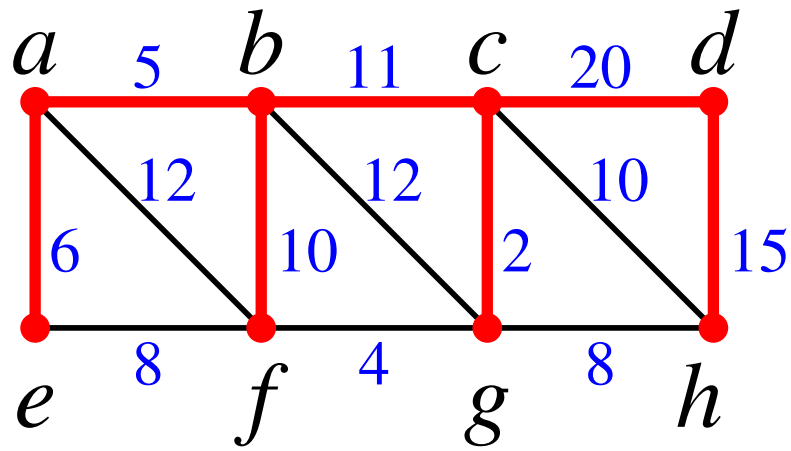
Chapter 3.5 – 3.6
Weighted graph algorithms

Prof. Tesler

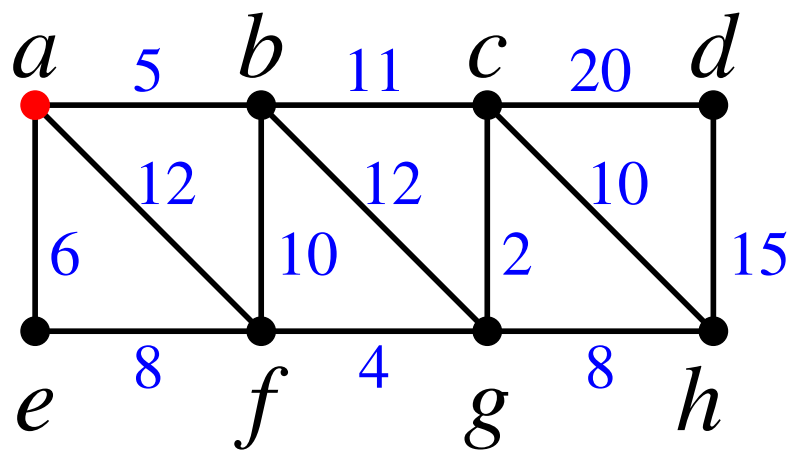
Math 154
Winter 2020

Minimum weight spanning trees



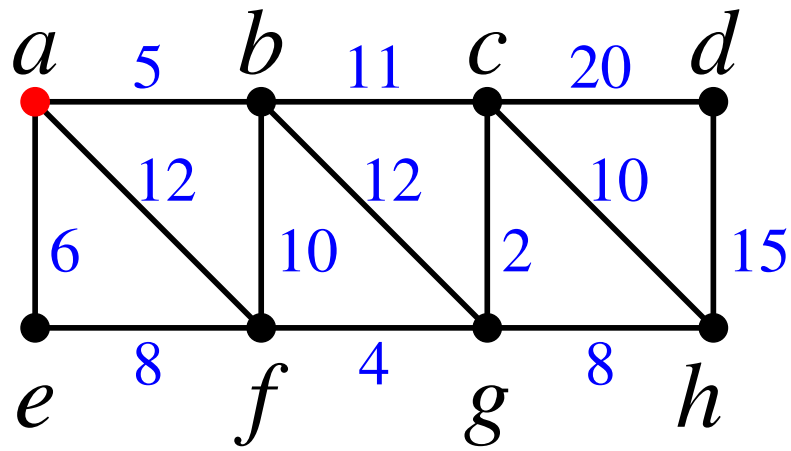
- Each edge is assigned a *weight* (or *cost*), denoted $\omega(e)$. It's a nonnegative real number, shown on each edge above.
- The *weight* of a spanning tree is the sum of the weights of its edges.
- The spanning tree shown in red above has weight
$$6 + 5 + 10 + 11 + 2 + 20 + 15 = 68.$$
- We'll cover two algorithms to find a minimum weight spanning tree in a connected graph, *Prim's Algorithm* and *Kruskal's Algorithm*.

Prim's Algorithm



- We'll build a tree T as a subgraph of G.
- Pick a vertex to start at. We'll use a.
- Initialize T to just vertex a (in red) and no edges.

Prim's Algorithm



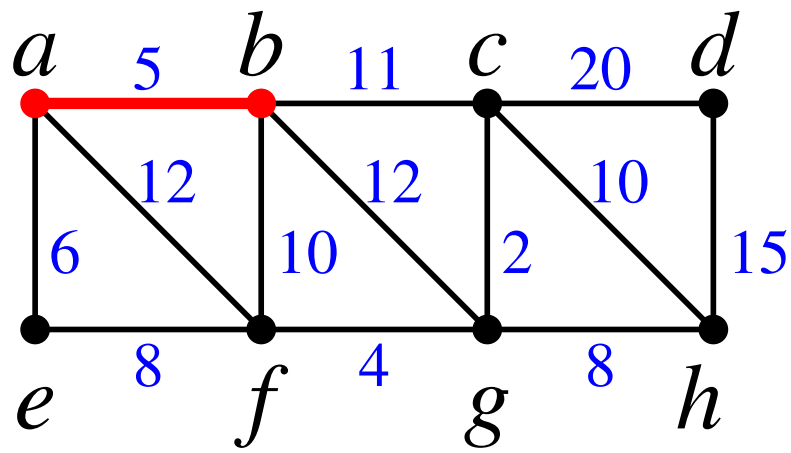
- Consider all edges with one vertex in the tree and one vertex not in the tree:

$$\{a, b\}, \quad \{a, f\}, \quad \{a, e\}$$

- Of those, pick one of smallest weight (if there's a tie, pick arbitrarily):

$$\{a, b\}$$

Prim's Algorithm



- Consider all edges with one vertex in the tree and the other isn't:

