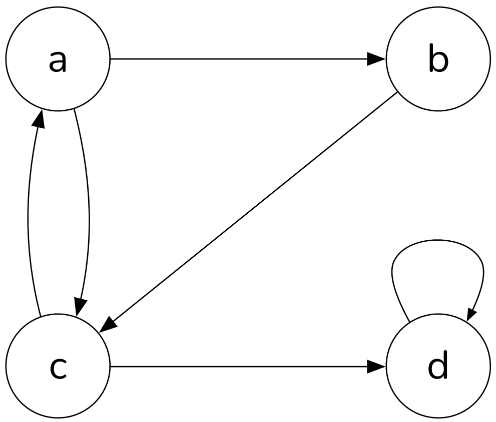
Optimal Substructure and the Handshaking Lemma

Kristian Guillaumier

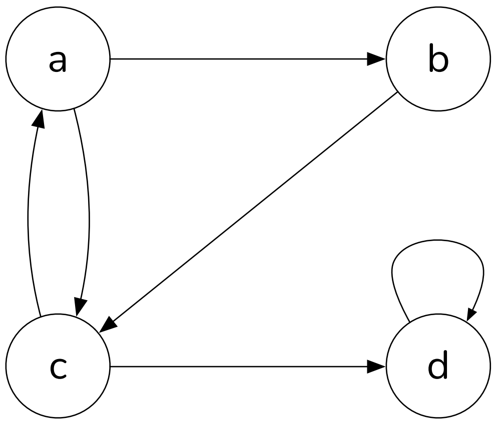
Revision: some graph jargon

- Vertices, edges.
- Labelled vertices, labelled edges.
- Directed, undirected graphs.
- Adjacent vertices (in $a \to b$ the vertex b is adjacent to a).
- The edge $a \to b$ emanates from a. The notation $A(v)$ denotes all the edges that emanate from v.
- The edge $a \to b$ is incident to b. The notation $I(v)$ denotes all the edges incident to the vertex v.



...continued

- Out-degree of a vertex:
  - The number of edges emanating from a node.
  - Written as $|A(v)|$
- In-degree of a vertex:
  - The number of edges incident on a node.
  - Written as $|I(v)|$



Vertex $\mathbf{v}$	$\mathbf{A(v)}$	Out-degree	$\mathbf{I(v)}$	In-degree
a	$\{(a,b),(a,c)\}$	2	$\{(c,a)\}$	1
b	$\{(b,c)\}$	1	$\{(a,b)\}$	1
c	$\{(c,a),(c,d)\}$	2	$\{(a,c),(b,c)\}$	2
d	$\{(d,d)\}$	1	$\{(c,d),(d,d)\}$	2

Example adapted http://www.brpreiss.com

…continued

- Subgraph:
  - The subgraph S of a graph G is a graph where:
    - Each vertex in S is a vertex in V of G.
    - Each edge in S:
      - Is a subset of the edges in G.
      - Each vertex in the edges of S is a vertex in S (this is implied because a subgraph is a graph).
- Connected graphs:
  - There is a path between every pair of vertices.

Graph representations

- Common representations are the adjacency list and the adjacency matrix.
- Suppose the vertices of a graph are labeled with numbers from 0 to $n-1$.
- Adjacency matrix:
  - An adjacency matrix would then be a 2D array (n by n) of Boolean values.
  - $A[i,j]$ true if there is an edge from i to j.
  - If the graph is weighted, then a value in the matrix is the weight.
  - If the graph is undirected $A[i,j] = A[j,i]$. This will make the matrix symmetrical about the diagonal.

...continued

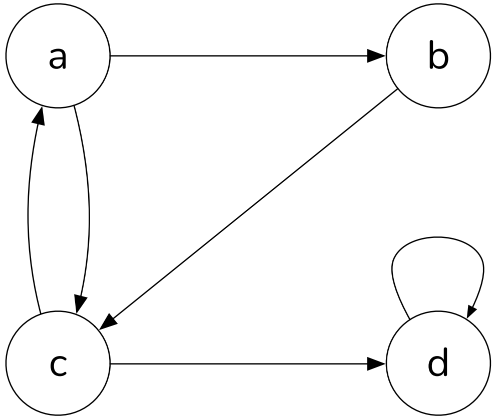
- Consider $G = (V, E)$ where $V = \{v_1, v_2, \dots, v_n\}$.
- We use an n by n matrix A of Boolean values where:

$$A[i,j] = \begin{cases} 1 \text{ if } (v_i, v_j) \in E \\ 0 \text{ otherwise} \end{cases}$$

$A =$

	a	b	c	d
a	0	1	1	0
b	0	0	1	0
c	1	0	0	1
d	0	0	0	1

$=$



...continued

- In adjacency matrices, space is $O(|V|^2)$.
- This is irrespective of the number of edges in the graph.
- If $|E| \ll |V|^2$ then the matrix will be sparse and will be inefficient – most of it will be zeros.
- Discuss sparse vs. dense.

...continued

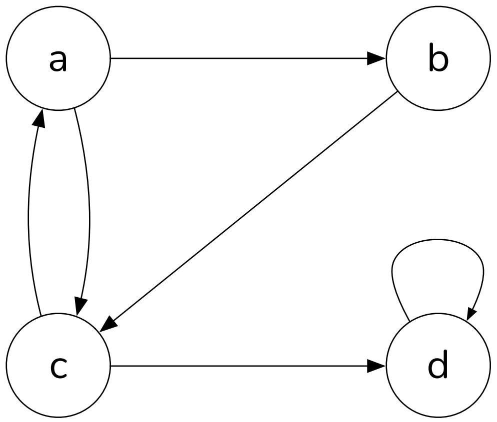
- Adjacency list:
  - Linked list of linked lists.
  - For a graph with n vertices, the primary linked list has n elements. That is, each vertex is a linked list of adjacent vertices.
  - Example:

$$a \to b \to c$$

$$b \to c$$

$$c \to a \to d$$

$$d \to d$$



Suitability

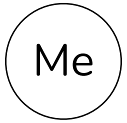
- Consider the operation of determining whether there is an edge between two vertices v_1 and v_2.
  - In adjacency matrix: examine value of $A[v_1, v_2]$.
  - In adjacency list: locate v_1 in primary list, look for v_2 in secondary list.
  - Winner: adjacency matrix.
- Find all vertices adjacent to a vertex v_1 in a graph having n vertices.
  - In adjacency matrix: go to the row for v_1 and iterate over n columns.
  - In adjacency list: go the element v_1 and the secondary list is the list of adjacent vertices.
  - Winner: adjacency list.

The handshaking lemma

- For undirected graphs.
- Colloquially:
  - There is a party.
  - Some people shake other people's hands.
  - Some people might not shake anyone’s hands.
  - I cannot shake hands with myself.
  - Note: if I shook hands with you it implies that you shook hands with me.
  - There will be an even number of handshakes.

Examples 1 and 2

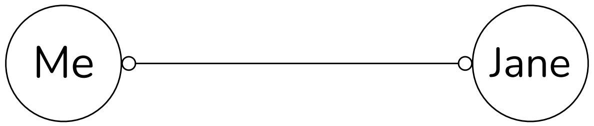
- I am alone. Zero shakes take place (even).



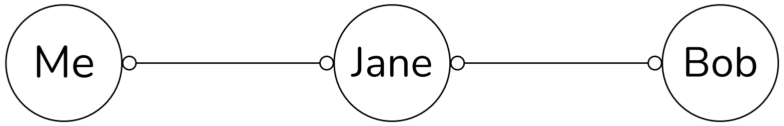
- There are two people at the party, but nobody shakes hands (even).

Examples 3 and 4

- Two people shake hands at the party. I shake hands with you, so you shook hands with me. There are two handshakes. Even.

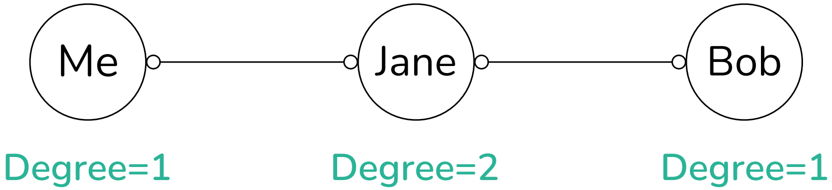


- Three people, shake hands as follows. There are four handshakes. Even.



