Lecture 30

Greedy: MST, Kruskal's Algorithm

Source: Introduction to Algorithms, CLRS

Cut Connection of MST

Cut Connection of MST

Let’s amend the previous lemma so that it can help us in building an MST!

Cut Connection of MST

Let's amend the previous lemma so that it can help us in building an MST!

Lemma: Let $G(V, E, w)$ an undirected, weighted and connected graph.

Cut Connection of MST

Let's amend the previous lemma so that it can help us in building an MST!

Lemma: Let $G(V, E, w)$ an undirected, weighted and connected graph. Let A be a subset of E

Cut Connection of MST

Let's amend the previous lemma so that it can help us in building an MST!

Lemma: Let $G(V, E, w)$ an undirected, weighted and connected graph. Let A be a subset of E that is included in some MST of G.

Cut Connection of MST

Let's amend the previous lemma so that it can help us in building an MST!

Lemma: Let $G(V, E, w)$ an undirected, weighted and connected graph. Let A be a subset of E that is included in some MST of G. Let $C = (S, T)$ be any cut in G so that cut set of C does not

Cut Connection of MST

Let's amend the previous lemma so that it can help us in building an MST!

Lemma: Let $G(V, E, w)$ an undirected, weighted and connected graph. Let A be a subset of E that is included in some MST of G. Let $C = (S, T)$ be any cut in G so that cut set of C does not have any edge from A.

Cut Connection of MST

Let's amend the previous lemma so that it can help us in building an MST!

Lemma: Let $G(V, E, w)$ an undirected, weighted and connected graph. Let A be a subset of E that is included in some MST of G. Let $C = (S, T)$ be any cut in G so that cut set of C does not have any edge from A. If e is the least weight edge in the cut-set of C,

Cut Connection of MST

Let's amend the previous lemma so that it can help us in building an MST!

Lemma: Let $G(V, E, w)$ an undirected, weighted and connected graph. Let A be a subset of E that is included in some MST of G. Let $C = (S, T)$ be any cut in G so that cut set of C does not have any edge from A. If e is the least weight edge in the cut-set of C, then $A \cup \{e\}$ is also part

Cut Connection of MST

Let's amend the previous lemma so that it can help us in building an MST!

Lemma: Let $G(V, E, w)$ an undirected, weighted and connected graph. Let A be a subset of E that is included in some MST of G. Let $C = (S, T)$ be any cut in G so that cut set of C does not have any edge from A. If e is the least weight edge in the cut-set of C, then $A \cup \{e\}$ is also part of some MST of G.

Cut Connection of MST

Let's amend the previous lemma so that it can help us in building an MST!

Lemma: Let $G(V, E, w)$ an undirected, weighted and connected graph. Let A be a subset of E that is included in some MST of G. Let $C = (S, T)$ be any cut in G so that cut set of C does not have any edge from A. If e is the least weight edge in the cut-set of C, then $A \cup \{e\}$ is also part of some MST of G.

Proof:

Cut Connection of MST

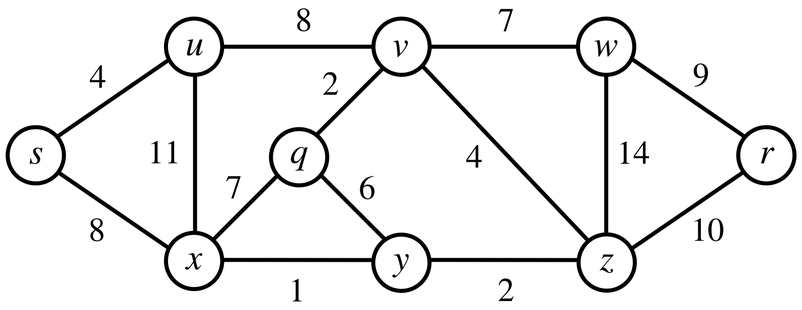
Let's amend the previous lemma so that it can help us in building an MST!

Lemma: Let $G(V, E, w)$ an undirected, weighted and connected graph. Let A be a subset of E that is included in some MST of G. Let $C = (S, T)$ be any cut in G so that cut set of C does not have any edge from A. If e is the least weight edge in the cut-set of C, then $A \cup \{e\}$ is also part of some MST of G.

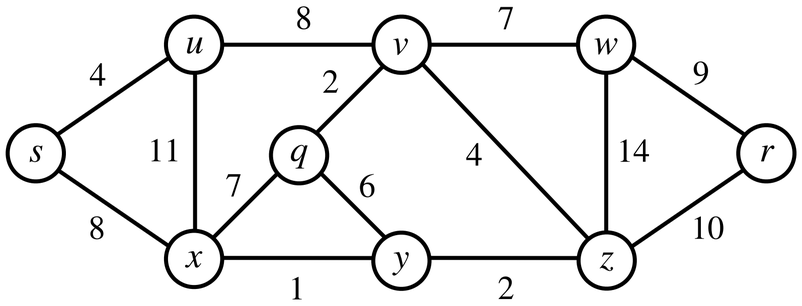
Proof: Similar to the previous proof.

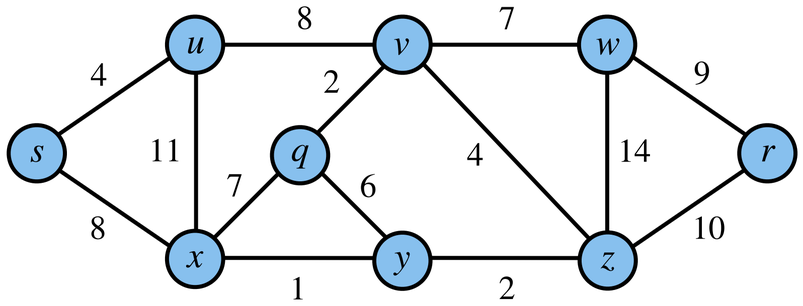
Kruskal's Algorithm: Demonstration

Kruskal's Algorithm: Demonstration

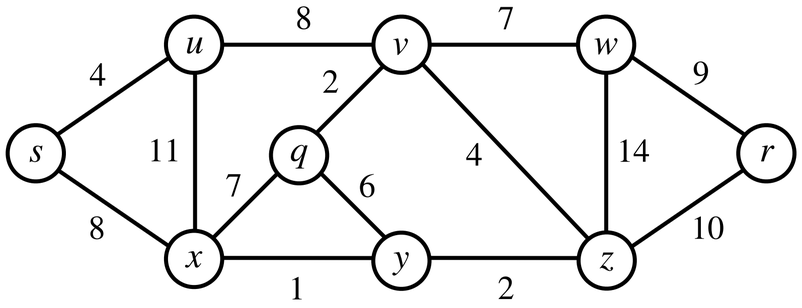


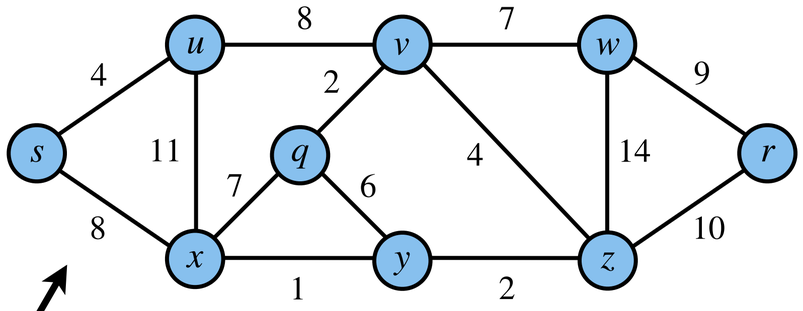
Kruskal's Algorithm: Demonstration





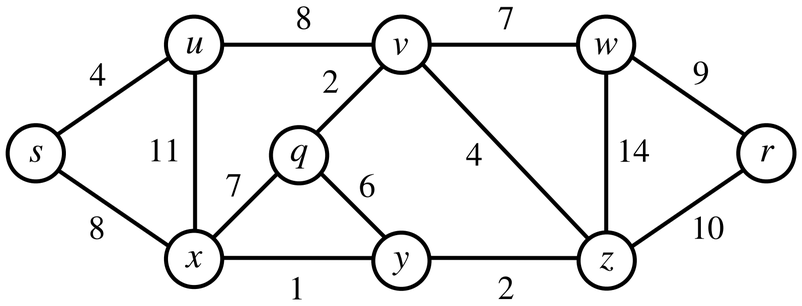
Kruskal's Algorithm: Demonstration

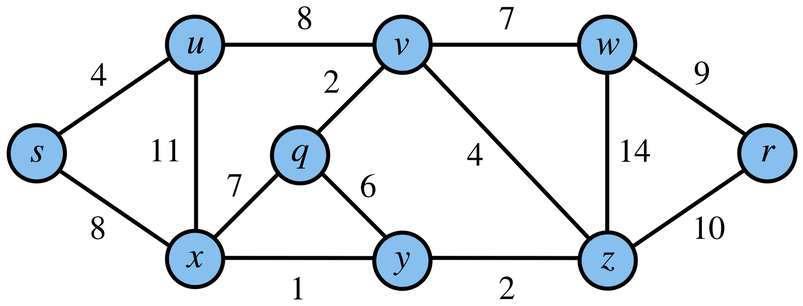




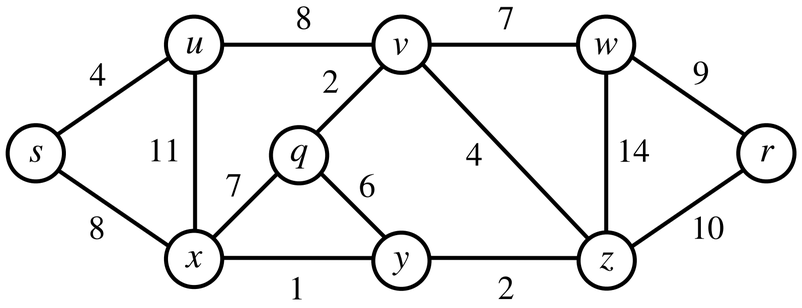
Start with single node subtrees.

