MINIMUM SPANNING TREES

Paula Rodríguez
Juan Mendivelso

Presentation by P. Rodríguez & J. Mendivelso. Contents and figures extracted from the book: Introduction to Algorithms, Third Edition. Cormen, Leiserson, Rivests and Stein. The MIT Press. 2009.

CONTENTS

1. MINIMUM SPANNING TREE

Minimum Spanning Tree

- Think in a telecommunications company trying to lay cable in a new neighborhood. If it is constrained to bury the cable only along certain paths (roads), then there would be a graph containing the points (houses) connected by those paths.

Minimum Spanning Tree

- Some of the paths might be more expensive, because they are longer, or require the cable to be buried deeper; these paths would be represented by edges with larger weights.

- Of all such arrangements, the one that uses the least amount of cable is usually the most desirable.

Minimum Spanning Tree

- Let G=(V,E) be a graph, where V is the set of houses, E is the set of possible interconnections between pairs of houses, and for each edge (u,v) $\in$ E, we have a weight w(u,v) specifying the cost (amount of cable needed) to connect u and v.
- We wish to find an acyclic subset T $\subseteq$ E that connects all the vertices and whose total weight

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

is minimized.

Minimum Spanning Tree

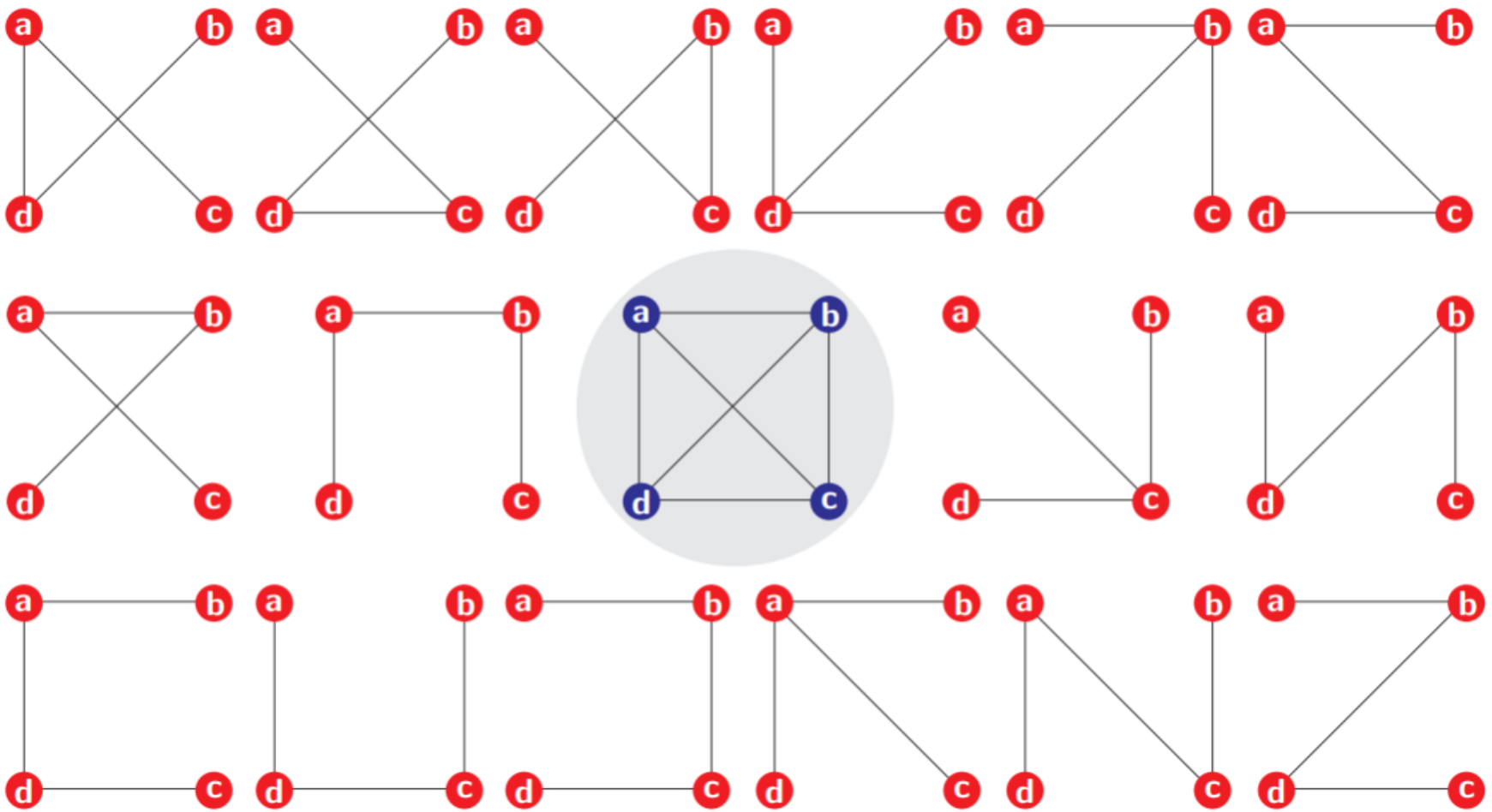


Image by Yoan Pinzón.

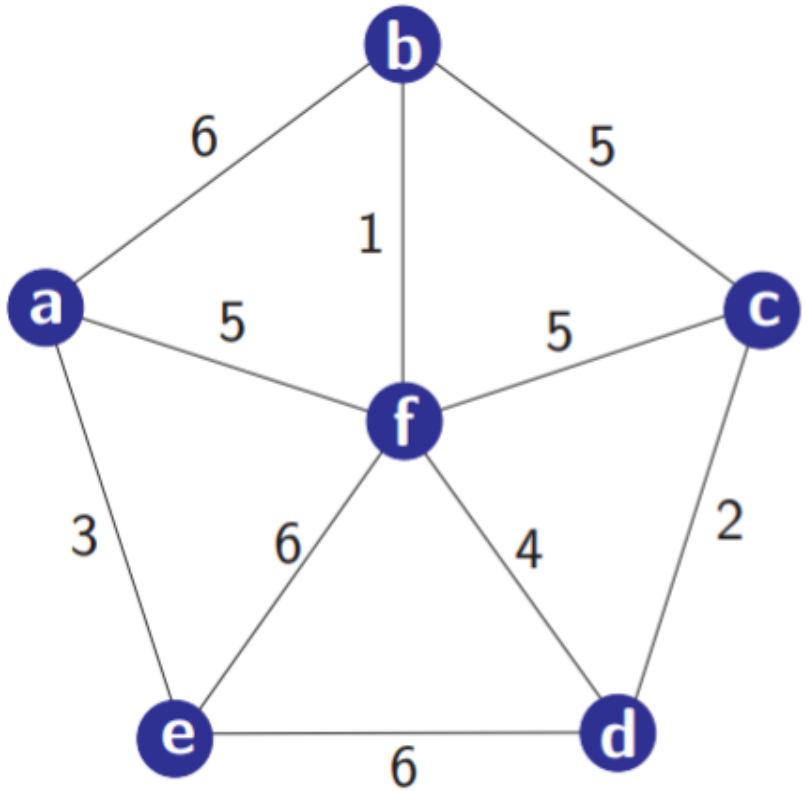
Minimum Spanning Tree

- Since T is acyclic and connects all the vertices, it must form a tree, which we call a spanning tree since it “spans” the graph G. We call the problem of determining the tree T the minimum spanning tree problem.

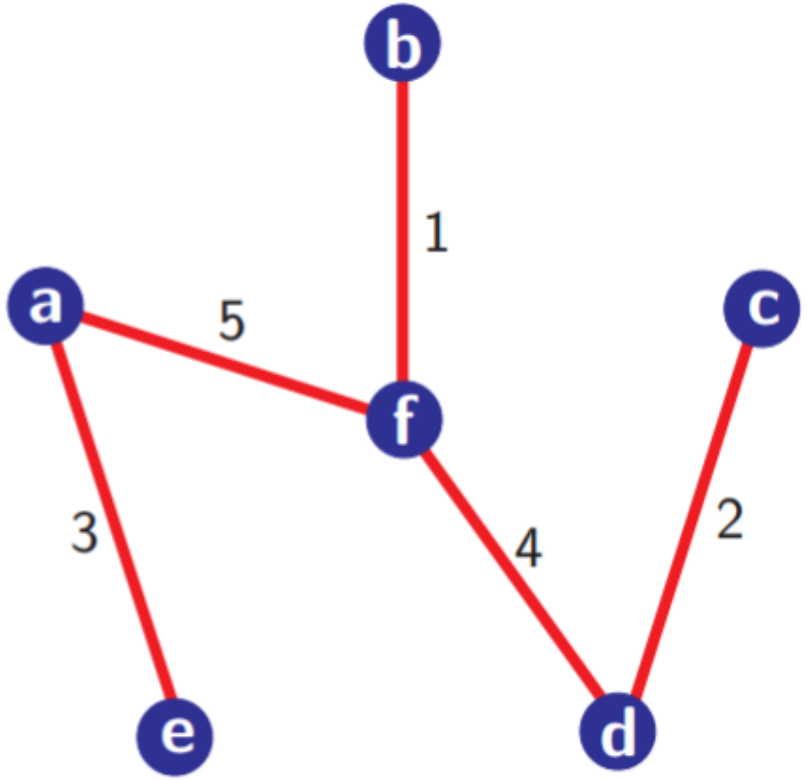
- A **spanning tree** of a graph is just a subgraph that contains all the vertices and is a tree.

- A **minimum spanning tree** of a graph is a spanning tree whose sum of edge weights is as small as possible.

Minimum Spanning Tree



$w = 43$



$w = 15$

Image by Yoan Pinzón.

2. GROWING A MINIMUM SPANNING TREE

Greedy Algorithms

- **Greedy algorithm:** Each step of a greedy algorithm must make one of several possible choices. The greedy strategy advocates making the choice that is the best at the moment.

- Such a strategy does not generally guarantee that it will always find globally optimal solutions to problems.

- For the minimum-spanning-tree problem, however, we can prove that certain greedy strategies do yield a spanning tree with minimum weight

Growing a Minimum Spanning Tree

- Assume that we have a connected, undirected graph G=(V, E) with a weight function $\boldsymbol{w}$: E $\rightarrow \mathbb{R}$, and we wish to find a minimum spanning tree for G.
- The greedy strategy to solve this problem is captured by the following generic method, which grows the minimum spanning tree one edge at a time.
- The generic method manages a set of edges A, maintaining the following loop invariant:

Prior to each iteration, A is a subset of some minimum spanning tree.

Growing a Minimum Spanning Tree

- **Loop Invariant:** Prior to each iteration, A is a subset of some minimum spanning tree.

- At each step, we determine an edge (u,v) that we can add to A without violating this invariant, in the sense that A U {(u,v)} is also a subset of a minimum spanning tree.
- We call such an edge a **safe edge** for A, since we can add it safely to A while maintaining the invariant.

Growing a Minimum Spanning Tree

Initialization: After line 1, the set A trivially satisfies the loop invariant.

Maintenance: The loop in lines 2–4 maintains the invariant by adding only safe edges.

Termination: All edges added to A are in a minimum spanning tree, and so the set A returned in line 5 must be a minimum spanning tree.

GENERIC-MST(G, w)

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

Finding a Safe Edge

- The tricky part is, of course, finding a **safe edge** in line 3.
- One must exist, since when line 3 is executed, the invariant dictates that there is a spanning tree T such that $A \subseteq T$.
- Within the while loop body, A must be a proper subset of T , and therefore there must be an edge $(u,v) \in T$ such that $(u,v) \notin A$ and (u,v) is safe for A.

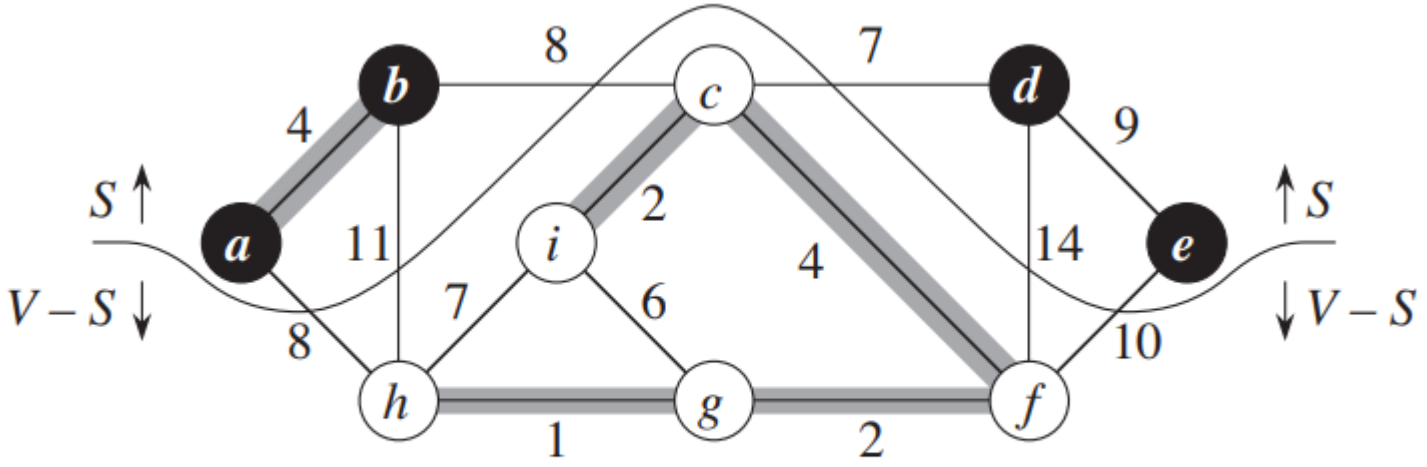
Cuts, Crossing Edges & Light Edges

To identify safe edges, we first need some definitions:

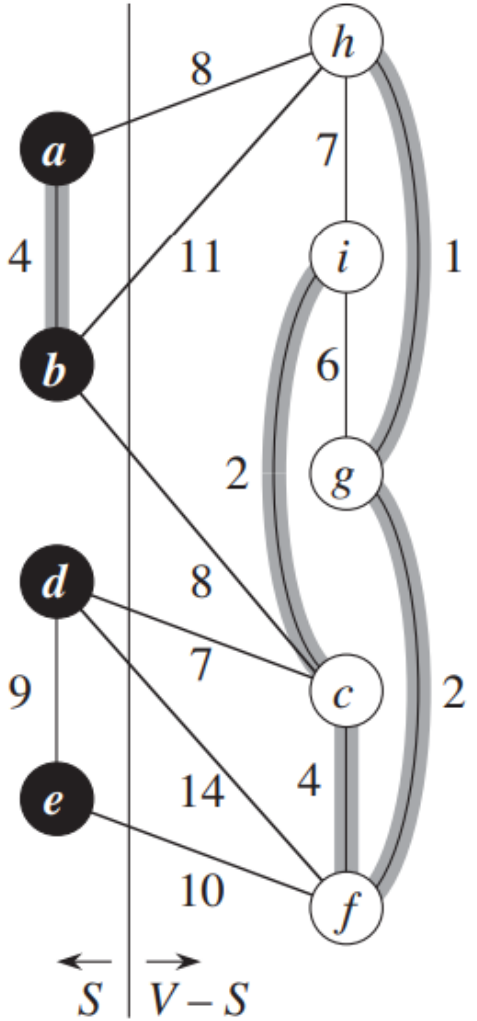
- A **cut** (S, V - S) of an undirected graph G=(V, E) is a partition of V.
- We say that an edge (u,v) $\in$ E **crosses** the cut (S, V - S) if one of its endpoints is in S and the other is in V - S.
- We say that a cut **respects** a set A of edges if no edge in A crosses the cut.
- An edge is a **light edge** crossing a cut if its weight is the minimum of any edge crossing the cut.

