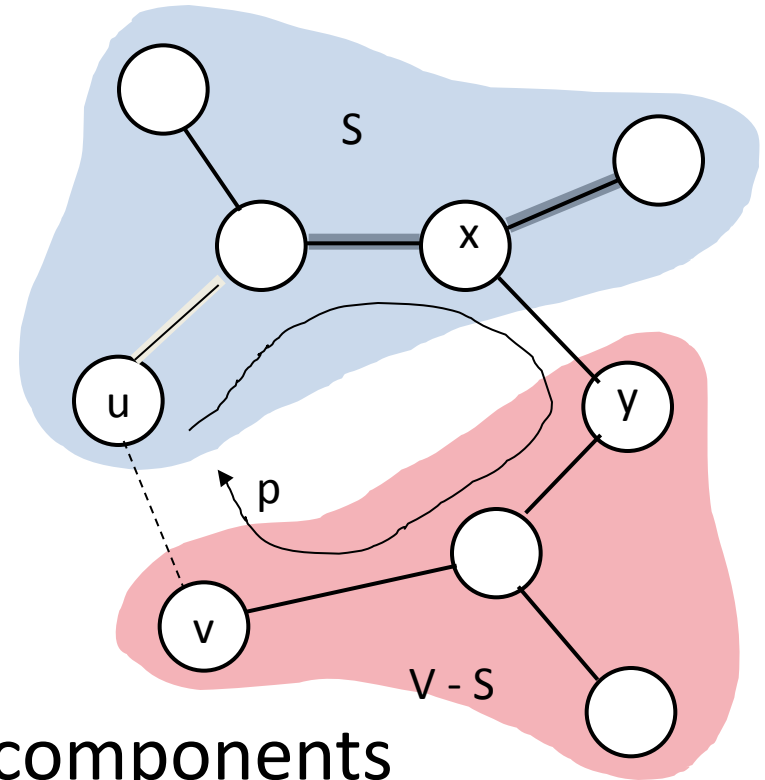
Proof:

- Let T be an MST that includes A
 - edges in A are shaded
- Case1: If T includes (u, v) , then it would be safe for A
- Case2: Suppose T does not include the edge (u, v)
- Idea**: construct another MST T' that includes $A \cup \{(u, v)\}$



Theorem - Proof

- T contains a unique path p between u and v
- Path p must cross the cut $(S, V - S)$ at least once: let (x, y) be that edge
- Let's remove $(x, y) \Rightarrow$ breaks T into two components.
- Adding (u, v) reconnects the components



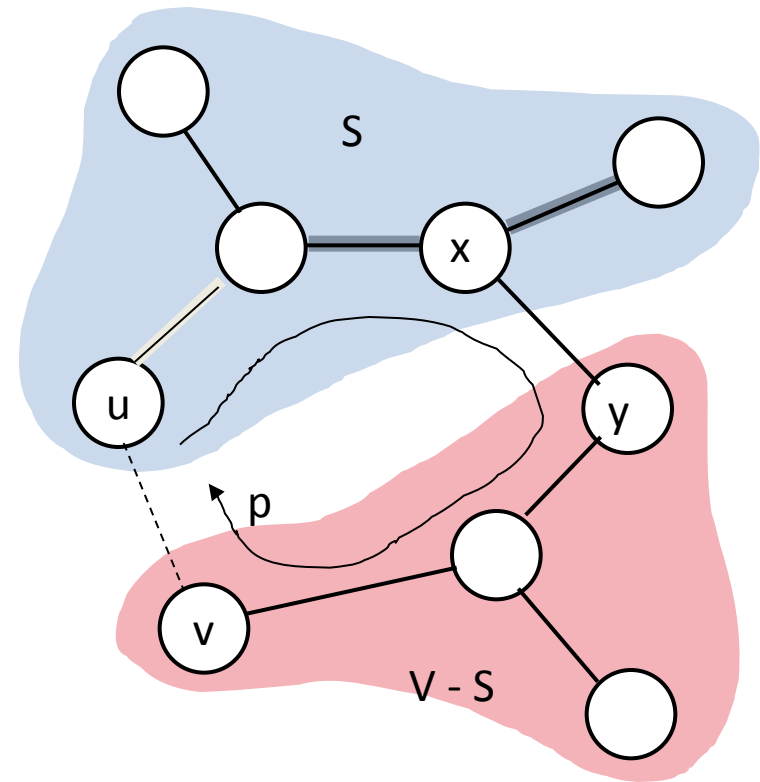
$$T' = T - \{(x, y)\} \cup \{(u, v)\}$$

Theorem – Proof (cont.)

$$T' = T - \{(x, y)\} \cup \{(u, v)\}$$

Have to show that T' is a MST:

- (u, v) is a light edge
 $\Rightarrow w(u, v) \leq w(x, y)$
- $w(T') = w(T) - w(x, y) + w(u, v) \leq w(T)$
- Since T is a spanning tree
 $w(T) \leq w(T') \Rightarrow T'$ must be an MST as well

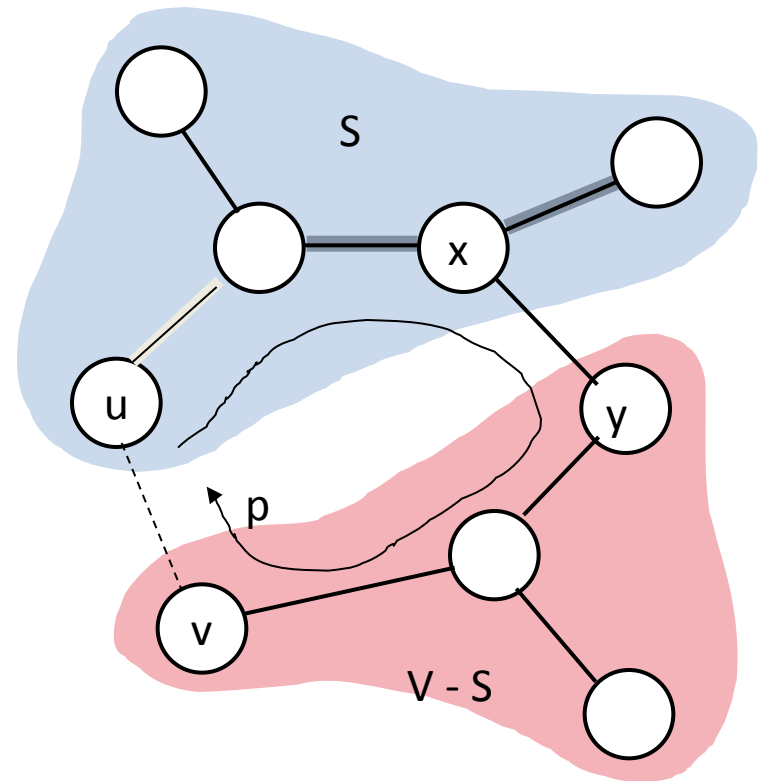


Theorem – Proof (cont.)