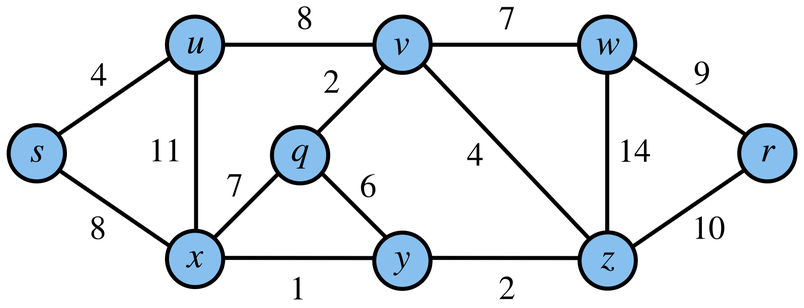
Kruskal's Algorithm: Demonstration

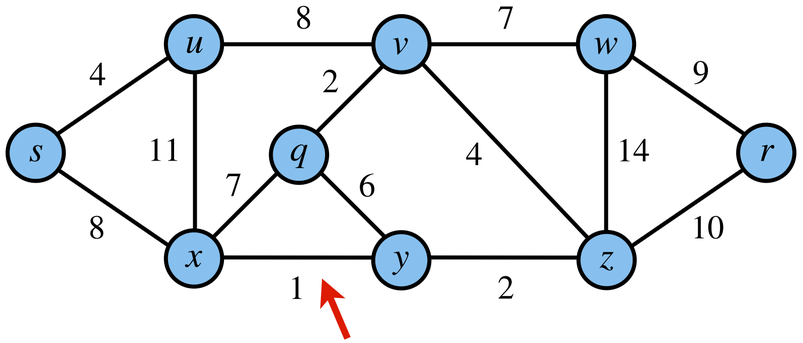




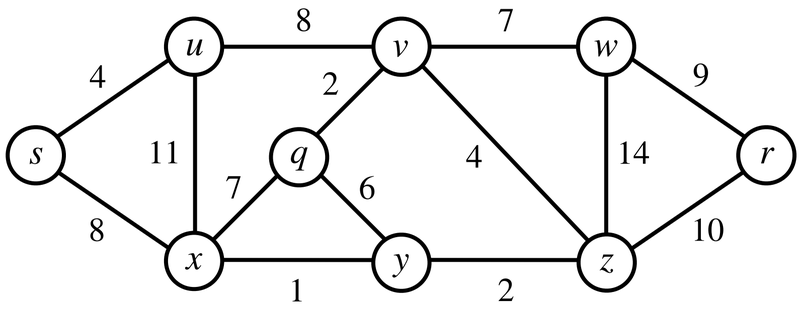
Kruskal's Algorithm: Demonstration

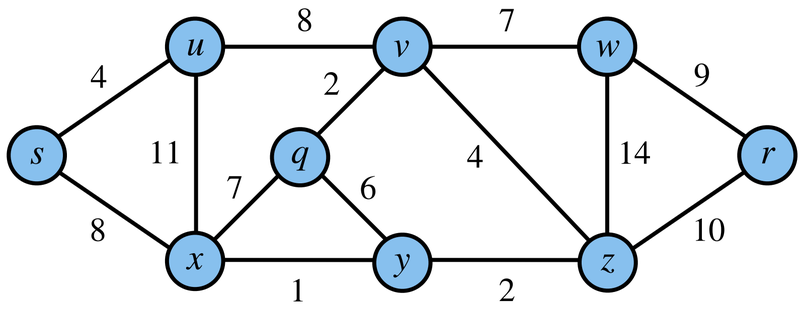


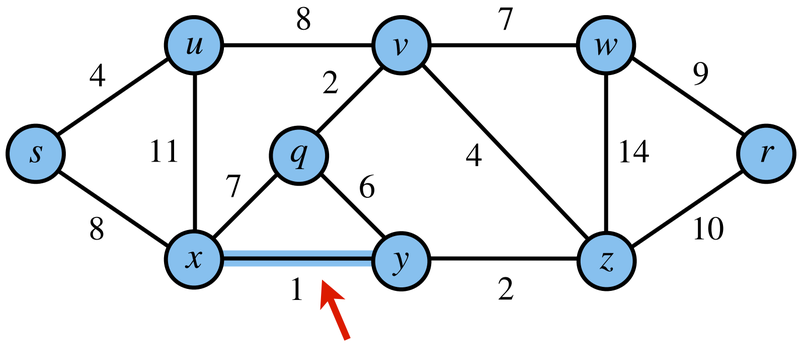




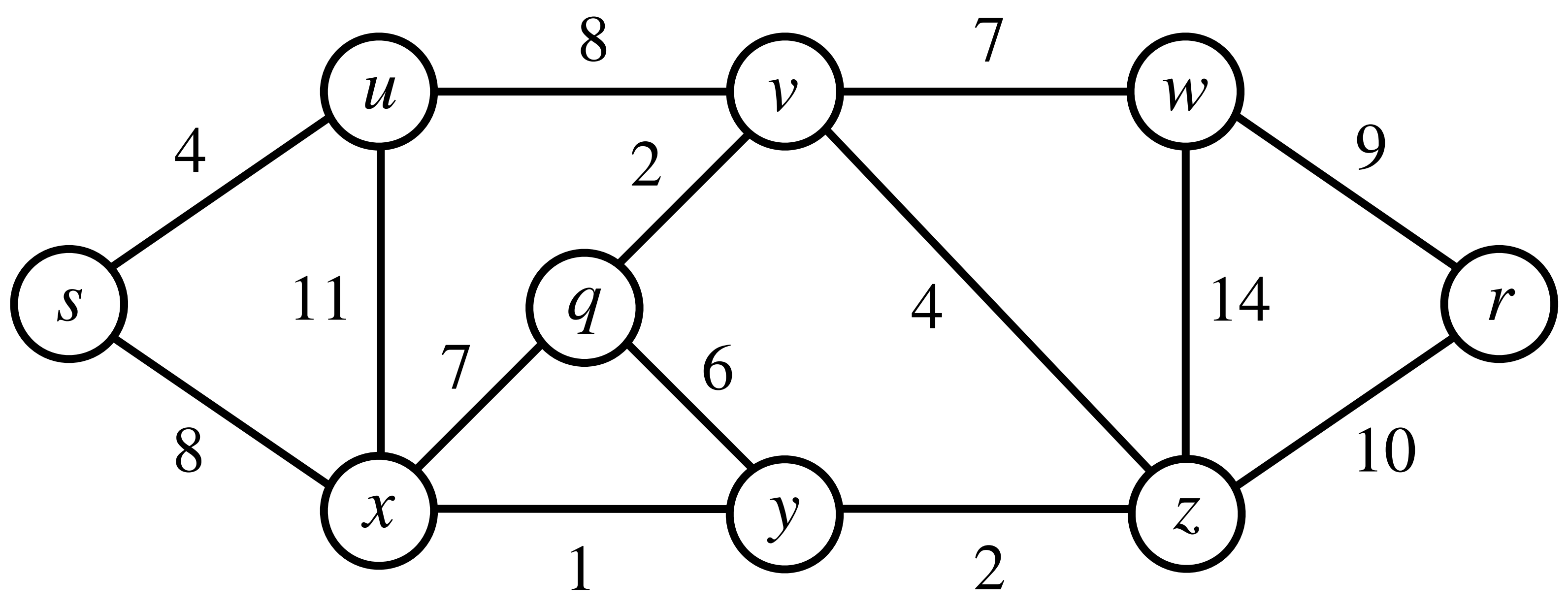
Kruskal's Algorithm: Demonstration

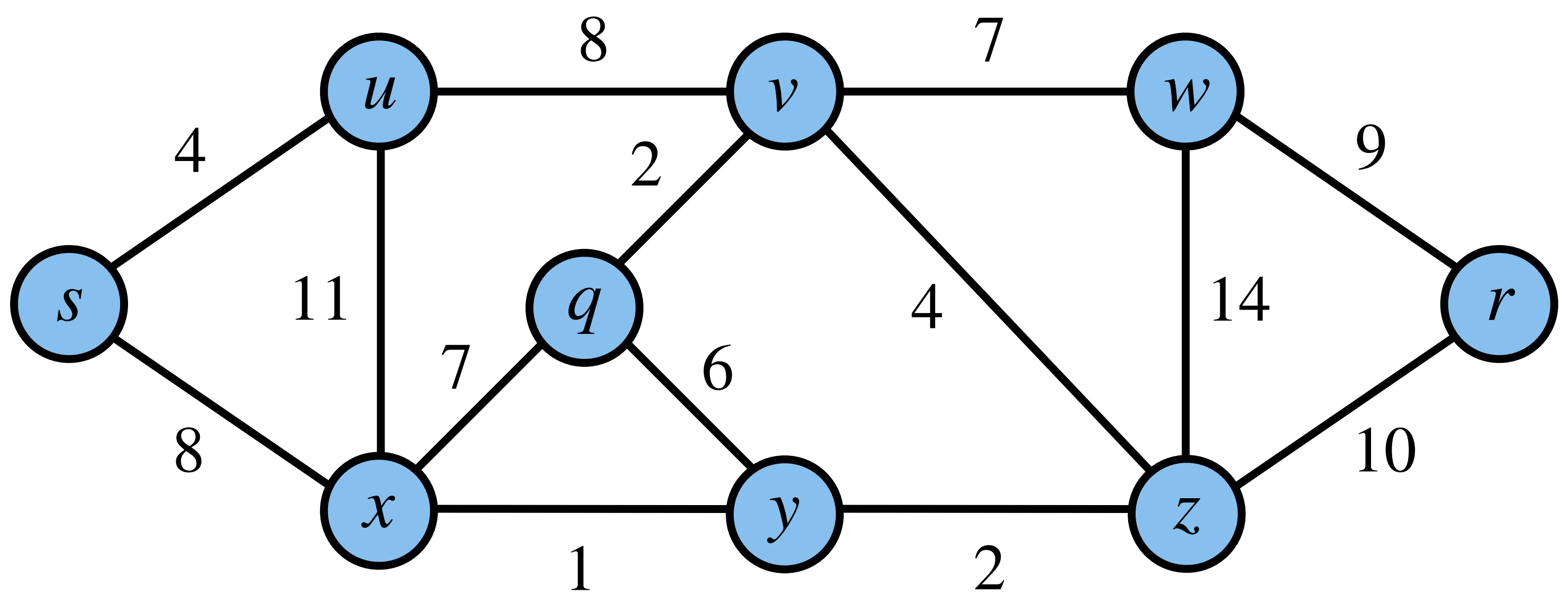


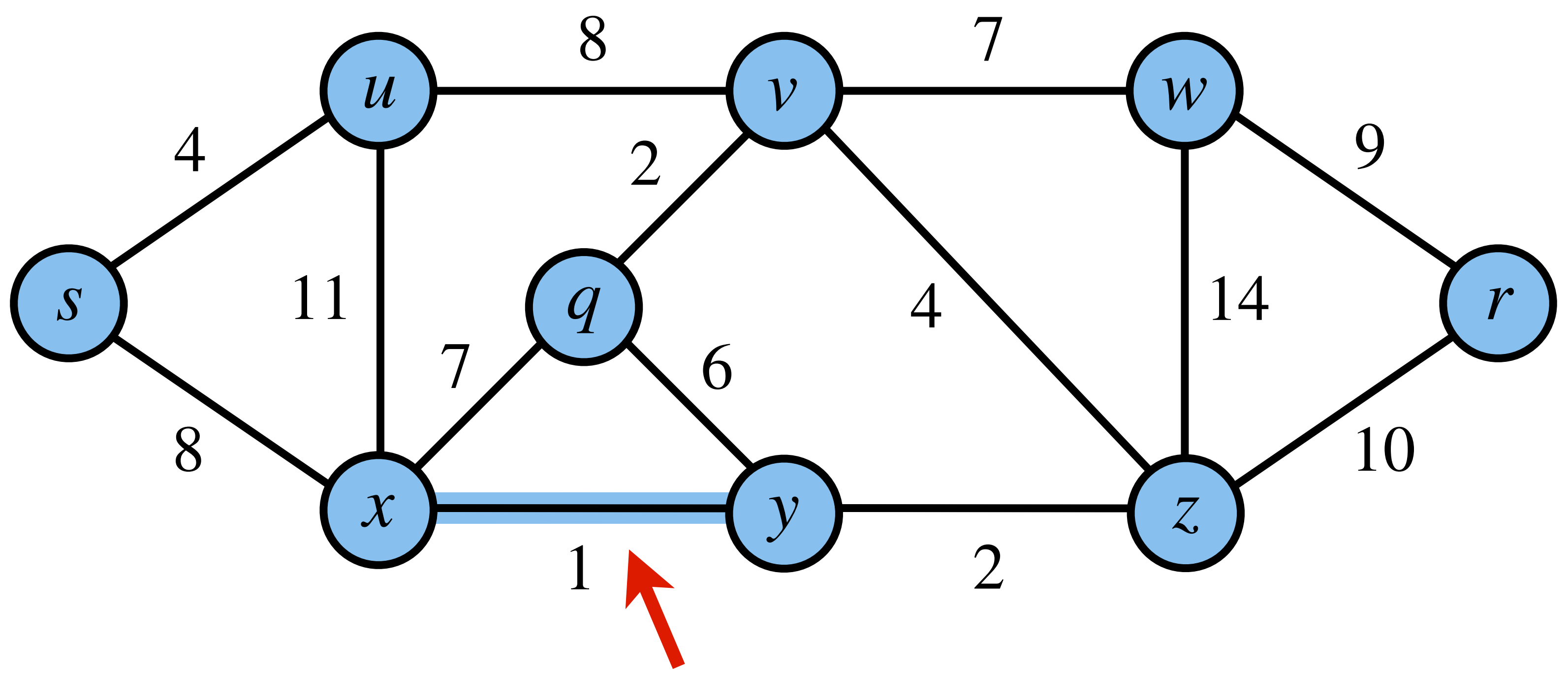


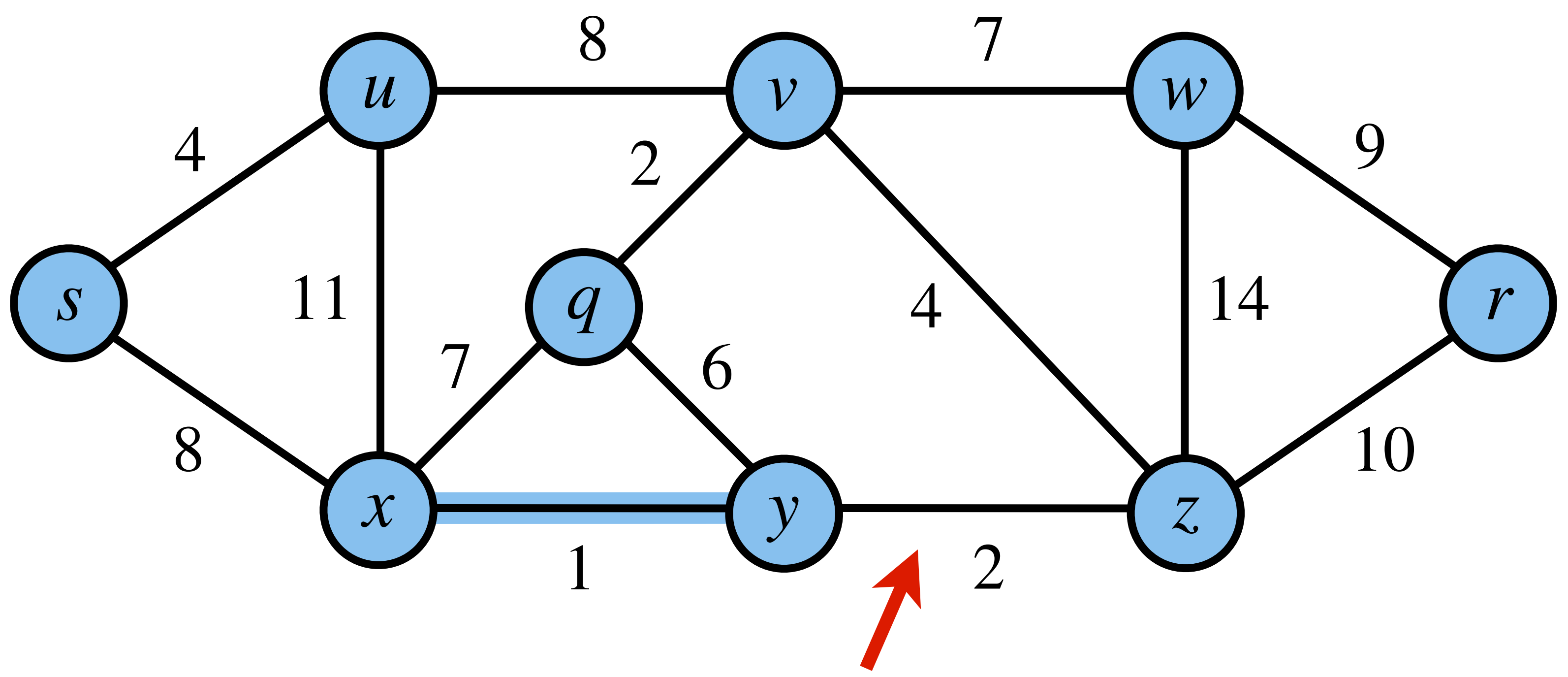