The handshaking lemma

- Note that I am essentially summing the degrees of every vertex.



$$\sum_{v \in V} \text{Degree}(v) = 2 \times |E|$$

Clearly, every edge is contributing +2 to the result. +1 to one vertex and +1 to another.

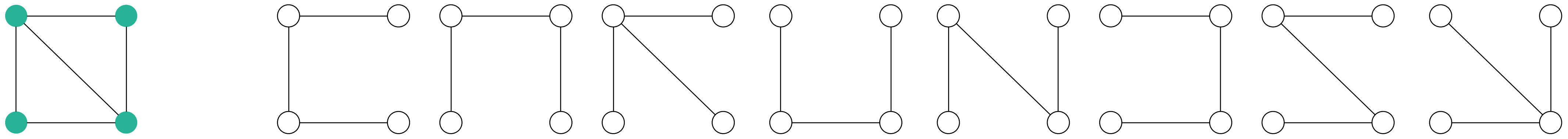
Important

We will be discussing a data structure called a spanning tree

This data structure and related algorithms with be covered as a dedicated topic in much more detail. We are mentioning it here to just as an example to illustrate the principle of optimal substructure and an application of the handshaking lemma.

Spanning trees

- Concerns connected and undirected graphs.
- A spanning tree is a tree that connects all the vertices in a graph and uses some edges.

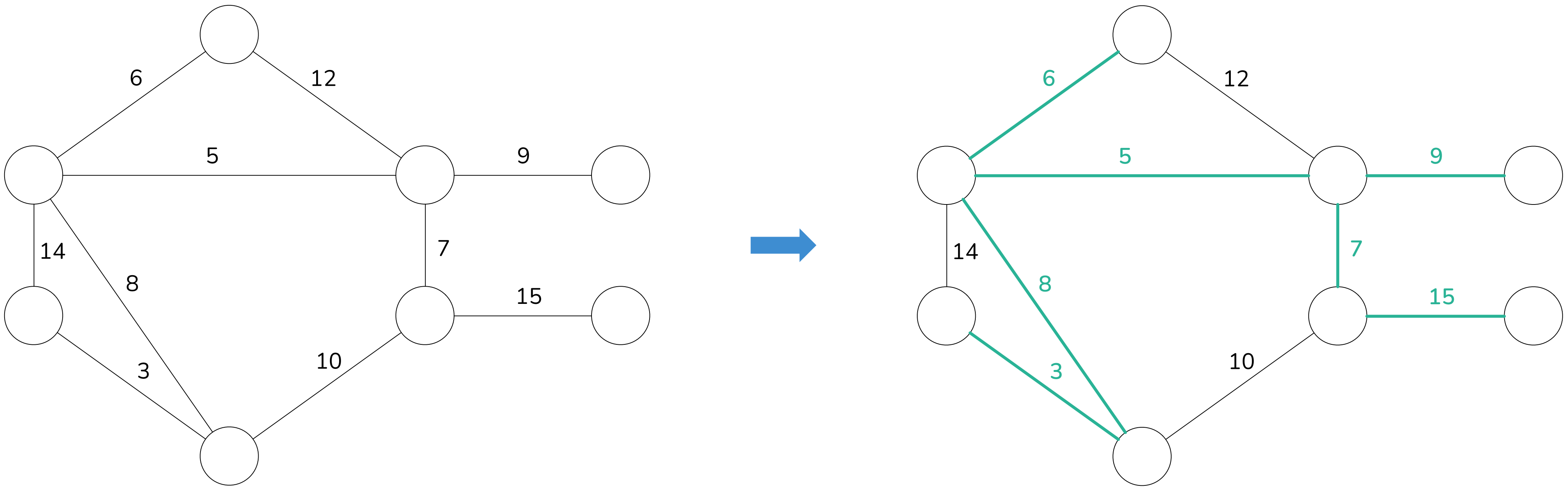


Example from Wolfram

The minimum spanning tree

- Again, concerns connected and undirected graphs.
- The graph is weighted (there is an edge weight function $w: E \rightarrow \mathcal{R}$).
- The weight of a spanning tree in the graph is the sum of the weights it uses.
- The minimum spanning tree is the spanning tree in the graph having the smallest weight.
- Major applications in distributed systems.
- From here on, we will assume that all edge weights are distinct.

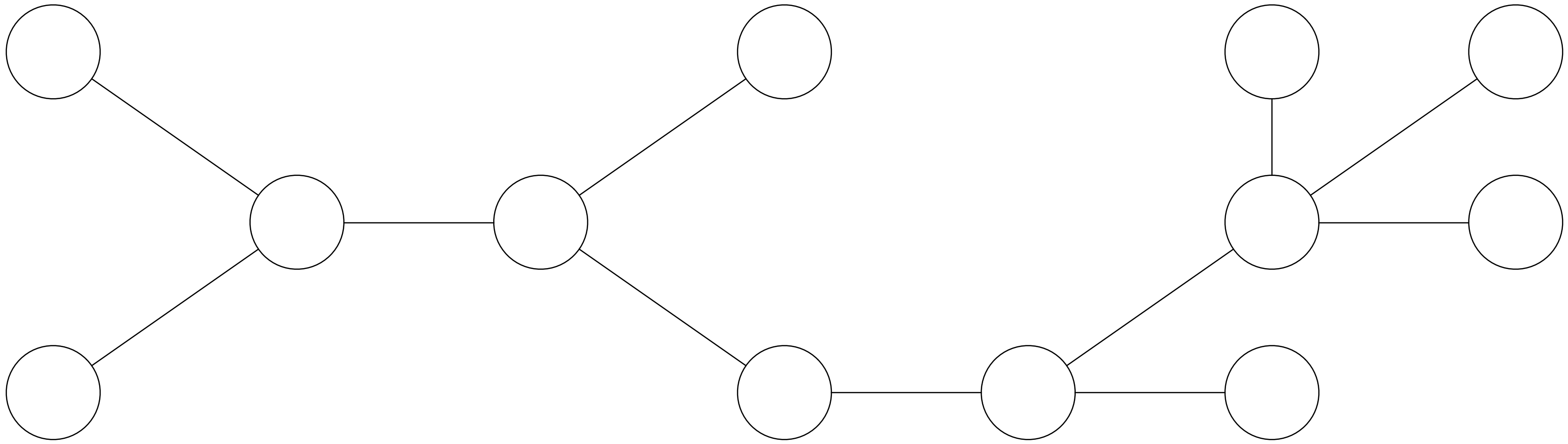
An example MST



Example from Eric Demaine

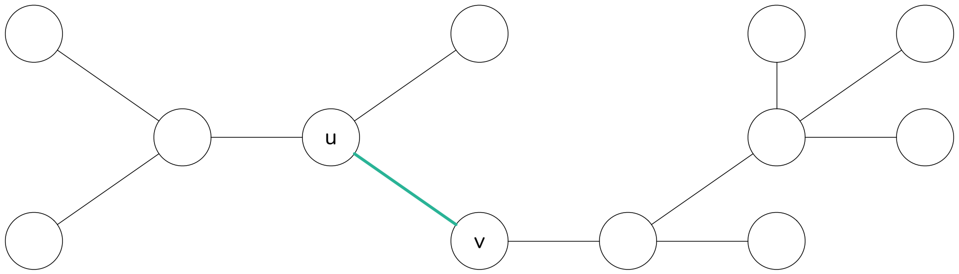
Claim: MSTs have optimal substructure

- Let's assume we have the following MST.
- Only the edges in the MST are shown (so we're just seeing the tree here).



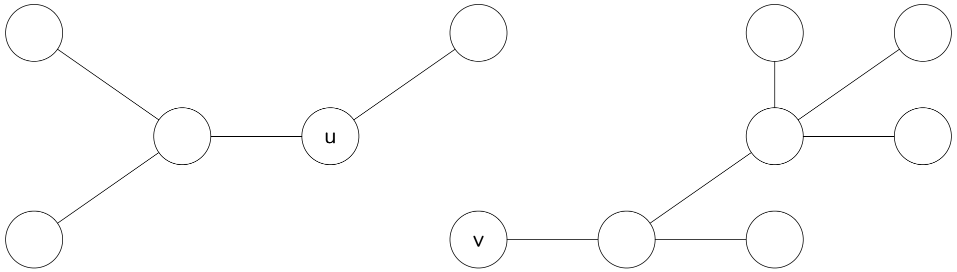
...continued

- If we remove any edge (u,v) we will end up partitioning the tree in two.