Need to show that (u, v) is safe for A :

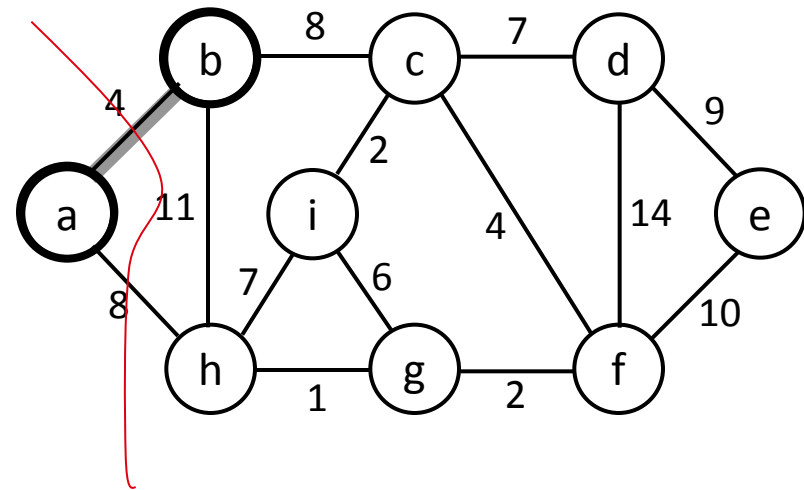
i.e., (u, v) can be a part of a MST

- $A \subseteq T$ and $(x, y) \notin T \Rightarrow$
 $(x, y) \notin A \Rightarrow A \subseteq T'$
- $A \cup \{(u, v)\} \subseteq T'$
- Since T' is a MST
 $\Rightarrow (u, v)$ is safe for A



Prim's Algorithm

- The edges in set A always form a single tree
- Starts from an arbitrary “root”: $V_A = \{a\}$
- At each step:
 - Find a light edge crossing $(V_A, V - V_A)$
 - Add this edge to A
 - Repeat until the tree spans all vertices

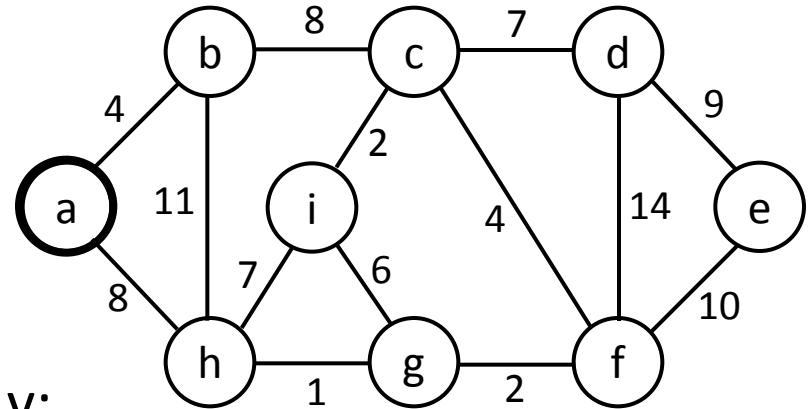


How to Find Light Edges Quickly?

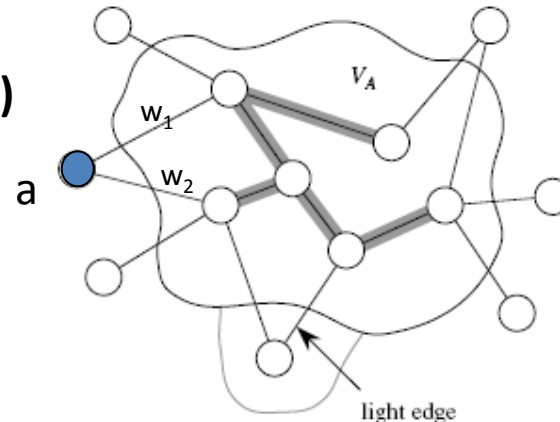
Use a priority queue Q:

- Contains vertices not yet included in the tree, i.e., $(V - V_A)$
 - $V_A = \{a\}$, $Q = \{b, c, d, e, f, g, h, i\}$
- We associate a key with each vertex v:

$\text{key}[v] = \text{minimum weight of any edge } (u, v)$
connecting v to V_A

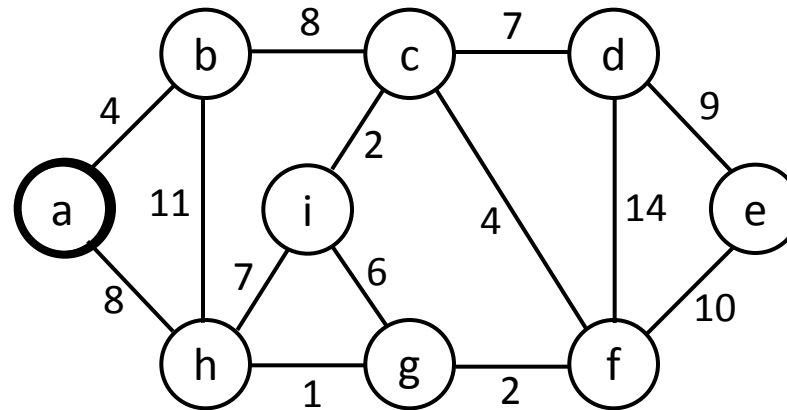


$$\text{Key}[a] = \min(w_1, w_2)$$

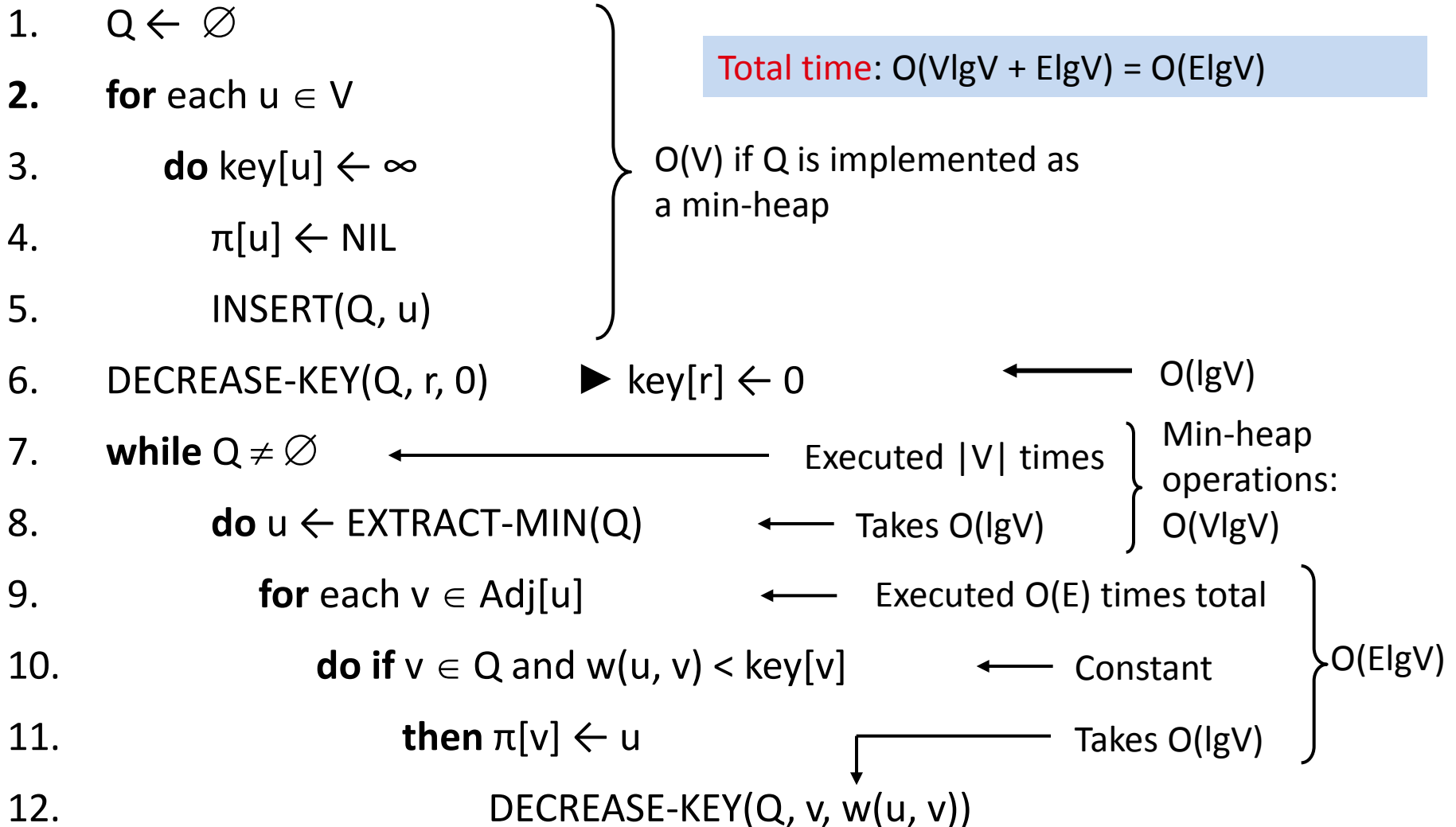


How to Find Light Edges Quickly? (cont.)

- After adding a new node to V_A we update the weights of all the nodes adjacent to it
e.g., after adding a to the tree, $k[b]=4$ and $k[h]=8$
- Key of v is ∞ if v is not adjacent to any vertices in V_A

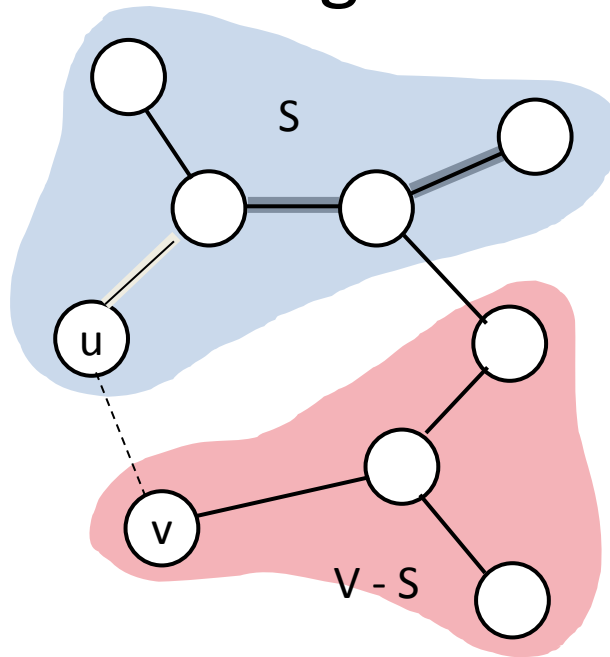


PRIM(V, E, w, r)

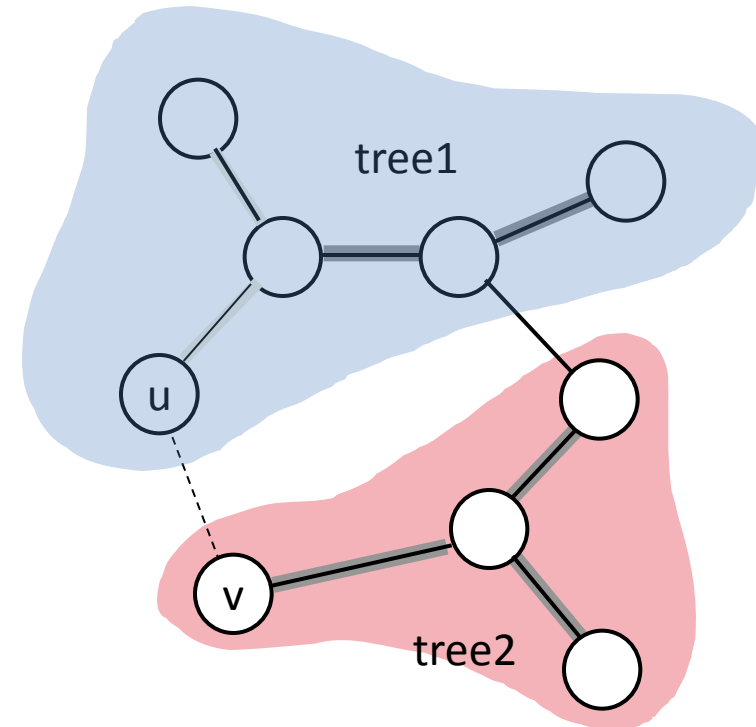


A different instance of the generic approach

(instance 1)