Cuts, Crossing Edges & Light Edges

- What are the edges crossing the cut?
- What are the light edges?
- If A is the set of shaded edges, does the cut respect A?



(a)



(b)

Light Edge satisfying a Property

- We say that an edge is a **light edge satisfying a given property** if its weight is the minimum of any edge satisfying the property.
- Our rule for recognizing safe edges is given by the following theorem.

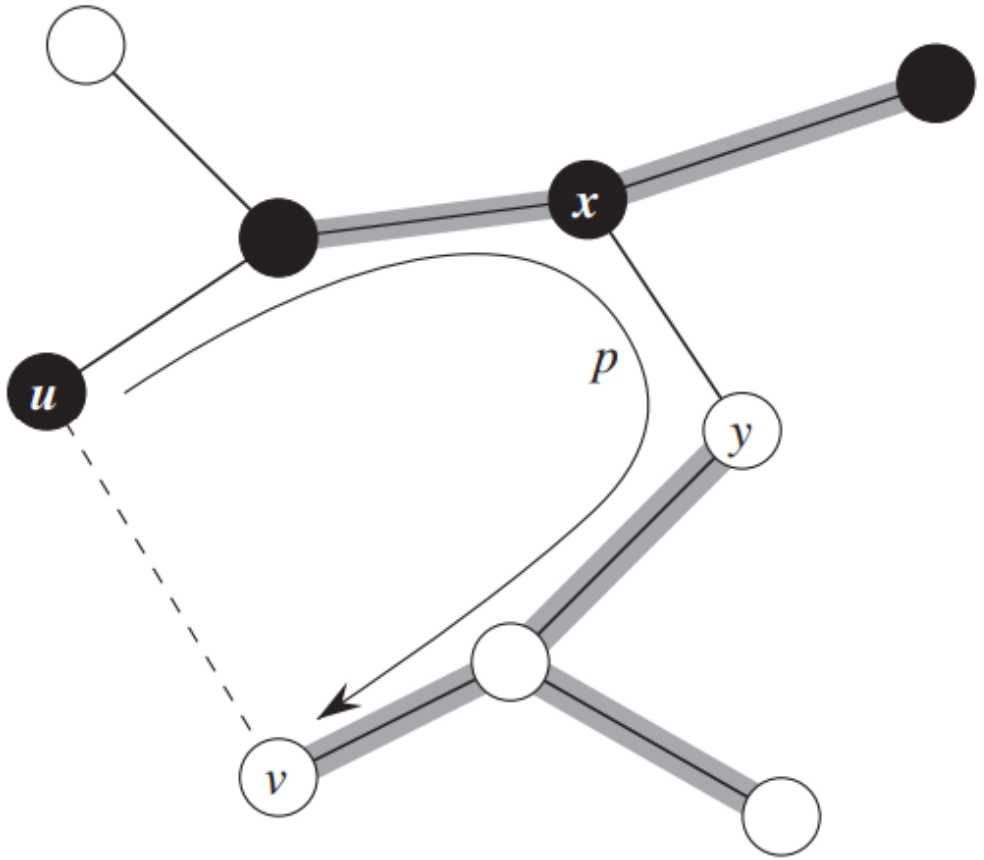
Theorem 23.1

Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, let $(S, V - S)$ be any cut of G that respects A, and let (u, v) be a light edge crossing $(S, V - S)$. Then, edge (u, v) is safe for A.

Finding a Safe Edge

Proof:

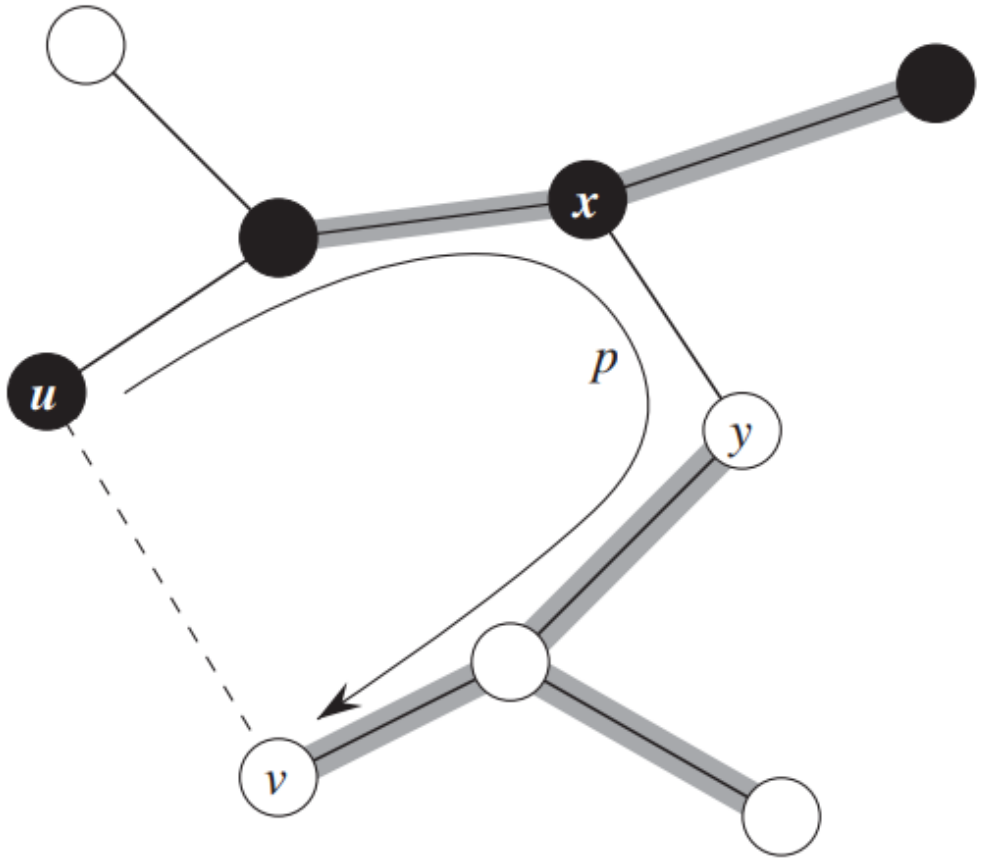
- Let *T* be a MST that includes *A* and assume that T does not contain the light edge *(u,v)*.
- We shall construct another MST *T'* that includes *A U {(u,v)}*, thereby showing that *(u,v)* is a safe edge for A.
- Black vertices are in S, and white vertices are in V - S.
- The edges in A are shaded, and *(u,v)* is a light edge crossing the cut *(S, V - S)*.



Finding a Safe Edge

Proof:

- Since *u* and *v* are on opposite sides of the cut (S, V - S), at least one edge in *T* lies on the simple path p and also crosses the cut.
- Let *(x, y)* be any such edge. The edge *(x, y)* is not in A, because the cut respects A.
- Since *(u,v)* is a light edge, $w(u,v) \leq w(x,y)$.
- Then, the edge *(u,v)* forms a cycle with the edges on the simple path *p* from *u* to *v* in *T*.

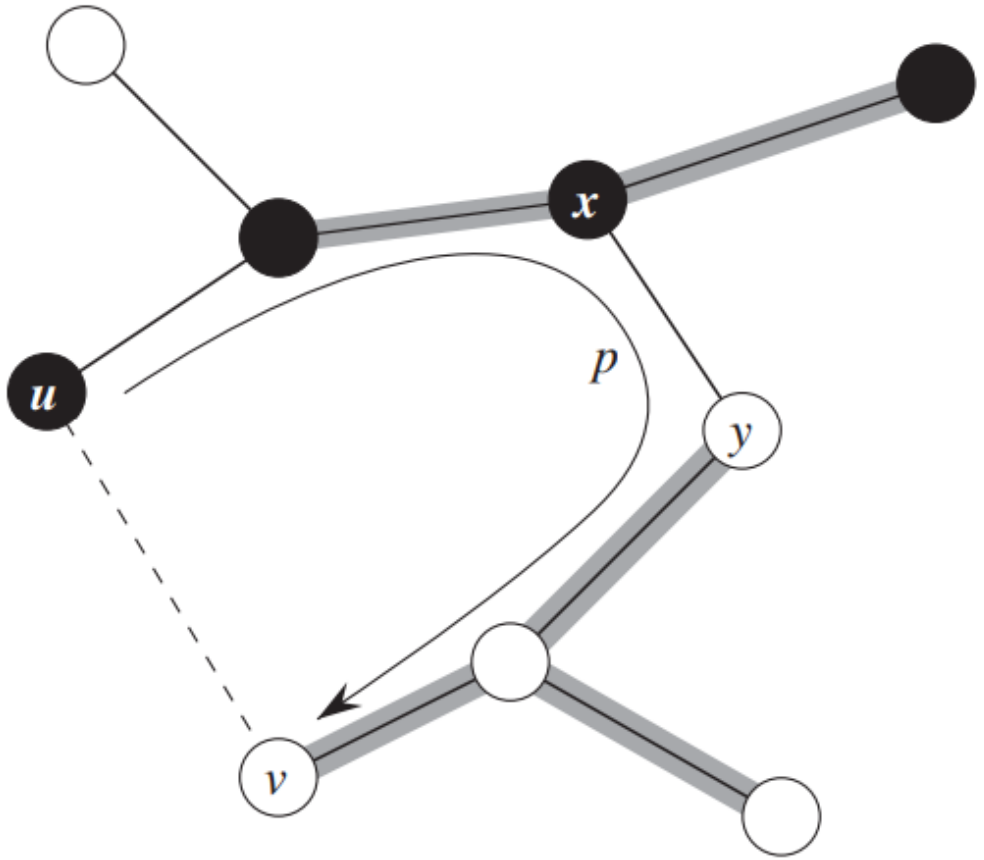


Finding a Safe Edge

Proof:

- Since *(x, y)* is on the unique simple path from u to v in T, removing *(x, y)* breaks *T* into two components.

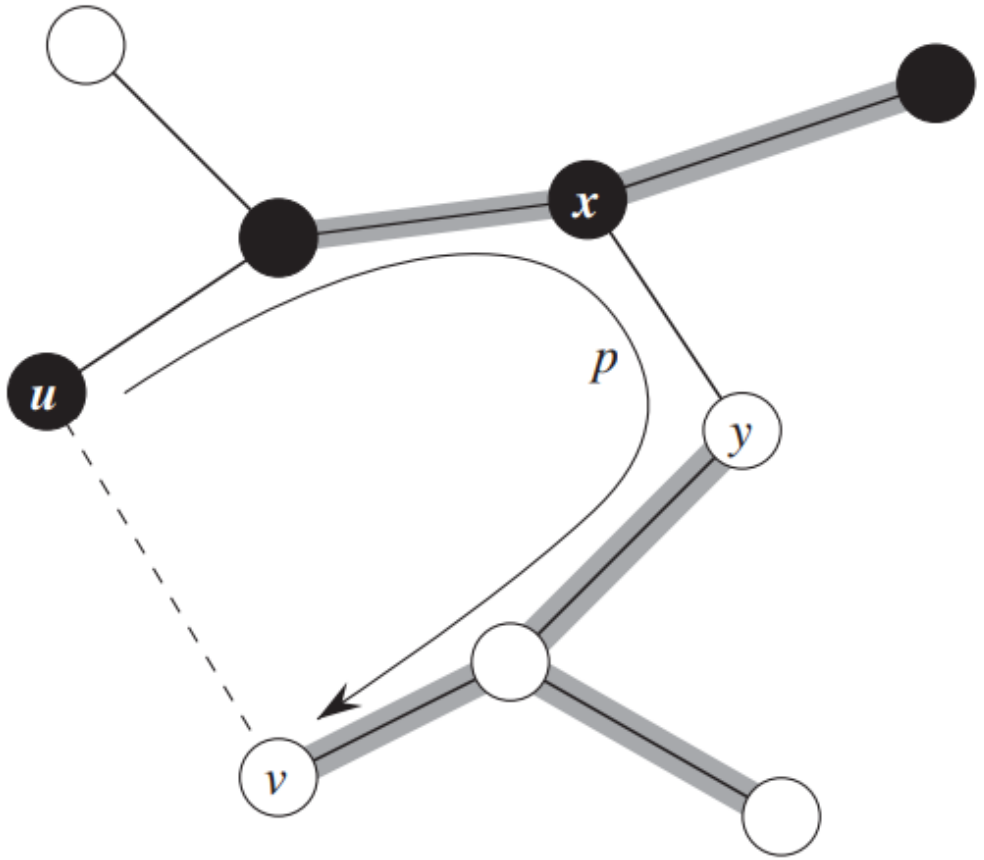
- Adding *(u,v)* reconnects them to form a new spanning tree
  T' = T - {(x,y)} U {(u,v)}.



Finding a Safe Edge

Proof:

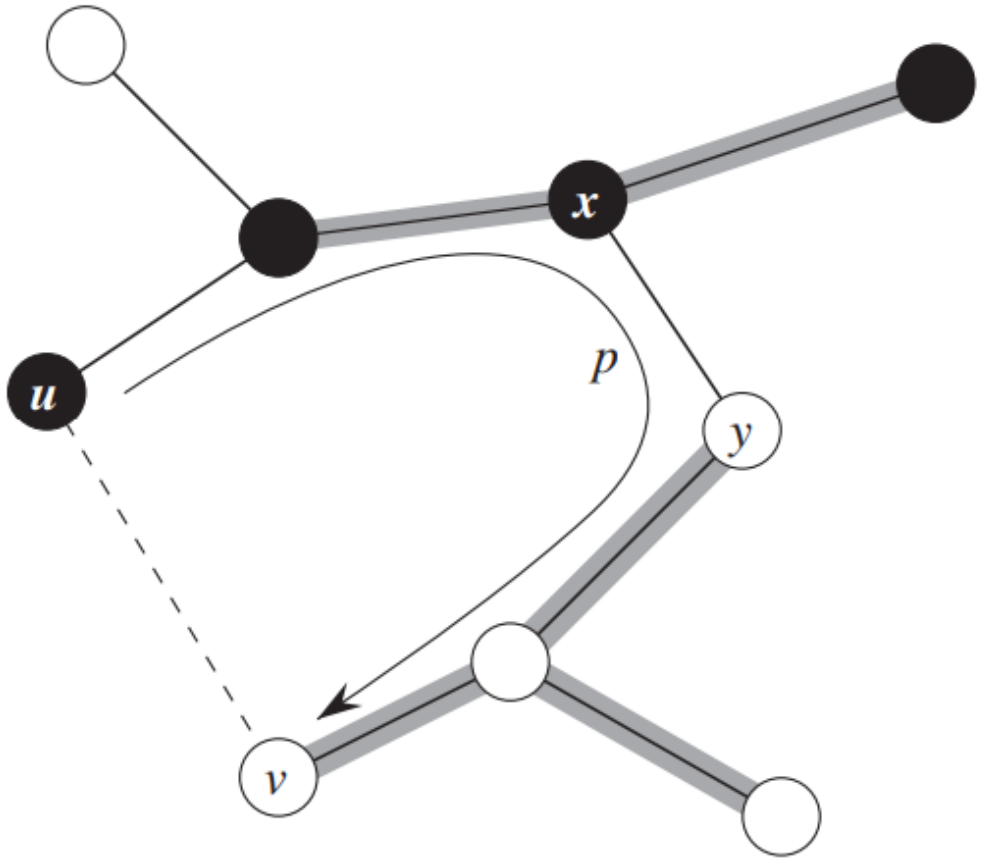
- Since *(u,v)* is a light edge, $w(u,v) \leq w(x,y)$.
- $T' = T - \{(x,y)\} \cup \{(u,v)\}$.
  $w(T') = w(T) - w(x,y) + w(u,v)$.
  $\leq w(T)$.
- But T is a MST, so
  $w(T) \leq w(T)$.
- So, $w(T) = w(T')$.
- Then, T' is also a MST.