We will end up with two trees T1 and T2.

T1 =

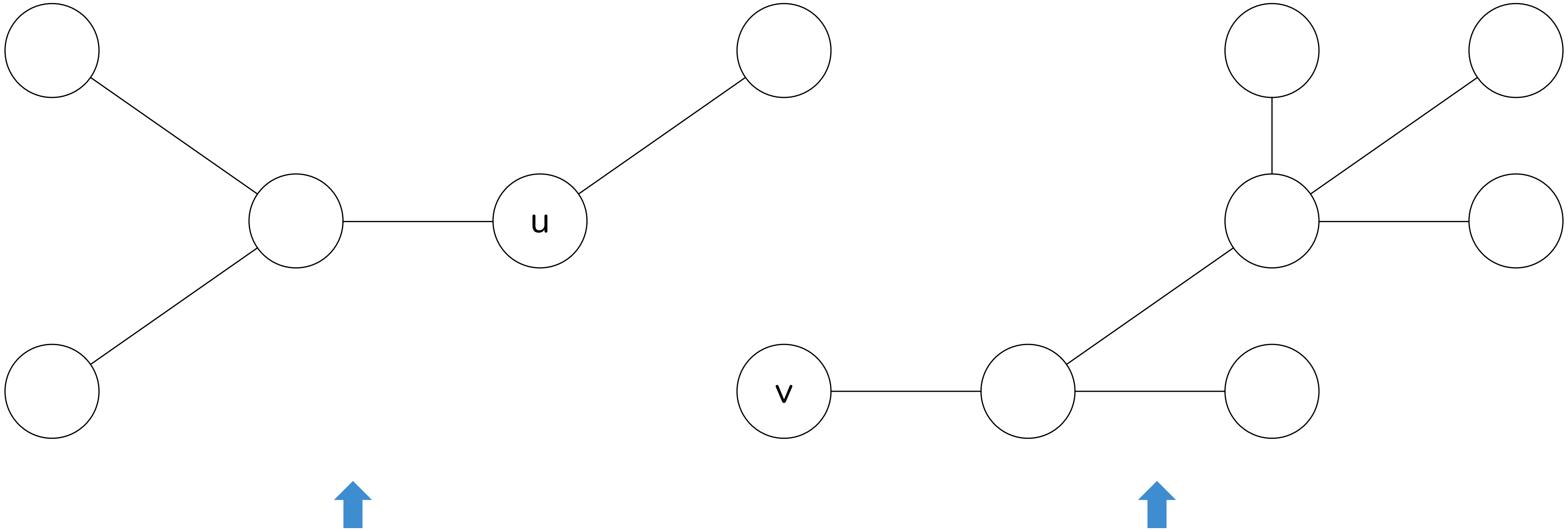


= T2

MSTs have optimal substructure

- Theorem – if T is an MST and we remove some edge (u,v):
  - T1 is an MST for the graph G1=(V1, E1) where G1 is the subgraph of G containing:
    - V1 = the vertices in T1.
    - E1 = (x,y) ϵ E : x, y ϵ V1.
  - T2 is an MST for the graph G2=(V2, E2) where G2 is the subgraph of G containing:
    - V2 = the vertices in T2.
    - E2 = (x,y) ϵ E : x, y ϵ V2.

...continued



T1 is an MST in this subgraph

T2 is an MST in this subgraph

Proof

- The weight of the whole MST is equal the weight of the edge removed plus the weight of the two subtrees.

$$w(T) = w(u, v) + w(T_1) + w(T_2)$$

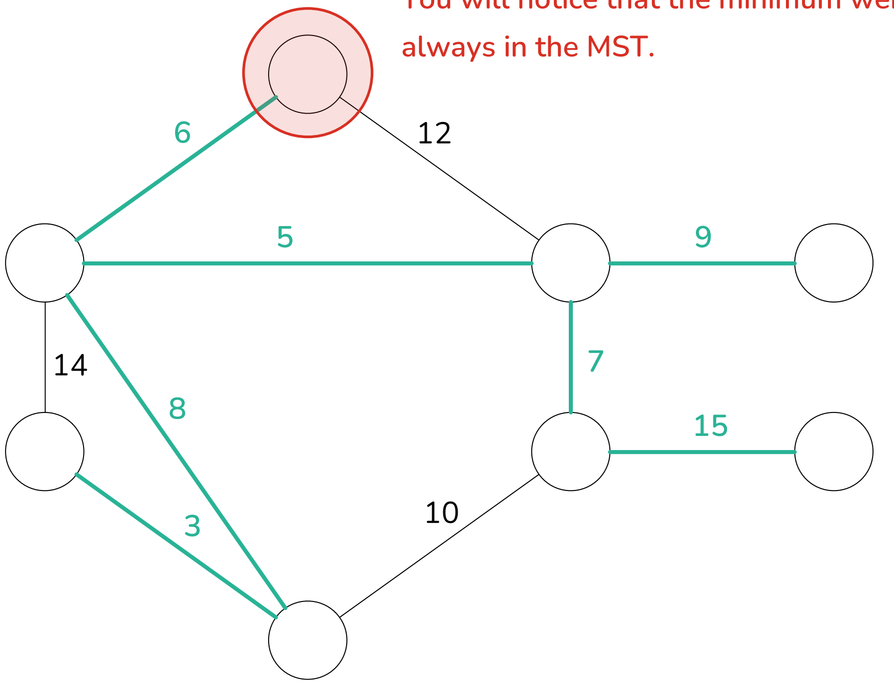
- Suppose some MST $T_1{}'$ existed that is lighter than T_1 for G_1 then:
  - T' is a spanning tree containing the edges: $\{u, v\} \cup T_1' \cup T_2$
  - Where T' is a lighter MST than T. This is a contradiction as this would imply that T was not an MST.
- Same applies when arguing for T2'.

Theorem

- Let T be the MST of $G = (V, E)$.
- Let $A \subseteq V$.
- Suppose $\{u, v\} \in E$ is the least-weight edge that connects A to $V - A$.
- Then $\{u, v\} \in T$. In other words, $\{u, v\}$ belongs to the MST T.

Illustration

- To illustrate, let's pick any single vertex:

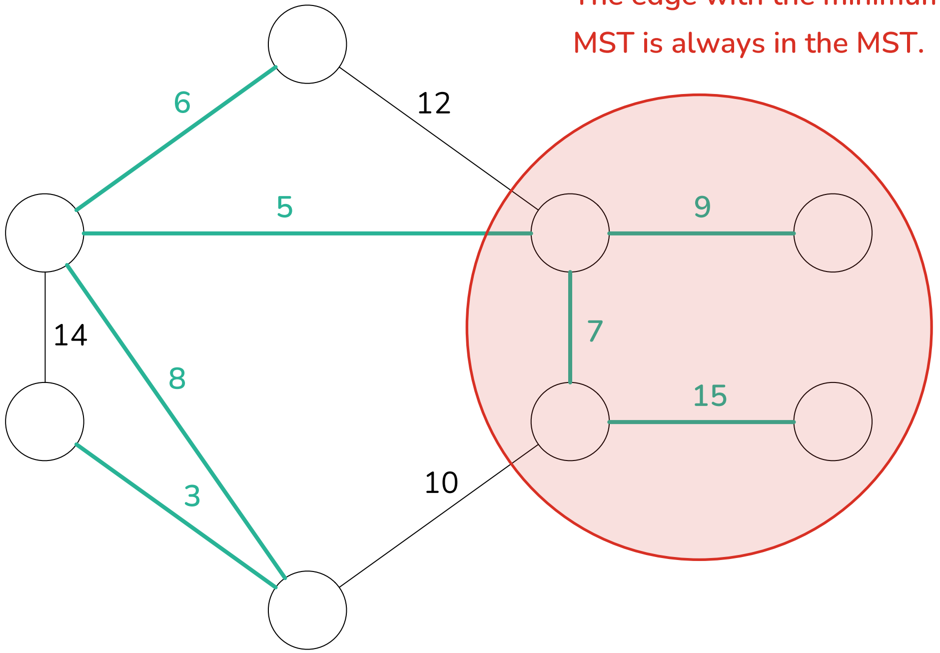


You will notice that the minimum weight edge emanating from any vertex is always in the MST.