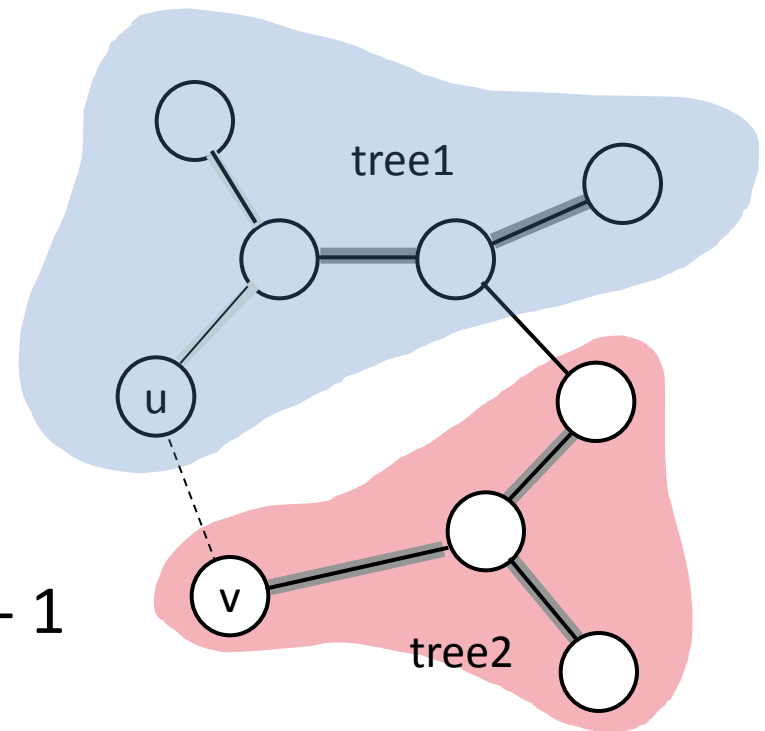
(instance 2)



- A is a forest containing connected components
 - Initially, each component is a single vertex
- Any safe edge merges two of these components into one
 - Each component is a tree

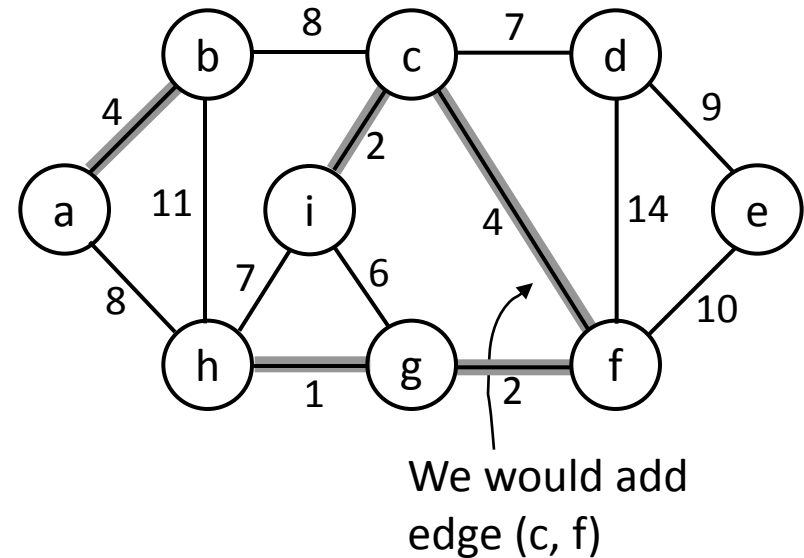
Kruskal's Algorithm

- How is it different from Prim's algorithm?
 - Prim's algorithm grows one tree all the time
 - Kruskal's algorithm grows multiple trees (i.e., a forest) at the same time.
 - Trees are merged together using **safe** edges
 - Since an MST has exactly $|V| - 1$ edges, after $|V| - 1$ merges, we would have only one component