Kruskal’s Algorithm: Demonstration

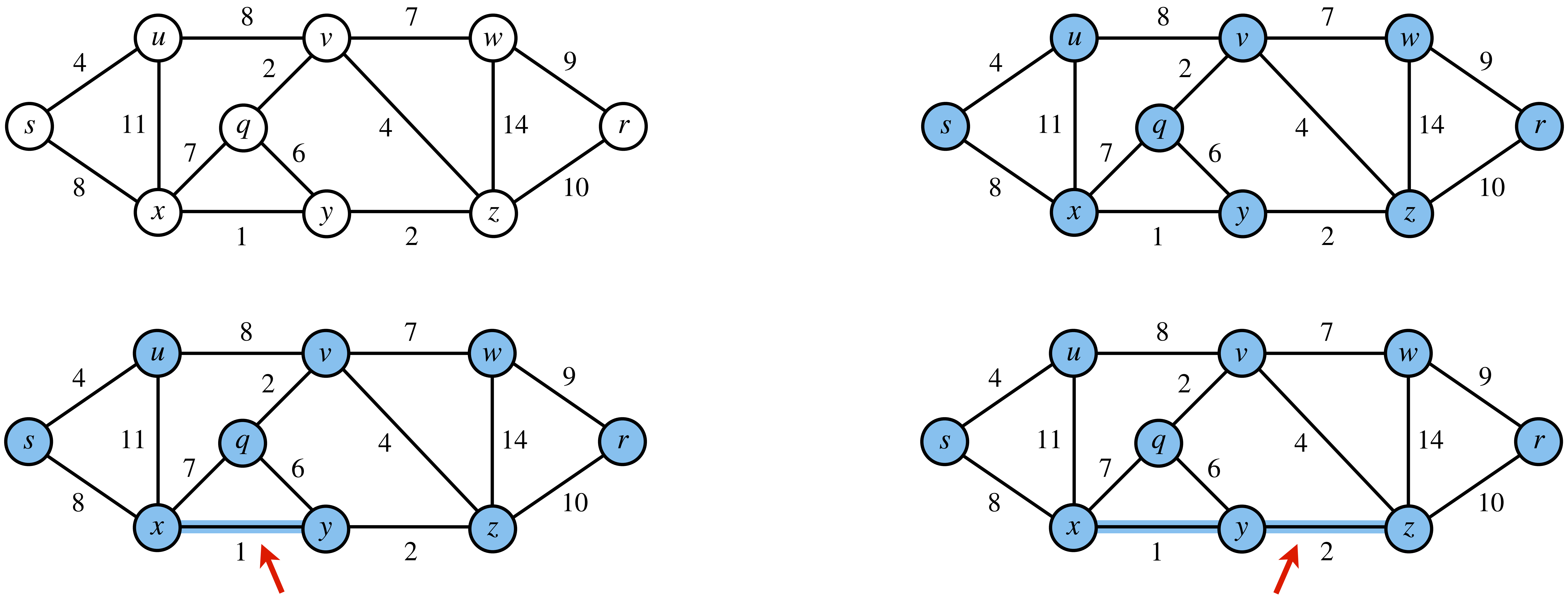




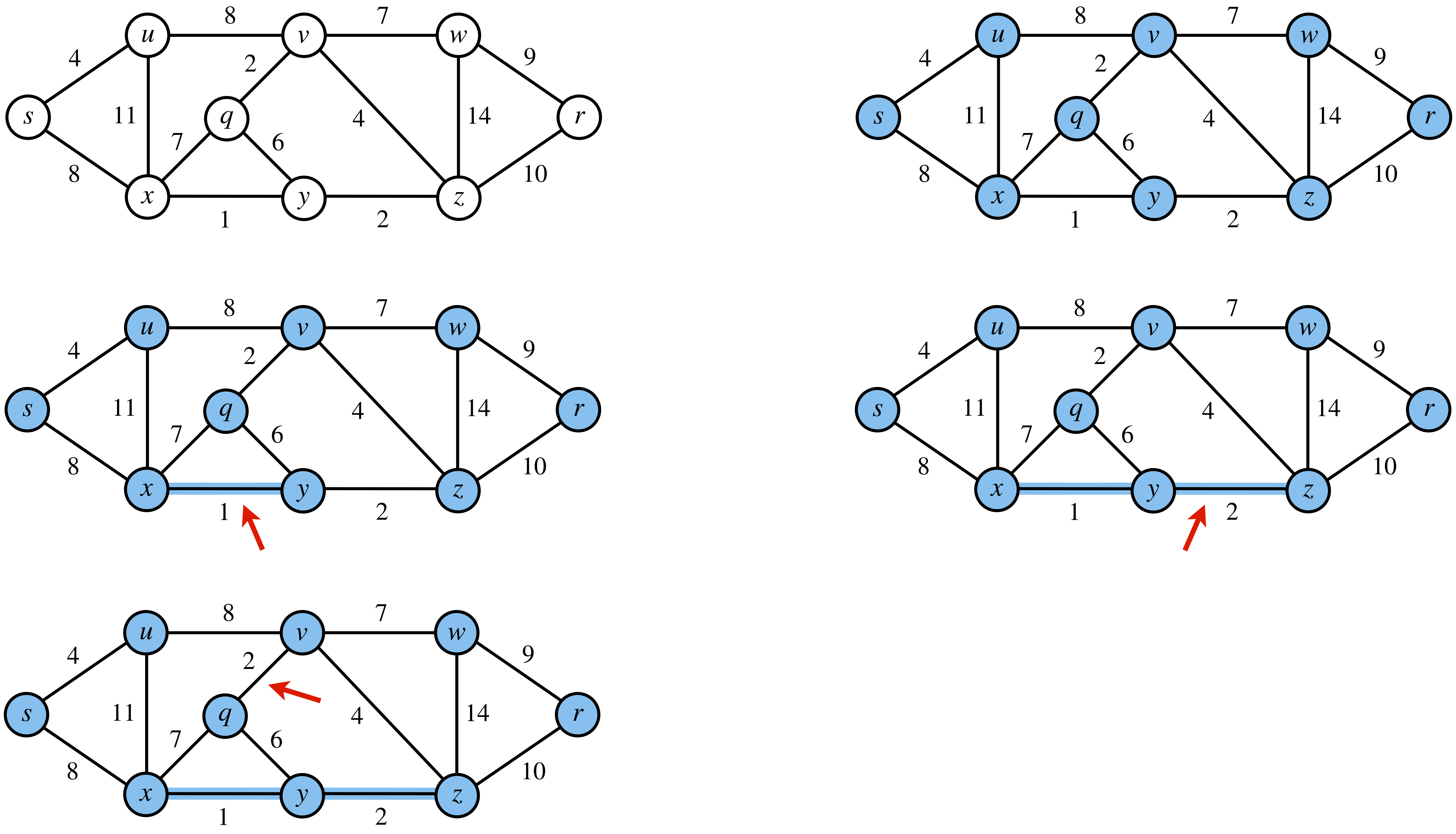




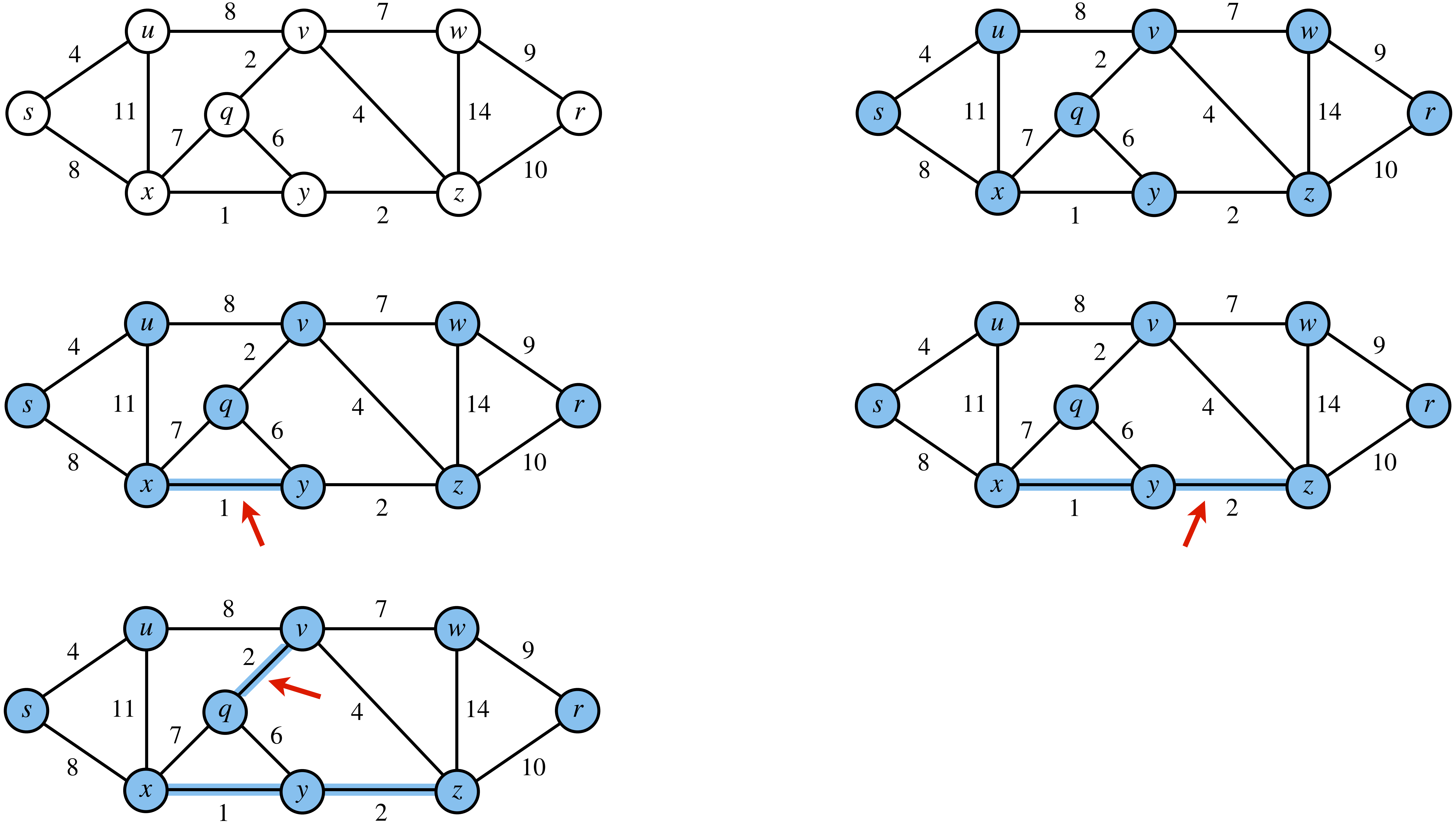
Kruskal’s Algorithm: Demonstration



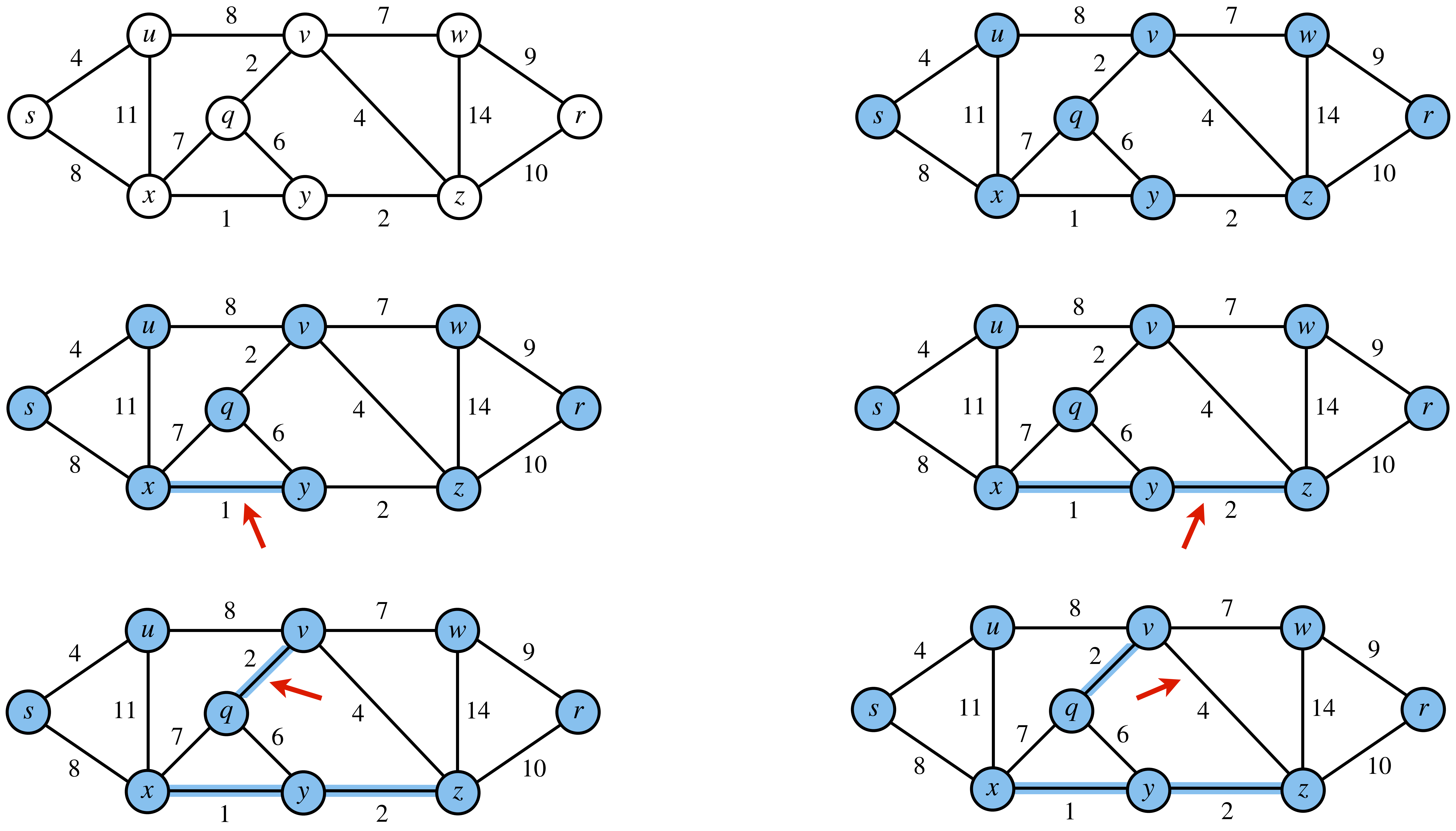
Kruskal's Algorithm: Demonstration



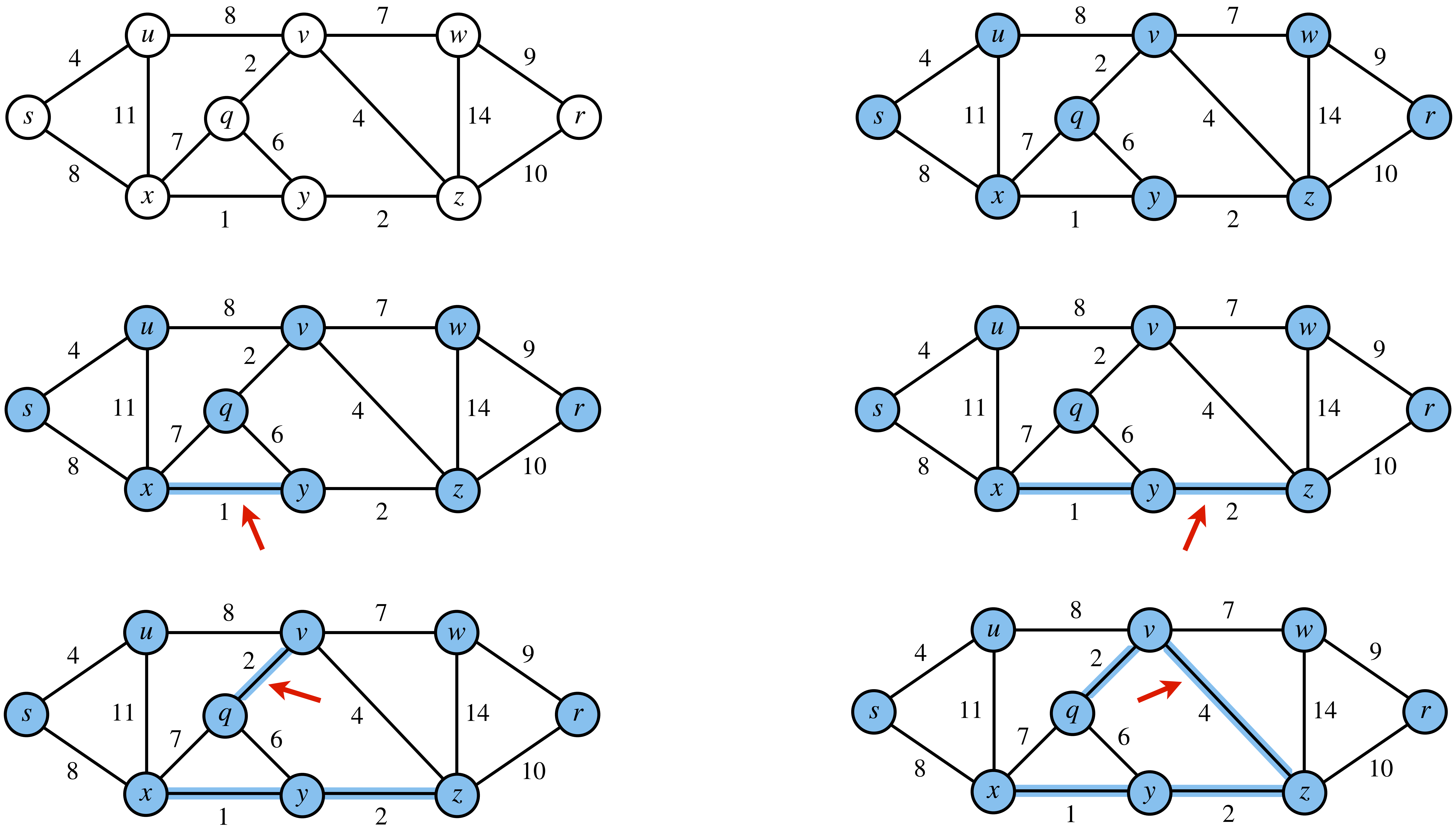
Kruskal's Algorithm: Demonstration



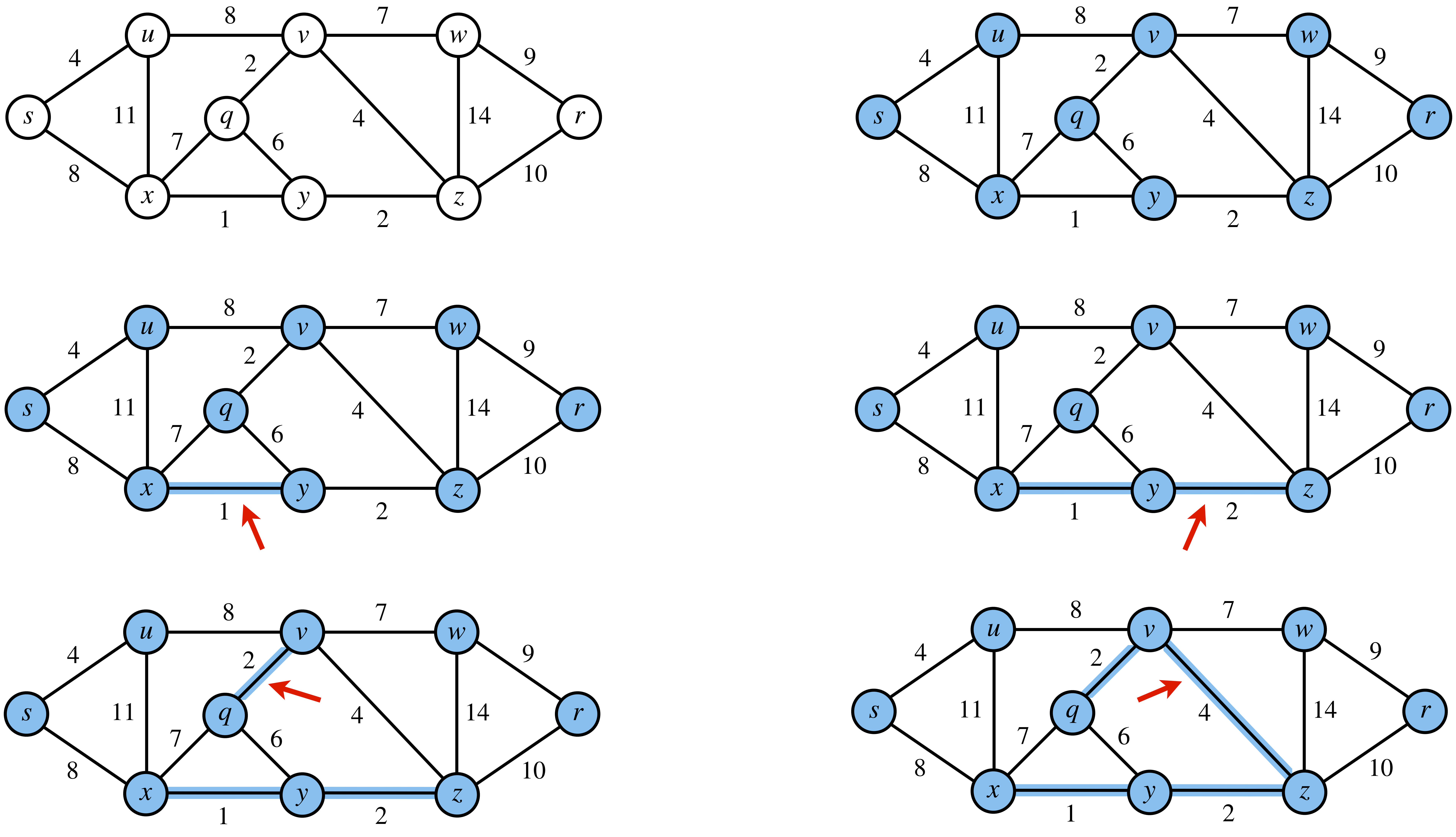
Kruskal’s Algorithm: Demonstration



Kruskal’s Algorithm: Demonstration



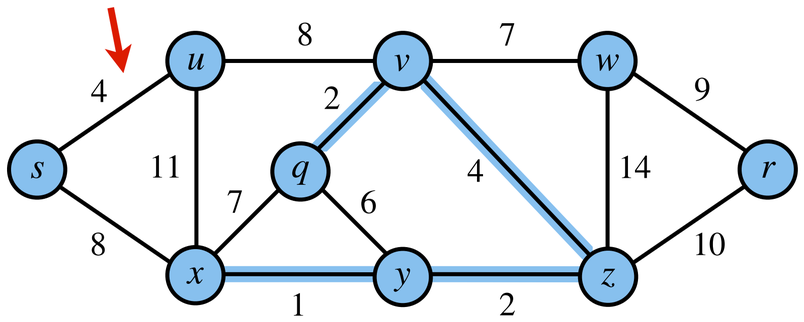