Illustration

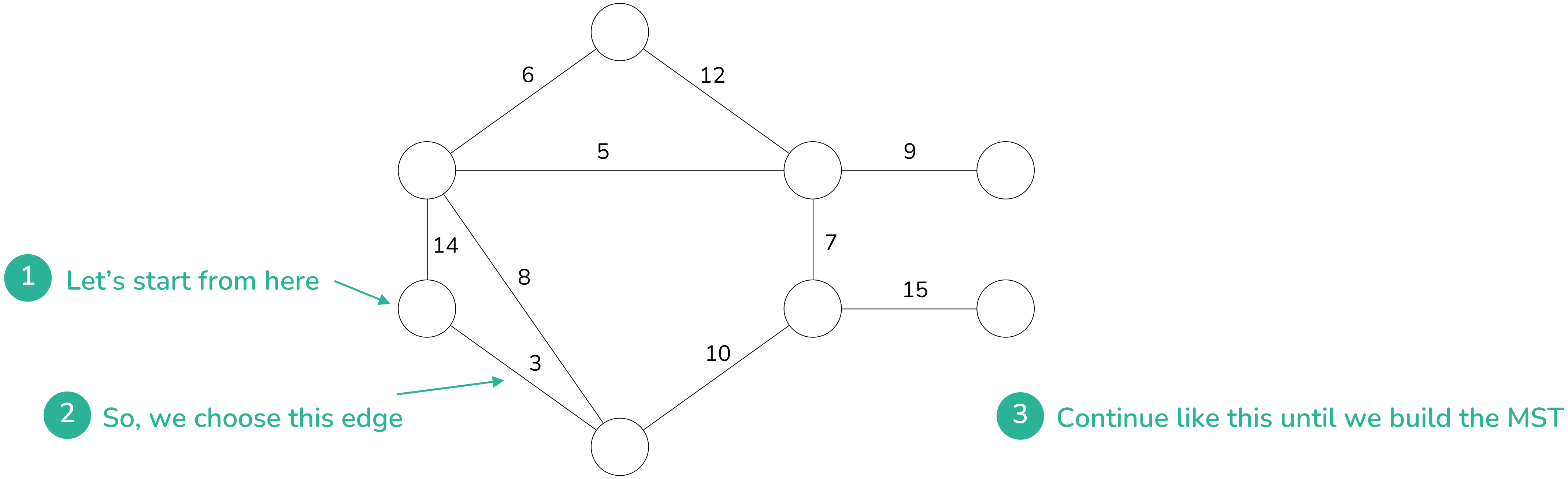
- More than one vertex.

The edge with the minimum weight emanating from a subset of vertices of the MST is always in the MST.



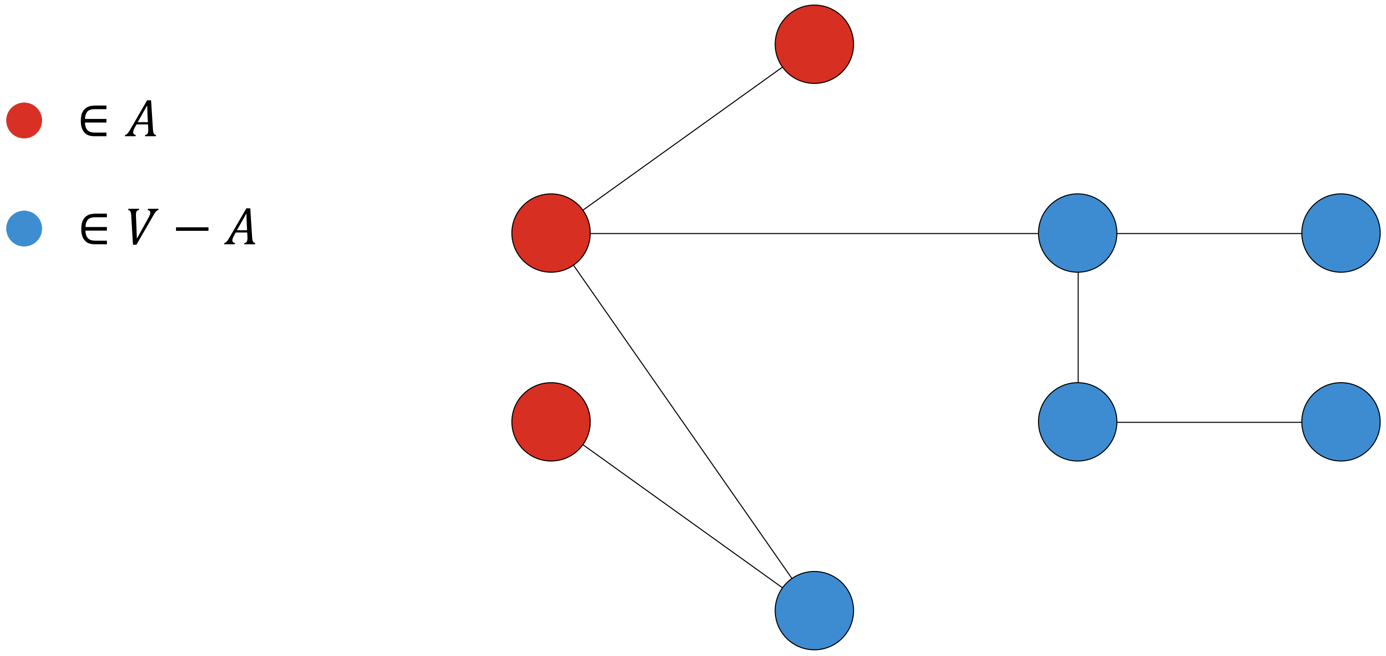
What does this mean?

- If I repeatedly connect a subtree of the MST to the rest of the graph and always pick the smallest weight edge to do so, I will end up with the MST for the graph.
- Try this for the following:



Proof

- Suppose I have this MST…



Our theorem:

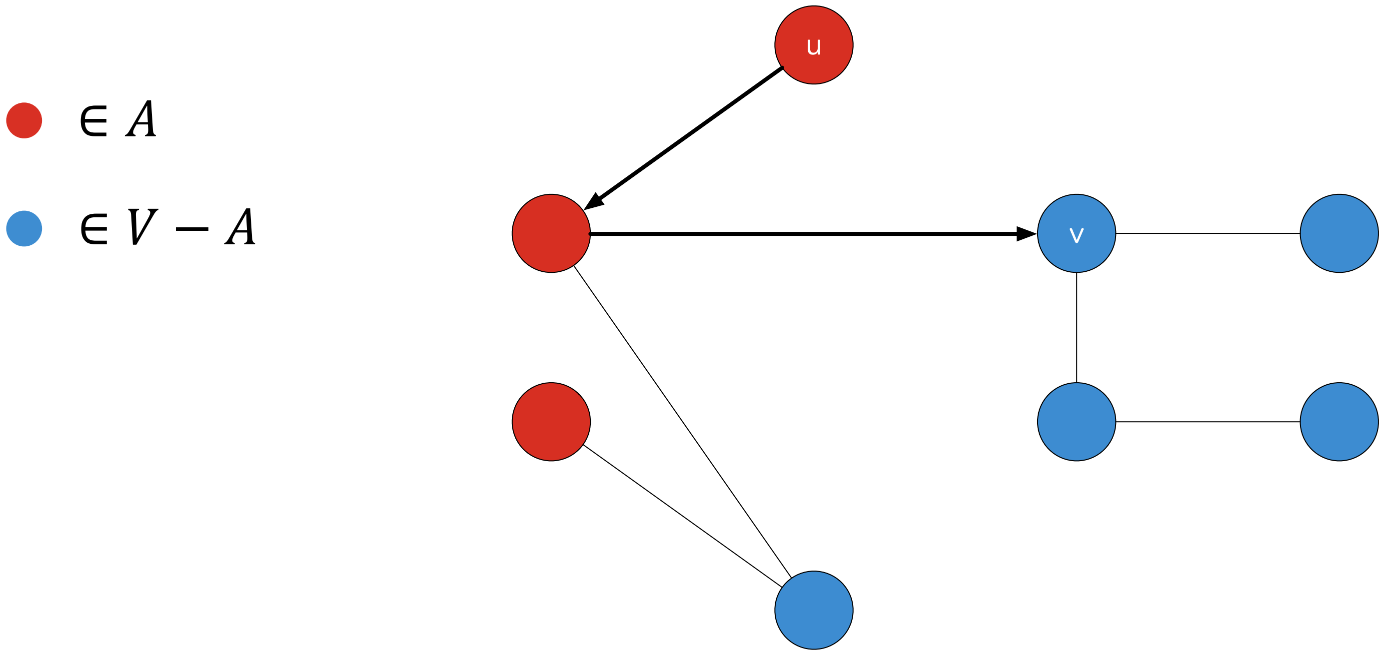
- Let T be the MST of $G = (V, E)$.
- Let $A \subseteq V$.
- Suppose $\{u, v\} \in E$ is the least-weight edge that connects A to $V - A$.
- Then $\{u, v\} \in T$. In other words, $\{u, v\}$ belongs to the MST T.