Finding a Safe Edge

Proof:

- Now, we show *(u,v)* is actually safe for *A*.
- We have $A \subseteq T'$, $A \subseteq T$ and $(x, y) \notin A$.
- Thus, $A \cup \{(u, v)\} \subseteq T'$.
- Since *T'* is a MST, *(u,v)* is safe for A.



Growing a Minimum Spanning Tree

- The **while** loop in lines 2–4 of GENERIC-MST executes |V| - 1 times because it finds one of the |V| - 1 edges of a minimum spanning tree in each iteration.
- Initially, when A = ∅, there are |V| trees in G'=(V,A), and each iteration reduces that number by 1.
- When the forest contains only a single tree, the method terminates.

GENERIC-MST(G, w)

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

Growing a Minimum Spanning Tree

Corollary 23.2

Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, and let $C = (V_C, E_C)$ be a connected component (tree) in the forest $G_A = (V, A)$. If (u, v) is a light edge connecting C to some other component in G_A, then (u, v) is safe for A.

Proof The cut $(V_C, V - V_C)$ respects A, and (u, v) is a light edge for this cut. Therefore, (u, v) is safe for A. ■

3. KRUSKAL'S ALGORITHM

Kruskal's Algorithm

- By Joseph Bernard Kruskal, 1956.
- Kruskal's algorithm is a greedy algorithm.
- In Kruskal's algorithm, the set A is a forest whose vertices are all those of the given graph. The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.
- In Kruskal's algorithm the edges are selected and added to the spanning tree in increasing order of their weights. An edge is added to the tree only if it does not create a cycle

Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3       MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6       if FIND-SET(u) ≠ FIND-SET(v)
7            A = A ∪ {(u, v)}
8            UNION(u, v)
9  return A
```

Kruskal's algorithm finds a safe edge to add to the growing forest by finding, of all the edges that connect any two trees in the forest, an edge (u,v) of least weight.

Kruskal's Algorithm

- Kruskal's algorithm uses a disjoint-set data structure to maintain several disjoint sets of elements. Each set contains the vertices in one tree of the current forest.

- The operation FIND-SET(u) returns a representative element from the set that contains u. Thus, we can determine whether two vertices u and v belong to the same tree by testing whether FIND-SET(u) equals FIND-SET(v).

- To combine trees, Kruskal's algorithm calls the UNION procedure.

Kruskal's Algorithm

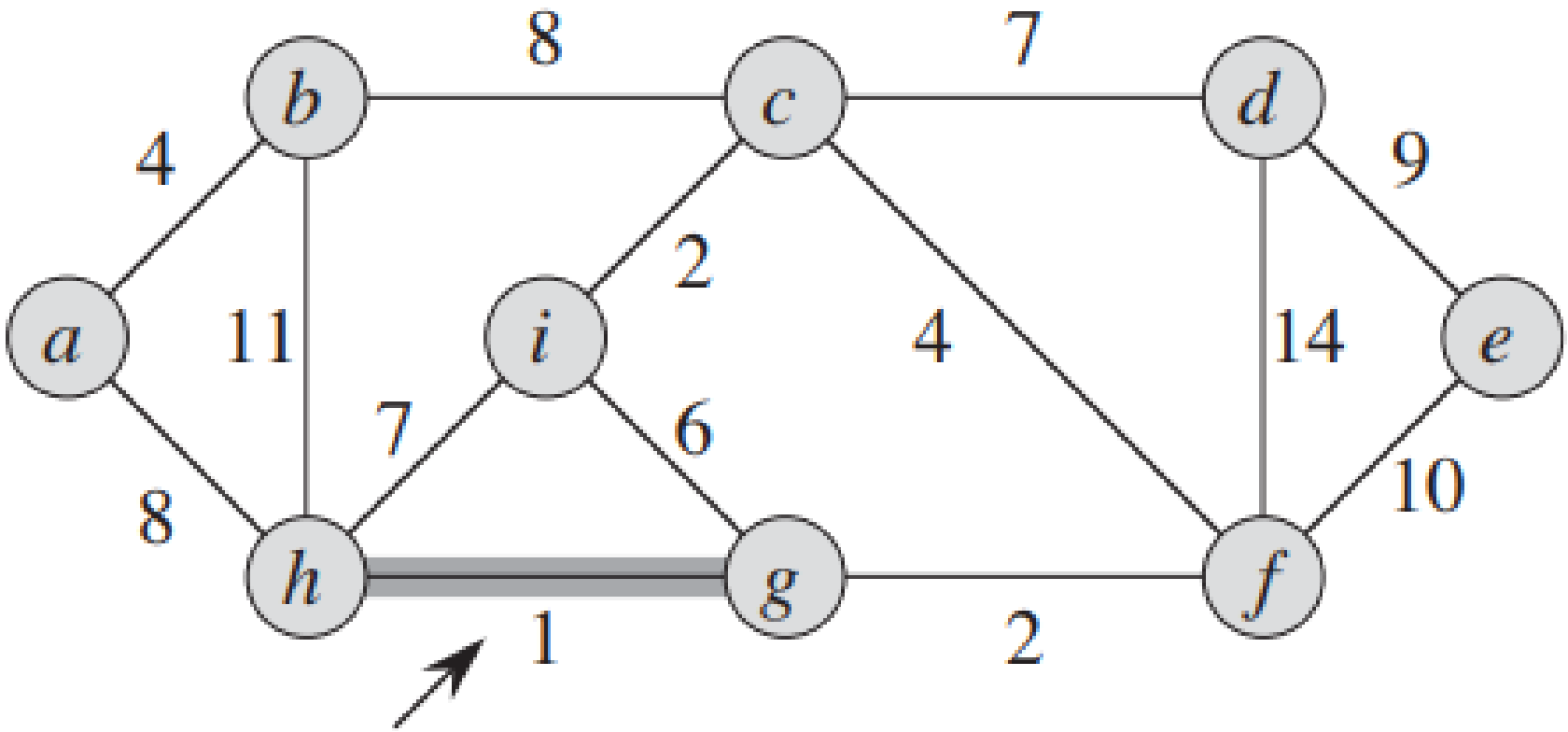


Image by Yoan Pinzón.

Kruskal's Algorithm

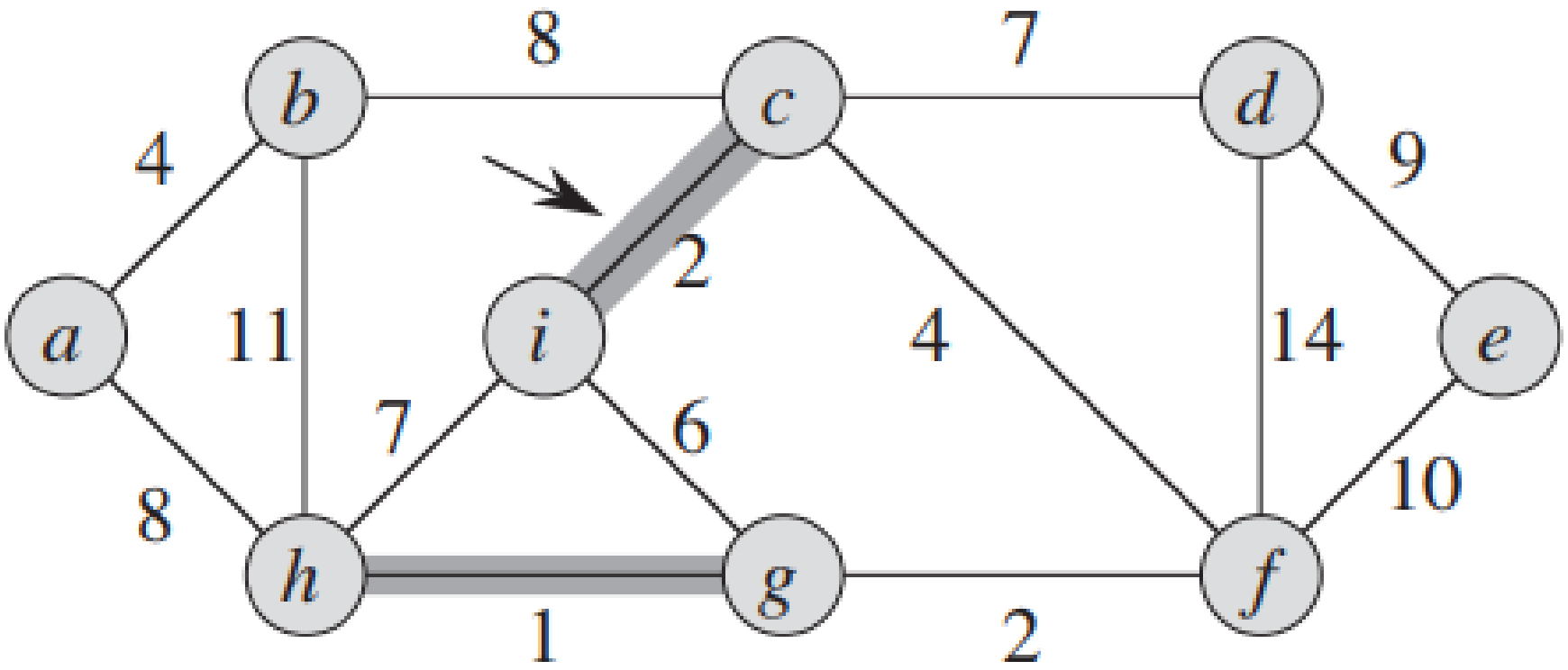


Image by Yoan Pinzón.