Kruskal’s Algorithm: Demonstration



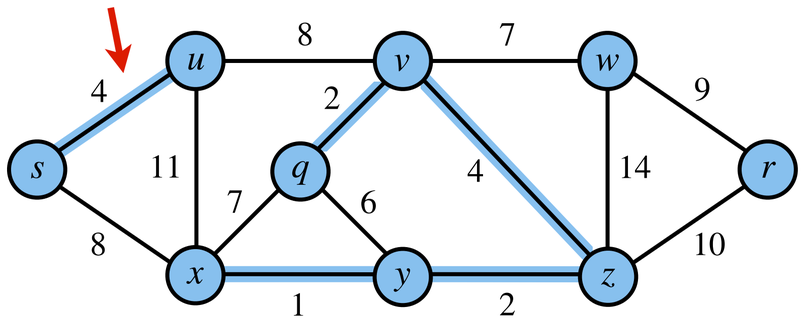
…

Kruskal’s Demonstration

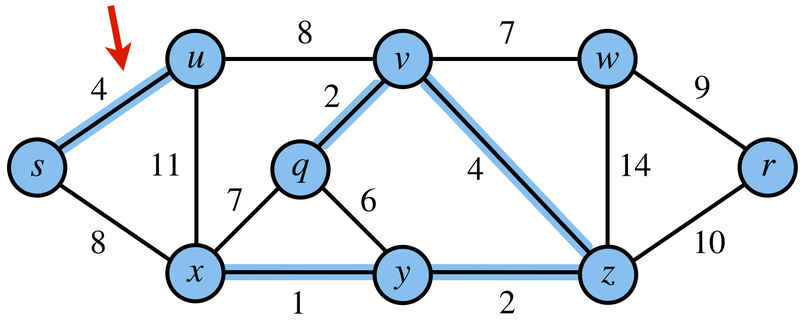
Kruskal's Demonstration

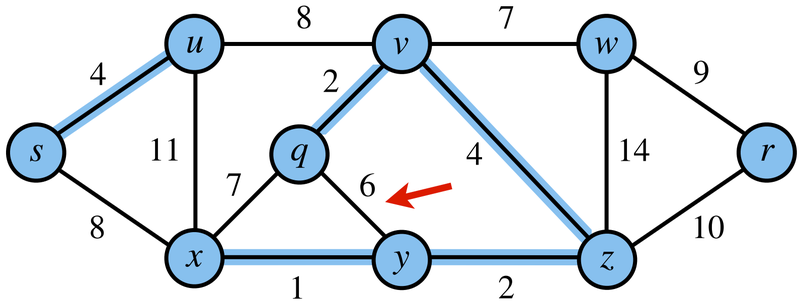


Kruskal's Demonstration

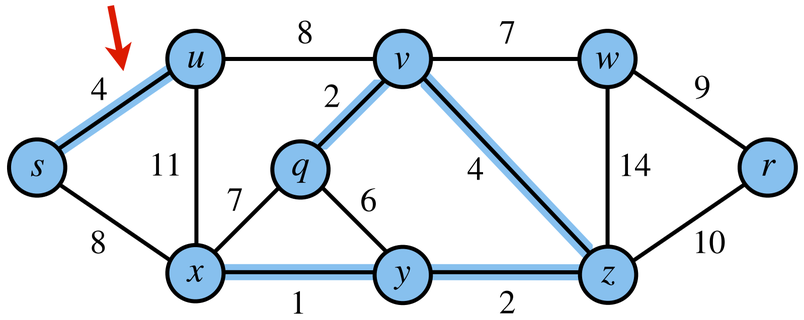


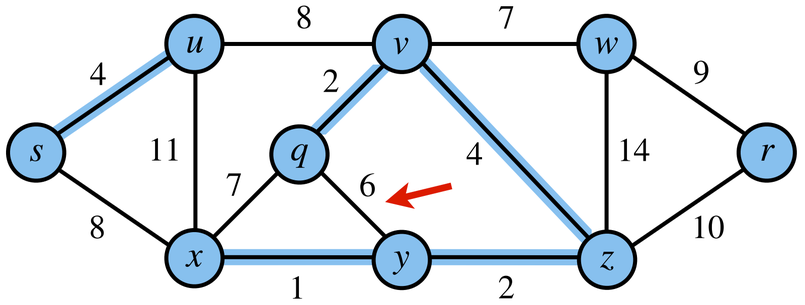
Kruskal's Demonstration

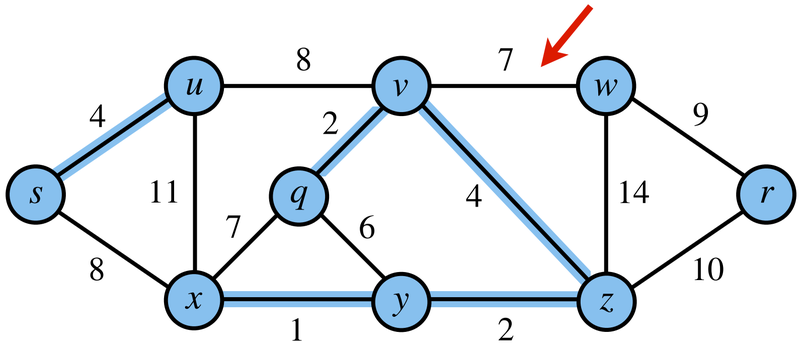




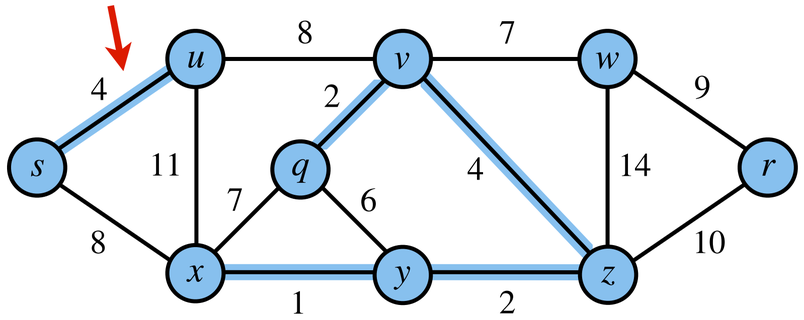
Kruskal's Demonstration

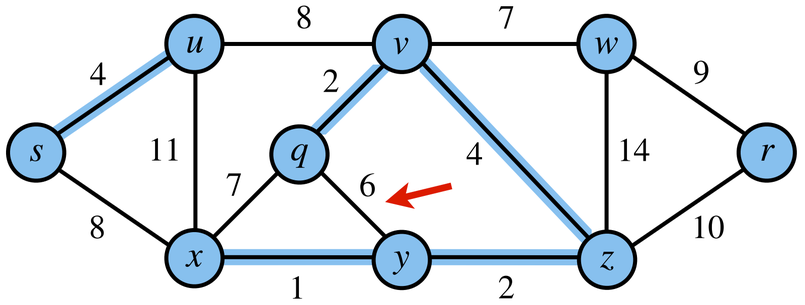