Proof

Our theorem:

- Let T be the MST of $G = (V, E)$.
- Let $A \subseteq V$.
- Suppose $\{u, v\} \in E$ is the least-weight edge that connects A to $V - A$.
- Then $\{u, v\} \in T$. In other words, $\{u, v\}$ belongs to the MST T.

- There is a unique simple path from u to v...



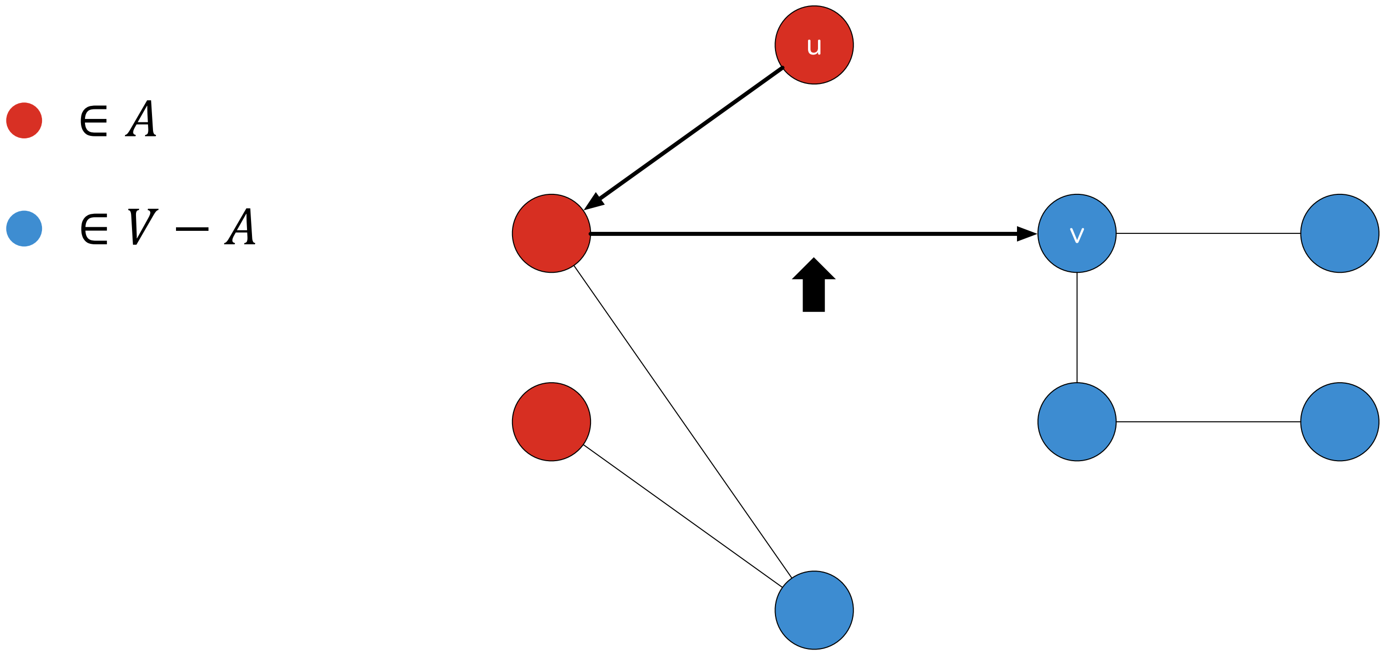
This 'unique simple path' property is true because we are dealing with a tree.

Proof

- At some point, the path will transition from A into (V-A)...

Our theorem:

- Let T be the MST of $G = (V, E)$.
- Let $A \subseteq V$.
- Suppose $\{u, v\} \in E$ is the least-weight edge that connects A to $V - A$.
- Then $\{u, v\} \in T$. In other words, $\{u, v\}$ belongs to the MST T.



Obviously because {u,v} connects A and (V-A)

Proof

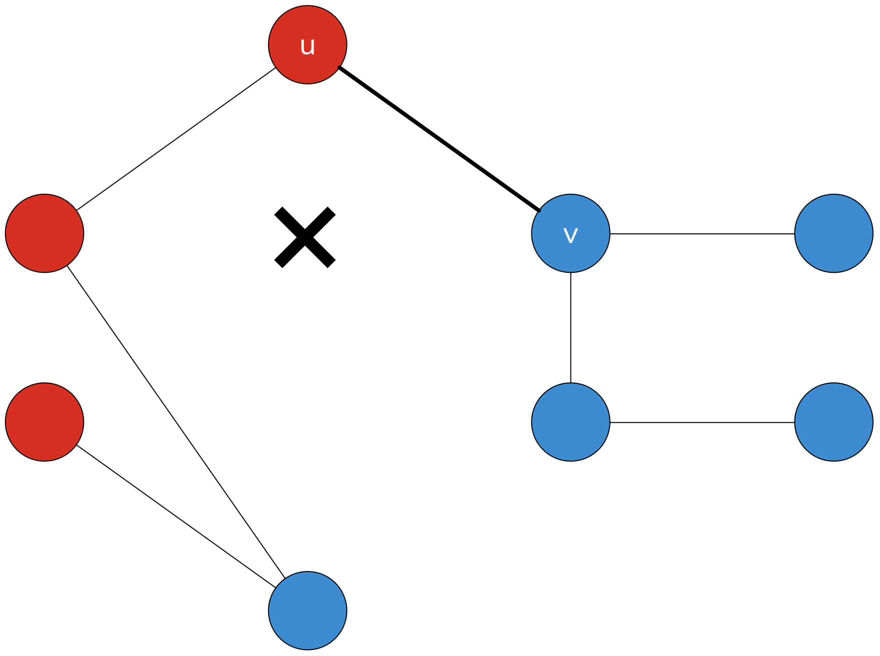
Our theorem:

- Let T be the MST of $G = (V, E)$.
- Let $A \subseteq V$.
- Suppose $\{u, v\} \in E$ is the least-weight edge that connects A to $V - A$.
- Then $\{u, v\} \in T$. In other words, $\{u, v\}$ belongs to the MST T.

- Assume that {u,v} ϵ T.
- And swap it with the edge in the path that transitions from A to (V-A)…

● $\in A$

● $\in V - A$

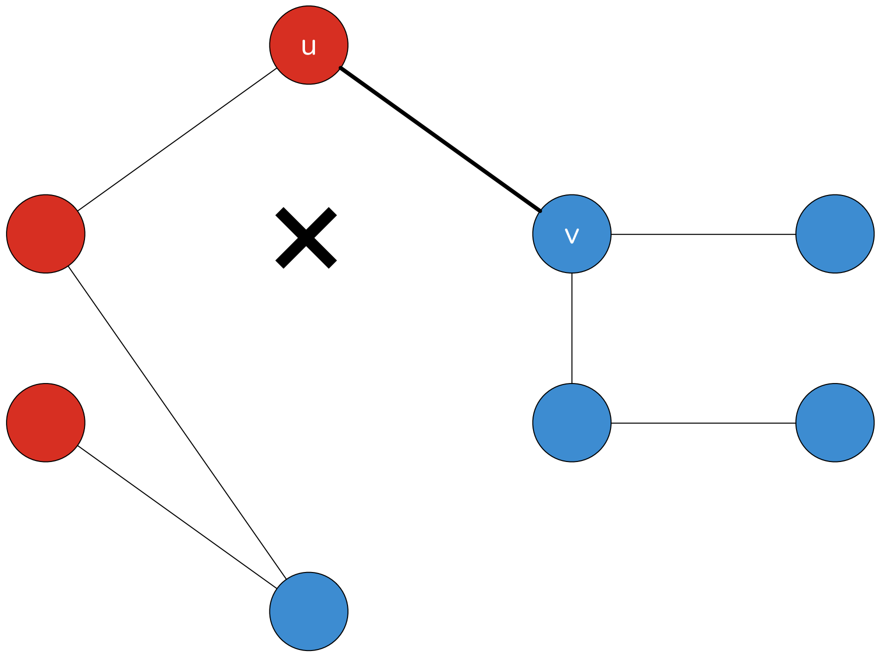


Proof

- If according to the requirement that the new edge is the lightest one connecting A and (V-A), then it is lighter than the edge we just removed.
- This is a contradiction because this would be a lighter tree than my starting one (which wouldn't have been an MST).