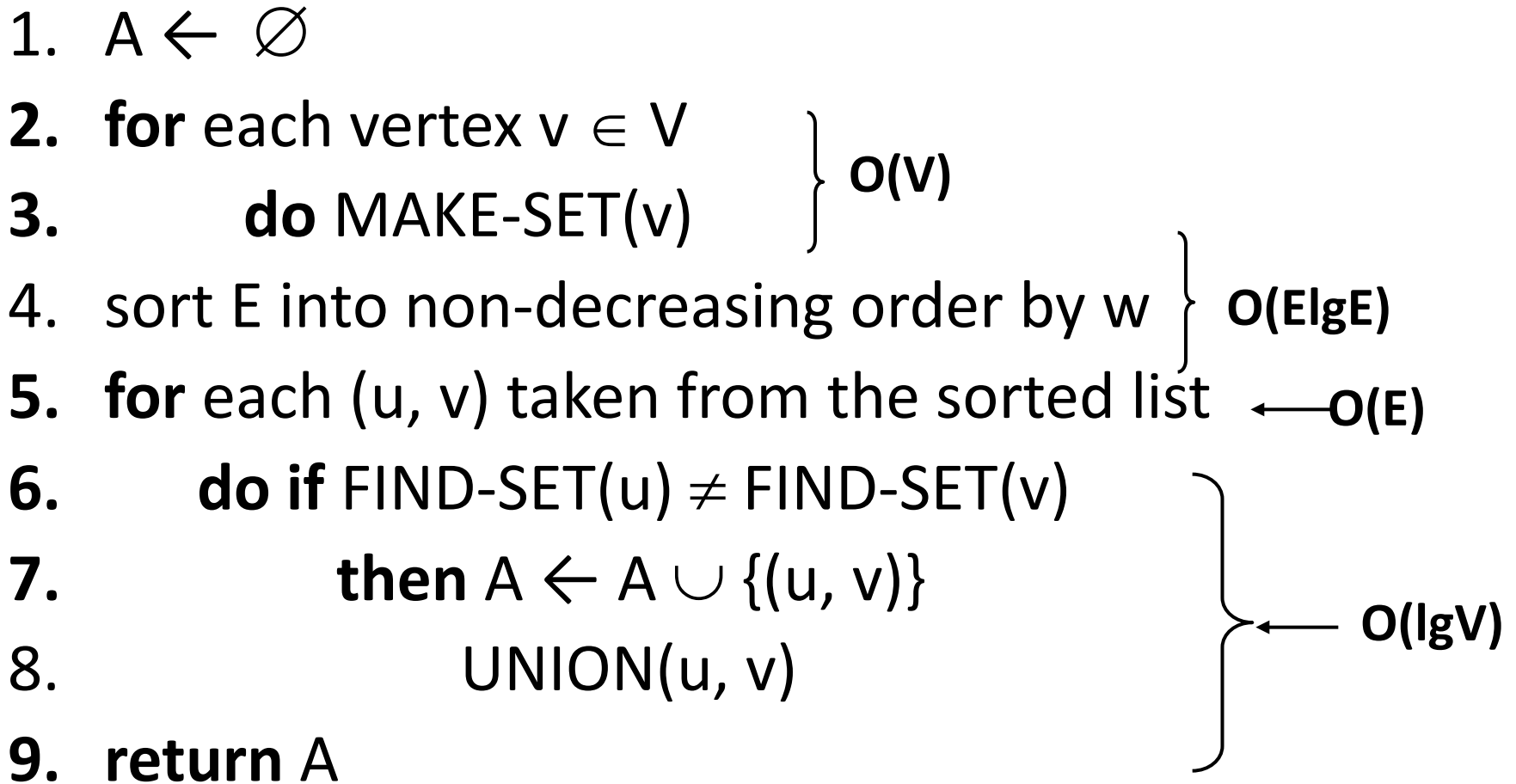


Kruskal's Algorithm

- Start with each vertex being its own component
- Repeatedly merge two components into one by choosing the **light** edge that connects them
- Which components to consider at each iteration?
 - Scan the set of edges in monotonically increasing order by weight



KRUSKAL(V, E, w)

1. $A \leftarrow \emptyset$
 2. **for** each vertex $v \in V$
 3. **do** MAKE-SET(v) } $O(V)$
 4. sort E into non-decreasing order by w } $O(E \lg E)$
 5. **for** each (u, v) taken from the sorted list ← $O(E)$
 6. **do if** FIND-SET(u) \neq FIND-SET(v)
 7. **then** $A \leftarrow A \cup \{(u, v)\}$
 8. UNION(u, v)
 9. **return** A
- 

Running time: $O(V + E \lg E + E \lg V) = O(E \lg E)$ – dependent on the implementation of the disjoint-set data structure

KRUSKAL(V, E, w) (cont.)

1. $A \leftarrow \emptyset$
 2. **for** each vertex $v \in V$
 3. **do** MAKE-SET(v) } $O(V)$
 4. sort E into non-decreasing order by w } $O(E \lg E)$
 5. **for** each (u, v) taken from the sorted list ← $O(E)$
 6. **do if** FIND-SET(u) \neq FIND-SET(v)
 7. **then** $A \leftarrow A \cup \{(u, v)\}$
 8. UNION(u, v)
 9. **return** A
- Running time: $O(V + E \lg E + E \lg V) = O(E \lg E)$
- Since $E = O(V^2)$, we have $\lg E = O(2 \lg V) = O(\lg V)$
-

In-Class Exercise/Question #1

- Suppose that some of the weights in a connected graph G are negative. Will Prim's algorithm still work? What about Kruskal's algorithm? Justify your answers.

In-Class Exercise/Question #2

- Find an algorithm for the “maximum” spanning tree. That is, given an undirected weighted graph G , find a spanning tree of G of maximum cost. Prove the correctness of your algorithm.

In-class Exercise

You are given a set $\{x_1, x_2, \dots, x_n\}$ of points that all lie on the same real line. Let us define a “unit-length closed interval” to be a line segment of length 1 that lies on the same real line as the points in the given set. This line segment “contains” all the points that lie along the segment, including the end points.

Describe a greedy algorithm that determines the smallest set of unit-length closed intervals that contains all of the given points.

Maximum Flow

Chapter 26

Flow Graph

- A common scenario is to use a graph to represent a “flow network” and use it to answer questions about material flows
- Flow is the rate that material moves through the network
- Each directed edge is a conduit for the material with some stated capacity
- Vertices are connection points but do not collect material
 - Flow into a vertex must equal the flow leaving the vertex, flow conservation

Sample Networks

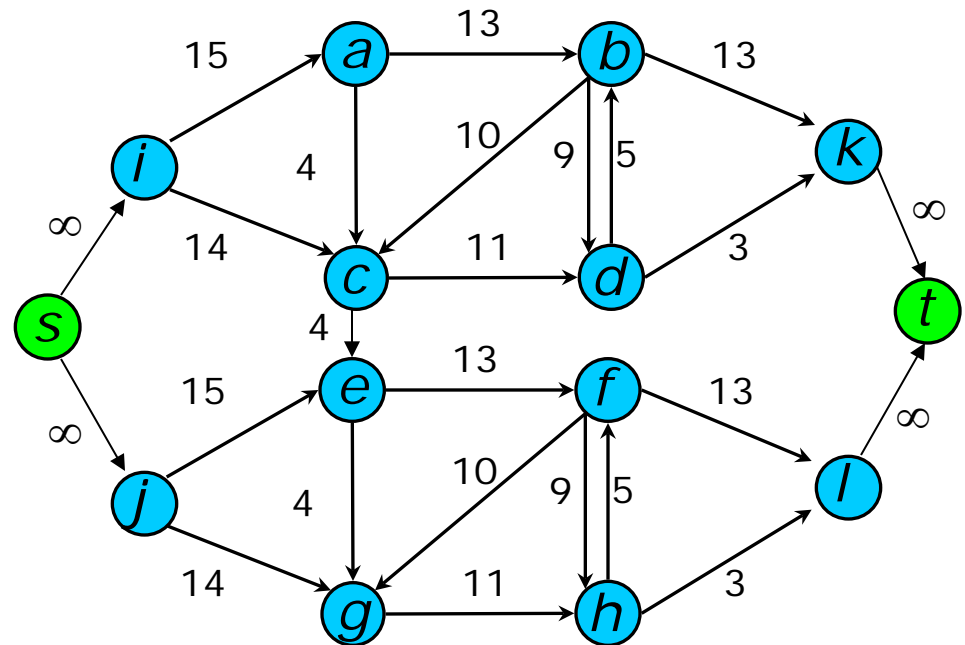
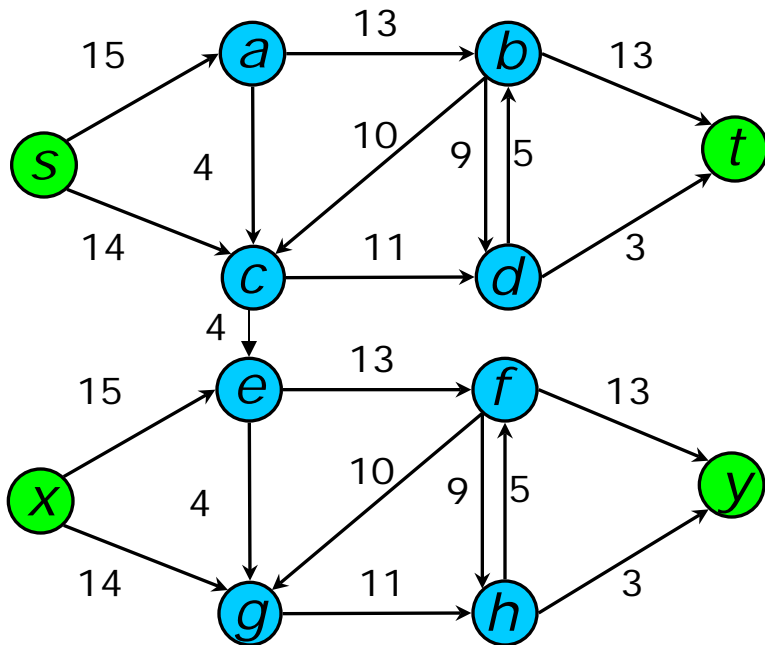
Network	Nodes	Arcs	Flow
communication	telephone exchanges, computers, satellites	cables, fiber optics, microwave relays	voice, video, packets
circuits	gates, registers, processors	wires	current
mechanical	joints	rods, beams, springs	heat, energy
hydraulic	reservoirs, pumping stations, lakes	pipelines	fluid, oil
financial	stocks, companies	transactions	money
transportation	airports, rail yards, street intersections	highways, railbeds, airway routes	freight, vehicles, passengers
chemical	sites	bonds	energy