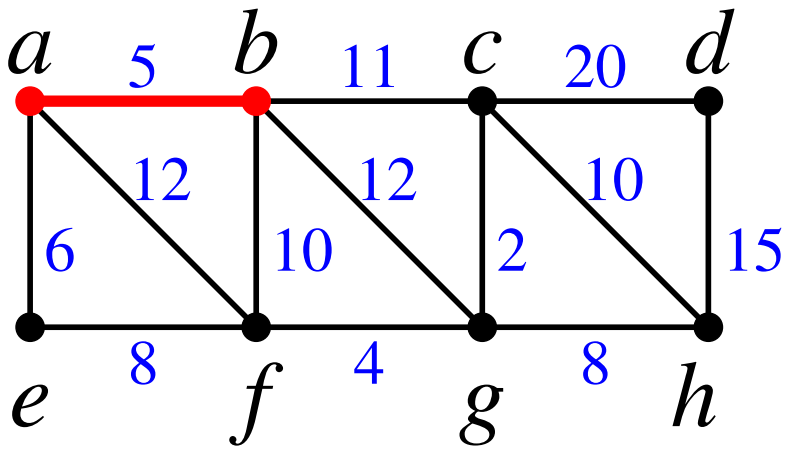
$$\{a,b\}, \quad \{a,f\}, \quad \{a,e\}$$

- Of those, pick one of smallest weight (break ties arbitrarily):

$$\{a,b\}$$

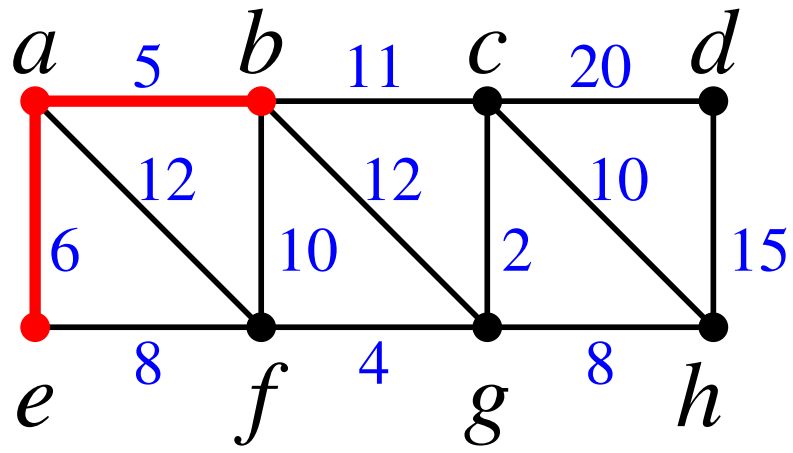
- Add it to the tree.
- Repeat until all vertices are in the tree.