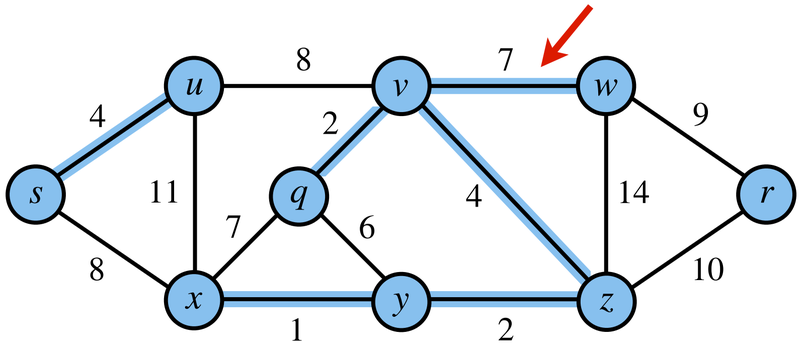




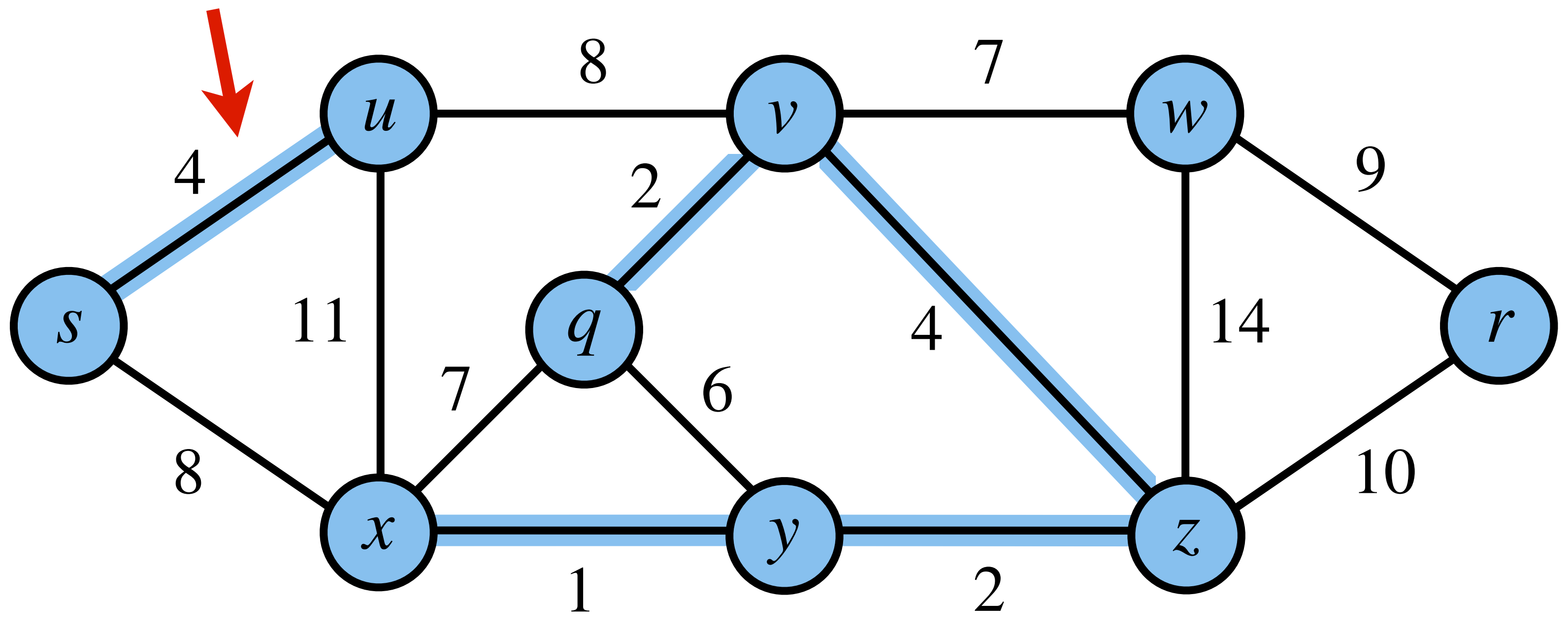
Kruskal's Demonstration

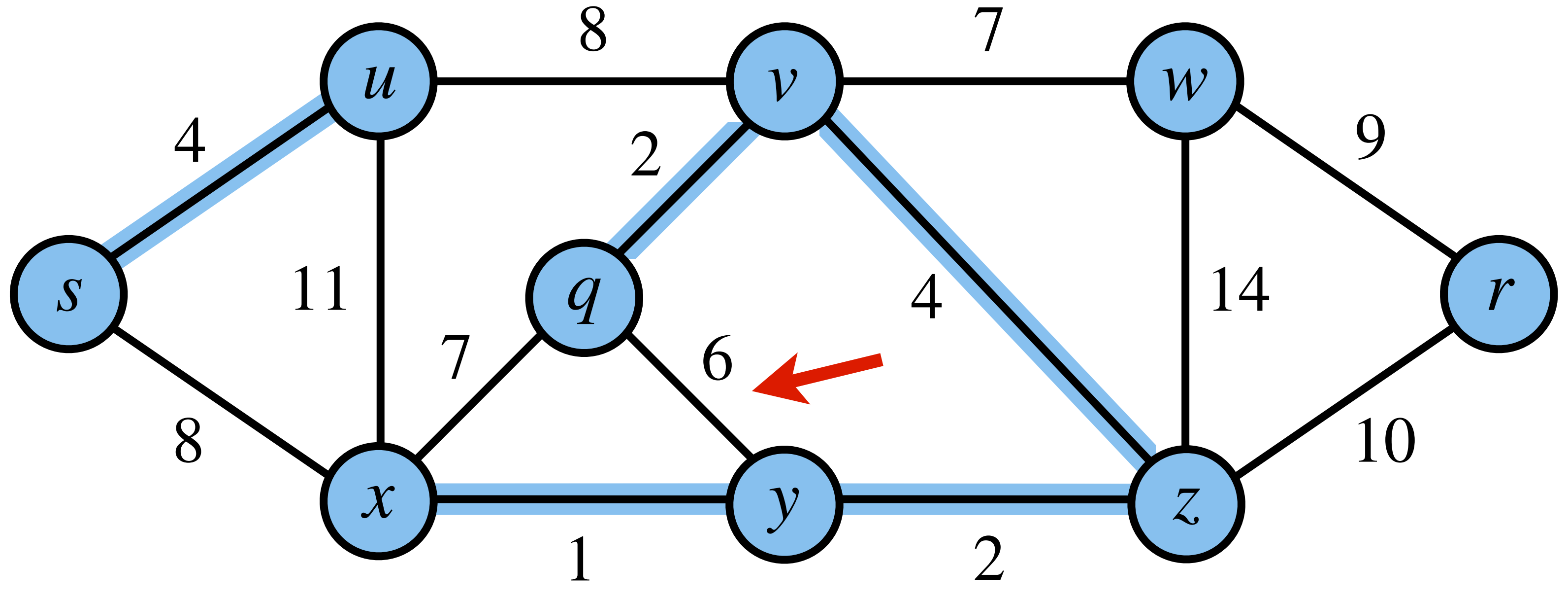


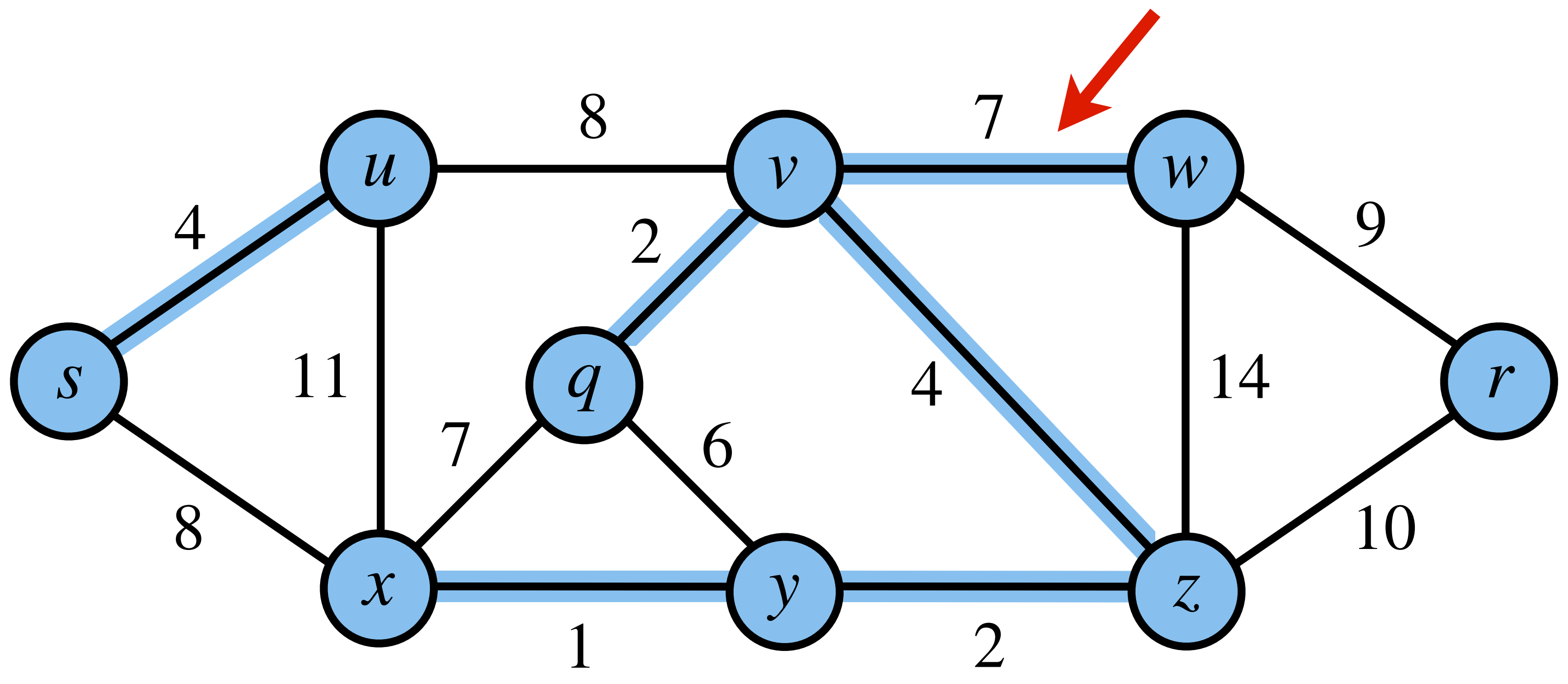


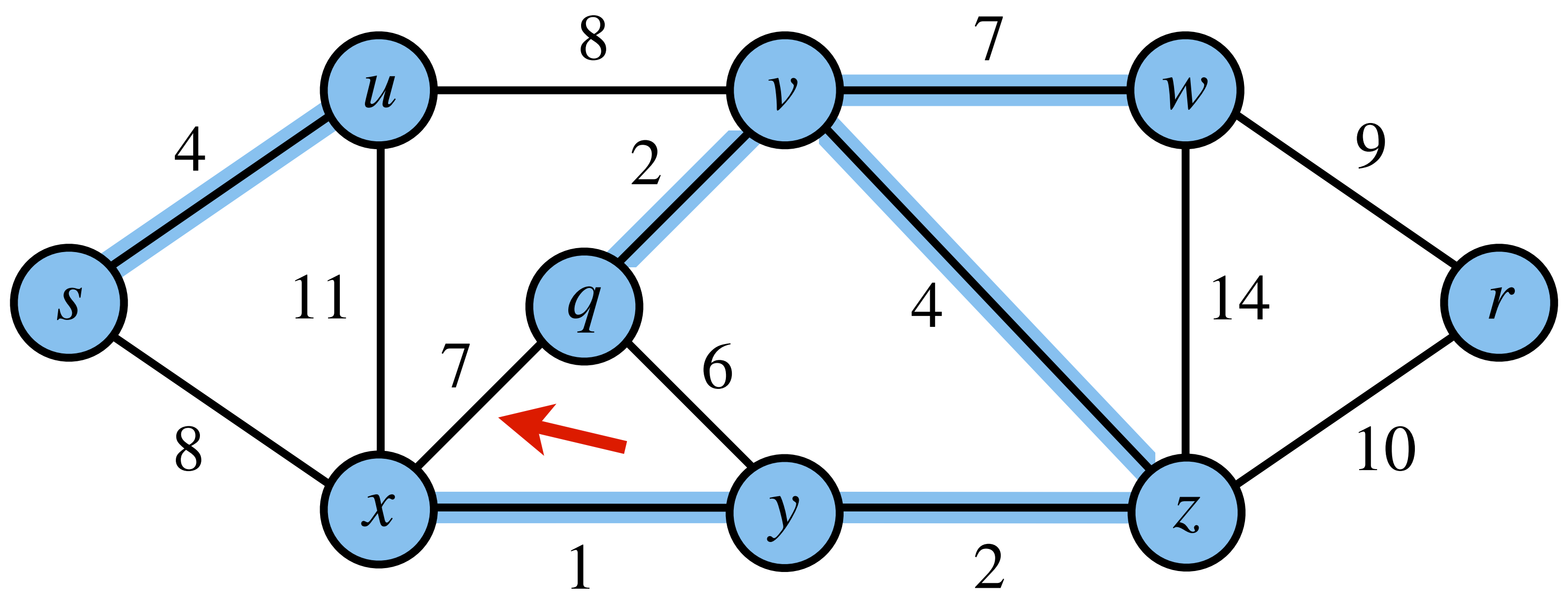


Kruskal's Demonstration

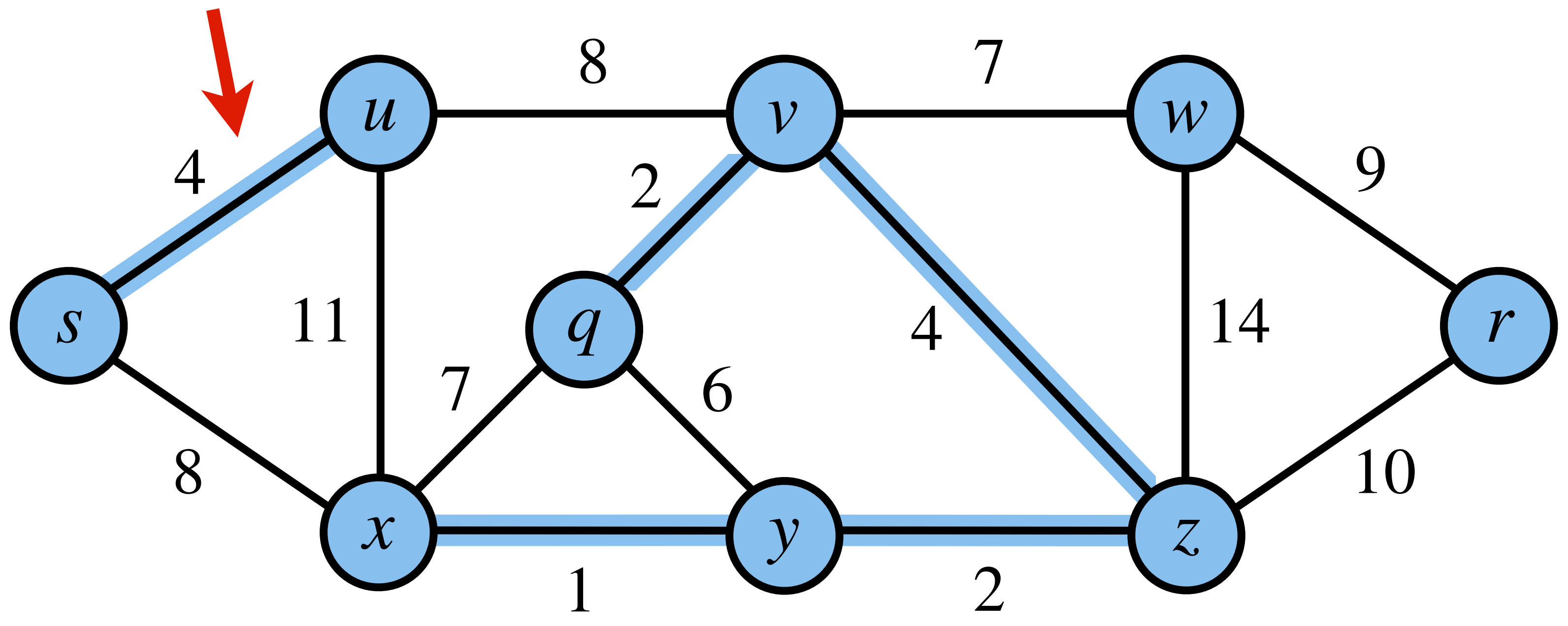


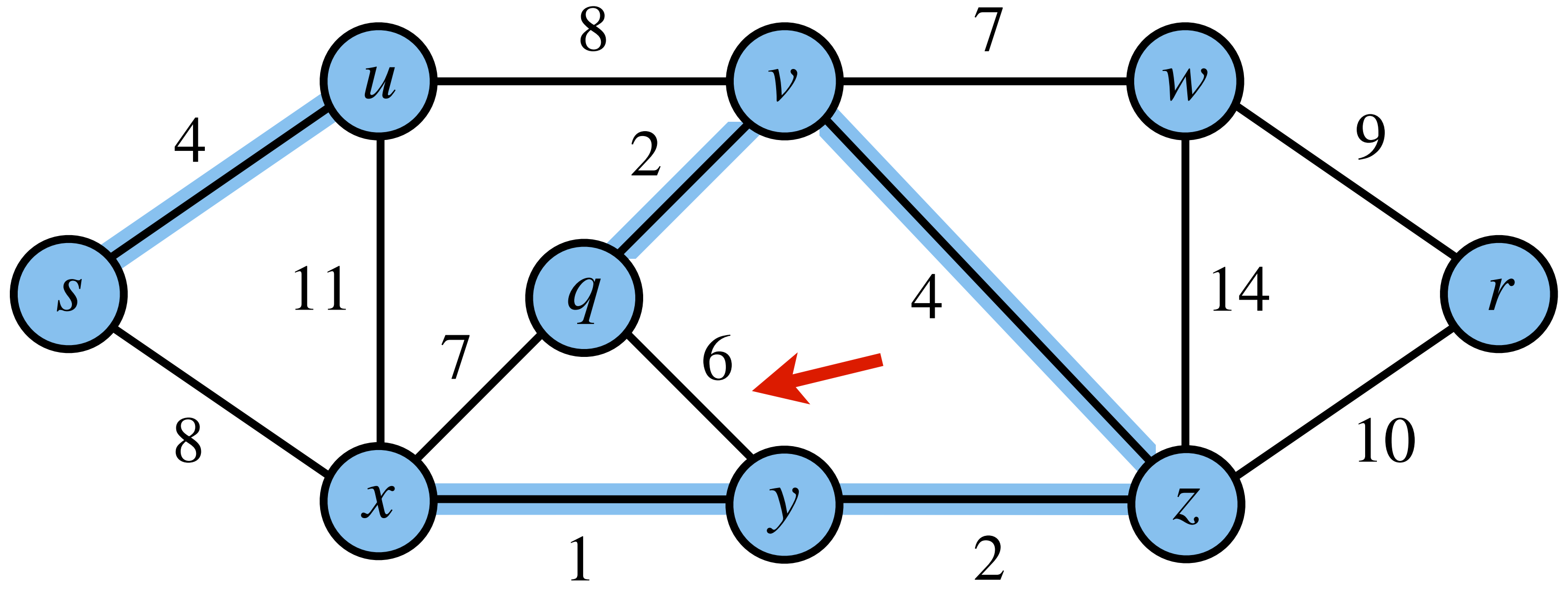


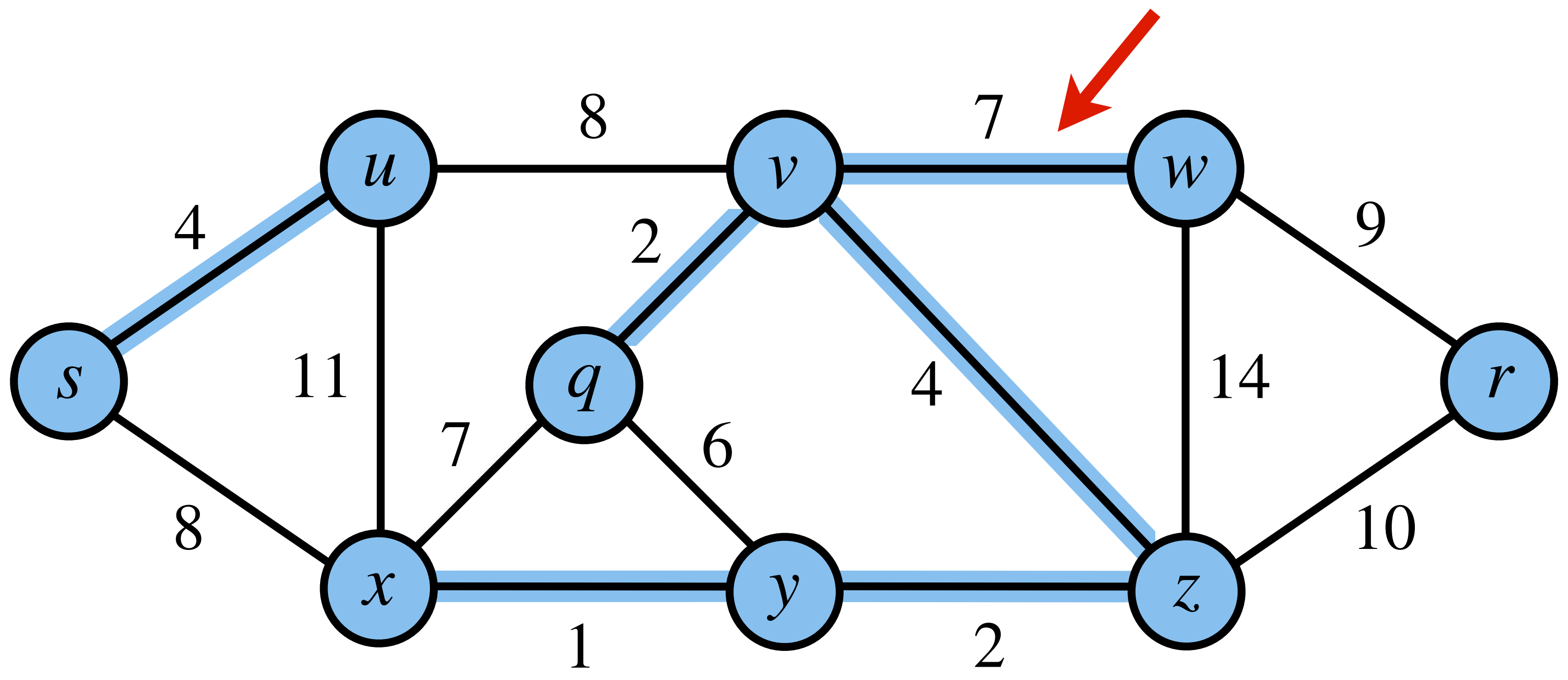


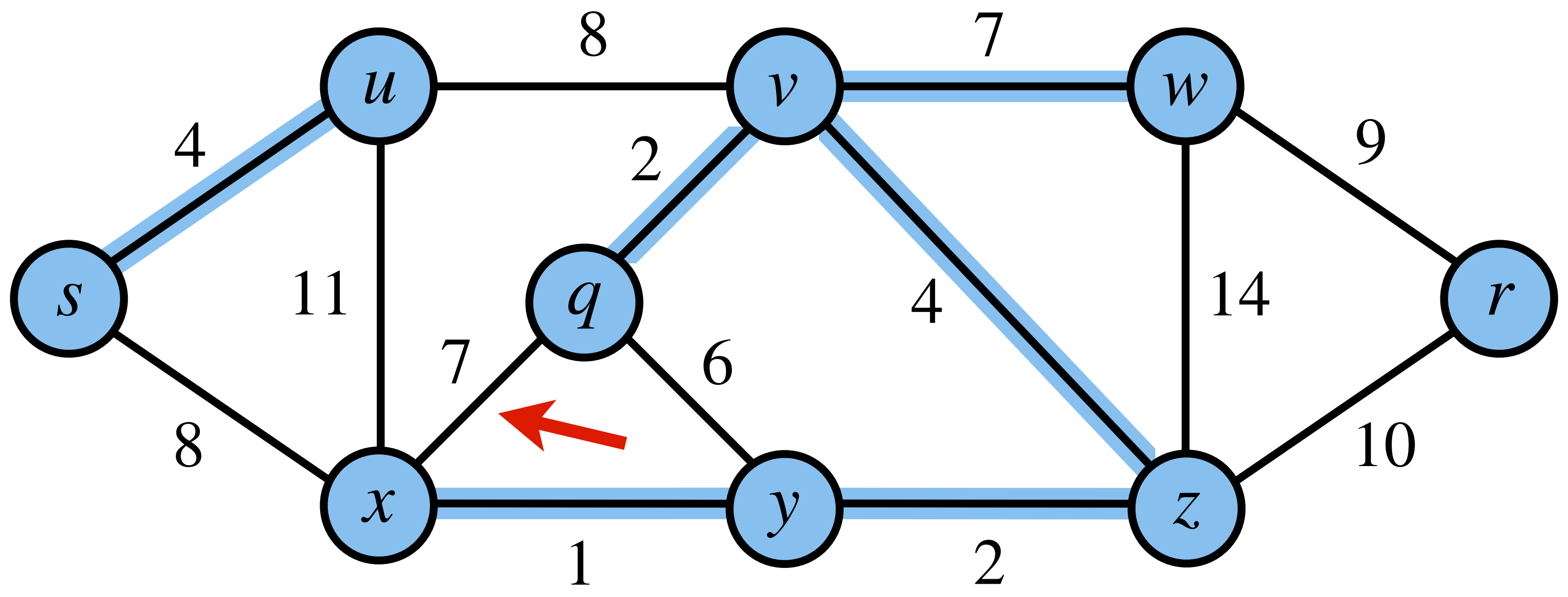


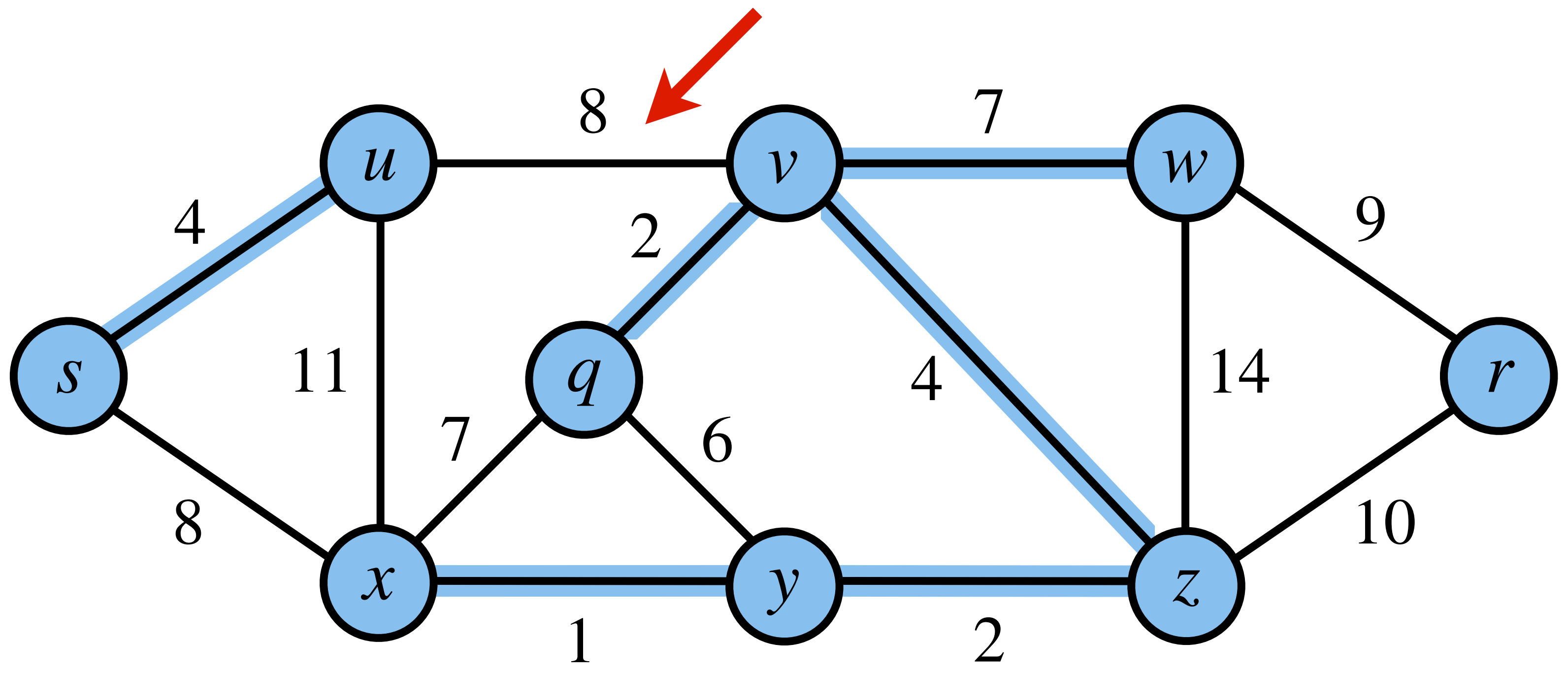
Kruskal's Demonstration