Kruskal's Algorithm

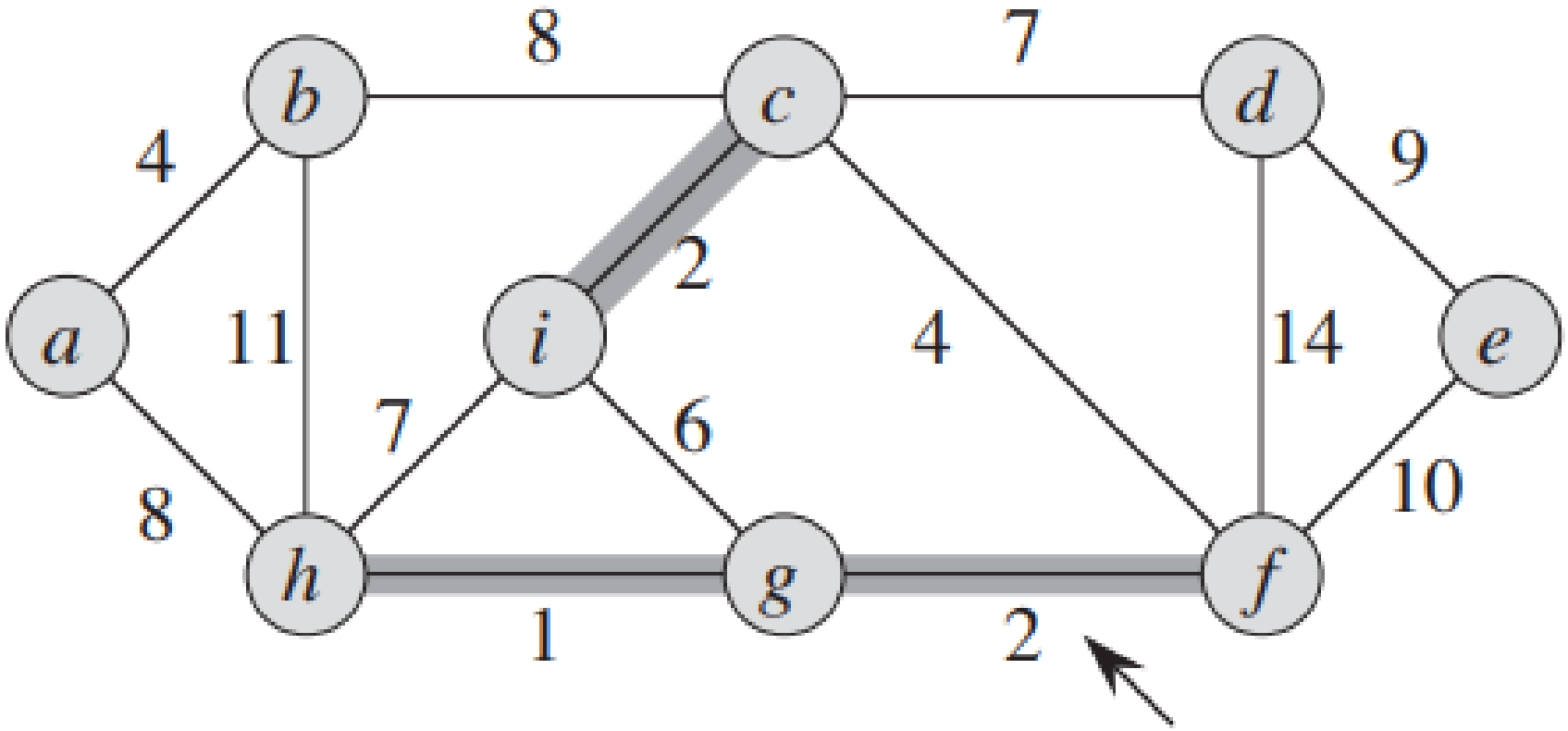


Image by Yoan Pinzón.

Kruskal's Algorithm

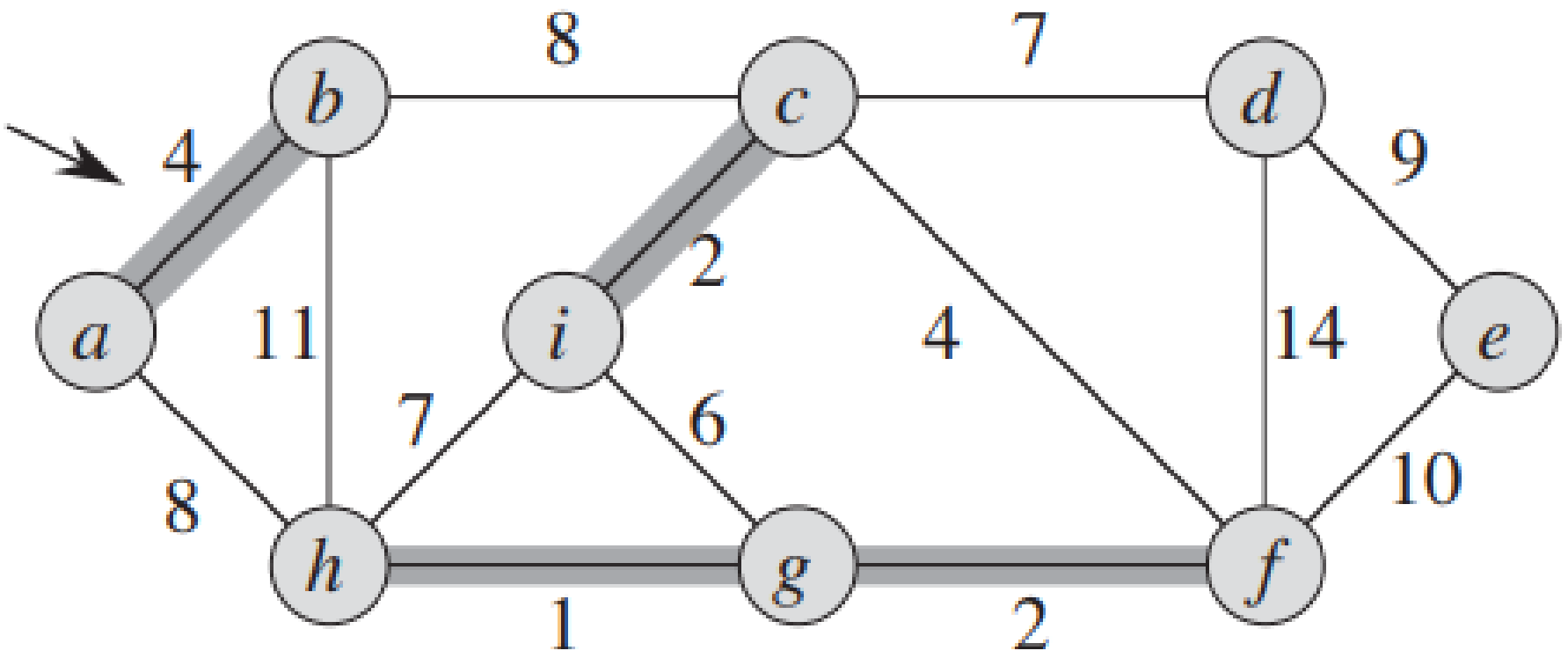


Image by Yoan Pinzón.

Kruskal's Algorithm

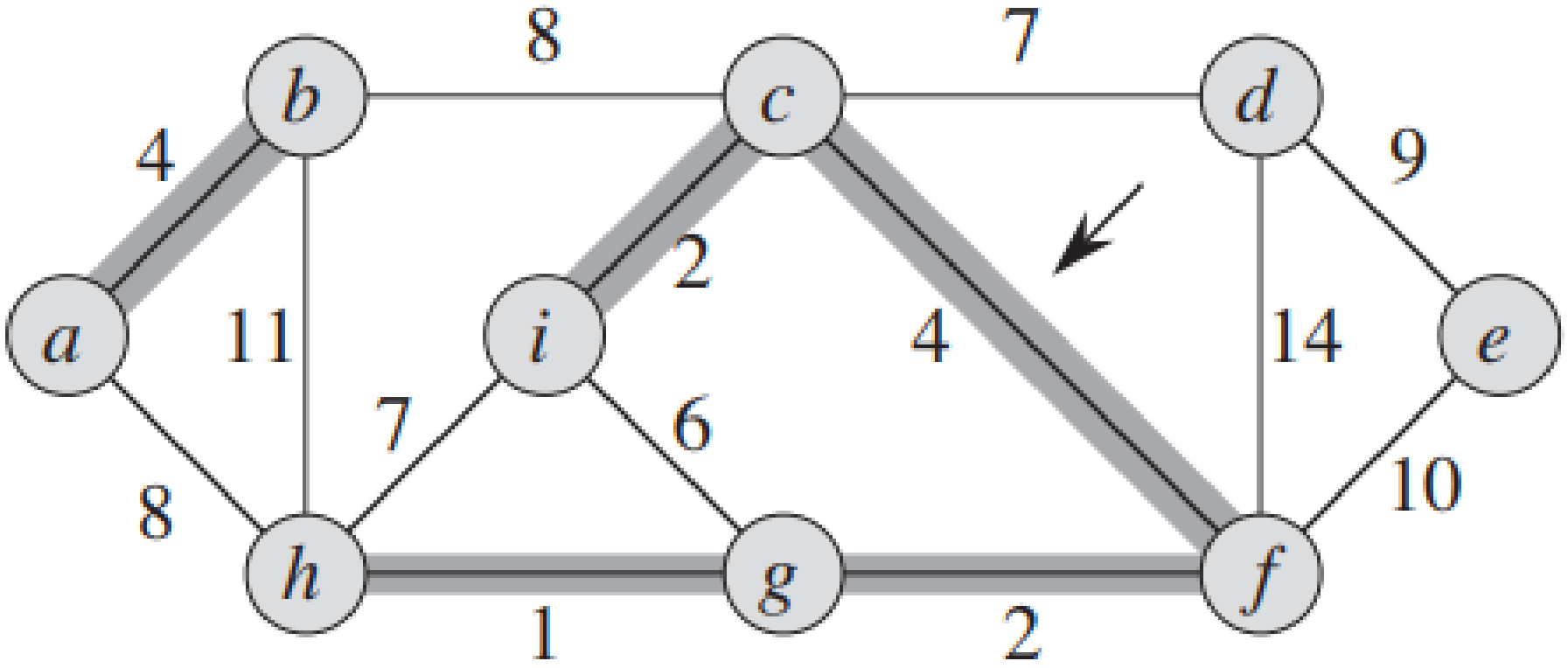


Image by Yoan Pinzón.

Kruskal's Algorithm

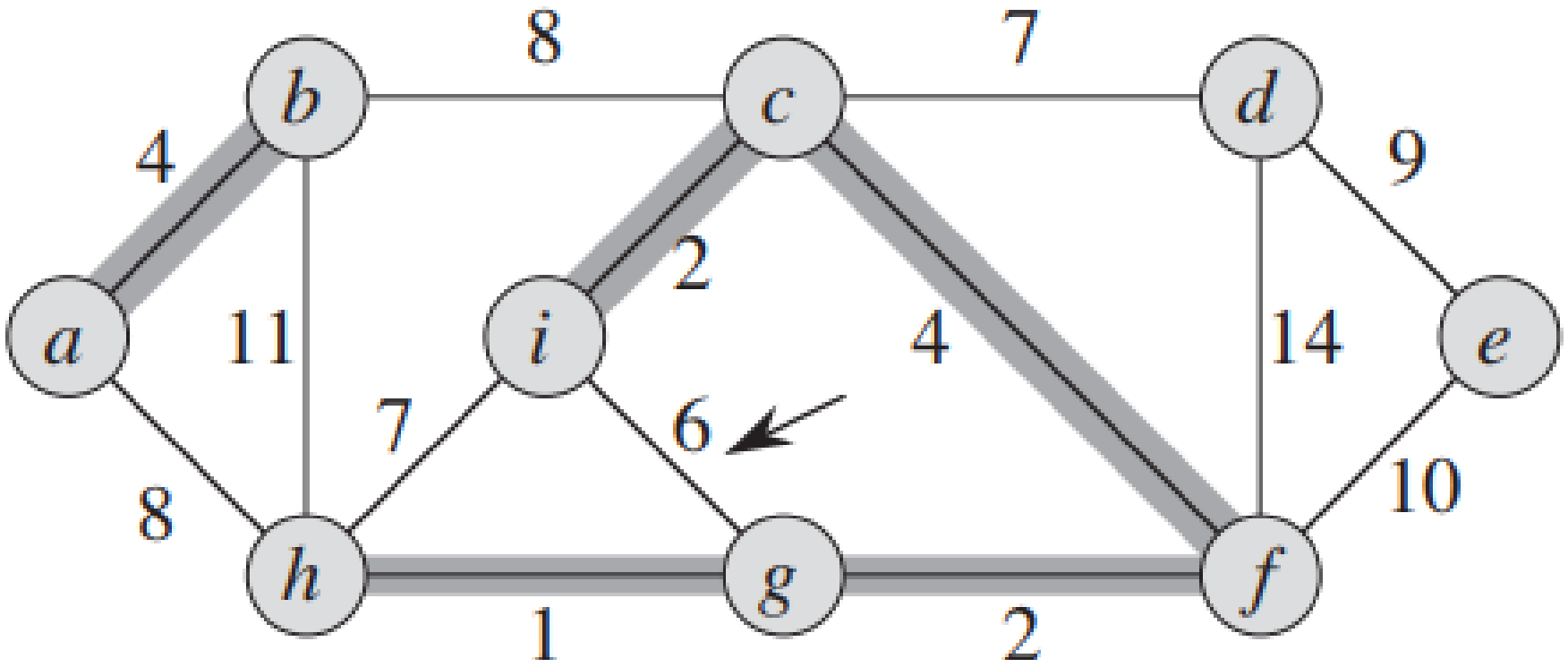


Image by Yoan Pinzón.

Kruskal's Algorithm

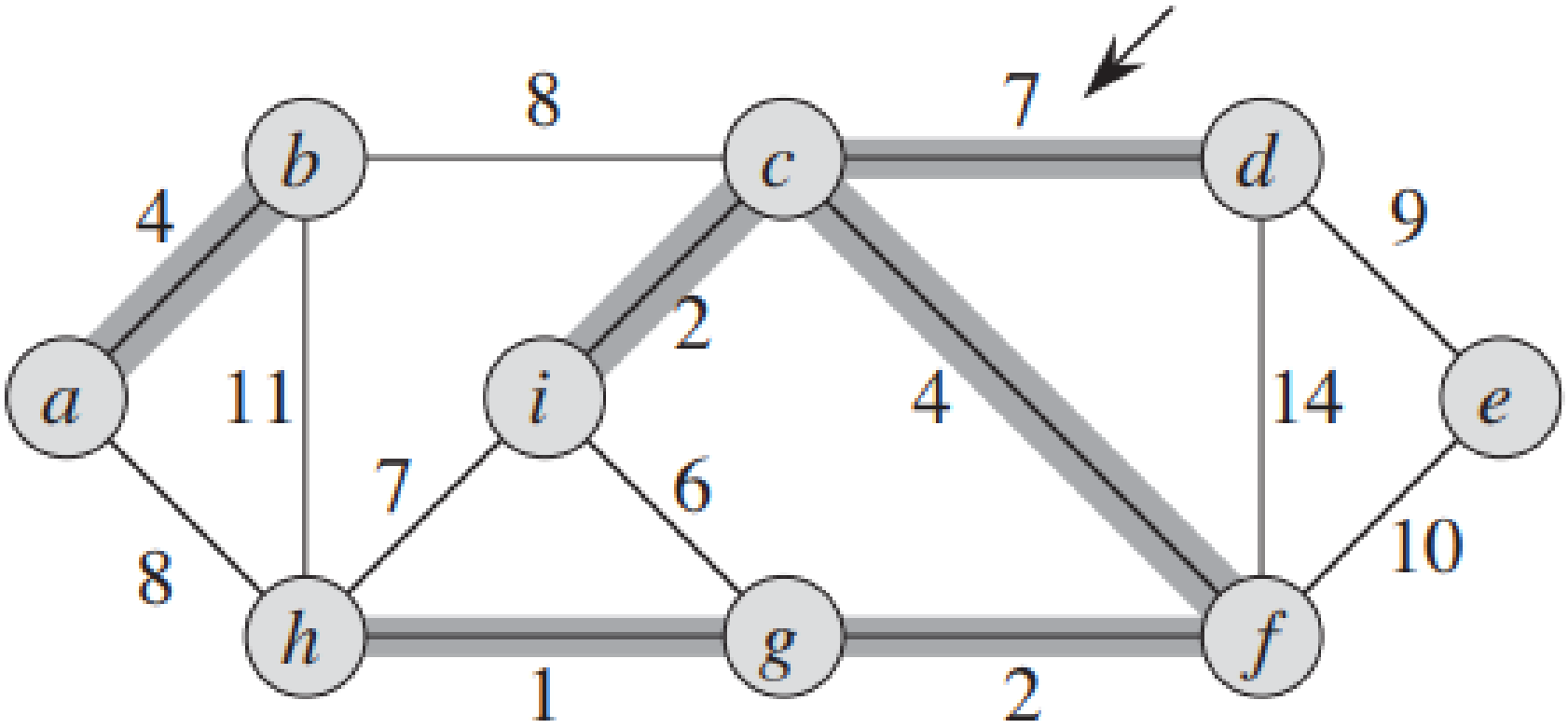


Image by Yoan Pinzón.

Kruskal's Algorithm

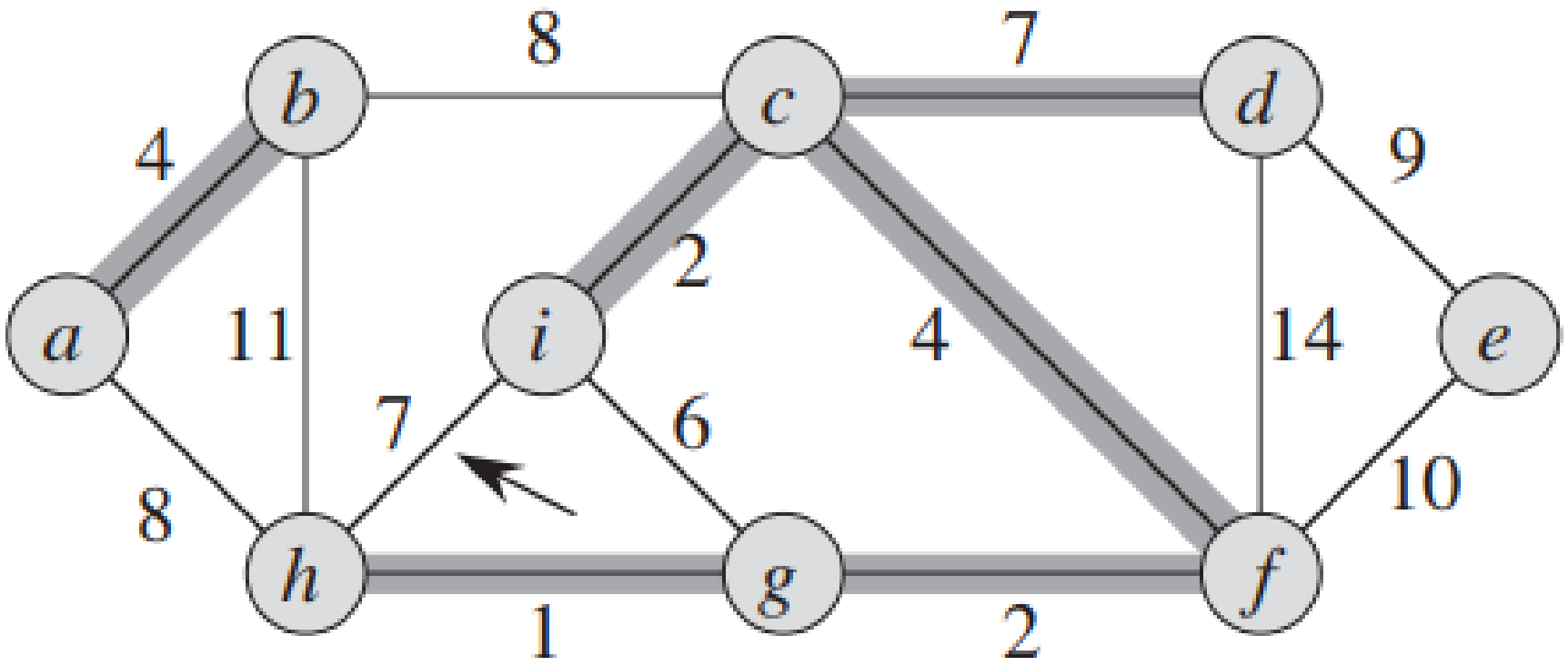


Image by Yoan Pinzón.

Kruskal's Algorithm

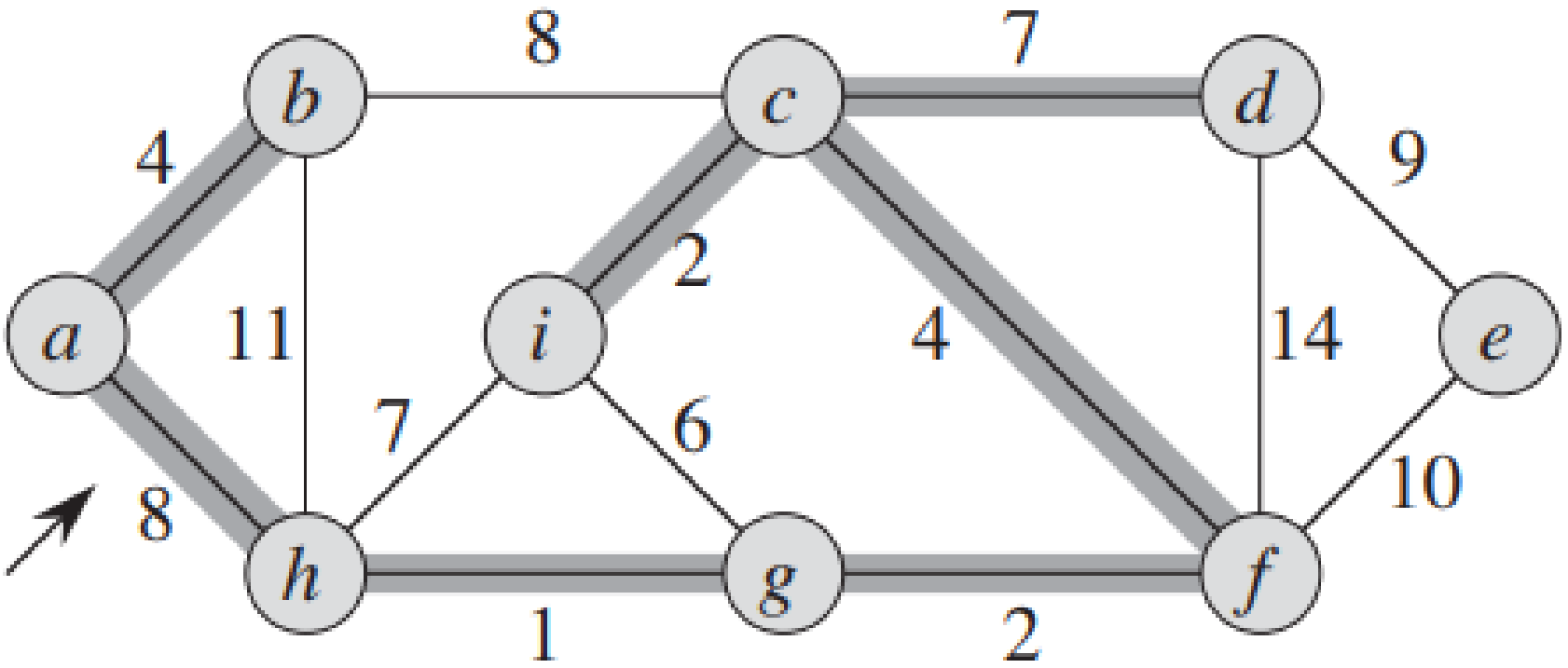


Image by Yoan Pinzón.

Kruskal's Algorithm

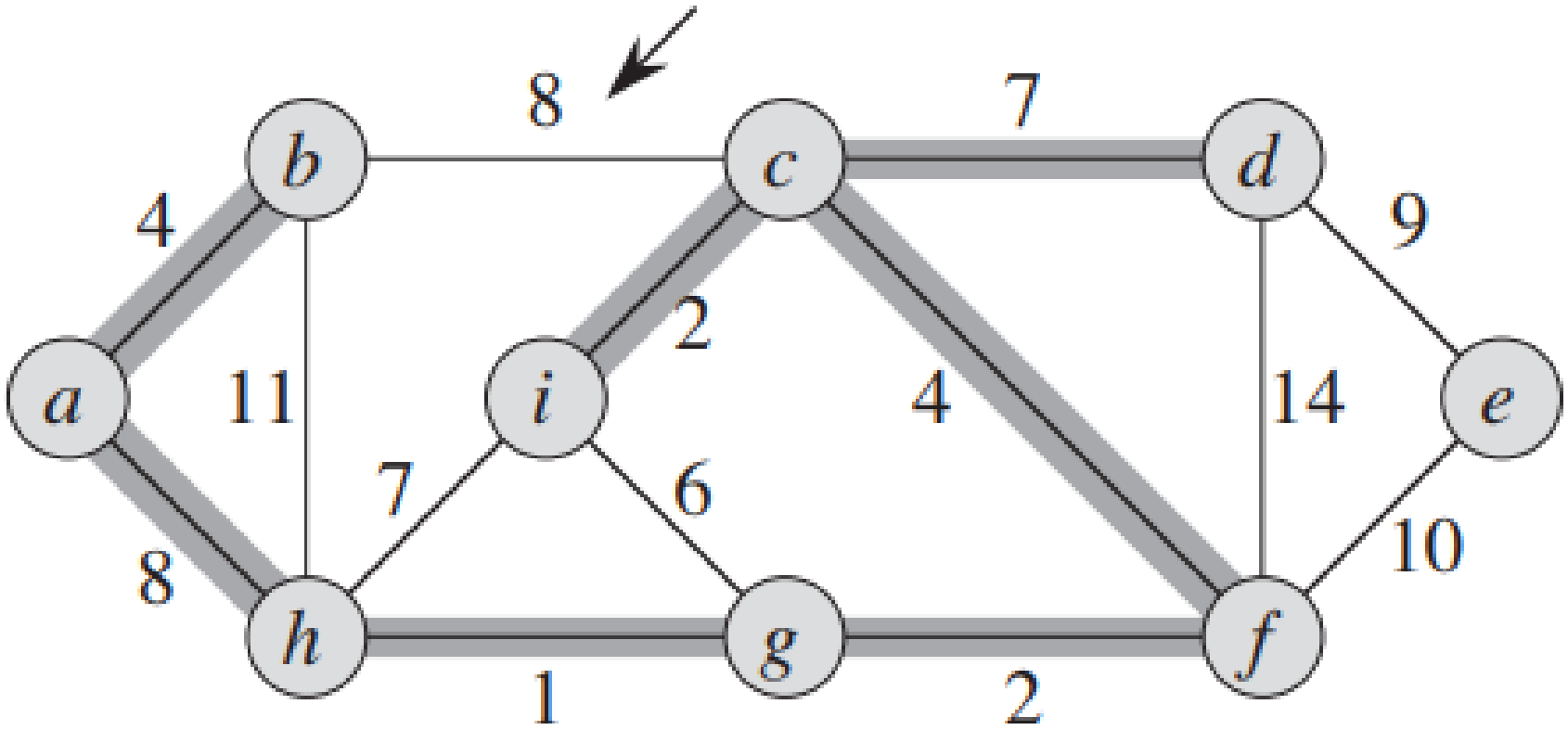


Image by Yoan Pinzón.

Kruskal's Algorithm

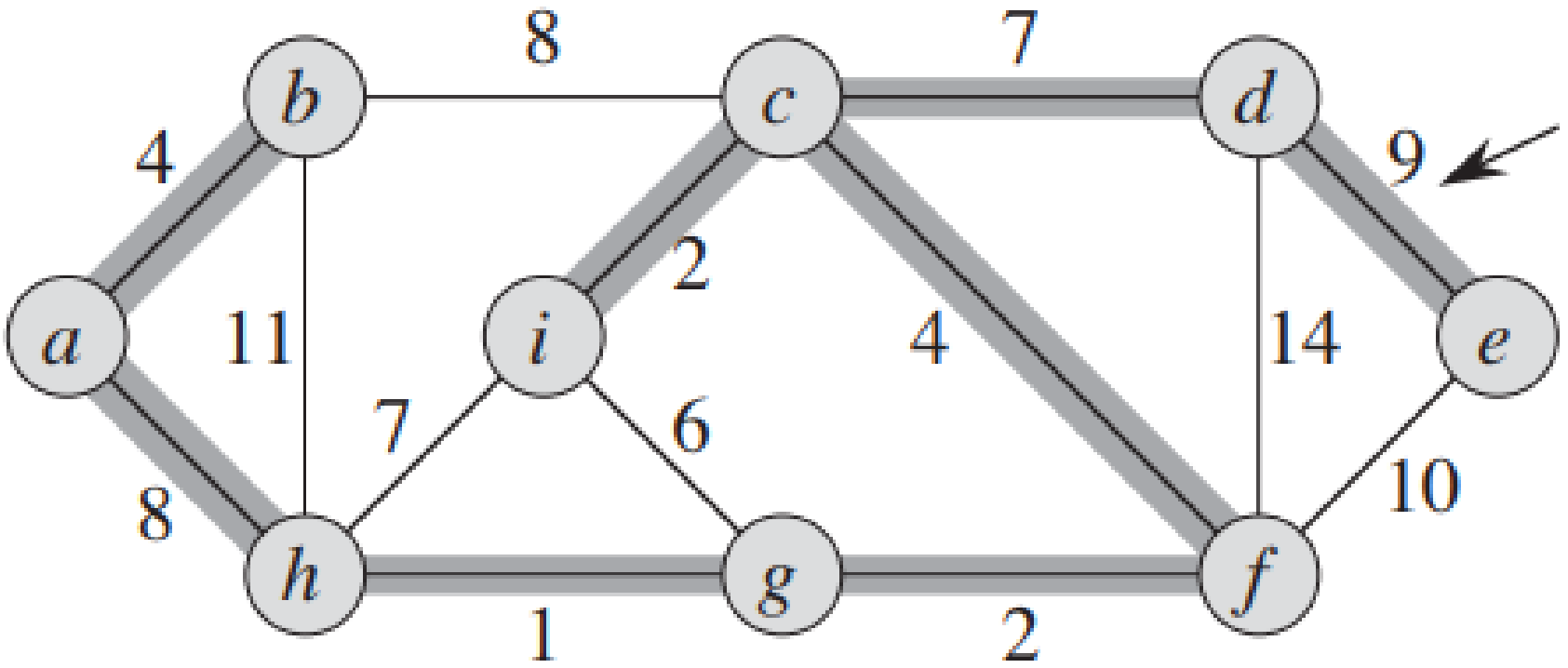


Image by Yoan Pinzón.

Kruskal's Algorithm

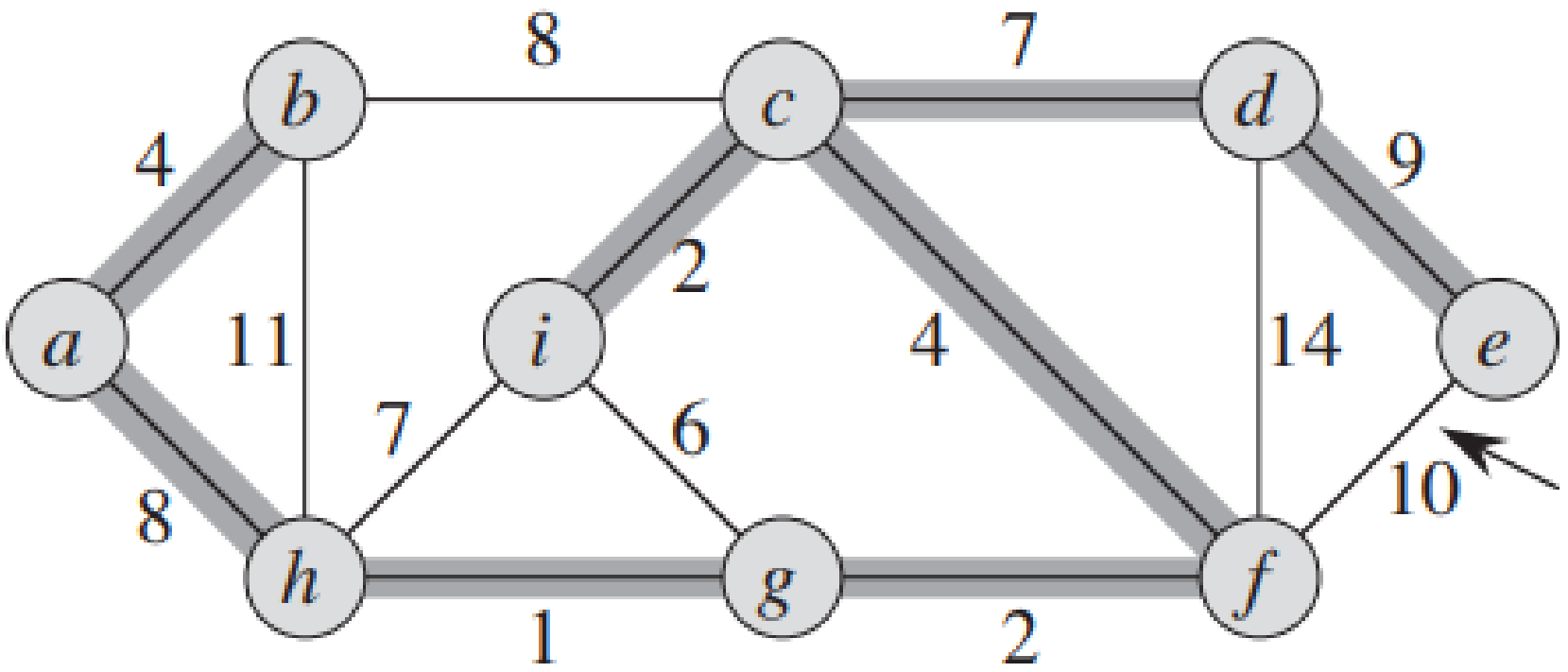


Image by Yoan Pinzón.

Kruskal's Algorithm

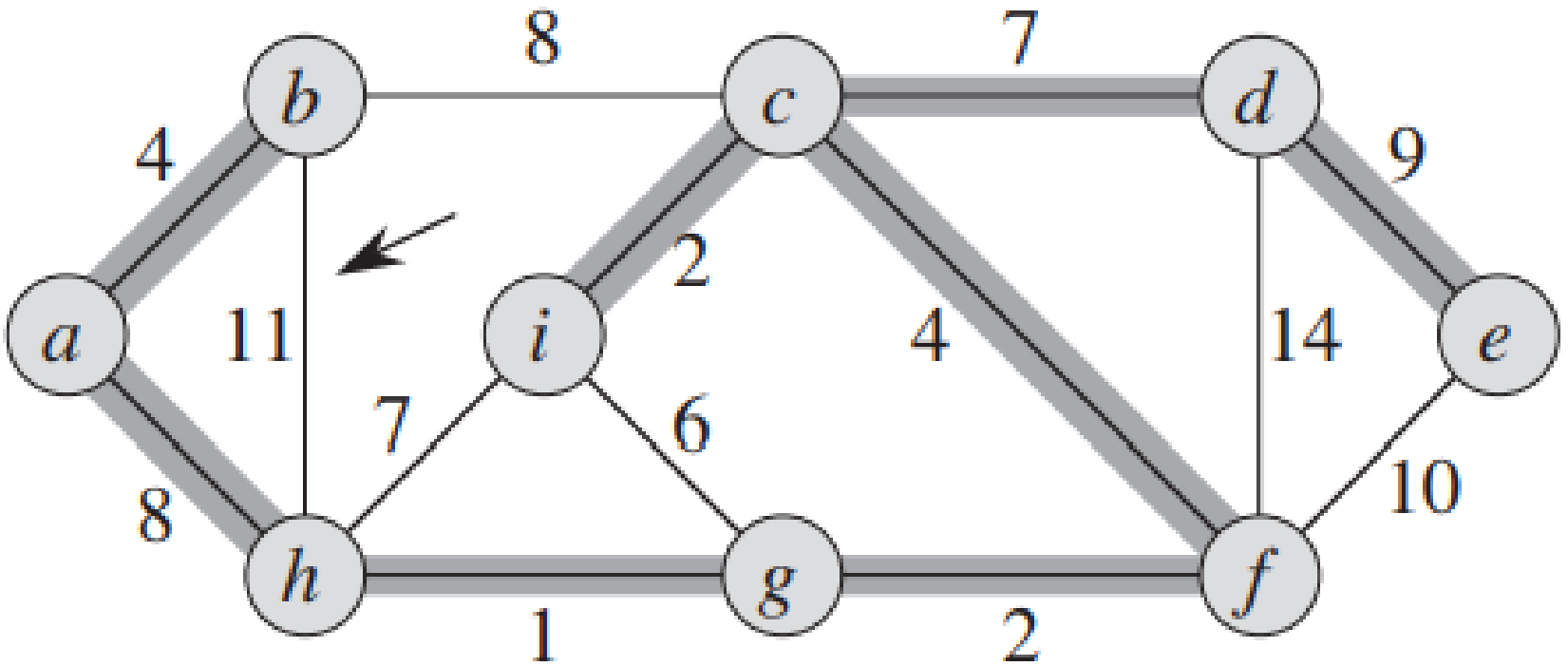


Image by Yoan Pinzón.

Kruskal's Algorithm

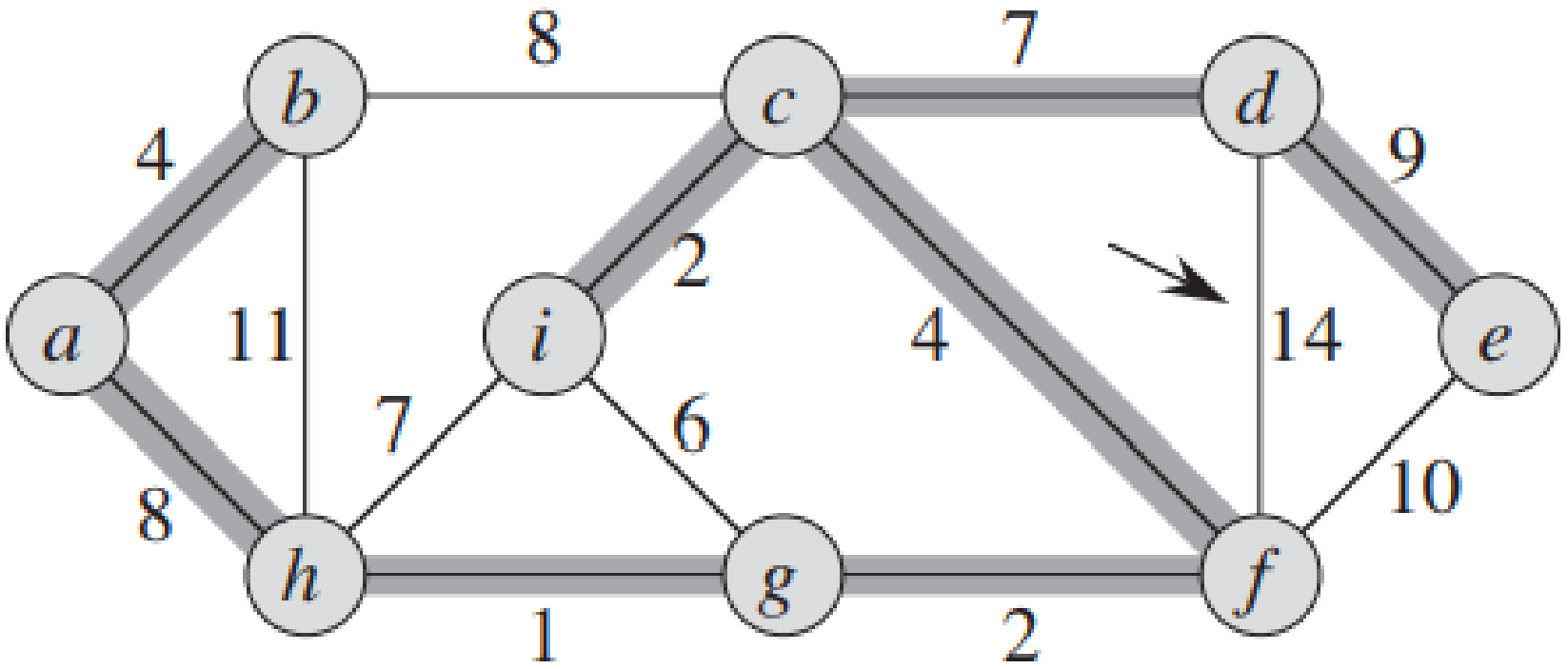


Image by Yoan Pinzón.