Flow Concepts

- Source vertex s
 - where material is produced
- Sink vertex t
 - where material is consumed
- For all other vertices – what goes in must go out
 - Flow conservation
- **Goal: determine maximum rate of material flow from source to sink**

Multiple Sources or Sinks

- What if you have a problem with more than one source and more than one sink?
- Modify the graph to create a single supersource and supersink



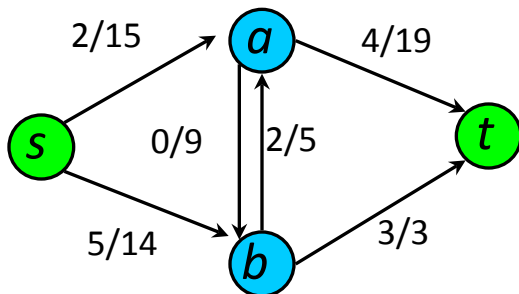
Formal Max Flow Problem

– Graph $G=(V,E)$ – a **flow network**

- Directed, each edge has **capacity** $c(u,v) \geq 0$
- Two special vertices: **source** s , and **sink** t
- For any other vertex v , there is a path $s \rightarrow \dots \rightarrow v \rightarrow \dots \rightarrow t$

– **Flow** – a function $f : V \times V \rightarrow \mathbf{R}$

- *Capacity constraint*: For all $u, v \in V$: $f(u,v) \leq c(u,v)$
- *Skew symmetry*: For all $u, v \in V$: $f(u,v) = -f(v,u)$
- *Flow conservation*: For all $u \in V - \{s, t\}$:

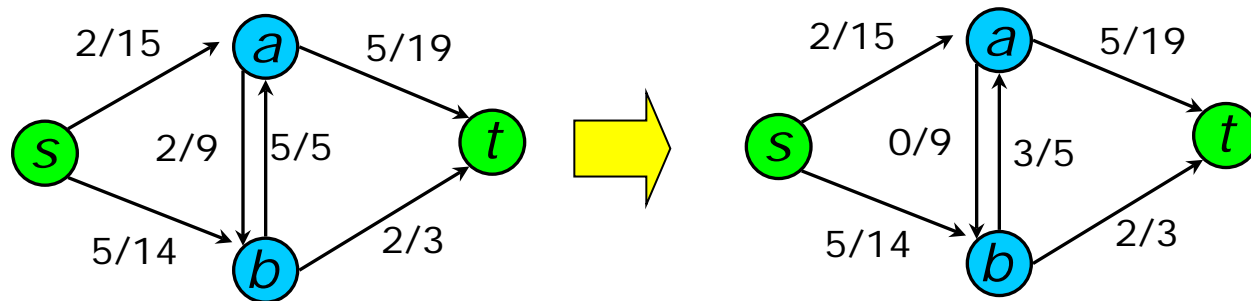


$$\sum_{v \in V} f(u,v) = f(u,V) = 0, \text{ or}$$

$$\sum_{v \in V} f(v,u) = f(V,u) = 0$$

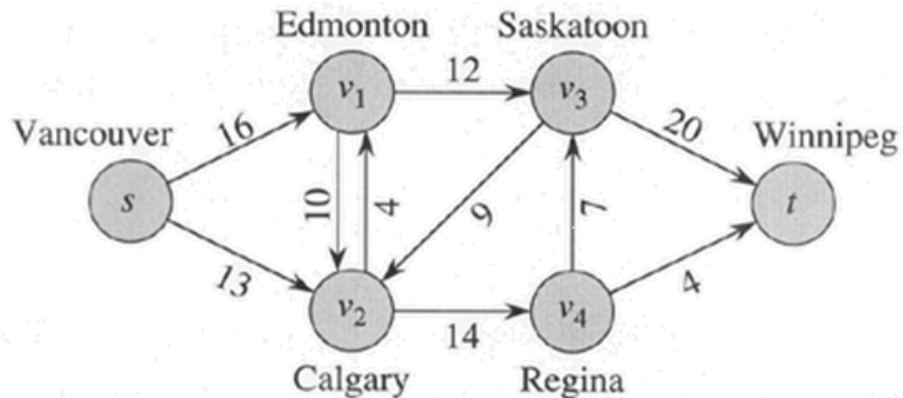
Cancellation of flows

- We would like to avoid two positive flows in opposite directions between the same pair of vertices
 - Such flows *cancel* (maybe partially) each other due to skew symmetry

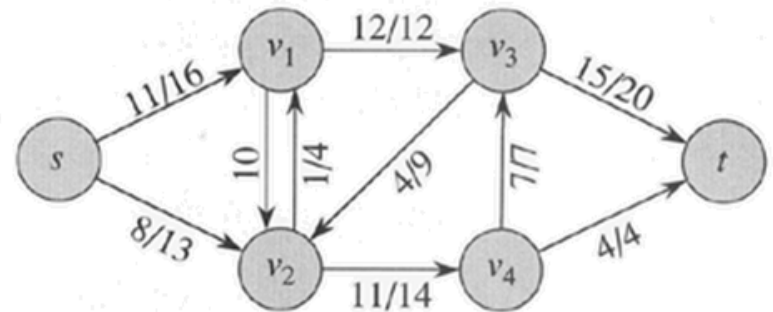


Max Flow

- We want to find a flow of maximum value from the source to the sink
 - Denoted by $|f|$



Lucky Puck Distribution Network



Max Flow, $|f| = 19$
Or is it?
Best we can do?