Prim's Algorithm



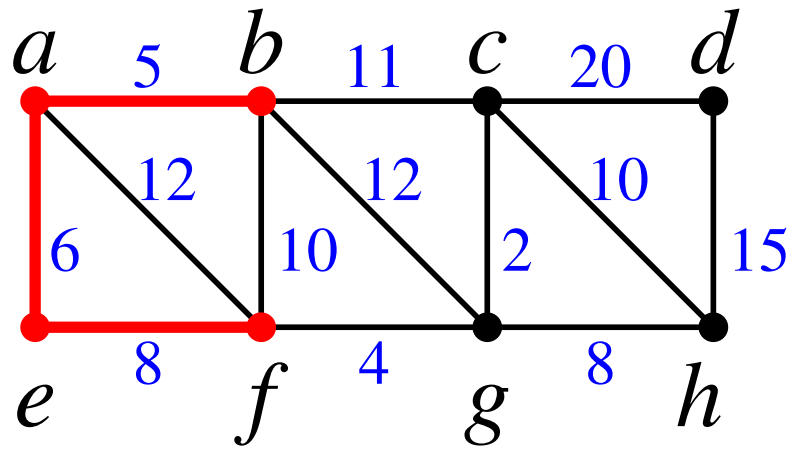
Weight: 5

Prim's Algorithm



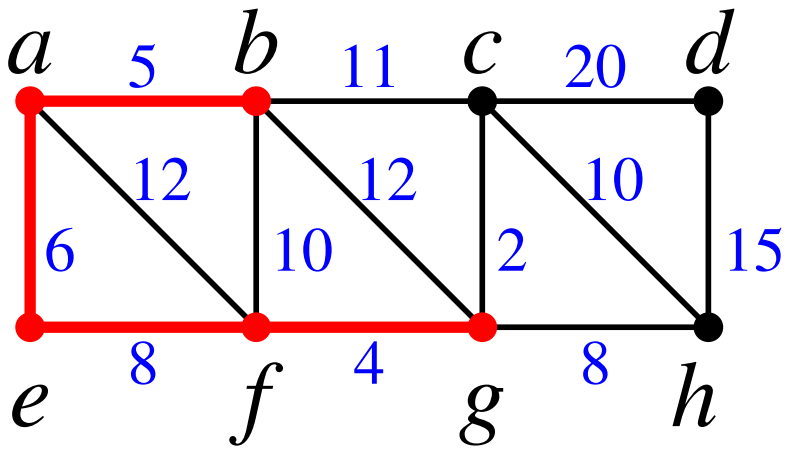
Weight: 11

Prim's Algorithm



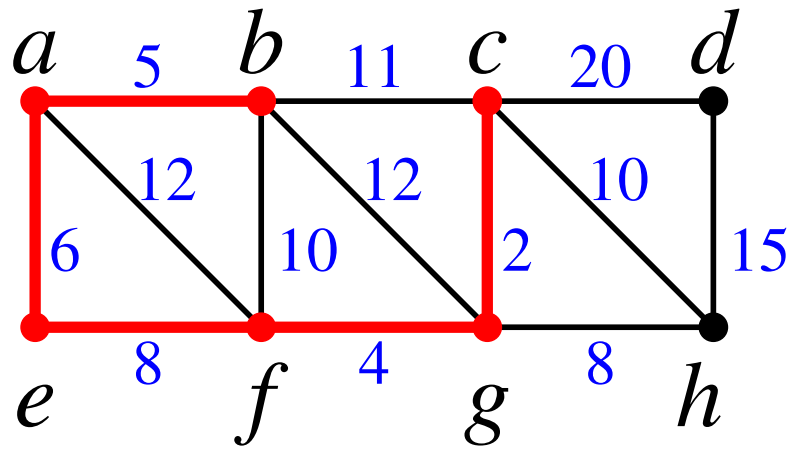
Weight: 19

Prim's Algorithm



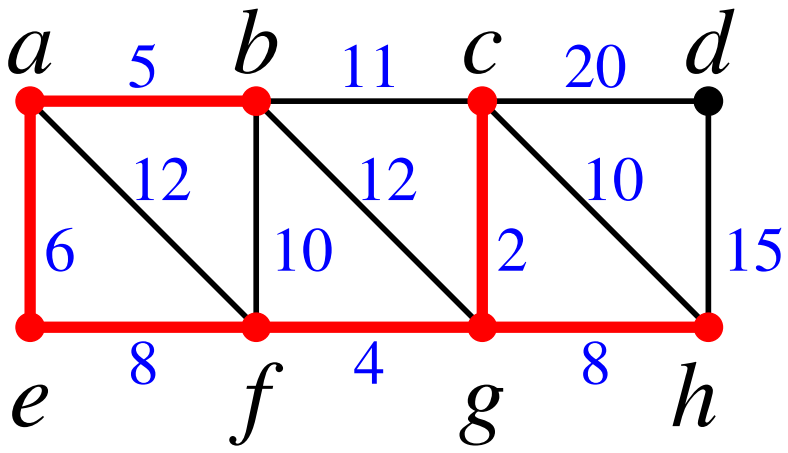
Weight: 23

Prim's Algorithm



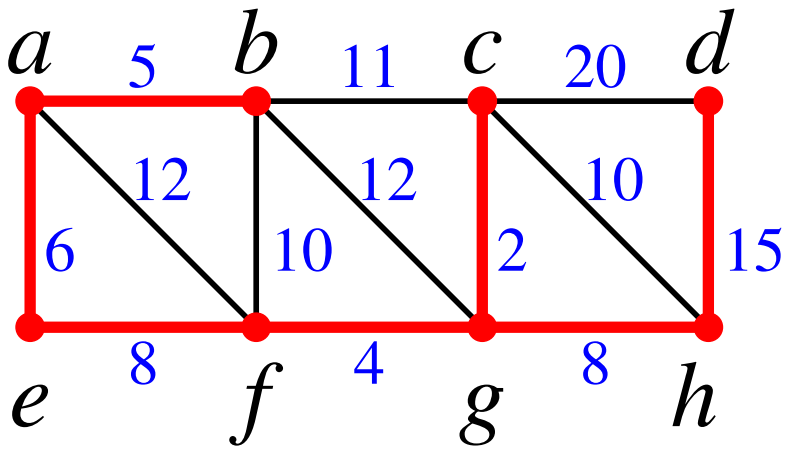
Weight: 25

Prim's Algorithm



Weight: 33

Prim's Algorithm



Weight: 48

Greedy Algorithms

- Prim's Algorithm (and the next two algorithms coming up) is a *greedy algorithm*.
- We use the optimal *local solution* for each step.
- We hope this gives the optimal *global solution* at the end, but greedy algorithms aren't guaranteed to work; it has to be proved for each specific algorithm.

Greedy algorithm for driving in rush hour

- At each intersection or fork, take the option (straight or a turn) that looks fastest and goes towards your destination.
- The overall time may be worse, since you could hit a traffic jam, lower speed limits, etc. in that direction.
- A GPS navigation system with real-time traffic information for the whole city considers all routes instead of just this one step.

Proof of Prim's Algorithm

- **Claim 1:** At each stage of Prim's Algorithm, T is a tree.
  - By the way we added edges (one vertex in T, the other vertex not in T), T is connected and doesn't have any cycles, so it's a tree.
- **Claim 2:** At each stage of Prim's Algorithm, T is contained in a minimum weight spanning tree of G.
  - We will prove this by induction on $m = |E(T)|$.

Proof of Prim's Algorithm

Claim

At each stage of Prim's Algorithm, T is contained in a minimum weight spanning tree of G.

Proof (base case):

- We prove this by induction on $m = |E(T)|$.
- When we start, T has one vertex and no edges, so it's contained in every minimum spanning tree.
- Since the graph is finite, there is a finite number of spanning trees (even if it's large), so there is one achieving a minimum weight.

Proof of Prim's Algorithm

Claim

At each stage of Prim's Algorithm, T is contained in a minimum weight spanning tree of G.

Proof (induction step):

- Tree T has m edges ($0 \leqslant m < |E(G)|$), and by the induction hypothesis, it's contained in a minimum weight spanning tree S.
- We select an edge e to add to T, to yield $T \cup \{e\}$ with $m + 1$ edges.
- **Case 1:** If $T \cup \{e\} \subset S$, we're done.
- **Case 2:** If $T \cup \{e\} \not\subset S$, we will show how to construct a different minimum weight spanning tree containing $T \cup \{e\}$.

Proof of Prim's Algorithm

Claim

At each stage of Prim's Algorithm, T is contained in a minimum weight spanning tree of G.

Proof (induction step, Case 2):

- S is a spanning tree and e isn't in S, so $S \cup \{e\}$ has a cycle C.
- $C - e$ is a path in S with one end in T and the other end not in T.
- So, somewhere along path $C - e$, there's at least one edge f that connects a vertex in T to a vertex not in T.
- Since e is a minimum weight edge connecting a vertex in T to a vertex not in T, we must have $\omega(e) \leqslant \omega(f)$.