Our theorem:

- Let T be the MST of $G = (V, E)$.
- Let $A \subseteq V$.
- Suppose $\{u, v\} \in E$ is the least-weight edge that connects A to $V - A$.
- Then $\{u, v\} \in T$. In other words, $\{u, v\}$ belongs to the MST T.



Prim's Algorithm

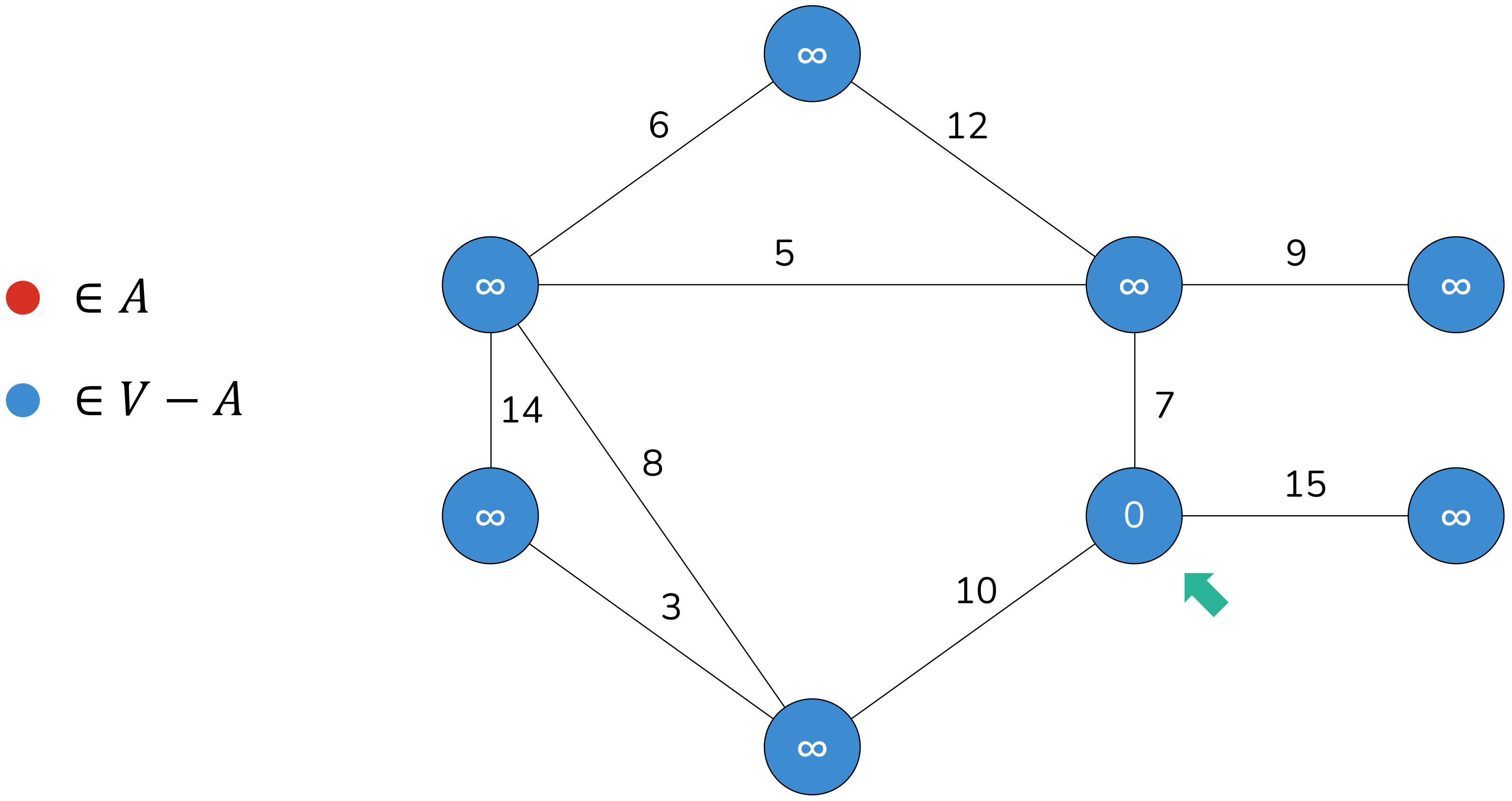
- Create a priority queue Q that contains the vertices (V-A), where:
  - The weight of a node in Q is the lightest node going to a vertex in A.
  - Initially A is empty.
  - The weight of each node in Q will be ∞. (A is empty so the lightest weight going to A is ∞).
- So far:
  - Q contains all of V (V-A actually, but A is empty).
  - All weights of the nodes in Q are ∞.
- Pick any node in Q and set its weight to zero.

…continued

Initialising the algorithm…

```
Q = V       // Q is a priority queue
Key[v] = ∞ // ∀v ∈ V
Key[s] = 0 // for any s ∈ V
```