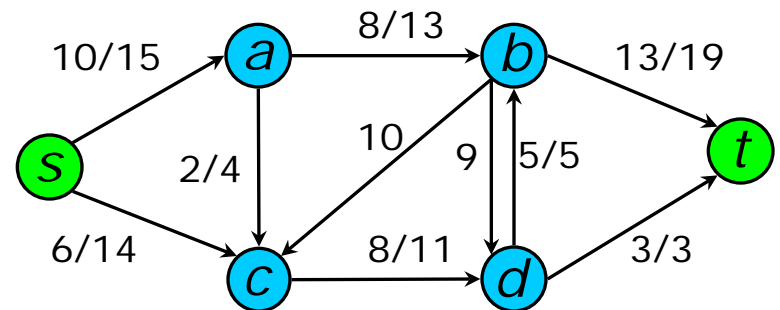
Ford-Fulkerson method

- Contains several algorithms:
 - Residual networks
 - Augmenting paths
 - Find a path p from s to t (**augmenting path**), such that there is some value $x > 0$, and for each edge (u,v) in p we can add x units of flow
 - $f(u,v) + x \leq c(u,v)$

FORD-FULKERSON-METHOD(G, s, t)

- 1 initialize flow f to 0
- 2 **while** there exists an augmenting path p
- 3 **do** augment flow f along p
- 4 **return** f

Augmenting Path?



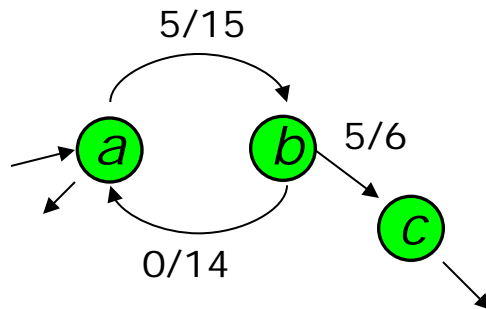
Residual Network

- To find augmenting path we can find any path in the **residual network**:
 - Residual capacities: $c_f(u,v) = c(u,v) - f(u,v)$
 - i.e. the actual capacity minus the net flow from u to v
 - Net flow may be negative
 - Residual network: $G_f=(V,E_f)$, where

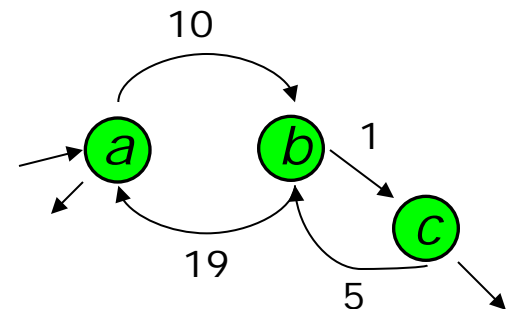
$$E_f = \{(u,v) \in V \times V : c_f(u,v) > 0\}$$
 - Observation – edges in E_f are either edges in E or their reversals:

$$|E_f| \leq 2|E|$$

Sub-graph with $c(u,v)$ and $f(u,v)$



Residual Sub-Graph



Residual Capacity and Augmenting Path

- Finding an Augmenting Path:
 - Find a path from s to t in the residual graph
 - The *residual capacity* of a path p in G_f :
$$c_f(p) = \min\{c_f(u,v) : (u,v) \text{ is in } p\}$$
 - i.e. find the minimum capacity along p
 - Doing augmentation: for all (u,v) in p , we just add this $c_f(p)$ to $f(u,v)$ (and subtract it from $f(v,u)$)
 - Resulting flow is a valid flow with a larger value.