Proof of Prim's Algorithm

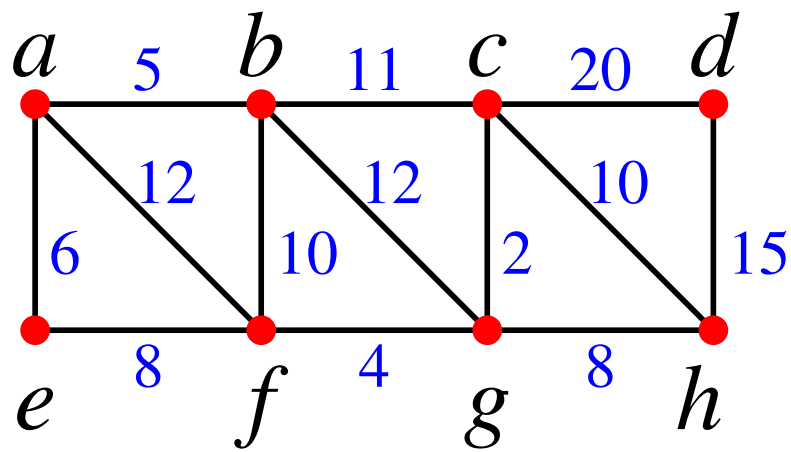
Claim

At each stage of Prim's Algorithm, T is contained in a minimum weight spanning tree of G.

Proof (induction step, Case 2, continued):

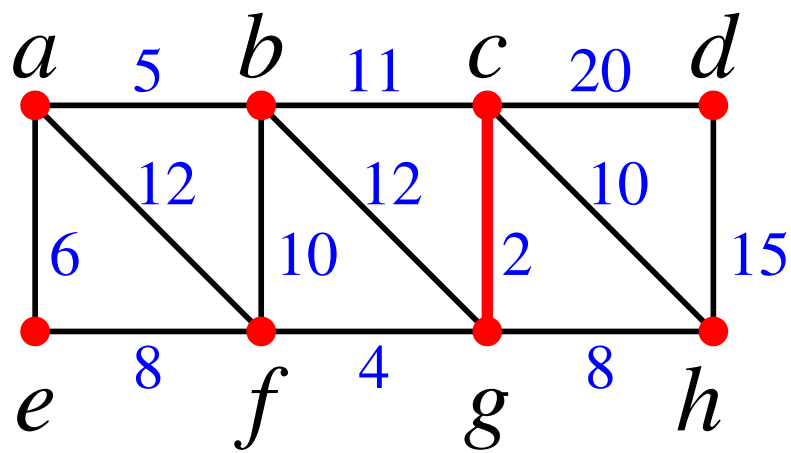
- Let $S' = (S \cup \{e\}) - \{f\}$.
  - We create a cycle by adding e, and break the cycle by removing f.
  - So S' is also a spanning tree of G (not necessarily min. weight), and S' contains $T \cup \{e\}$.
- $\omega(S') = \omega(S) + \omega(e) - \omega(f)$, and $\omega(S') \leqslant \omega(S)$ since $\omega(e) \leqslant \omega(f)$.
- But S is a minimum weight spanning tree, so $\omega(S) \leqslant \omega(S')$.
- Thus, $\omega(S) = \omega(S')$, so S' is also a minimum weight spanning tree of G, and it contains $T \cup \{e\}$. The claim has been proved.

Kruskal's Algorithm



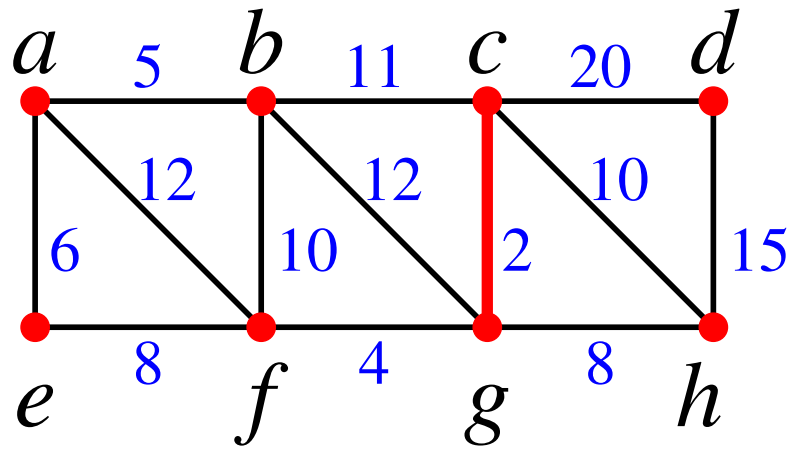
- We'll build a forest F (red) as a subgraph of G.
- Initialize F to all of the vertices of G, but no edges.

Kruskal's Algorithm



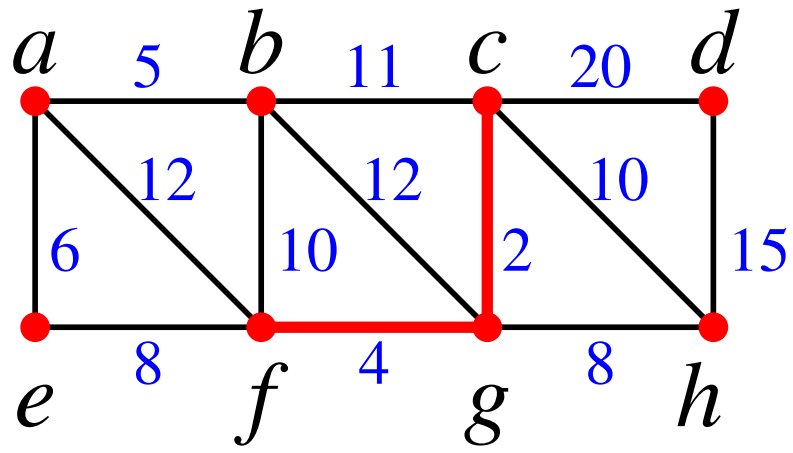
- Pick an edge of min. weight connecting different components of F: $\{c, g\}$.
- Add the edge to F.
- Repeat until there are no edges connecting different components.
- At each step, F is a forest.
- **If G is connected:** The final result is a minimum weight spanning tree (similar proof to Prim's Algorithm).
- **If G isn't connected:** The final result is a forest with a minimum weight spanning tree of each connected component.