Edge Processed	Collection of disjoint sets								
initial state	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}
1 (h,g)✓	{a}	{b}	{c}	{d}	{e}	{f}	{g,h}		{i}
2 (i,c)✓	{a}	{b}	{c,i}	{d}	{e}	{f}	{g,h}		
2 (g,f)✓	{a}	{b}	{c,i}	{d}	{e}		{g,h,f}		
4 (a,b)✓	{a,b}		{c,i}	{d}	{e}		{g,h,f}		
4 (c,f)✓	{a,b}			{d}	{e}		{g,h,f,c,i}		
6 (i,g)	{a,b}			{d}	{e}		{g,h,f,c,i}		
7 (c,d)✓	{a,b}				{e}		{g,h,f,c,i,d}		
7 (h,i)	{a,b}				{e}		{g,h,f,c,i,d}		
8 (a,h)✓					{e}		{g,h,f,c,i,d,a,b}		
8 (b,c)					{e}		{g,h,f,c,i,d,a,b}		
9 (d,e)✓							{g,h,f,c,i,d,a,b,e}		
10 (f,e)							{g,h,f,c,i,d,a,b,e}		
11 (b,h)							{g,h,f,c,i,d,a,b,e}		
14 (d,f)							{g,h,f,c,i,d,a,b,e}		

Image by Yoan Pinzón.

$$A = \{(h,g),(i,c),(g,f),(a,b),(c,f),(c,d),(a,h),(d,e)\}, w = 37$$

Kruskal's Algorithm

- What is the running time if the sets are implemented as linked lists?

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3       MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6       if FIND-SET(u) ≠ FIND-SET(v)
7            A = A ∪ {(u, v)}
8            UNION(u, v)
9  return A
```