Residual network and augmenting path

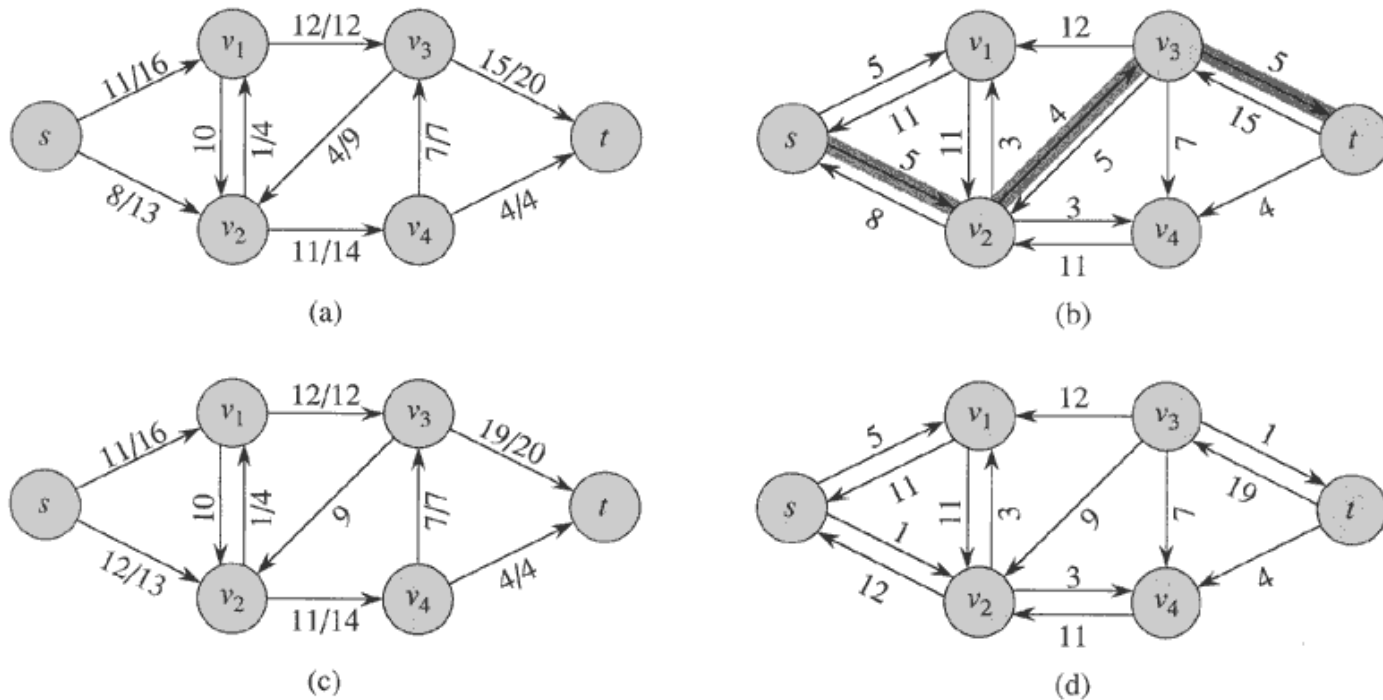


Figure 26.3 (a) The flow network G and flow f of Figure 26.1(b). (b) The residual network G_f with augmenting path p shaded; its residual capacity is $c_f(p) = c(v_2, v_3) = 4$. (c) The flow in G that results from augmenting along path p by its residual capacity 4. (d) The residual network induced by the flow in (c).

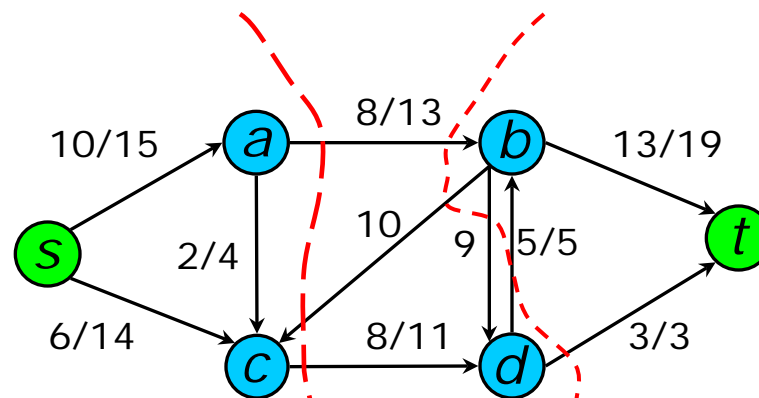
The Ford-Fulkerson method

```
Ford-Fulkerson( $G, s, t$ )
1 for each edge  $(u, v)$  in  $G.E$  do
2    $f(u, v) \leftarrow f(v, u) \leftarrow 0$ 
3 while there exists a path  $p$  from  $s$  to  $t$  in residual
   network  $G_f$  do
4    $c_f = \min\{c_f(u, v) : (u, v) \text{ is in } p\}$ 
5   for each edge  $(u, v)$  in  $p$  do
6      $f(u, v) \leftarrow f(u, v) + c_f$ 
7      $f(v, u) \leftarrow -f(u, v)$ 
8 return  $f$ 
```

The algorithms based on this method differ in how they choose p in step 3. If chosen poorly the algorithm might not terminate.

Cuts

- Does the method find the minimum flow?
 - Yes, if we get to the point where the residual graph has no path from s to t
 - A **cut** is a partition of V into S and $T = V - S$, such that $s \in S$ and $t \in T$
 - The **net flow** ($f(S,T)$) through the cut is the sum of flows $f(u,v)$, where $s \in S$ and $t \in T$
 - Includes negative flows back from T to S
 - The **capacity** ($c(S,T)$) of the cut is the sum of capacities $c(u,v)$, where $s \in S$ and $t \in T$
 - The sum of positive capacities
 - **Minimum cut** – a cut with the smallest capacity of all cuts.
- $|f| = f(S,T)$ i.e. the value of a max flow is equal to the capacity of a min cut.



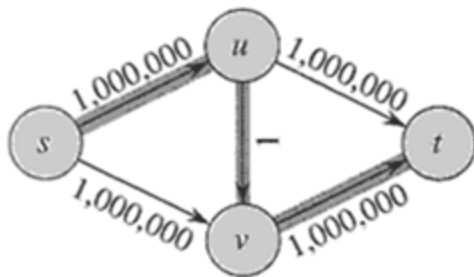
Cut capacity = 24 Min Cut capacity = 21

Max Flow / Min Cut Theorem

1. Since $|f| \leq c(S,T)$ for all cuts of (S,T) then if $|f| = c(S,T)$ then $c(S,T)$ must be the min cut of G
2. This implies that f is a maximum flow of G
3. This implies that the residual network G_f contains no augmenting paths.
 - If there were augmenting paths this would contradict that we found the maximum flow of G
 - $1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \dots$ and from $2 \rightarrow 3$ we have that the Ford Fulkerson method finds the maximum flow if the residual graph has no augmenting paths.

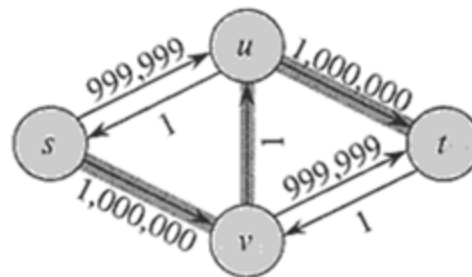
Worst Case Running Time

- Assuming integer flow
- Each augmentation increases the value of the flow by some positive amount.
- Augmentation can be done in $O(E)$.
- Total worst-case running time $O(E |f^*|)$, where f^* is the max-flow found by the algorithm.
- Example of worst case:



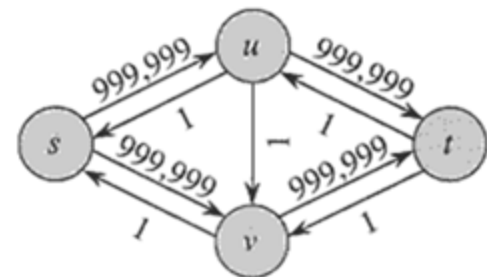
(a)

Augmenting path of 1



(b)

Resulting Residual Network



(c)

Resulting Residual Network

Edmonds Karp

- Take **shortest path** (in terms of number of edges) as an augmenting path – Edmonds-Karp algorithm
 - How do we find such a shortest path?
 - Running time $O(VE^2)$, because the number of augmentations is $O(VE)$
 - Skipping the proof here