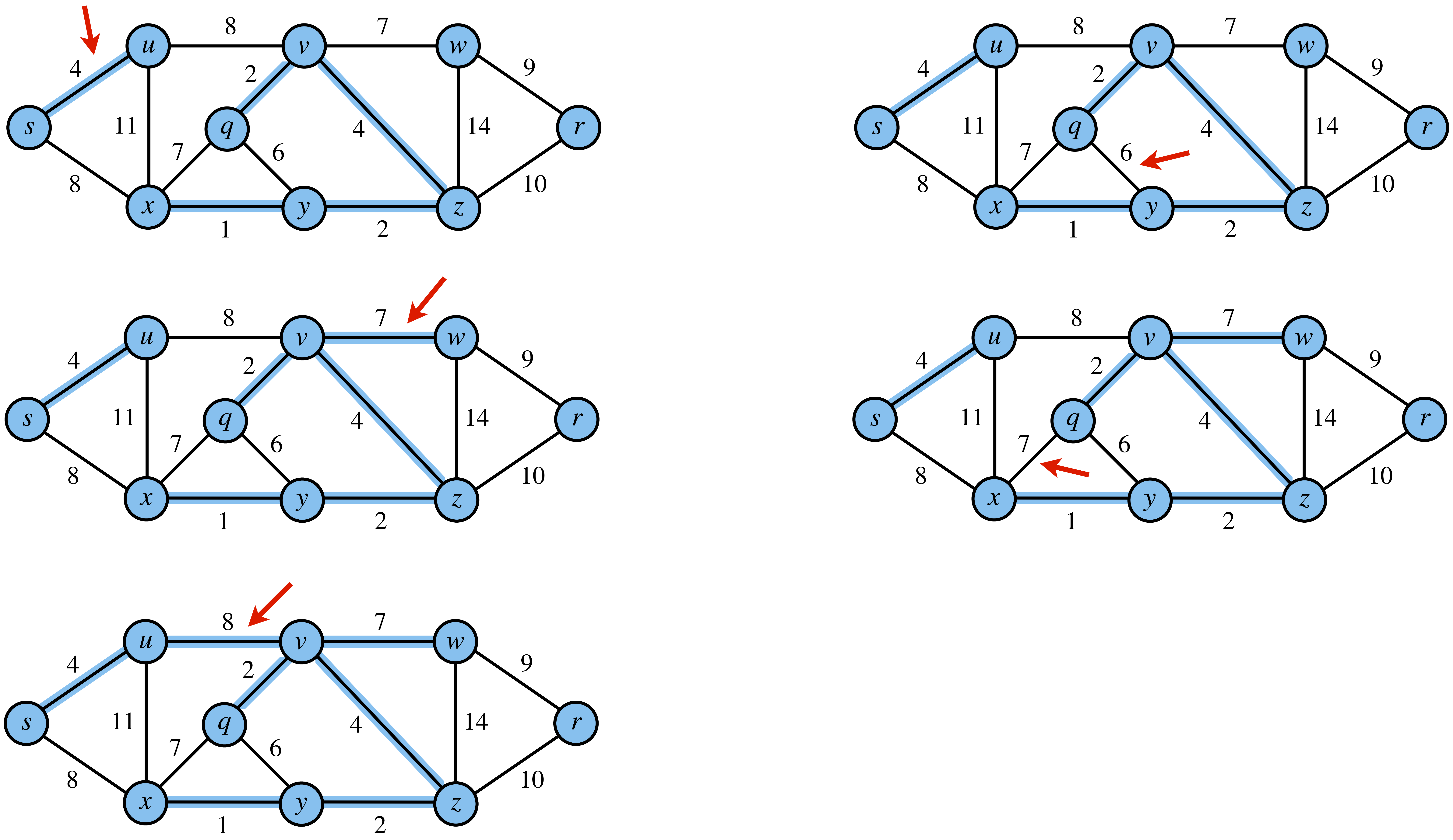




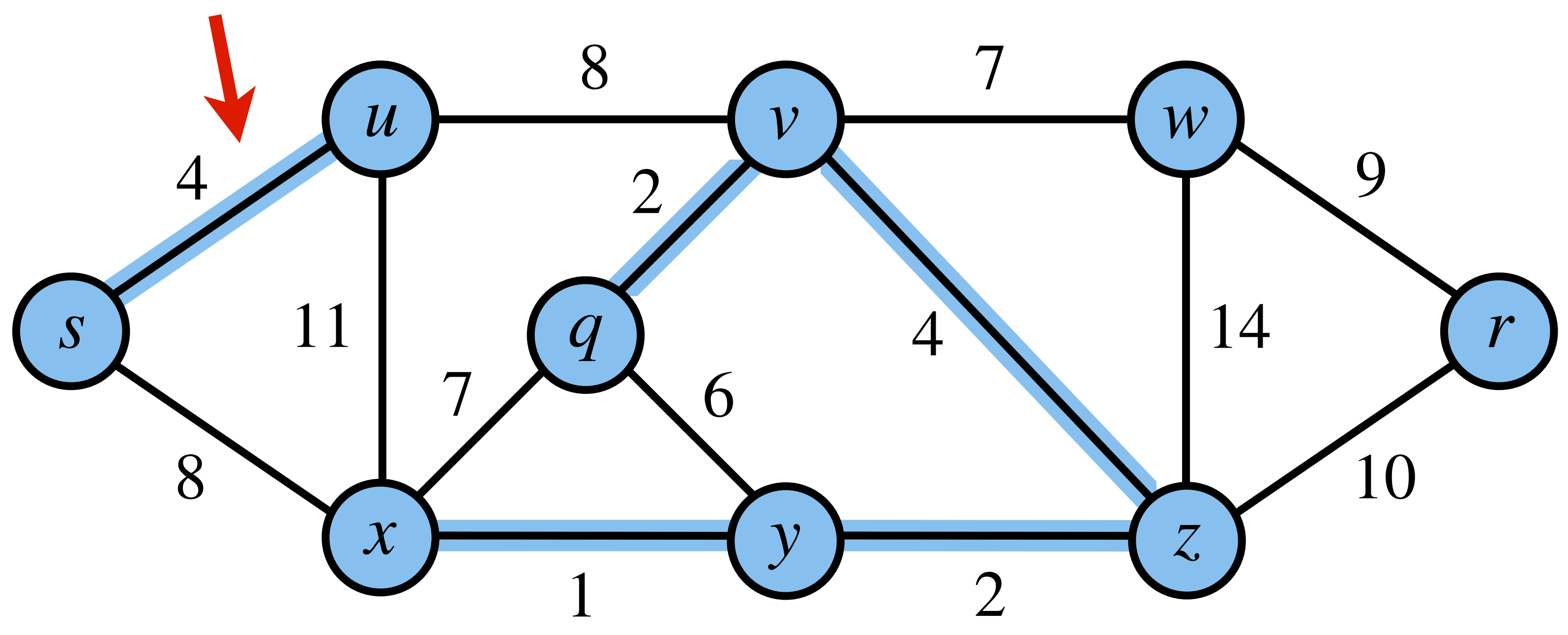


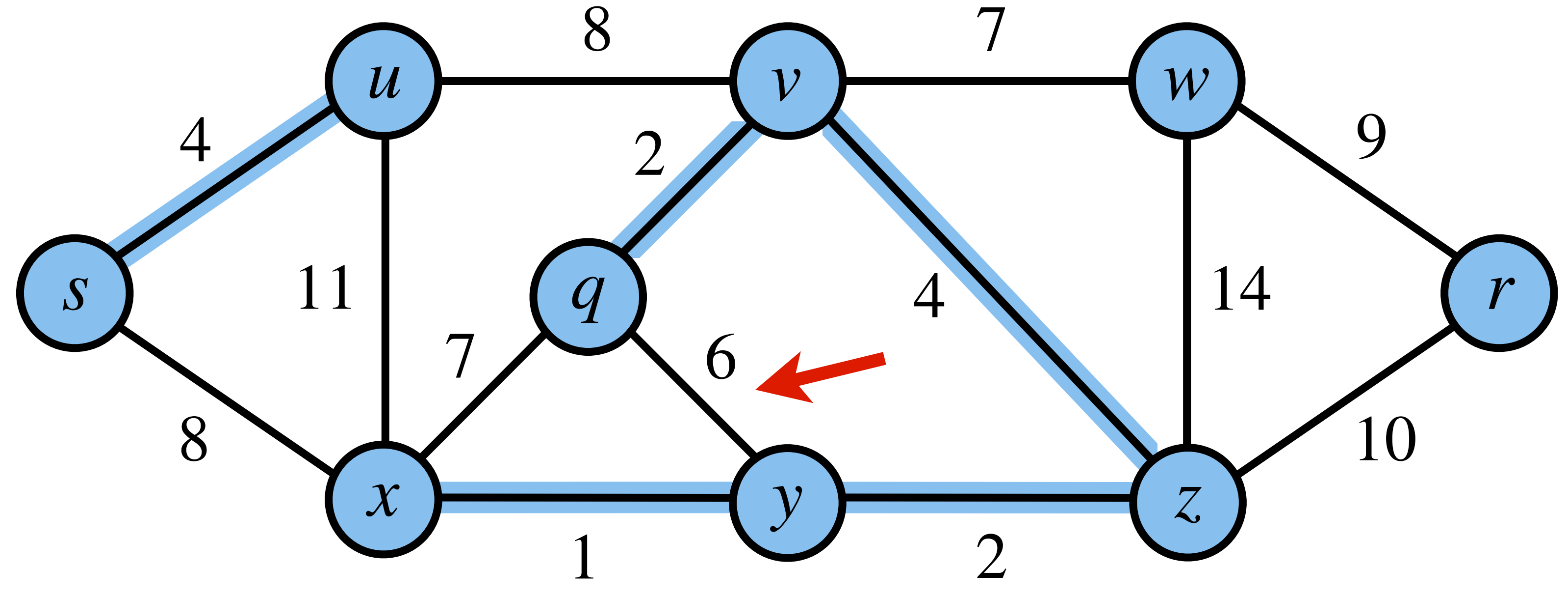


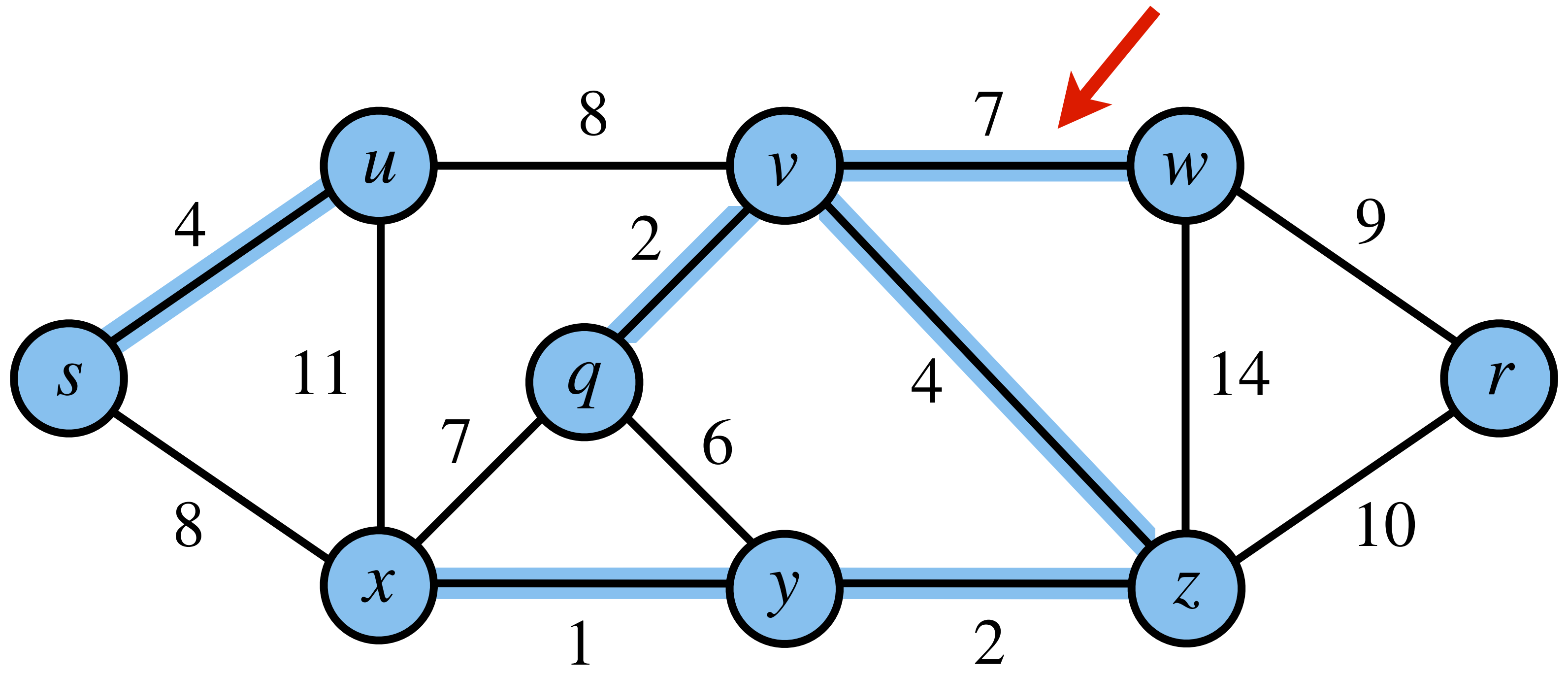
Kruskal's Demonstration

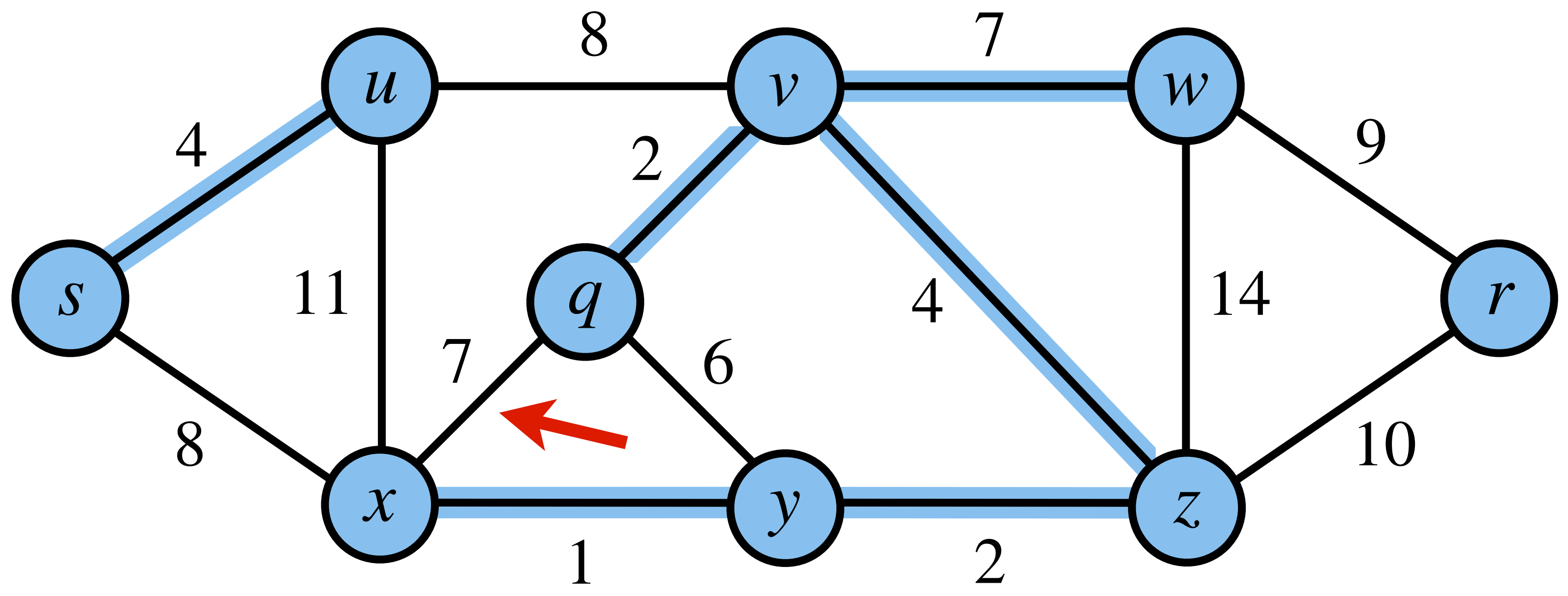


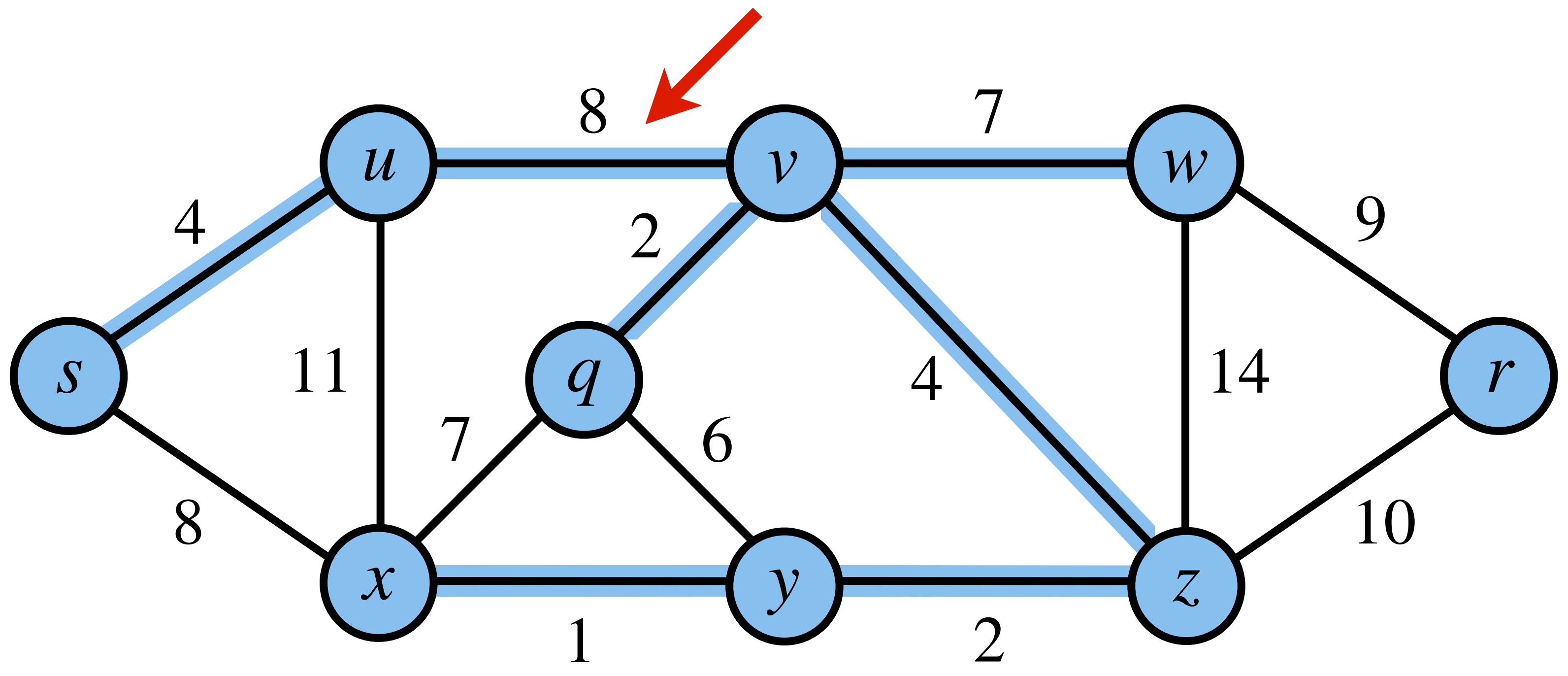
Kruskal's Demonstration

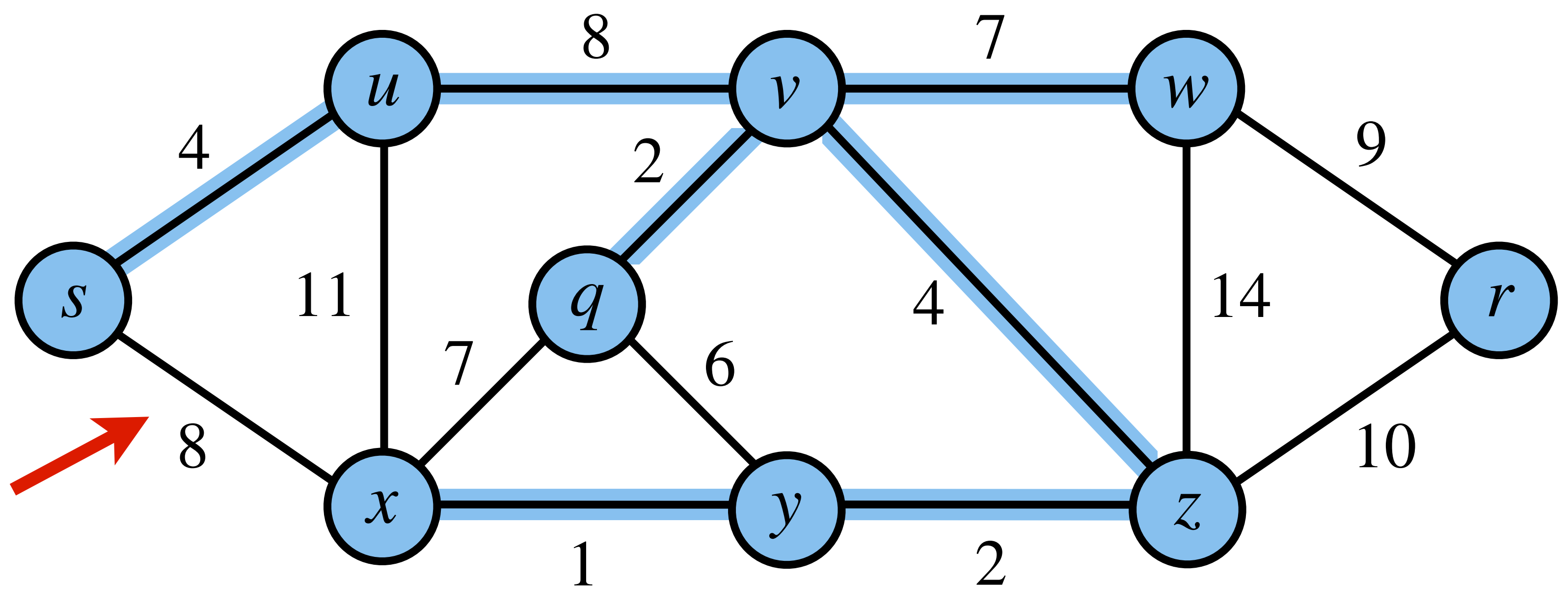




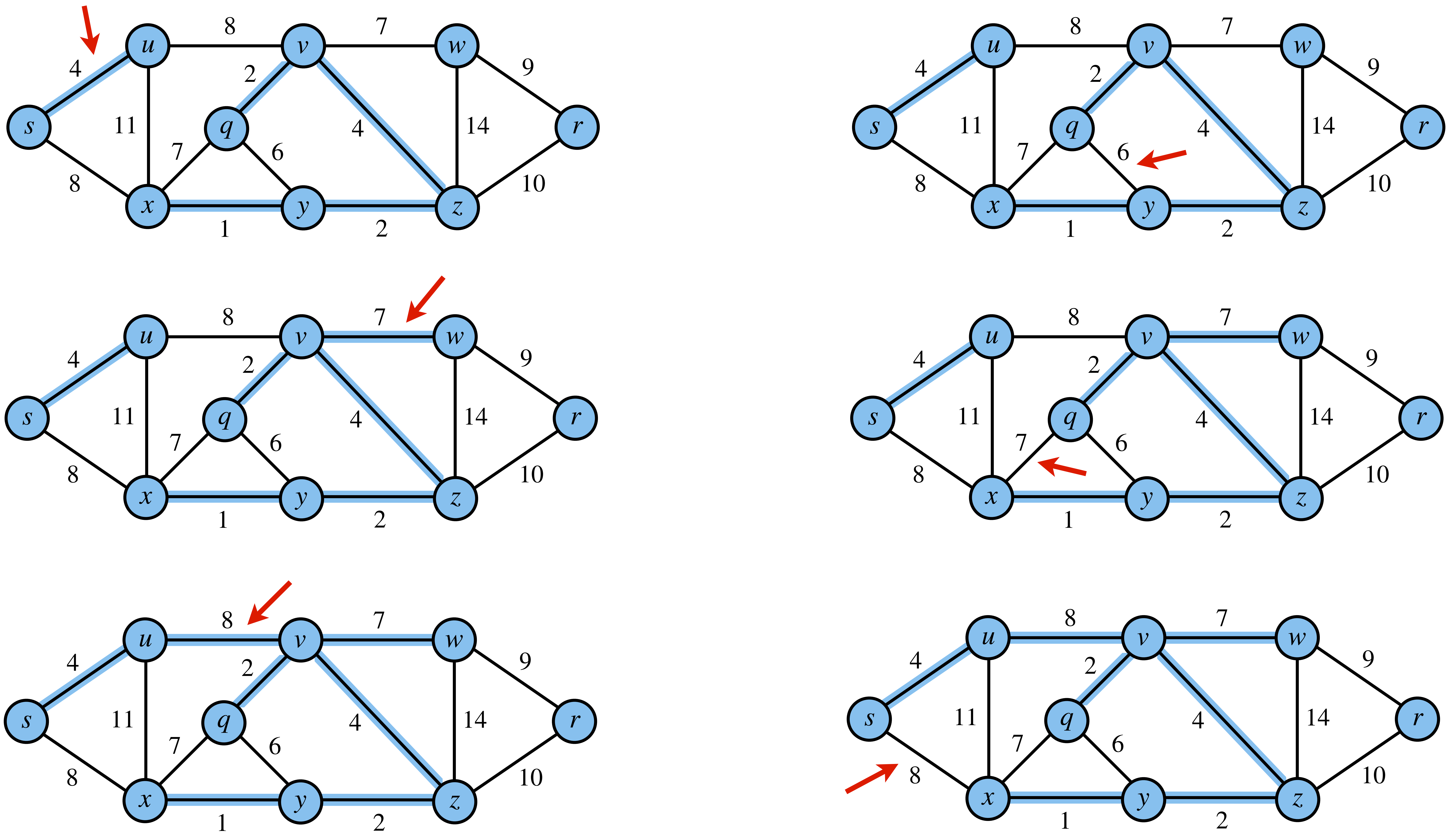








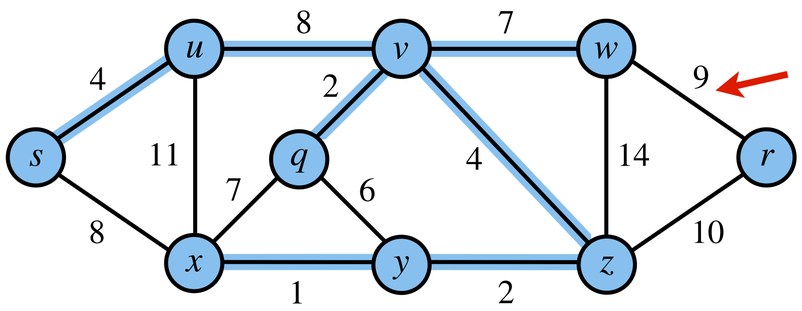
Kruskal’s Demonstration



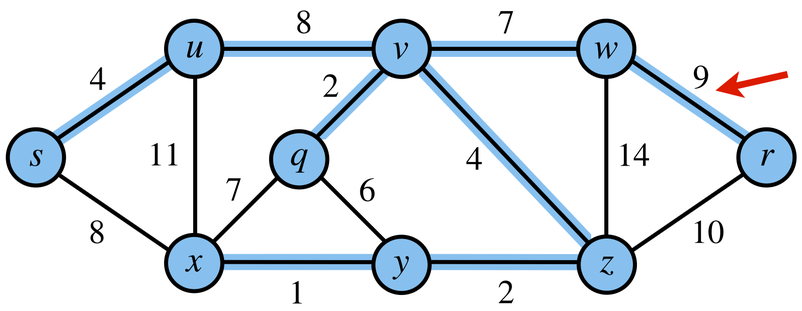
...