Kruskal's Algorithm

- If the sets are implemented as a disjoint-set-forest, the total complexity is O(E log V).

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3       MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6       if FIND-SET(u) ≠ FIND-SET(v)
7            A = A ∪ {(u, v)}
8            UNION(u, v)
9  return A
```

4. Prim’s algorithm

Prim’s Algorithm

- By Robert Clay Prim, 1957
- Prim’s algorithm is a greedy algorithm.
- In Prim’s algorithm, the set A forms a single tree. The safe edge added to A is always a least-weight edge connecting the tree to a vertex not in the tree.
- Rather than build a subgraph one edge at a time, Prim’s algorithm builds a tree one vertex at a time

Prim's Algorithm

- The tree starts from an arbitrary root vertex r and grows until the tree spans all the vertices in V . Each step adds to the tree A a light edge that connects A to an isolated vertex.

- By Corollary 23.2, this rule adds only edges that are safe for A; therefore, when the algorithm terminates, the edges in A form a minimum spanning tree.

- This strategy qualifies as greedy since at each step it adds to the tree an edge that contributes the minimum amount possible to the tree's weight.

Prim's Algorithm

- During execution of the algorithm, all vertices that are not in the tree reside in a min-priority queue Q based on a key attribute. For each vertex v, the attribute v.key is the minimum weight of any edge connecting to a vertex in the tree; by convention, v.key = ∞ if there is no such edge.
- The attribute v.π names the parent of v in the tree. The algorithm implicitly maintains the set A from GENERIC-MST as

$$A = \{(v, v.\pi) : v \in V - \{r\} - Q\} .$$

When the algorithm terminates, the min-priority queue Q is empty; the minimum spanning tree A for G is thus

$$A = \{(v, v.\pi) : v \in V - \{r\}\} .$$

Prim's Algorithm

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

Prim's Algorithm

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

The algorithm maintains the following three-part loop invariant:

Prior to each iteration of the **while** loop of lines 6–11,

1. $A = \{(v, v.\pi) : v \in V - \{r\} - Q\}$.
2. The vertices already placed into the minimum spanning tree are those in $V - Q$.