...continued

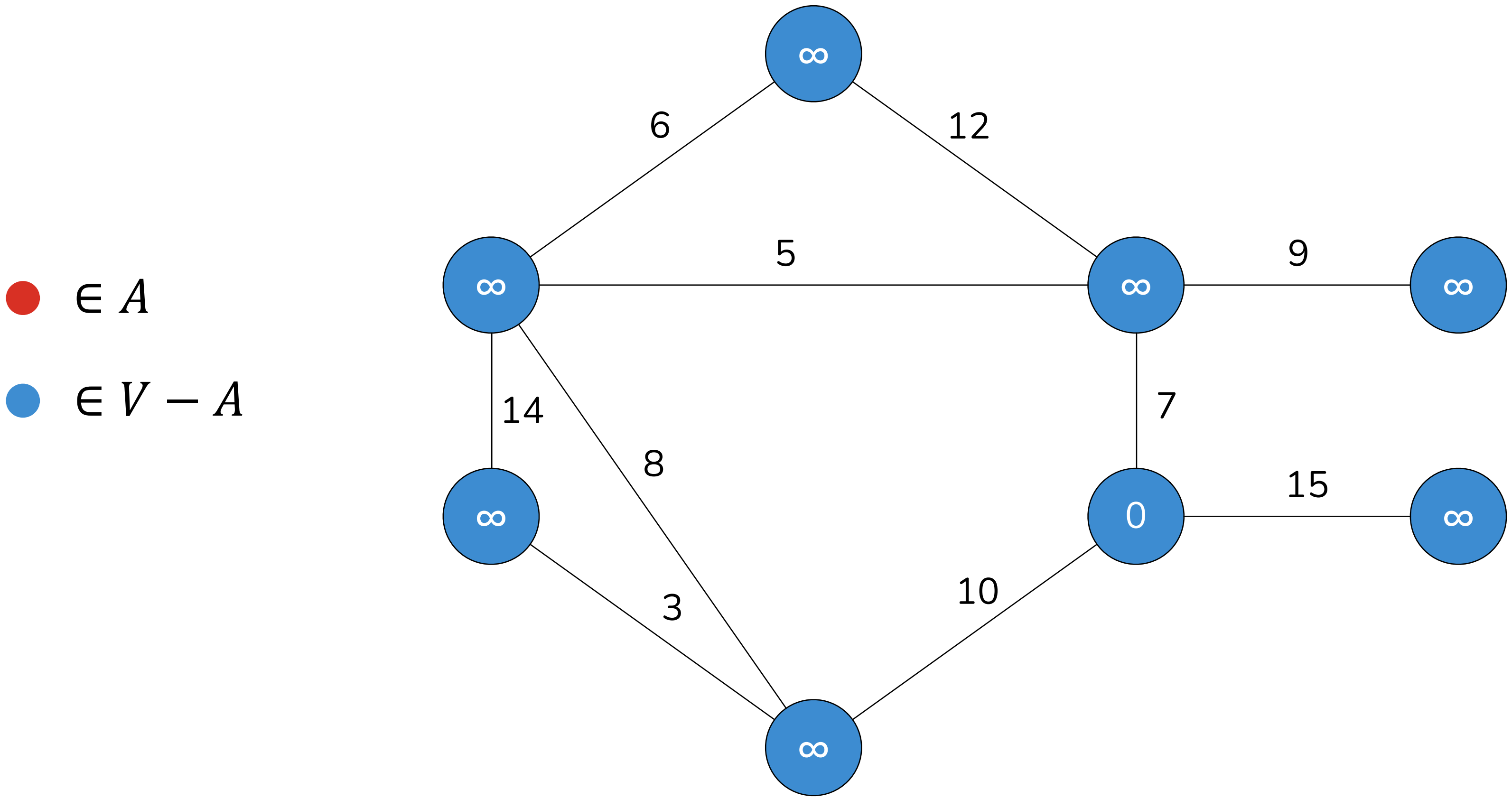


…continued

```
while Q ≠ empty {
  u = DequeueMin(Q) // Get the min (lightest) element.

  for each v ∈ Adjacent[u] {
    if v ∈ Q and w(u,v) < Key[v] then {
      Key[v] = w(u,v)
      Parent[v] = u
    }
  }
}
```

Start: initialise

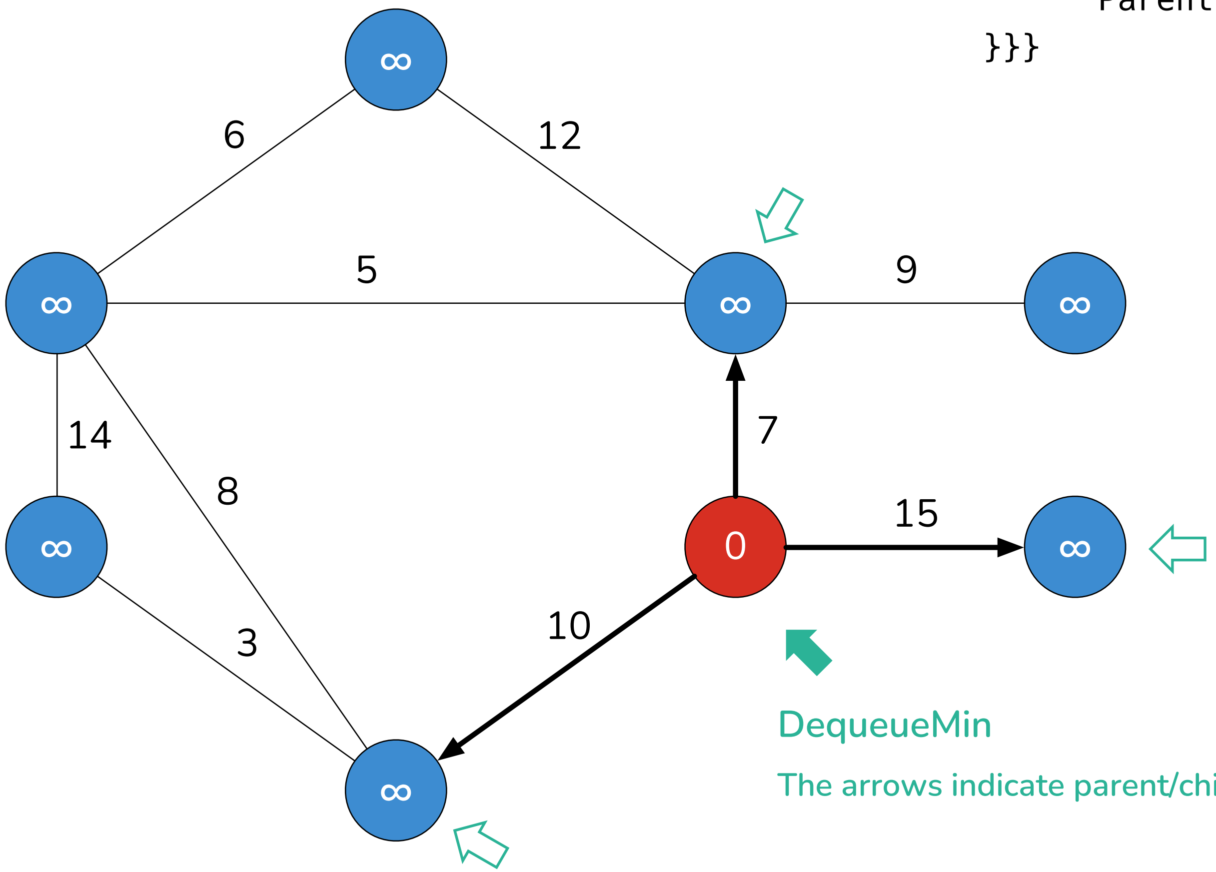


While queue is not empty

```
while Q ≠ empty {
  u = DequeueMin(Q)
  for each v ∈ Adjacent[u] {
    if v ∈ Q and w(u,v) < Key[v] then {
      Key[v] = w(u,v)
      Parent[v] = u
}}}
```

● $\in A$

● $\in V - A$

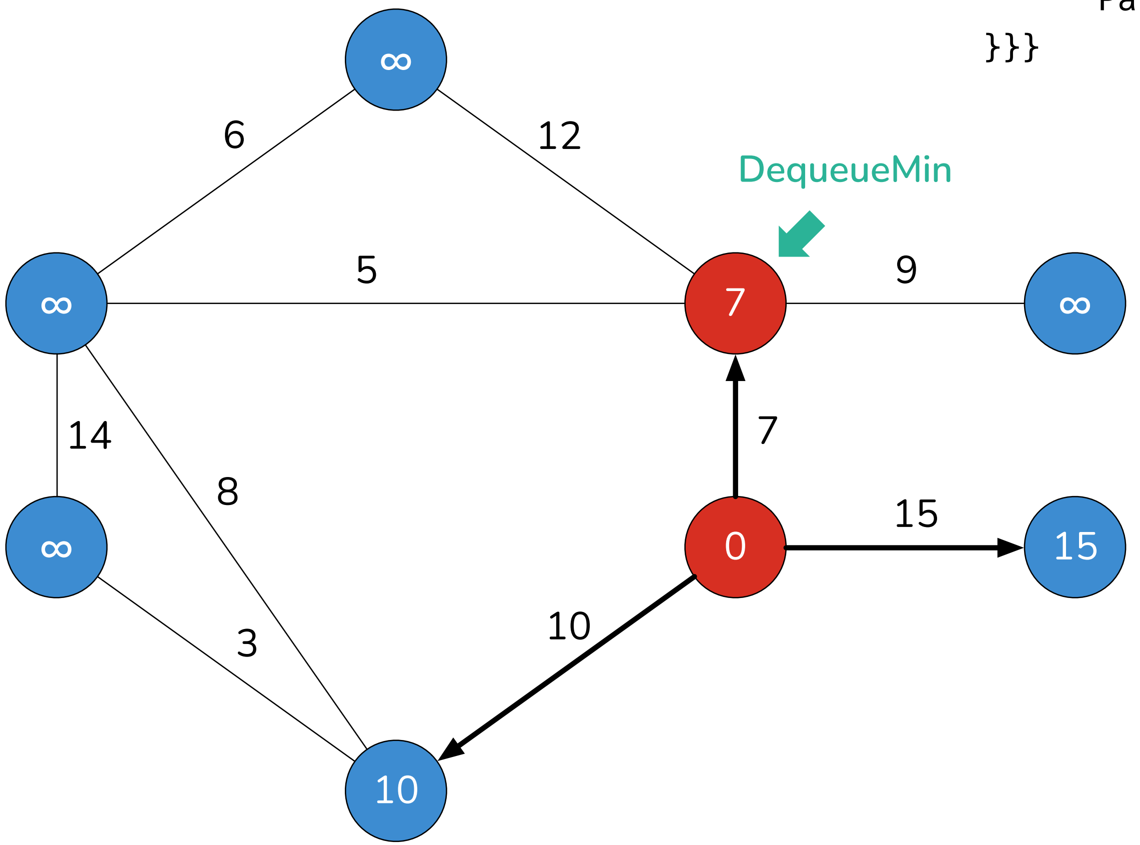


The arrows indicate parent/child relationships in the tree.