Kruskal's Algorithm



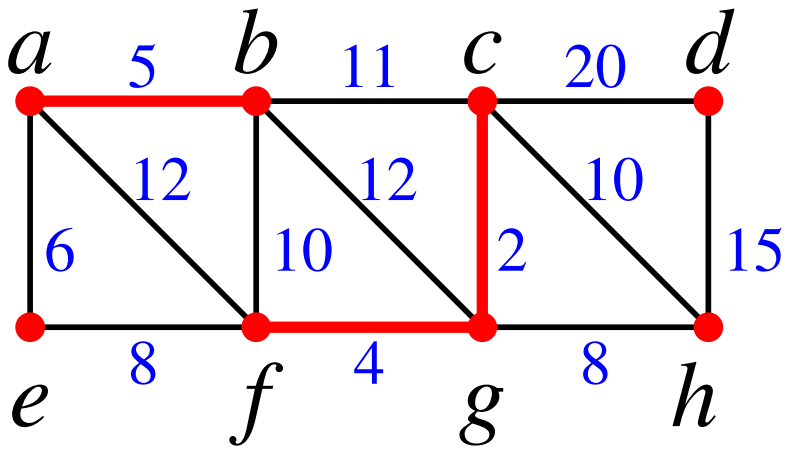
Weight: 2

Kruskal's Algorithm



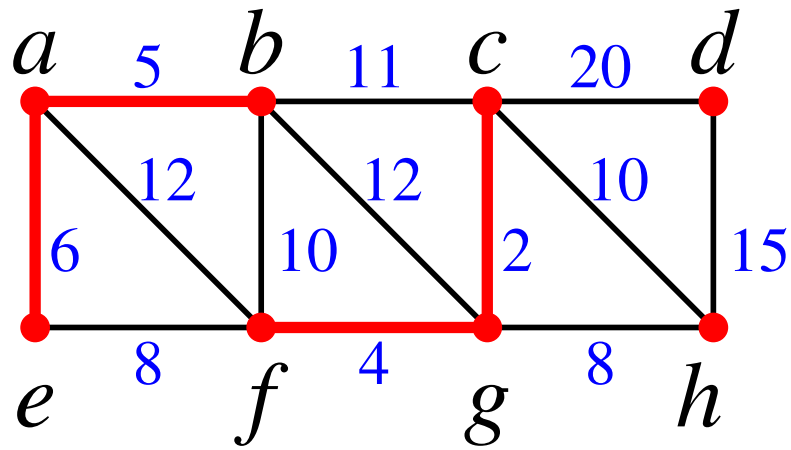
Weight: 6

Kruskal's Algorithm



Weight: 11

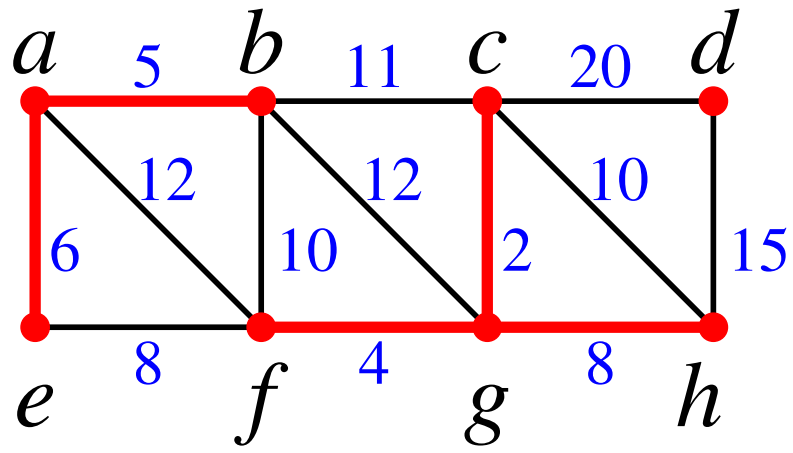
Kruskal's Algorithm



Weight: 17

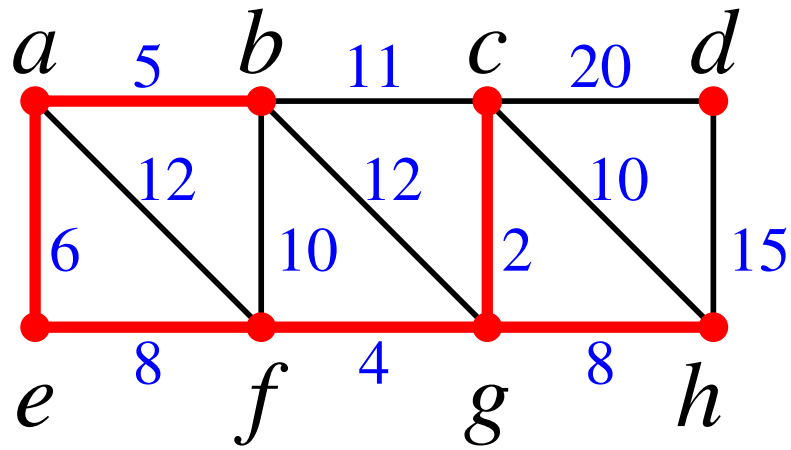
- Two edges are candidates, $\{e, f\}$ and $\{g, h\}$, both with weight 8. Pick either one.