3. For all vertices $v \in Q$, if $v.\pi \neq \text{NIL}$, then $v.key < \infty$ and $v.key$ is the weight of a light edge $(v, v.\pi)$ connecting v to some vertex already placed into the minimum spanning tree.

Prim's Algorithm

Time Complexity:

- The running time of Prim's algorithm depends on how we implement the min-priority queue Q.

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

Prim's Algorithm

Time Complexity (Min Heap):

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

- If we implement Q as a binary min-heap, we can use
- BUILD-MIN-HEAP: **O(V)** time.
- The while loop executes V times.
- EXTRACT-MIN: O(lg V) time, total **O(V lg V)**.
- Lines 8–11 executes O(E) times altogether.
- Implicit DECREASE-KEY: O(lg V), total **O(E lg V)**.
- The total time for Prim's algorithm is

  O(V lg V + E lg V) = O(E lg V).

Prim's Algorithm

Time Complexity (Fibonacci Heap):

- If we implement Q as a binary min-heap, we can use
- BUILD-MIN-HEAP: **O(V)** time.
- The while loop executes V times.
- EXTRACT-MIN: O(lg V) time, total **O(V lg V)**.
- Lines 8–11 executes O(E) times altogether.
- Implicit DECREASE-KEY: O(1), total **O(E)**.
- The total time for Prim's algorithm is

  O(V lg V + E).

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

Prim's Algorithm

Find the MST for the following graph:

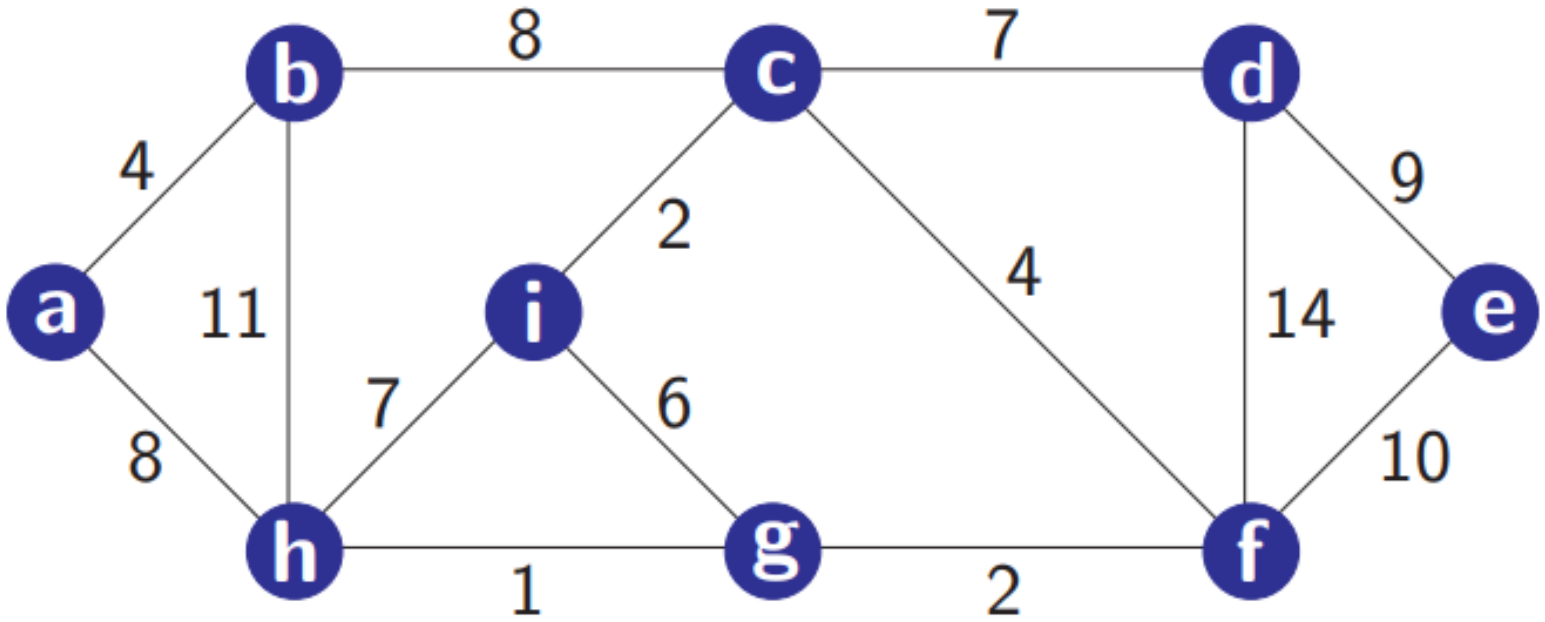


Image by Yoan Pinzón.

Prim’s Algorithm

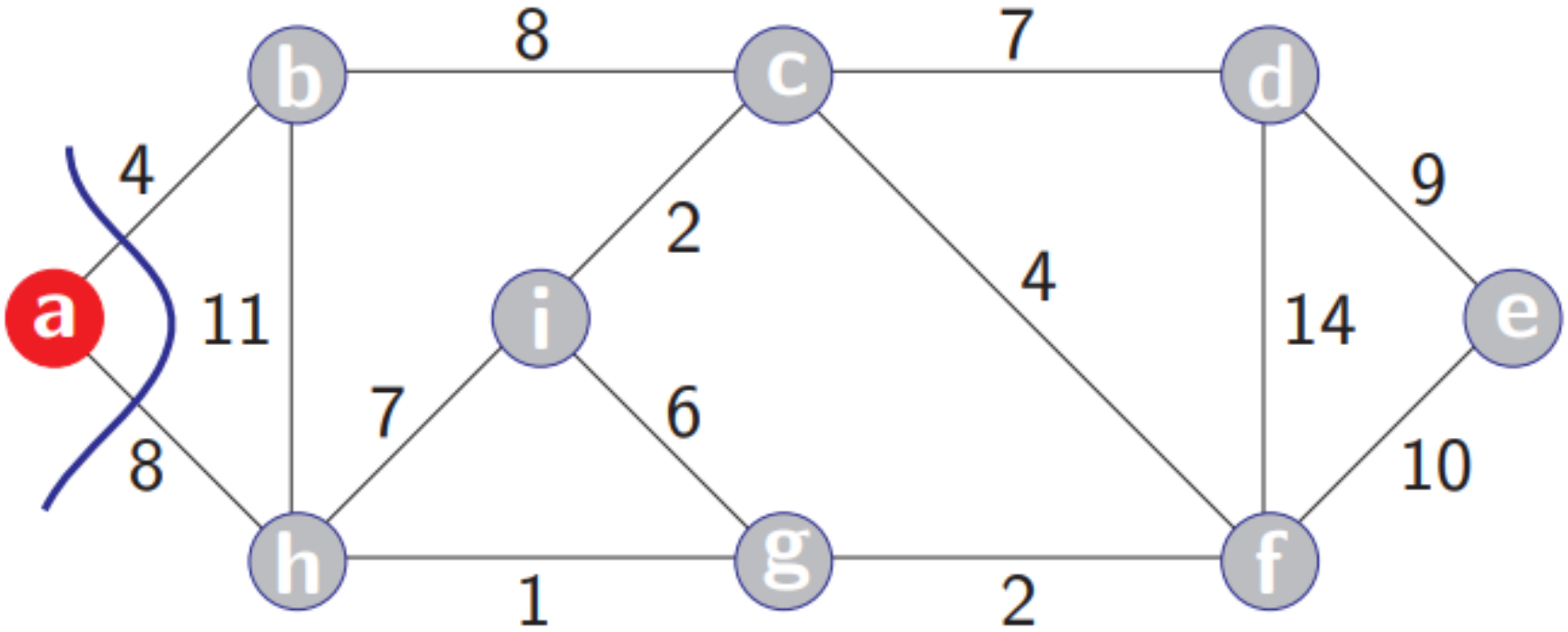


Image by Yoan Pinzón.

Prim’s Algorithm

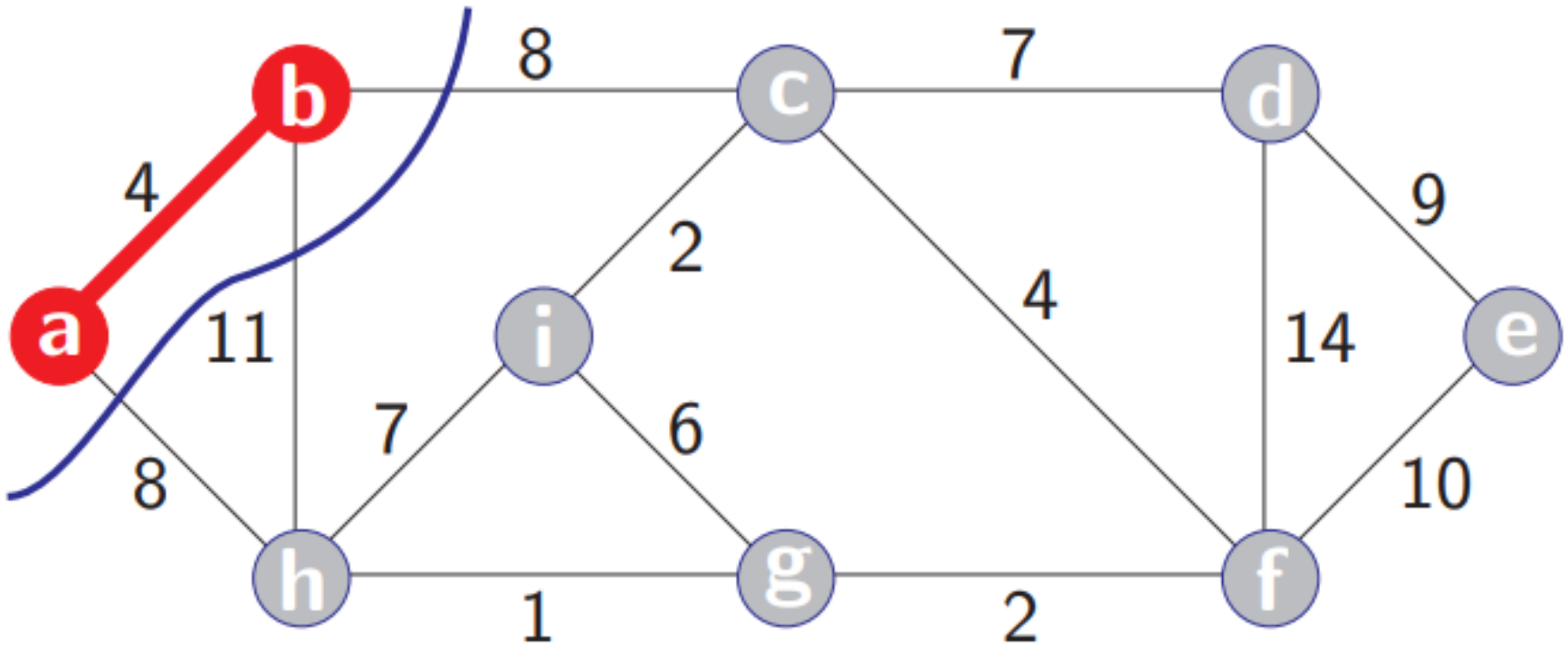


Image by Yoan Pinzón.

Prim's Algorithm

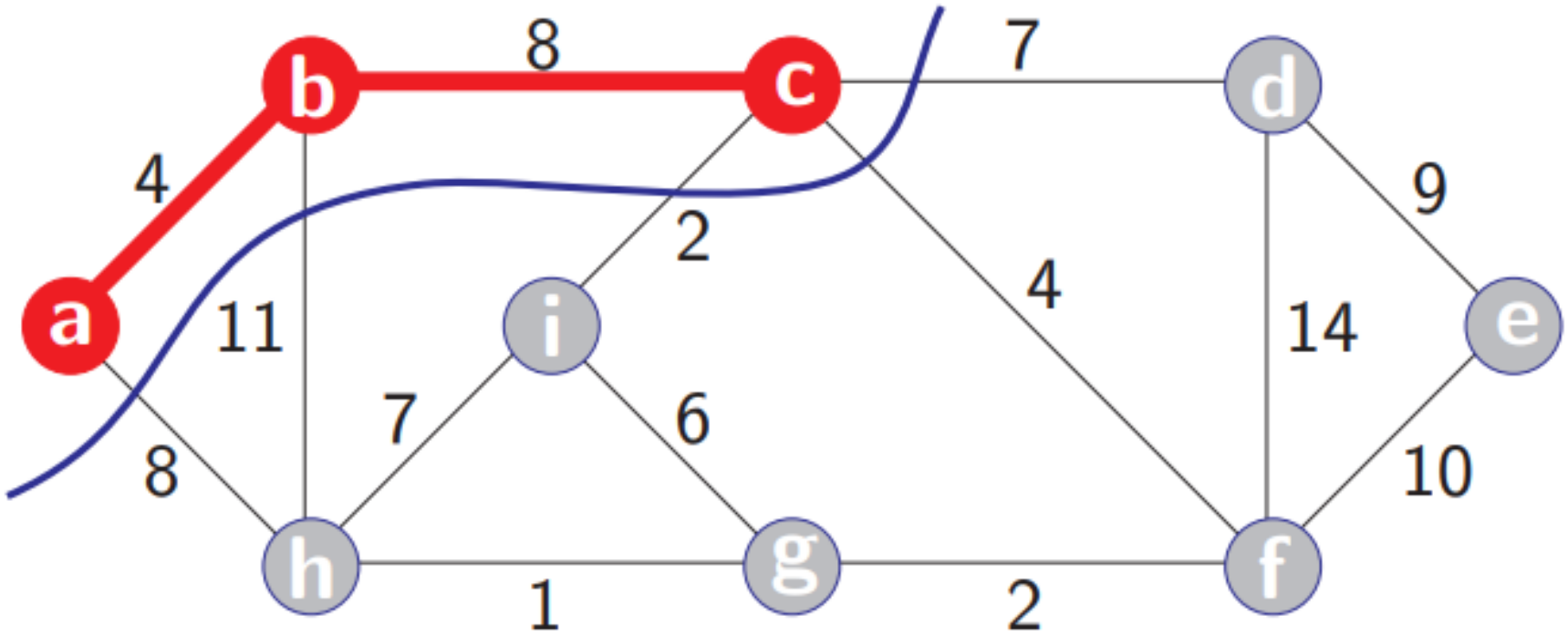


Image by Yoan Pinzón.

Prim's Algorithm

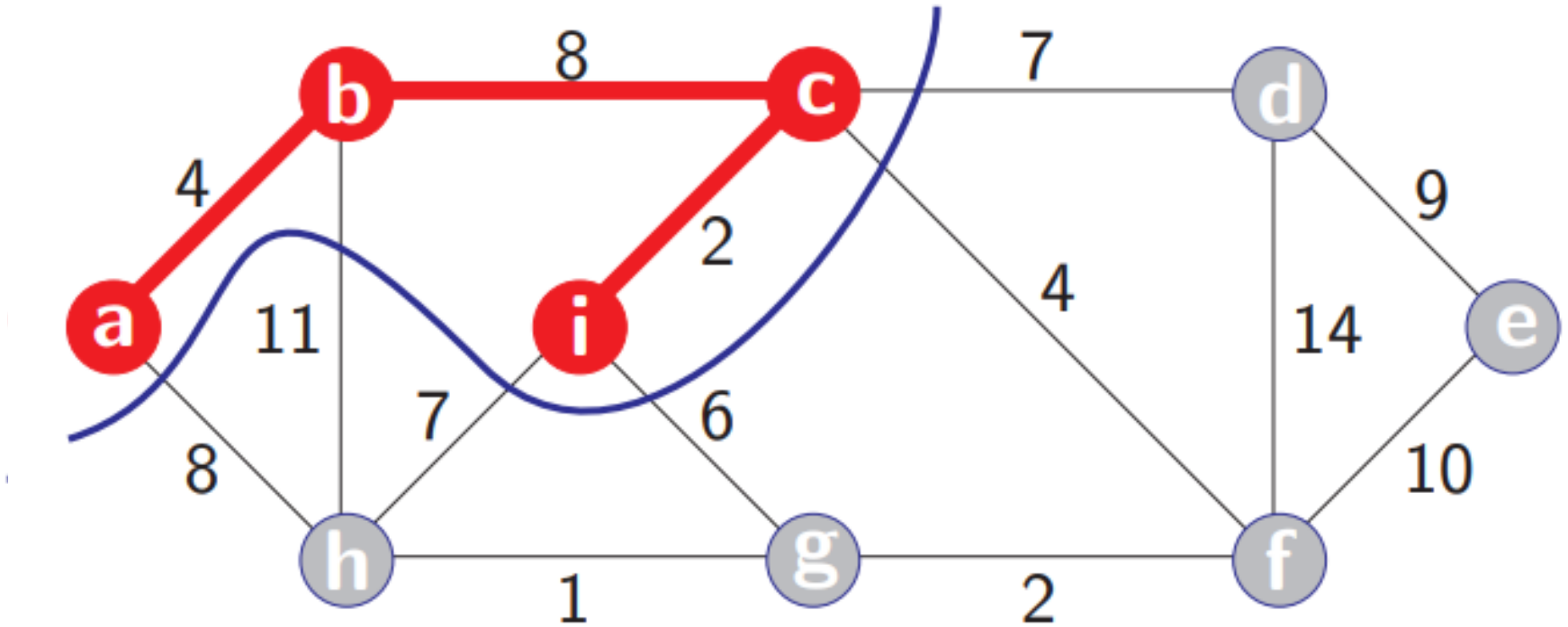


Image by Yoan Pinzón.

Prim's Algorithm

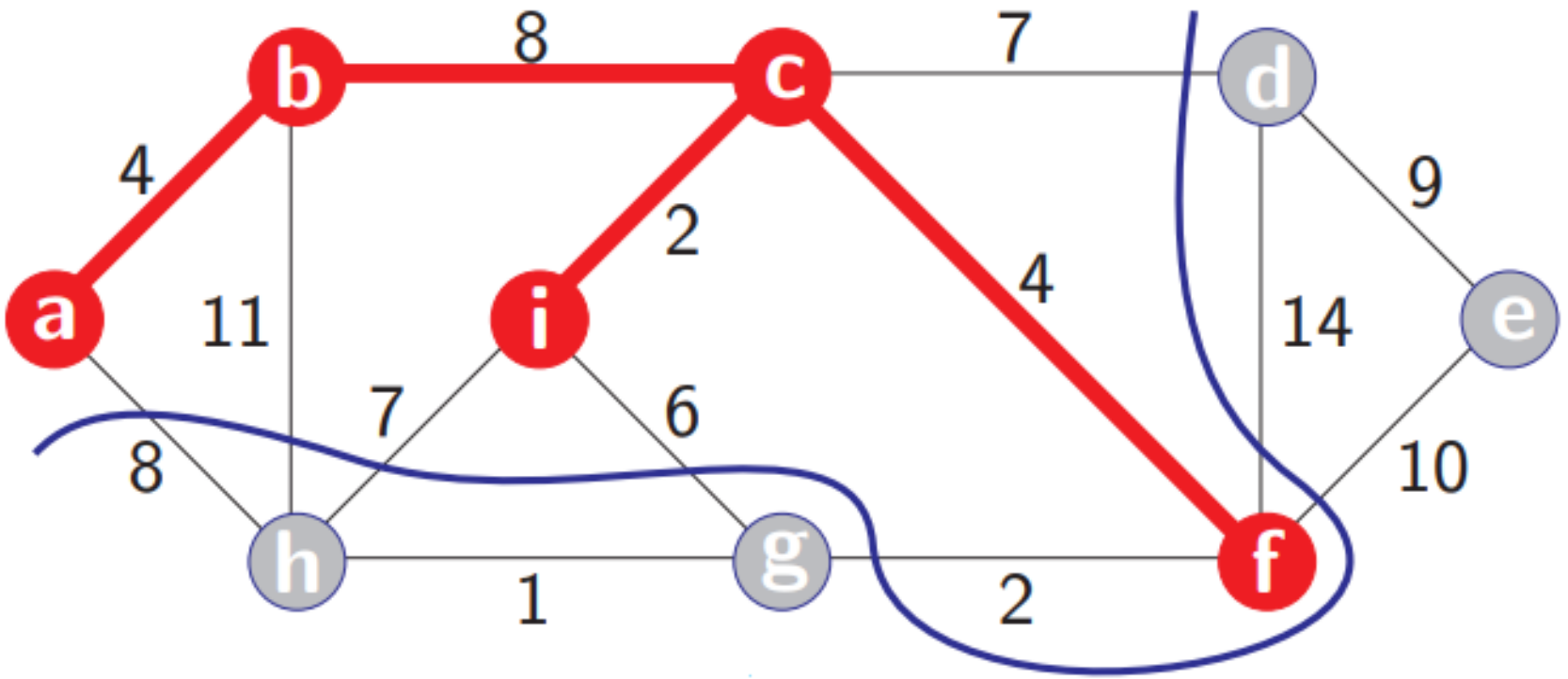


Image by Yoan Pinzón.

Prim's Algorithm

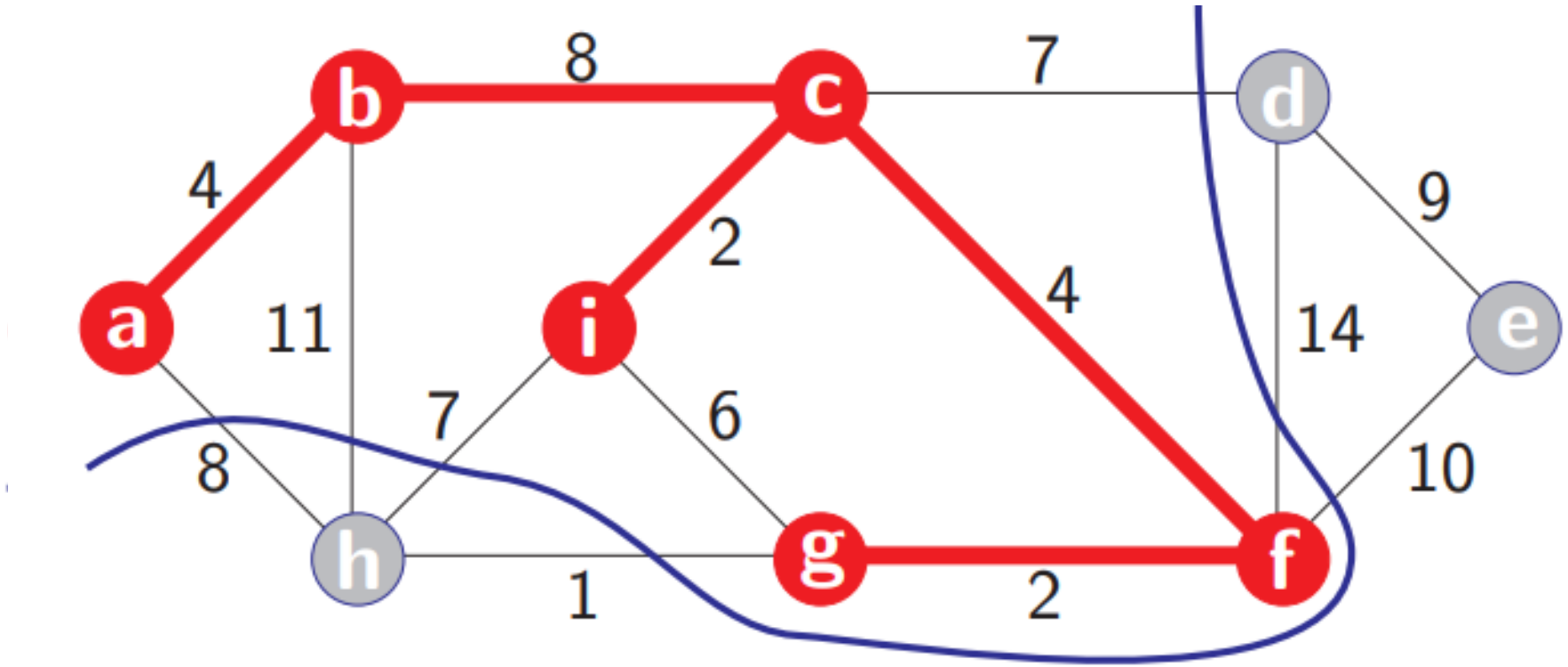


Image by Yoan Pinzón.

Prim's Algorithm

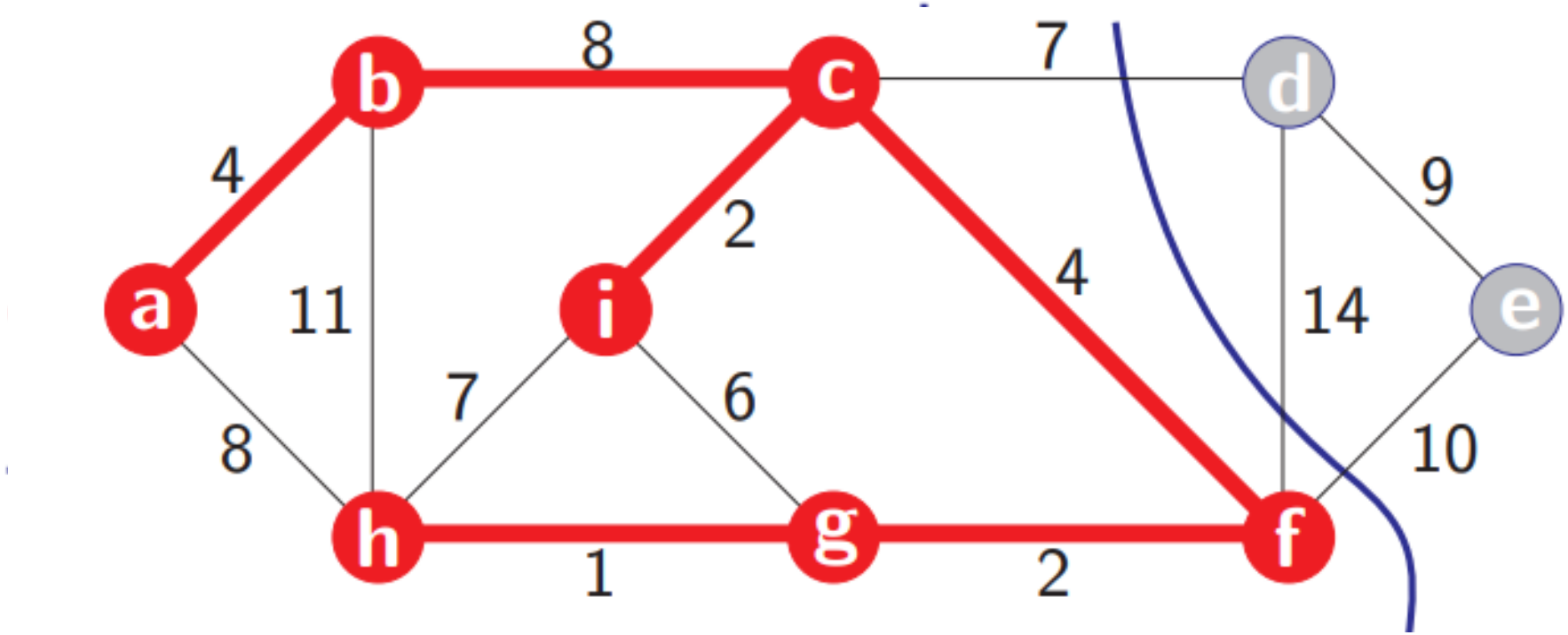


Image by Yoan Pinzón.

Prim's Algorithm

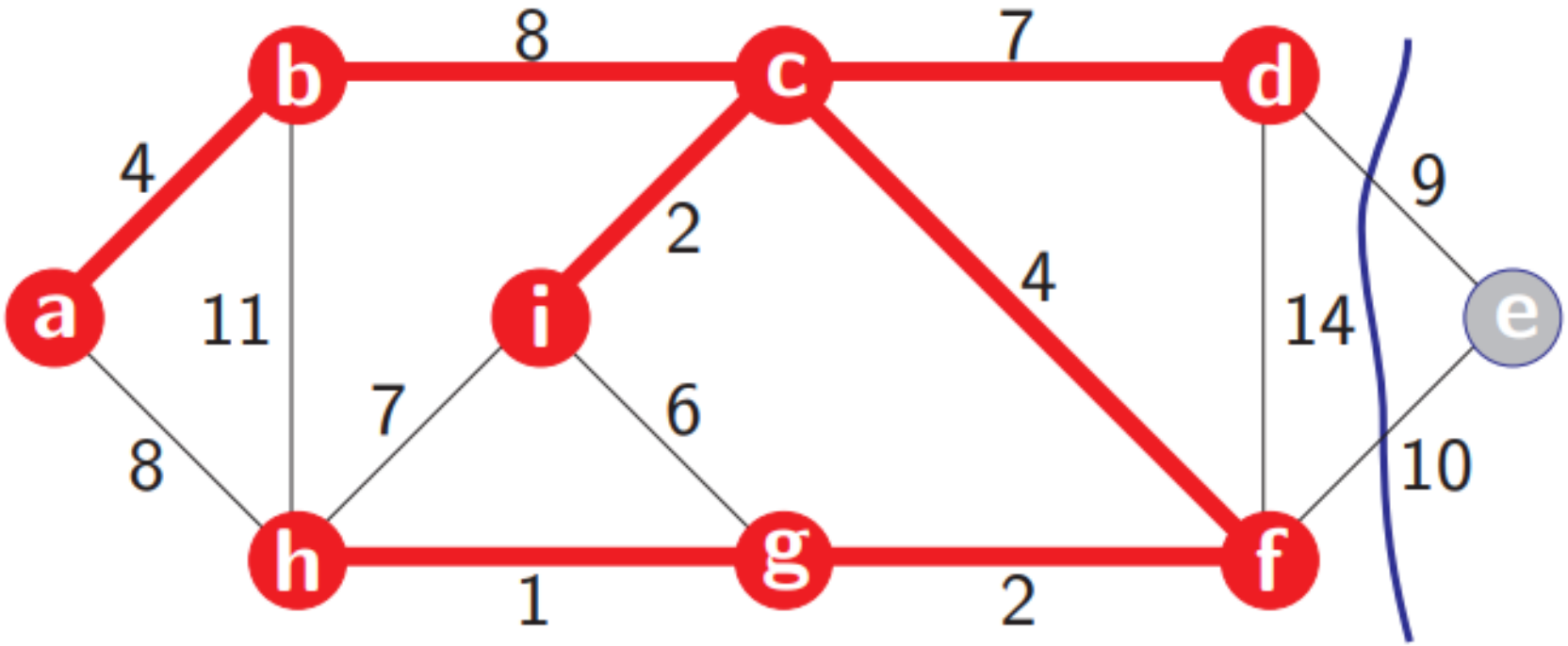


Image by Yoan Pinzón.

Prim’s Algorithm

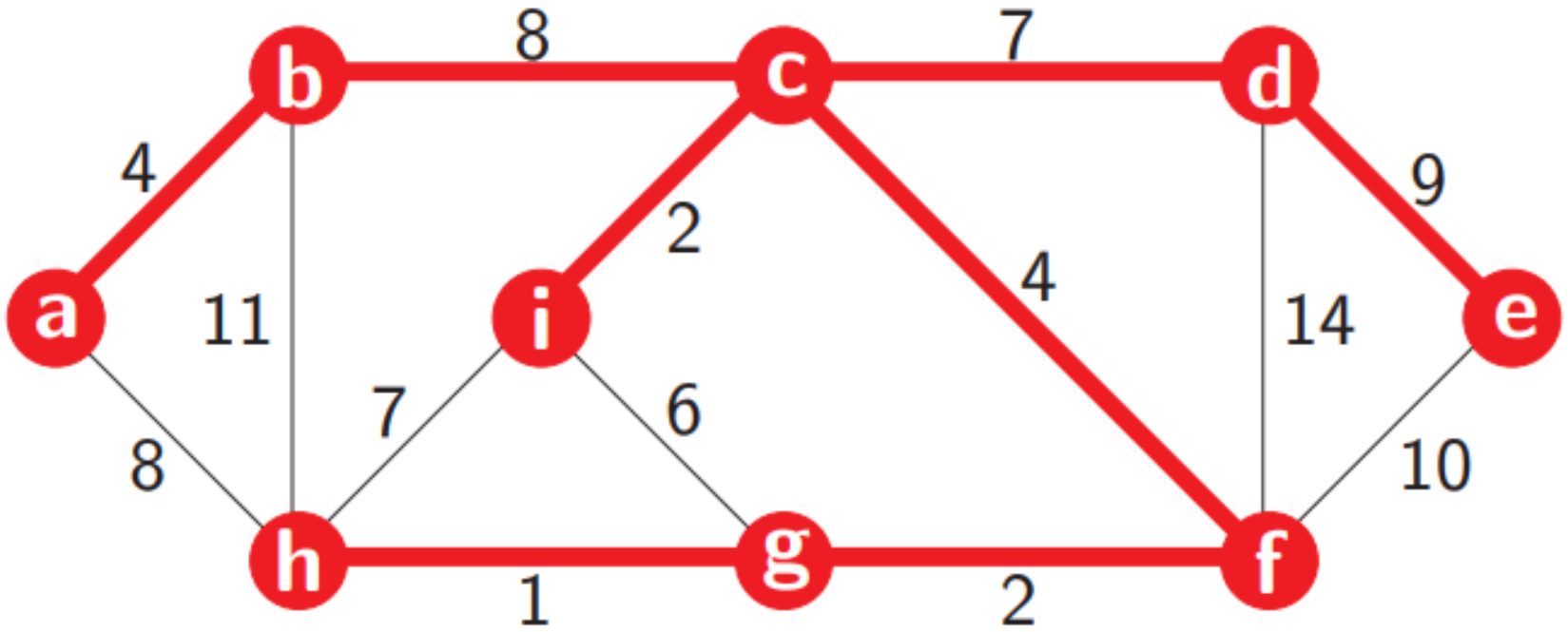


Image by Yoan Pinzón.

Prim/Kruskal – Exercise

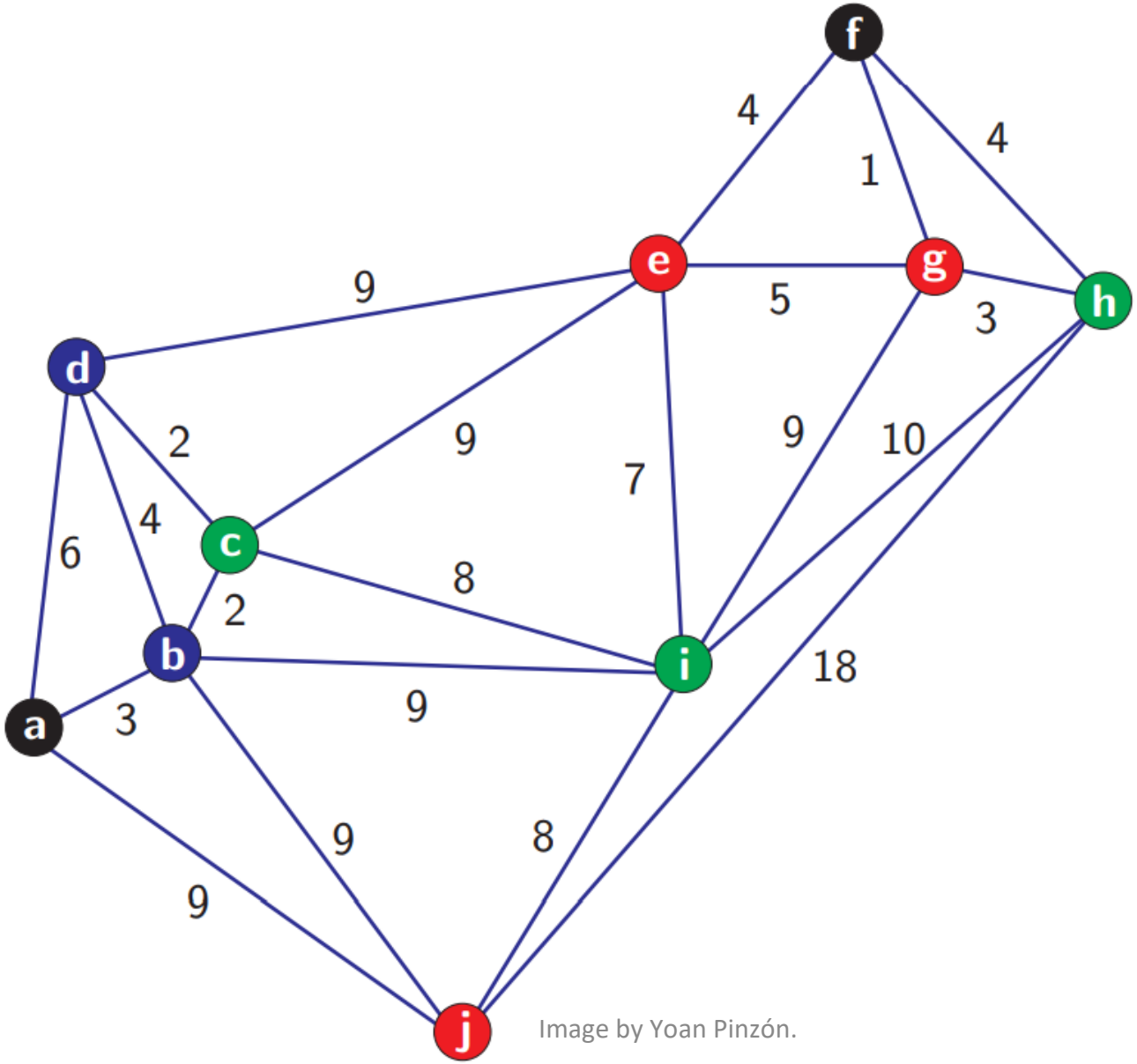


Image by Yoan Pinzón.

BIBLIOGRAPHY

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms, Third Edition. The MIT Press. 2009.
- Advanced Algorithms Slides by Yoan Pinzón. Figures from this presentation were included.