Kruskal’s Demonstration

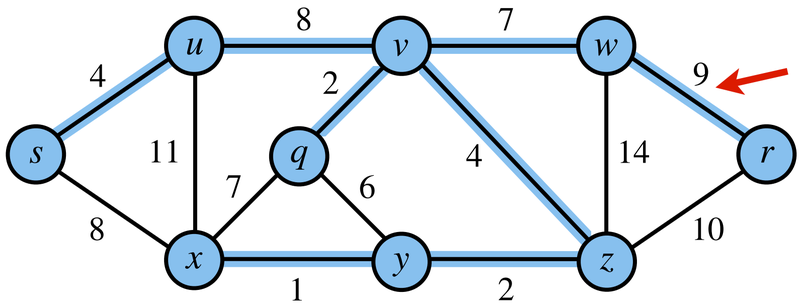
Kruskal's Demonstration

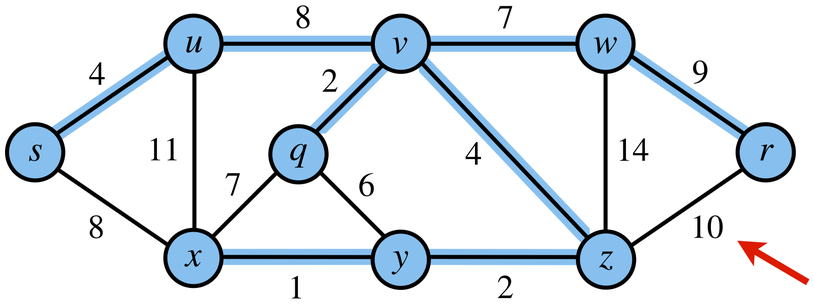


Kruskal's Demonstration

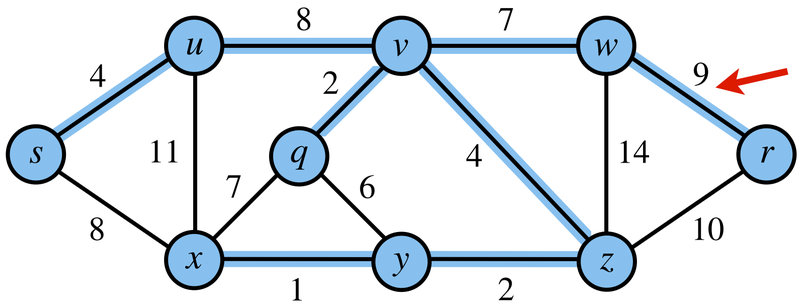


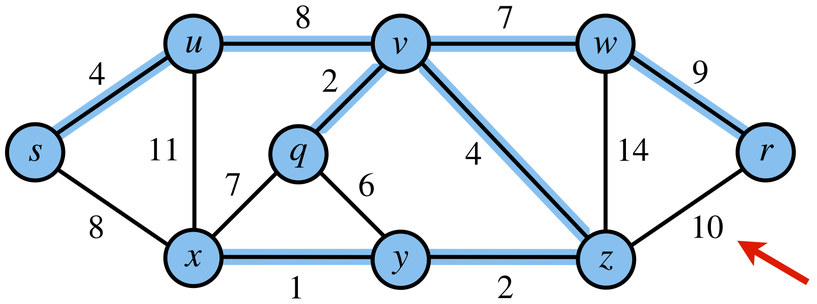
Kruskal's Demonstration

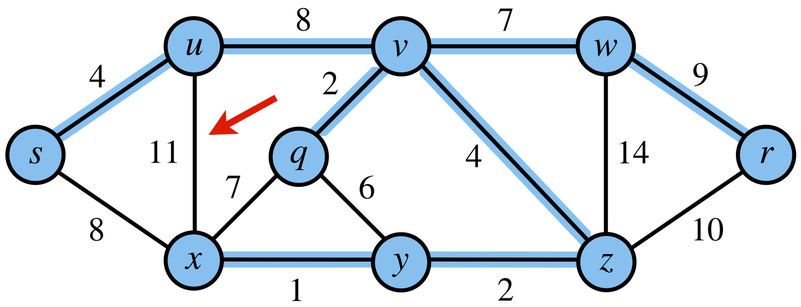




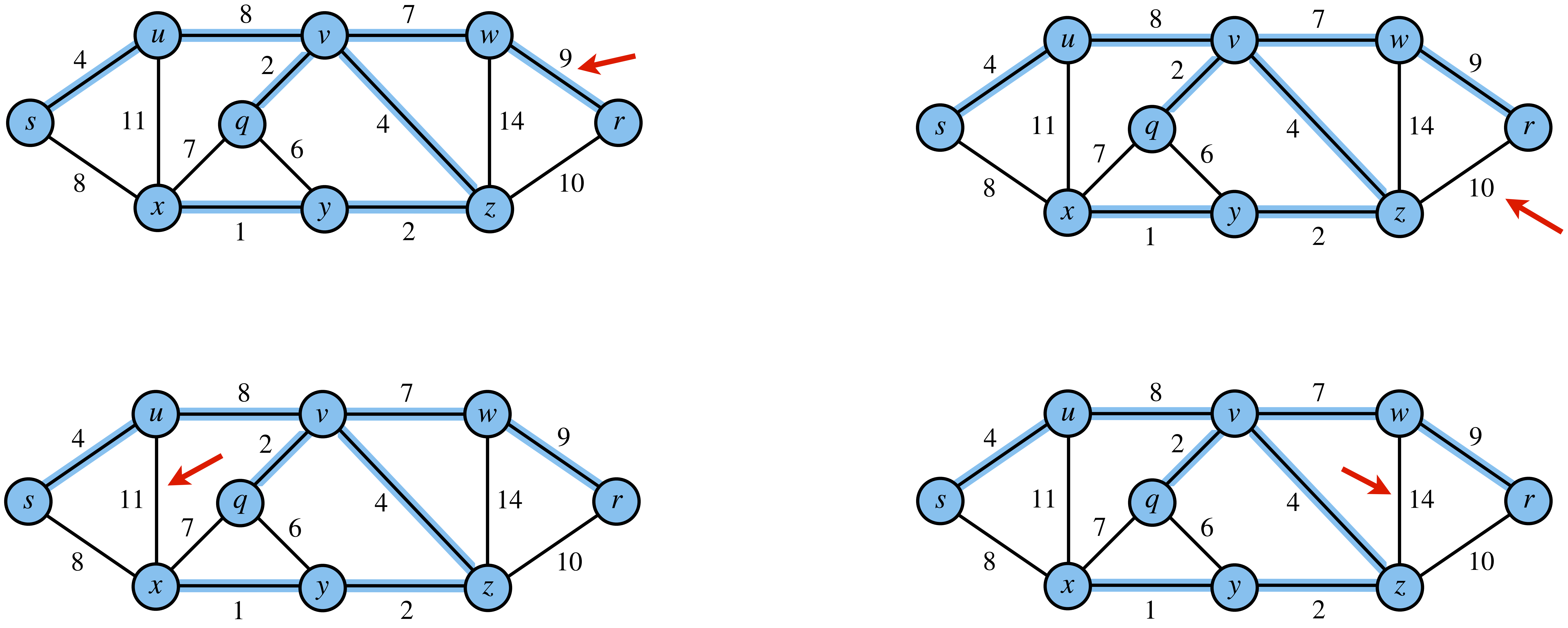
Kruskal's Demonstration







Kruskal’s Demonstration



Kruskal's Algorithm: Idea

Kruskal's Algorithm: Idea

Kruskal's algorithm works in the following manner.

Kruskal's Algorithm: Idea

Kruskal's algorithm works in the following manner.

- Start with $|V|$ many subtrees. Each vertex forms one subtree.

Kruskal's Algorithm: Idea

Kruskal's algorithm works in the following manner.

- Start with $|V|$ many subtrees. Each vertex forms one subtree.
- Start with edge set, $A = \emptyset$.

Kruskal's Algorithm: Idea

Kruskal's algorithm works in the following manner.

- Start with $|V|$ many subtrees. Each vertex forms one subtree.
- Start with edge set, $A = \emptyset$.
- Go over edges in ascending order of their weights.

Kruskal's Algorithm: Idea

Kruskal's algorithm works in the following manner.

- Start with $|V|$ many subtrees. Each vertex forms one subtree.
- Start with edge set, $A = \emptyset$.
- Go over edges in ascending order of their weights.
  - For an edge $\{u, v\}$, if u and v belong to different subtrees:

Kruskal's Algorithm: Idea

Kruskal's algorithm works in the following manner.

- Start with $|V|$ many subtrees. Each vertex forms one subtree.
- Start with edge set, $A = \emptyset$.
- Go over edges in ascending order of their weights.
  - For an edge $\{u, v\}$, if u and v belong to different subtrees:
    - Add $\{u, v\}$ in A.

Kruskal's Algorithm: Idea

Kruskal's algorithm works in the following manner.

- Start with $|V|$ many subtrees. Each vertex forms one subtree.
- Start with edge set, $A = \emptyset$.
- Go over edges in ascending order of their weights.
  - For an edge $\{u, v\}$, if u and v belong to different subtrees:
    - Add $\{u, v\}$ in A.
    - Join the different subtrees containing u and v via $\{u, v\}$.

Kruskal's Algorithm

Kruskal's Algorithm

Kruskal(G):

Kruskal's Algorithm

Kruskal(G)**:**

1. **for** each vertex $v \in V(G)$ *// Starting with $|V|$ many subtree*

Kruskal's Algorithm

Kruskal(G)**:**

1. **for** each vertex $v \in V(G)$ *// Starting with $|V|$ many subtree*
2. **Make-Set**(v)

Kruskal's Algorithm

Kruskal(G)**:**

1. **for** each vertex $v \in V(G)$ *// Starting with* $|V|$ *many subtree*
2. **Make-Set**(v)
3. $A = \emptyset$

Kruskal's Algorithm

Kruskal(G)**:**

1. **for** each vertex $v \in V(G)$ *// Starting with* $|V|$ *many subtree*
2. **Make-Set**(v)
3. $A = \emptyset$
4. sort the list of edges into monotonically increasing order by their weight

Kruskal's Algorithm

Kruskal(G)**:**

1. **for** each vertex $v \in V(G)$ *// Starting with $|V|$ many subtree*
2. **Make-Set**(v)
3. $A = \emptyset$
4. sort the list of edges into monotonically increasing order by their weight
5. **for** each edge $\{u, v\}$ taken from the sorted list in order

Kruskal's Algorithm

Kruskal(G)**:**

1. **for** each vertex $v \in V(G)$ *// Starting with $|V|$ many subtree*
2. **Make-Set**(v)
3. $A = \emptyset$
4. sort the list of edges into monotonically increasing order by their weight
5. **for** each edge $\{u, v\}$ taken from the sorted list in order
6. **if Find-Set**$(u) \neq$ **Find-Set**(v) *// u and v are in different trees*

Kruskal's Algorithm

Kruskal(G)**:**

1. **for** each vertex $v \in V(G)$ *// Starting with $|V|$ many subtree*
2. **Make-Set**(v)
3. $A = \emptyset$
4. sort the list of edges into monotonically increasing order by their weight
5. **for** each edge $\{u, v\}$ taken from the sorted list in order
6. **if Find-Set**$(u) \neq$ **Find-Set**(v) *// u and v are in different trees*
7. $A = A \cup \{\{u, v\}\}$

Kruskal's Algorithm

Kruskal(G)**:**

1. **for** each vertex $v \in V(G)$ *// Starting with $|V|$ many subtree*
2. **Make-Set**(v)
3. $A = \emptyset$
4. sort the list of edges into monotonically increasing order by their weight
5. **for** each edge $\{u, v\}$ taken from the sorted list in order
6. **if Find-Set**$(u) \neq$ **Find-Set**(v) *// u and v are in different trees*
7. $A = A \cup \{\{u, v\}\}$
8. **Union**(u, v)

Kruskal's Algorithm

Kruskal(G)**:**

1. **for** each vertex $v \in V(G)$ *// Starting with $|V|$ many subtree*
2. **Make-Set**(v)
3. $A = \emptyset$
4. sort the list of edges into monotonically increasing order by their weight
5. **for** each edge $\{u, v\}$ taken from the sorted list in order
6. **if Find-Set**$(u) \neq$ **Find-Set**(v) *// u and v are in different trees*
7. $A = A \cup \{\{u, v\}\}$
8. **Union**(u, v)
9. **return** A

Kruskal's Algorithm

Kruskal(G)**:**

$\Theta(|V|)$

1. **for** each vertex $v \in V(G)$ *// Starting with $|V|$ many subtree*
2. **Make-Set**(v)
3. $A = \emptyset$
4. sort the list of edges into monotonically increasing order by their weight
5. **for** each edge $\{u, v\}$ taken from the sorted list in order
6. **if Find-Set**$(u) \neq$ **Find-Set**(v) *// u and v are in different trees*
7. $A = A \cup \{\{u, v\}\}$
8. **Union**(u, v)
9. **return** A

Kruskal's Algorithm

Kruskal(G)**:**

1. **for** each vertex $v \in V(G)$ $\Theta(|V|)$ *// Starting with $|V|$ many subtree*
2. **Make-Set**(v)
3. $A = \emptyset$
4. sort the list of edges into monotonically increasing order by their weight $\Theta(|E|\log|E|)$
5. **for** each edge $\{u, v\}$ taken from the sorted list in order
6. **if Find-Set**$(u) \neq$ **Find-Set**(v) *// u and v are in different trees*
7. $A = A \cup \{\{u, v\}\}$
8. **Union**(u, v)
9. **return** A

Kruskal's Algorithm

Kruskal(G)**:**

1. **for** each vertex $v \in V(G)$ — $\Theta(|V|)$ — *// Starting with $|V|$ many subtree*
2. **Make-Set**(v)
3. $A = \emptyset$
4. sort the list of edges into monotonically increasing order by their weight — $\Theta(|E| \log |E|)$
5. **for** each edge $\{u, v\}$ taken from the sorted list in order
6. **if Find-Set**$(u) \neq$ **Find-Set**(v) — *// u and v are in different trees*
7. $A = A \cup \{\{u, v\}\}$
8. **Union**(u, v)
9. **return** A

$O(|E| \log |V|)$

Kruskal's Algorithm

Kruskal(G)**:**

1. **for** each vertex $v \in V(G)$ — $\Theta(|V|)$ — *// Starting with $|V|$ many subtree*
2. **Make-Set**(v)
3. $A = \emptyset$
4. sort the list of edges into monotonically increasing order by their weight — $\Theta(|E| \log |E|)$
5. **for** each edge $\{u, v\}$ taken from the sorted list in order
6. **if Find-Set**$(u) \neq$ **Find-Set**(v) *// u and v are in different trees*
7. $A = A \cup \{\{u, v\}\}$
8. **Union**(u, v)
9. **return** A

$O(|E| \log |V|)$

(Disjoint-Set with rank heuristic)

Kruskal's Algorithm

Kruskal(G)**:**

1. **for** each vertex $v \in V(G)$ — $\Theta(|V|)$ — *// Starting with $|V|$ many subtree*
2. **Make-Set**(v)
3. $A = \emptyset$
4. sort the list of edges into monotonically increasing order by their weight — $\Theta(|E| \log |E|)$
5. **for** each edge $\{u, v\}$ taken from the sorted list in order
6. **if Find-Set**$(u) \neq$ **Find-Set**(v) *// u and v are in different trees*
7. $A = A \cup \{\{u, v\}\}$
8. **Union**(u, v)
9. **return** A

$O(|E| \log |V|)$

(Disjoint-Set with rank heuristic)

Time Complexity:

Kruskal's Algorithm

Kruskal(G)**:**

1. **for** each vertex $v \in V(G)$ — $\Theta(|V|)$ — *// Starting with $|V|$ many subtree*
2. **Make-Set**(v)
3. $A = \emptyset$
4. sort the list of edges into monotonically increasing order by their weight — $\Theta(|E| \log |E|)$
5. **for** each edge $\{u, v\}$ taken from the sorted list in order
6. **if Find-Set**$(u) \neq$ **Find-Set**(v) *// u and v are in different trees*
7. $A = A \cup \{\{u, v\}\}$
8. **Union**(u, v)
9. **return** A

$O(|E| \log |V|)$

(Disjoint-Set with rank heuristic)

Time Complexity: $O(|E| \log |E|)$

Kruskal's Algorithm

Kruskal(G)**:**

1. **for** each vertex $v \in V(G)$ — $\Theta(|V|)$ — *// Starting with $|V|$ many subtree*
2. **Make-Set**(v)
3. $A = \emptyset$
4. sort the list of edges into monotonically increasing order by their weight — $\Theta(|E| \log |E|)$
5. **for** each edge $\{u, v\}$ taken from the sorted list in order
6. **if Find-Set**$(u) \neq$ **Find-Set**(v) *// u and v are in different trees*
7. $A = A \cup \{\{u, v\}\}$
8. **Union**(u, v)
9. **return** A

$O(|E| \log |V|)$ (Disjoint-Set with rank heuristic)

Time Complexity: $O(|E| \log |E|) = O(|E| \log |V|)$

Kruskal's Algorithm

Kruskal(G)**:**

1. **for** each vertex $v \in V(G)$ — $\Theta(|V|)$ — *// Starting with $|V|$ many subtree*
2. **Make-Set**(v)
3. $A = \emptyset$
4. sort the list of edges into monotonically increasing order by their weight — $\Theta(|E| \log |E|)$
5. **for** each edge $\{u, v\}$ taken from the sorted list in order
6. **if Find-Set**$(u) \neq$ **Find-Set**(v) *// u and v are in different trees*
7. $A = A \cup \{\{u, v\}\}$
8. **Union**(u, v)
9. **return** A

$O(|E| \log |V|)$ (Disjoint-Set with rank heuristic)

Time Complexity: $O(|E| \log |E|) = O(|E| \log |V|)$ $(\because |E| \leq |V|^2)$

Kruskal's Algorithm: Correctness (Spanning)

Kruskal's Algorithm: Correctness (Spanning)

The algorithm starts with $|V|$ many subtrees.

Kruskal's Algorithm: Correctness (Spanning)

The algorithm starts with $|V|$ many subtrees.

Why should the algorithm terminate with just one tree?

Kruskal's Algorithm: Correctness (Spanning)

The algorithm starts with $|V|$ many subtrees.

Why should the algorithm terminate with just one tree?

Suppose algorithm terminates with $T_1, T_2, \ldots, T_k$ disconnected subtrees, where $k \geq 2$.

Kruskal's Algorithm: Correctness (Spanning)

The algorithm starts with $|V|$ many subtrees.

Why should the algorithm terminate with just one tree?

Suppose algorithm terminates with $T_1, T_2, \ldots, T_k$ disconnected subtrees, where $k \geq 2$.

Consider the cut:

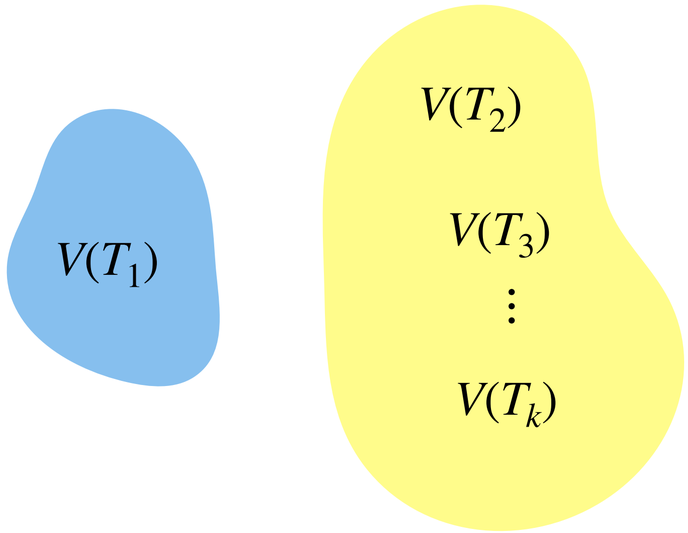
Kruskal's Algorithm: Correctness (Spanning)

The algorithm starts with $|V|$ many subtrees.

Why should the algorithm terminate with just one tree?

Suppose algorithm terminates with $T_1, T_2, \ldots, T_k$ disconnected subtrees, where $k \geq 2$.

Consider the cut:



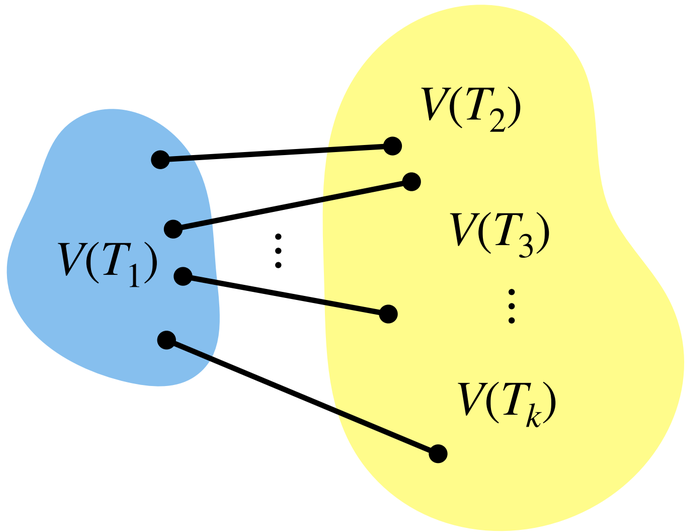
Kruskal's Algorithm: Correctness (Spanning)

The algorithm starts with $|V|$ many subtrees.

Why should the algorithm terminate with just one tree?

Suppose algorithm terminates with $T_1, T_2, \ldots, T_k$ disconnected subtrees, where $k \geq 2$.

Consider the cut:



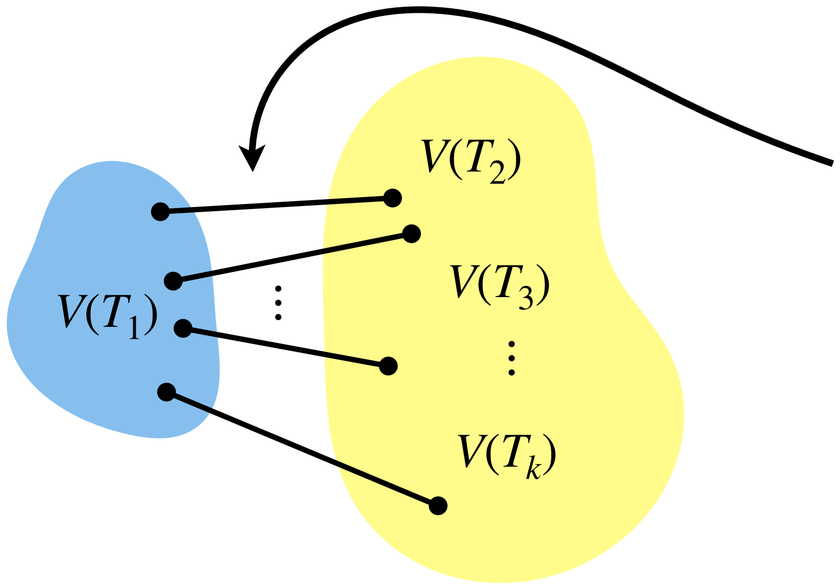
Kruskal's Algorithm: Correctness (Spanning)

The algorithm starts with $|V|$ many subtrees.

Why should the algorithm terminate with just one tree?

Suppose algorithm terminates with $T_1, T_2, \ldots, T_k$ disconnected subtrees, where $k \geq 2$.

Consider the cut:



The edges of the cut set must have been considered in some iteration to be included in A.