Kruskal's Algorithm



Weight: 25

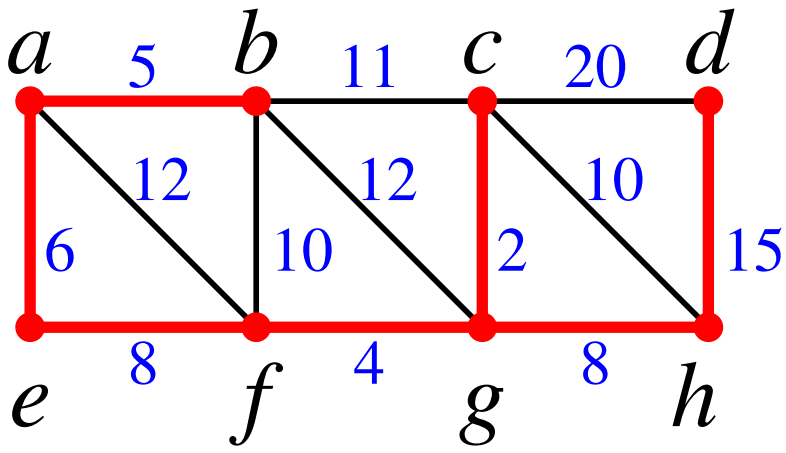
Kruskal's Algorithm



Weight: 33

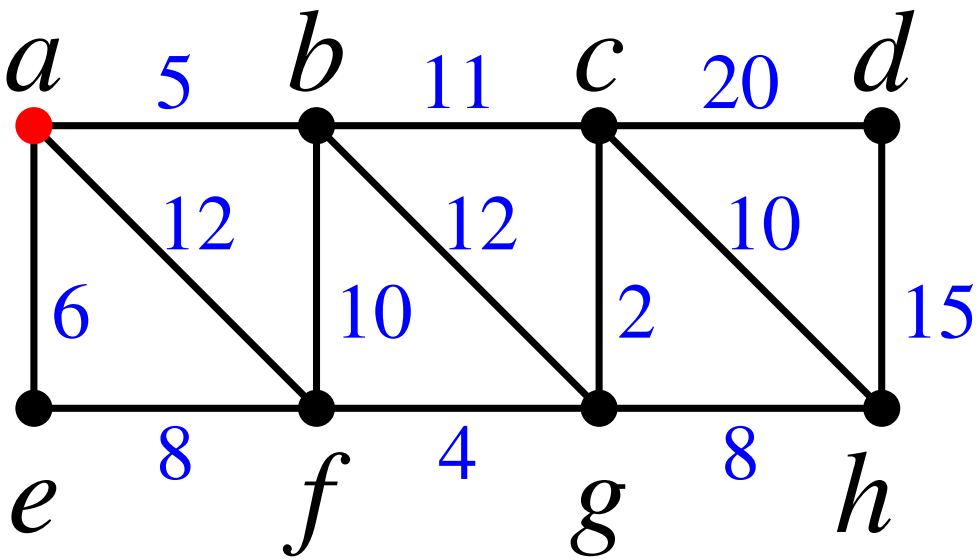
- The edges with vertices in different components are
  $\{c, d\}$ (weight 20) and $\{d, h\}$ (weight 15)
  .
- There are lower weight edges not in F, but adding any of them would form a cycle.

Kruskal's Algorithm



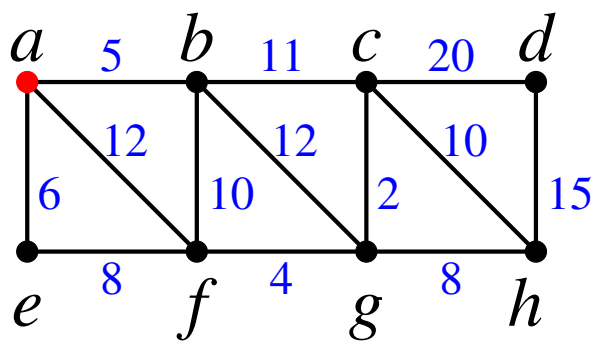
Weight: 48

Dijkstra's Algorithm



- Pick a vertex; we'll use a.
- We want to find minimum weight paths from a to all vertices.
- Minimum length paths (in # edges) corresponds to all edges having weight 1. We already solved that using breadth first search.

Dijkstra's Algorithm



v	a	b	c	d	e	f	g	h
dist(v)	0	∞	∞	∞	∞	∞	∞	∞
prev(v)	—	—	—	—	—	—	—	—

$$W = \{a, b, c, d, e, f, g, h\}$$

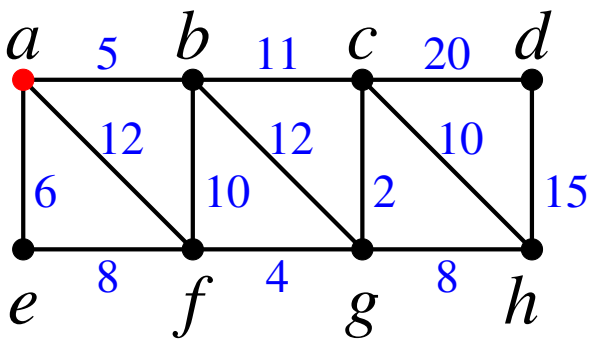
Create two arrays concerning the best path (smallest weight) we have found so far from a to v, for each vertex v:

- dist(v) is its weight.
  The only path we know so far is a to itself (one vertex, length 0).
  Initialize to dist$(a) = 0$ and dist$(v) = \infty$ for all other vertices.

- prev$(v) = u$ means that its last edge is $\{u, v\}$.
  Initialize to prev$(v) =$ undefined for all vertices.
  We will see that prev$()$ encodes a tree.

- Also initialize $W = V(G)$. We will examine vertices one at a time.

Dijkstra's Algorithm

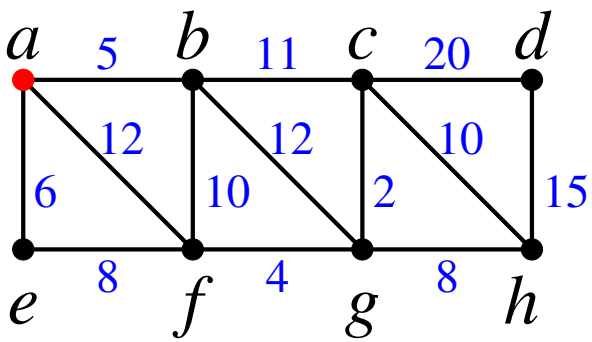


v	a	b	c	d	e	f	g	h
dist(v)	0	∞	∞	∞	∞	∞	∞	∞
prev(v)	—	—	—	—	—	—	—	—

$$W = \{a, b, c, d, e, f, g, h\}$$

- Pick any vertex $u \in W$ with dist(u) minimum. Here, $u = a$.
- Remove u from W.
- For each neighbor $v \in N_G(u)$:
  - Consider the best path from a to u so far, plus edge $\{u, v\}$. This goes from a to v.
  - It has weight $\quad$ dist$(u) + \omega(\{u, v\})$.
  - If this is smaller than dist(v), then set
    $$\text{dist}(v) = \text{dist}(u) + \omega(\{u, v\}) \quad \text{and} \quad \text{prev}(v) = u.$$
- Repeat the above steps until W is emptied.

Dijkstra's Algorithm

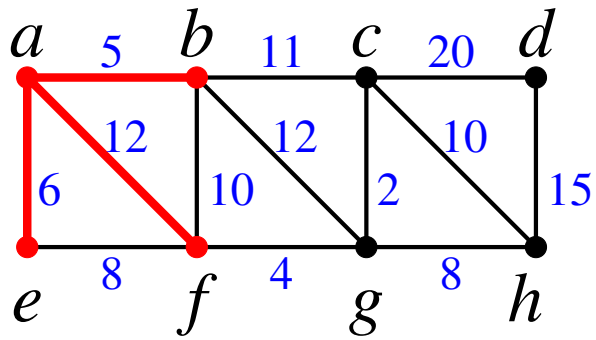


v	a	b	c	d	e	f	g	h
dist(v)	0	∞	∞	∞	∞	∞	∞	∞
prev(v)	—	—	—	—	—	—	—	—

$$W = \{a, b, c, d, e, f, g, h\}$$

- **Vertex:** $u = a$
- **Neighbors:** b, f
- **Paths through u to neighbors:**
  - **To b:** $0 + 5 = 5 < \infty$, so set dist$(b) = 5$ and prev$(b) = a$.
  - **To e:** $0 + 6 = 6 < \infty$, so set dist$(e) = 6$ and prev$(e) = a$.

Dijkstra's Algorithm

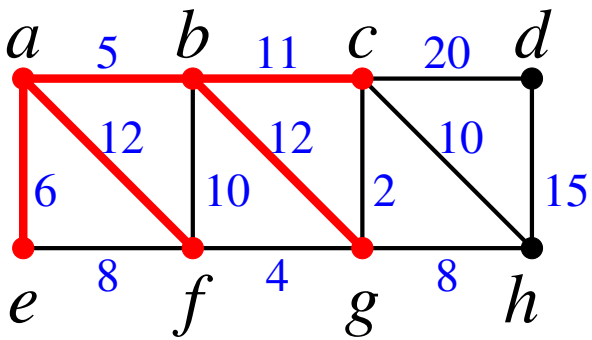


v	a	b	c	d	e	f	g	h
dist(v)	0	5	∞	∞	6	12	∞	∞
prev(v)	$-$	a	$-$	$-$	a	a	$-$	$-$

$$W = \{b, c, d, e, f, g, h\}$$

- **Vertex:** $u = b$
- **Neighbors:** a, c, f, g
- **Paths through u to neighbors:**
  - **To a:** $5 + 5 = 10 > 0$, so it's not better than the prior best path to a. Also, it goes back to u's parent ($a - b - a$), so it's not a path. We can always skip u's parent; it never improves the prior best path.
  - **To c:** $5 + 11 = 16 < \infty$, so set dist$(c) = 16$ and prev$(c) = b$.
  - **To f:** $5 + 10 = 15 > 12$, so it's not better.
  - **To g:** $5 + 12 = 17 < \infty$, so set dist$(g) = 17$ and prev$(e) = b$.

Dijkstra's Algorithm

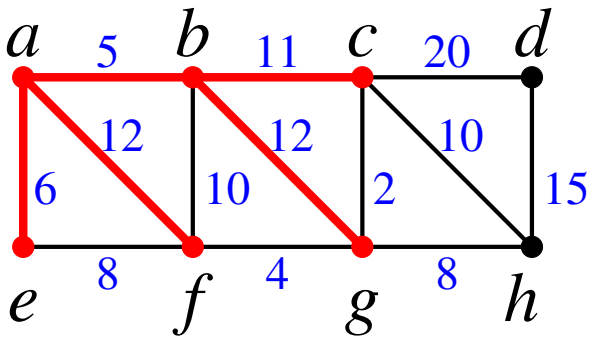


v	a	b	c	d	e	f	g	h
dist(v)	0	5	16	∞	6	12	17	∞
prev(v)	$-$	a	b	$-$	a	a	b	$-$

$$W = \{c, d, e, f, g, h\}$$

- **Vertex:** $u = e$
- **Neighbors:** $\cancel{a}, f$
- **Paths through u to neighbors:**
  - **To f:** $6 + 8 = 14 > 12$, so it's not better.

Dijkstra's Algorithm

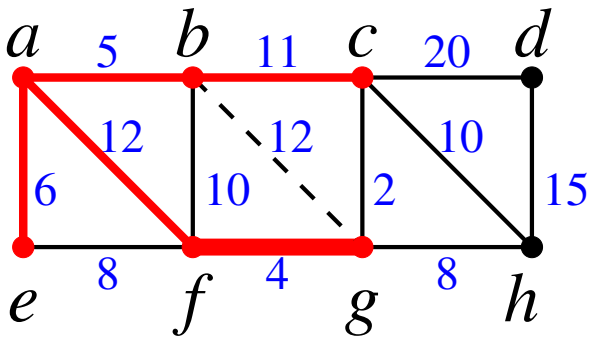


v	a	b	c	d	e	f	g	h
dist(v)	0	5	16	∞	6	12	17	∞
prev(v)	—	a	b	—	a	a	b	—

$$W = \{c, d, f, g, h\}$$

- **Vertex:** $u = f$
- **Neighbors:** $\cancel{a}, b, e, g$
- **Paths through u to neighbors:**
  - **To b:** $12 + 10 = 22 > 5$, so it's not better.
  - **To e:** $12 + 8 = 20 > 6$, so it's not better.
  - **To g:** $12 + 4 = 16 < 17$, so it's better than the prior path!
    Set dist$(g) = 16$ and prev$(g) = f$.
    This replaces the edge we previously had to g.