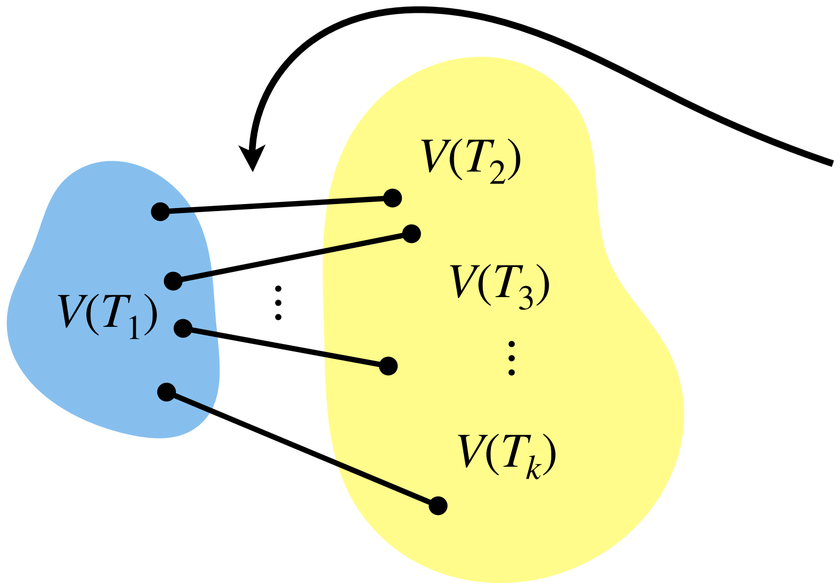
Kruskal's Algorithm: Correctness (Spanning)

The algorithm starts with $|V|$ many subtrees.

Why should the algorithm terminate with just one tree?

Suppose algorithm terminates with $T_1, T_2, \ldots, T_k$ disconnected subtrees, where $k \geq 2$.

Consider the cut:



The edges of the cut set must have been considered in some iteration to be included in A.

The first encountered edge, say $\{u, v\}$, of the above cut-set must have been added in A

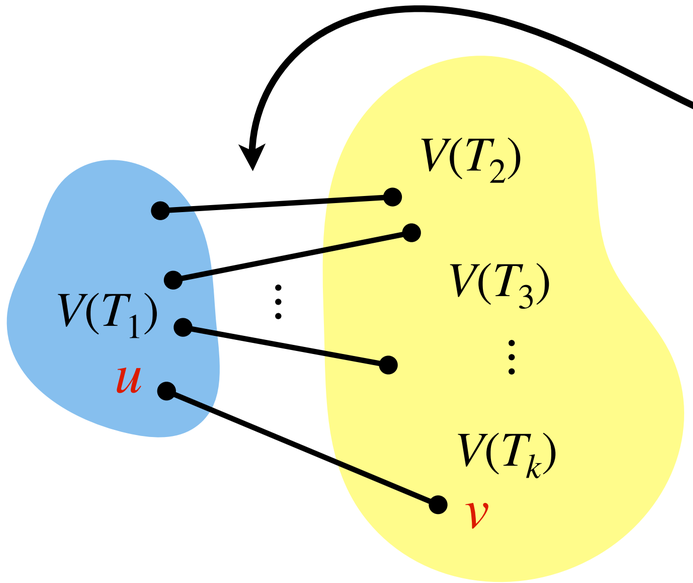
Kruskal's Algorithm: Correctness (Spanning)

The algorithm starts with $|V|$ many subtrees.

Why should the algorithm terminate with just one tree?

Suppose algorithm terminates with $T_1, T_2, \ldots, T_k$ disconnected subtrees, where $k \geq 2$.

Consider the cut:



The edges of the cut set must have been considered in some iteration to be included in A.

The first encountered edge, say $\{u, v\}$, of the above cut-set must have been added in A

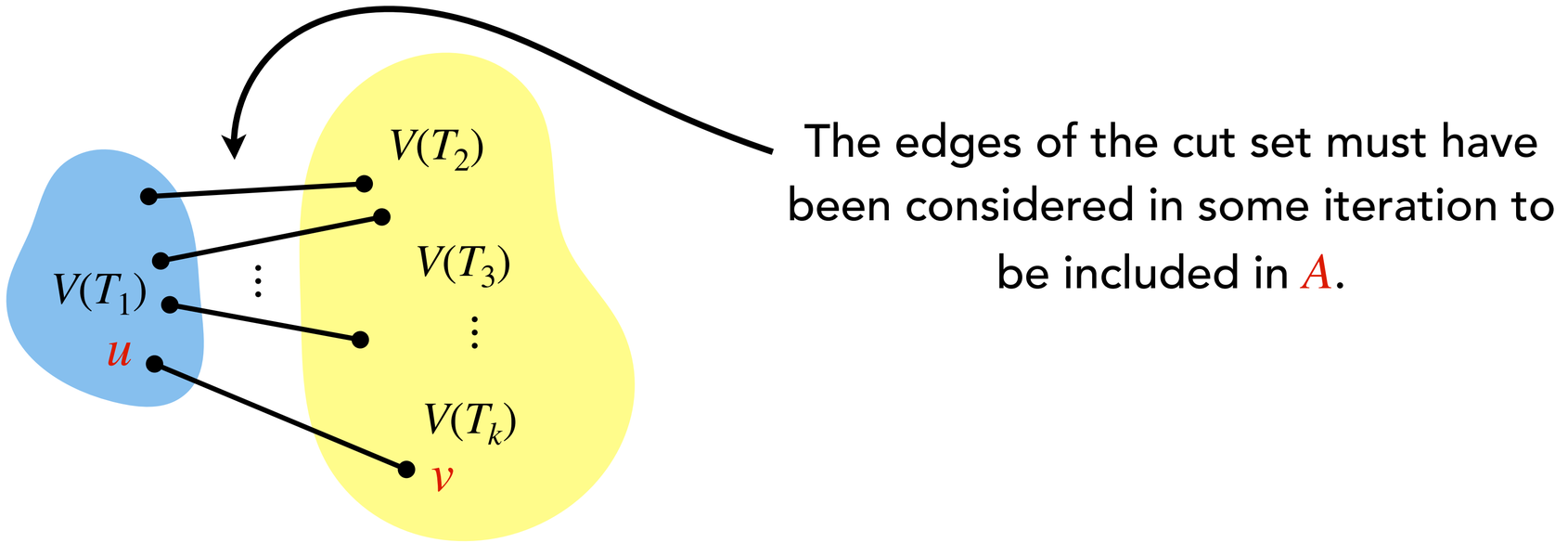
Kruskal's Algorithm: Correctness (Spanning)

The algorithm starts with $|V|$ many subtrees.

Why should the algorithm terminate with just one tree?

Suppose algorithm terminates with $T_1, T_2, \ldots, T_k$ disconnected subtrees, where $k \geq 2$.

Consider the cut:



The first encountered edge, say $\{u, v\}$, of the above cut-set must have been added in A putting u and v in the same subtree.

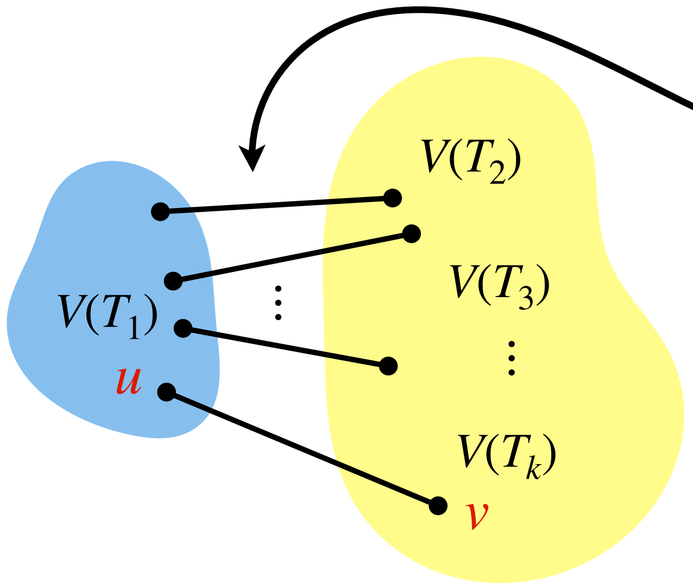
Kruskal's Algorithm: Correctness (Spanning)

The algorithm starts with $|V|$ many subtrees.

Why should the algorithm terminate with just one tree?

Suppose algorithm terminates with $T_1, T_2, \ldots, T_k$ disconnected subtrees, where $k \geq 2$.

Consider the cut:



The edges of the cut set must have been considered in some iteration to be included in A.

The first encountered edge, say $\{u, v\}$, of the above cut-set must have been added in A putting u and v in the same subtree. Contradiction!

Kruskal's Algorithm: Correctness (Minimum)

Kruskal's Algorithm: Correctness (Minimum)

Now, we will prove that any stage of the algorithm A is part of some MST.

Kruskal's Algorithm: Correctness (Minimum)

Now, we will prove that any stage of the algorithm A is part of some MST.

Lemma: Let $G(V, E, w)$ an undirected, weighted and connected graph. Let A be a subset of E that is included in some MST of G. Let $C = (S, T)$ be any cut in G so that cut set of C does not have any edge from A. If e is the least weight edge in the cut-set of C, then $A \cup \{e\}$ is also part of some MST of G.

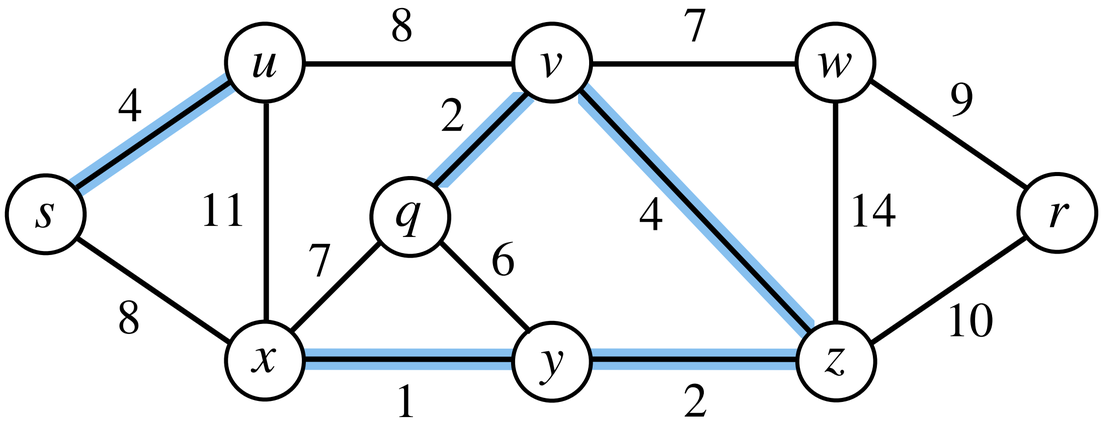
Recall this lemma.

Kruskal's Algorithm: Correctness (Minimum)

Now, we will prove that any stage of the algorithm A is part of some MST.

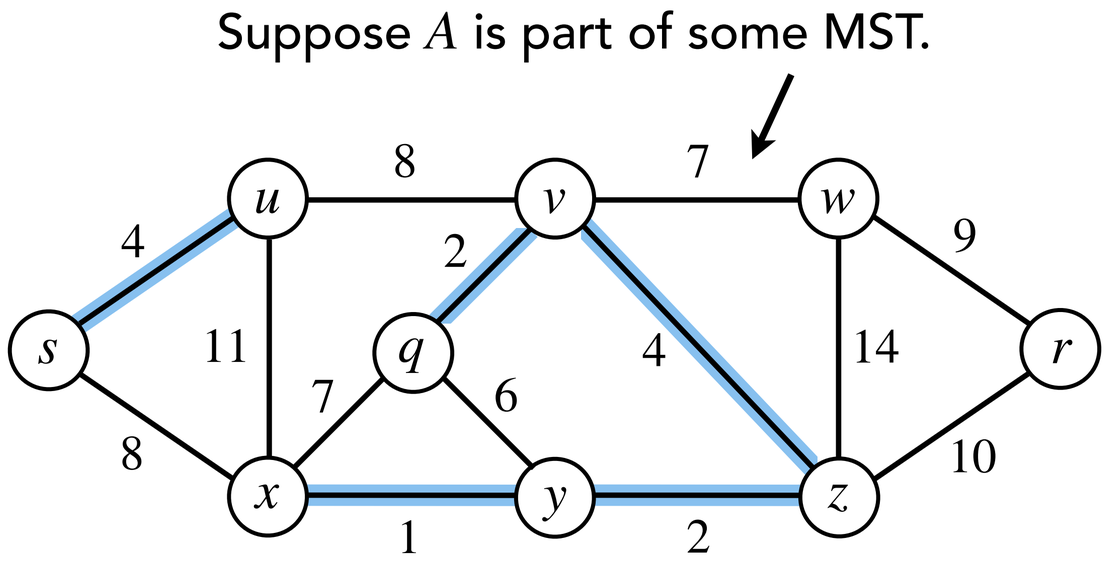
Kruskal's Algorithm: Correctness (Minimum)

Now, we will prove that any stage of the algorithm A is part of some MST.



Kruskal's Algorithm: Correctness (Minimum)

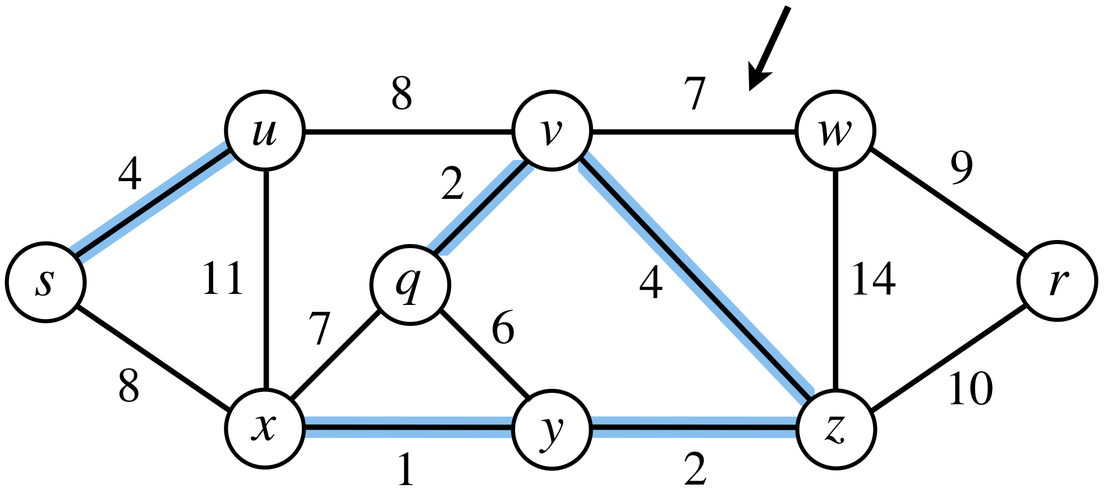
Now, we will prove that any stage of the algorithm A is part of some MST.



Kruskal's Algorithm: Correctness (Minimum)

Now, we will prove that any stage of the algorithm A is part of some MST.

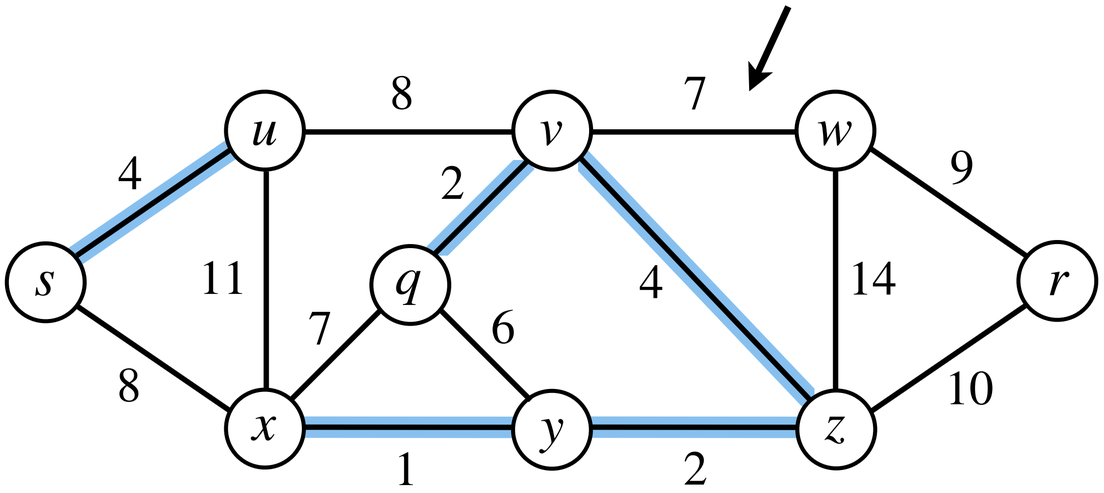
Suppose A is part of some MST. Edge $\{v, w\}$ is about be added to A.



Kruskal's Algorithm: Correctness (Minimum)

Now, we will prove that any stage of the algorithm A is part of some MST.

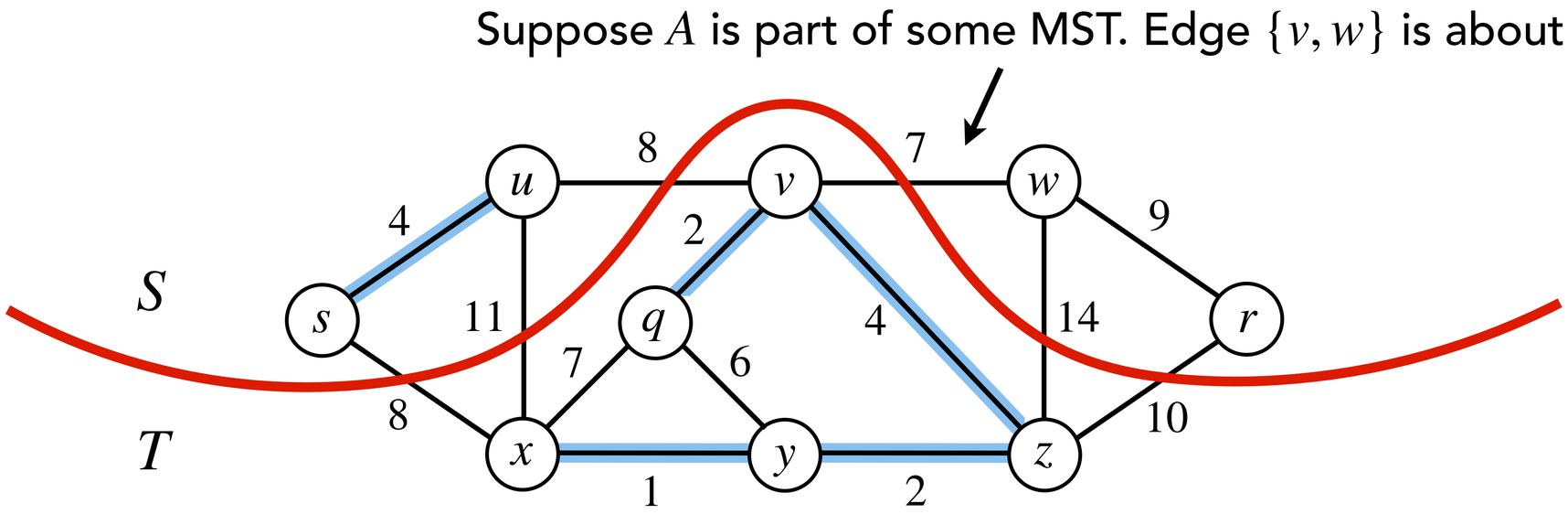
Suppose A is part of some MST. Edge $\{v, w\}$ is about be added to A.



Can you think of a cut so that its cut-set does not have edges from A and $\{v, w\}$ is least weight in it?

Kruskal's Algorithm: Correctness (Minimum)

Now, we will prove that any stage of the algorithm A is part of some MST.



Can you think of a cut so that its cut-set does not have edges from A and $\{v, w\}$ is least weight in it?