Dijkstra's Algorithm

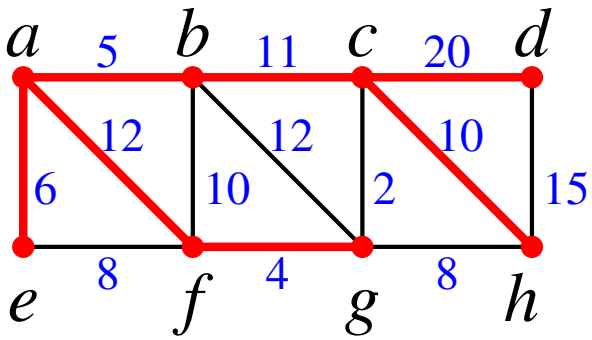


v	a	b	c	d	e	f	g	h
dist(v)	0	5	16	∞	6	12	16	∞
prev(v)	$-$	a	b	$-$	a	a	f	$-$

$$W = \{c, d, g, h\}$$

- **Vertex:** c and g are tied! We'll use $u = c$.
- **Neighbors:** $\cancel{b}, d, g, h$
- **Paths through u to neighbors:**
  - **To d:** $16 + 20 = 36 < \infty$, so set dist$(d) = 36$ and prev$(d) = c$.
  - **To g:** $16 + 2 = 18 > 16$, so it's not better.
  - **To h:** $16 + 10 = 26 < \infty$, so set dist$(h) = 26$ and prev$(h) = c$.

Dijkstra's Algorithm

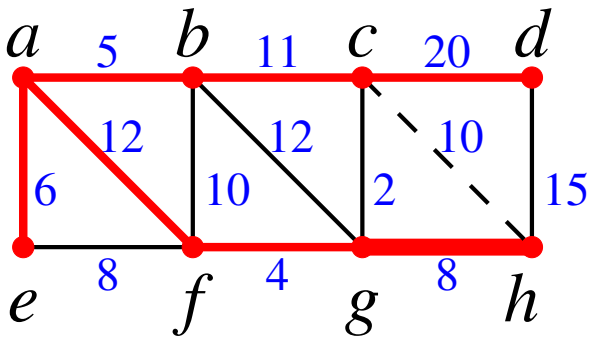


v	a	b	c	d	e	f	g	h
dist(v)	0	5	16	36	6	12	16	26
prev(v)	—	a	b	c	a	a	f	c

$$W = \{d, g, h\}$$

- **Vertex:** $u = g$
- **Neighbors:** $b, c, \cancel{f}, h$
- **Paths through u to neighbors:**
  - **To b:** $16 + 12 = 28 > 5$, so it's not better.
  - **To c:** $16 + 2 = 18 > 16$, so it's not better.
  - **To h:** $16 + 8 = 24 < 26$, so it's better than the prior path!
    Set dist$(h) = 24$ and prev$(h) = g$.
    This replaces the edge we previously had to h.

Dijkstra's Algorithm

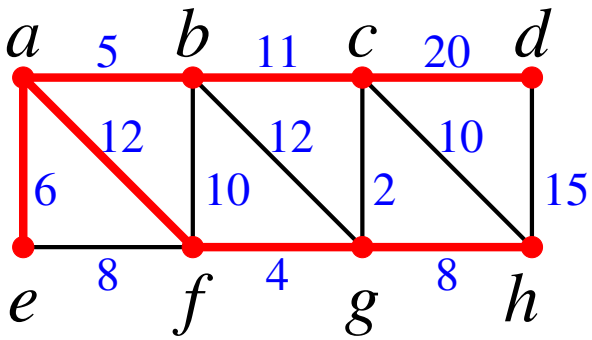


v	a	b	c	d	e	f	g	h
dist(v)	0	5	16	36	6	12	16	24
prev(v)	$-$	a	b	c	a	a	f	g

$$W = \{d, h\}$$

- **Vertex:** $u = h$
- **Neighbors:** $c, d, \cancel{g}$
- **Paths through u to neighbors:**
  - **To c:** $24 + 10 = 34 > 16$, so it's not better.
  - **To d:** $24 + 15 = 39 > 36$, so it's not better.

Dijkstra's Algorithm

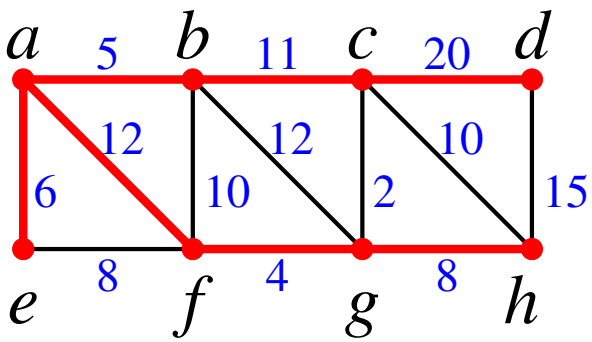


v	a	b	c	d	e	f	g	h
dist(v)	0	5	16	36	6	12	16	24
prev(v)	$-$	a	b	c	a	a	f	g

$$W = \{d\}$$

- **Vertex:** $u = d$
- **Neighbors:** $\cancel{c}, h$
- **Paths through u to neighbors:**
  - **To h:** $36 + 15 = 51 > 24$, so it's not better.

Dijkstra's Algorithm



v	a	b	c	d	e	f	g	h
dist(v)	0	5	16	36	6	12	16	24
prev(v)	$-$	a	b	c	a	a	f	g

$$W = \emptyset$$

- We are done!
- For a minimum weight path from a to h, work backwards from h by iterating prev(v) until we reach a:
  - prev$(h) = g$ prev$(g) = f$ prev$(f) = a$
  - Put these in the forwards direction: a, f, g, h
- It works the same for minimum weight paths from a to any vertex.

More algorithms for minimum weight paths

There are also algorithms beyond what we're covering, including:

- The *Bellman-Ford Algorithm* allows for negative edge weights (as long as that doesn't result in any cycles of negative weight).
- The *Floyd-Warshall Algorithm* allows for negative edge weights (same restriction) and it simultaneously computes the minimum distance between all pairs of vertices.