…continued

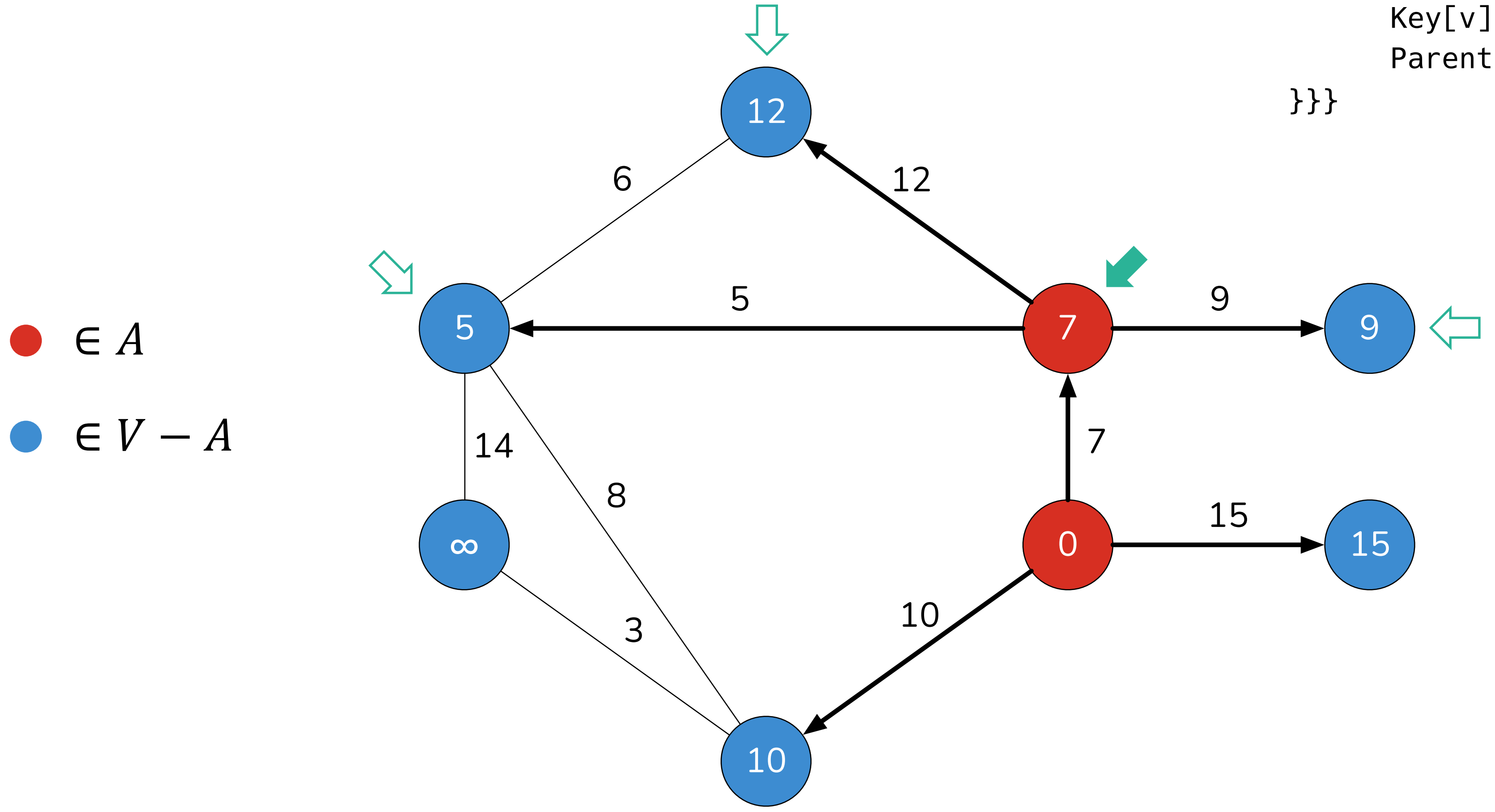
```
while Q ≠ empty {
  u = DequeueMin(Q)
  for each v ∈ Adjacent[u] {
    if v ∈ Q and w(u,v) < Key[v] then {
      Key[v] = w(u,v)
      Parent[v] = u
}}}
```

- $\in A$
- $\in V - A$



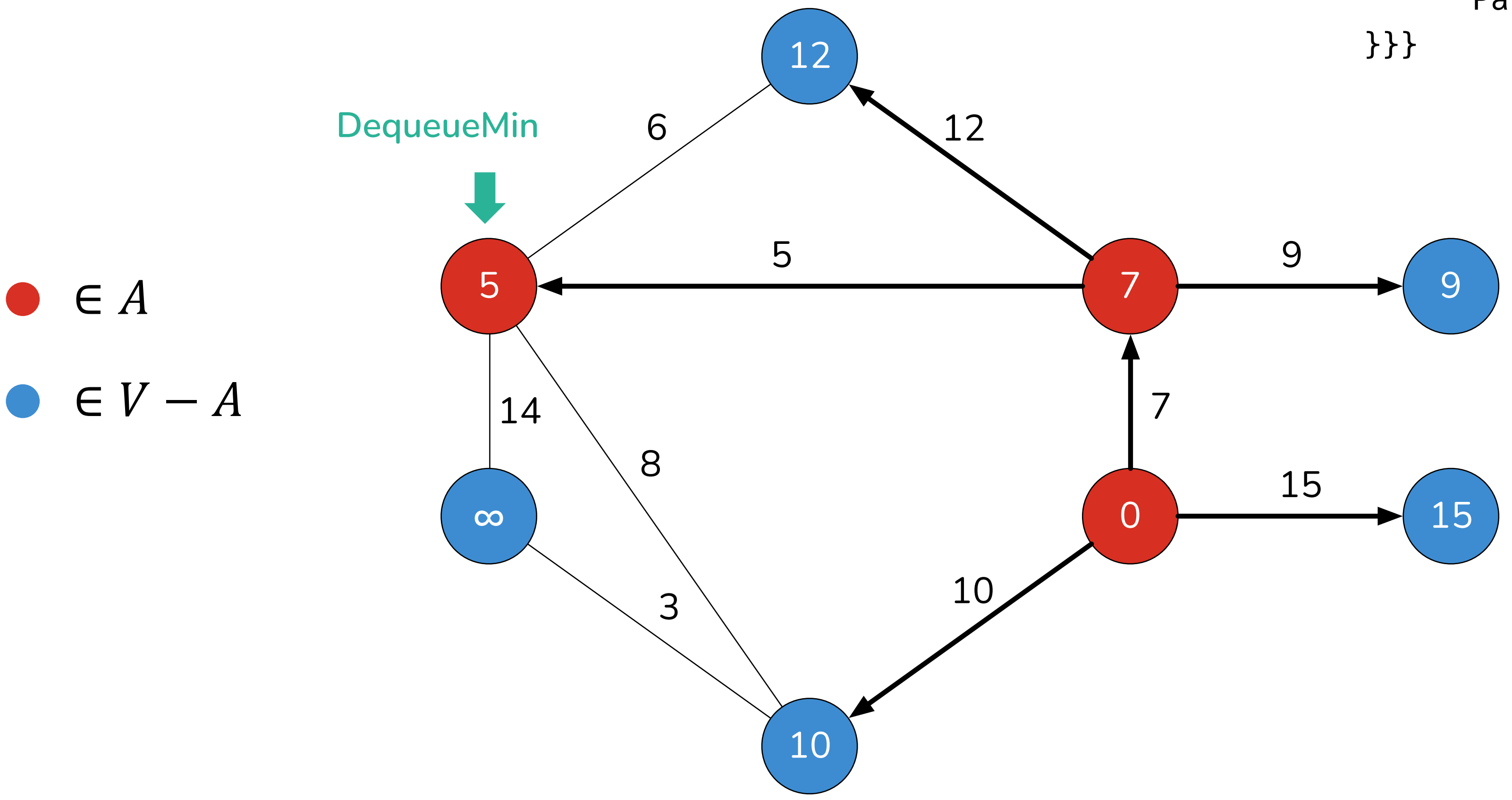
...continued

```
while Q ≠ empty {
  u = DequeueMin(Q)
  for each v ∈ Adjacent[u] {
    if v ∈ Q and w(u,v) < Key[v] then {
      Key[v] = w(u,v)
      Parent[v] = u
}}}
```



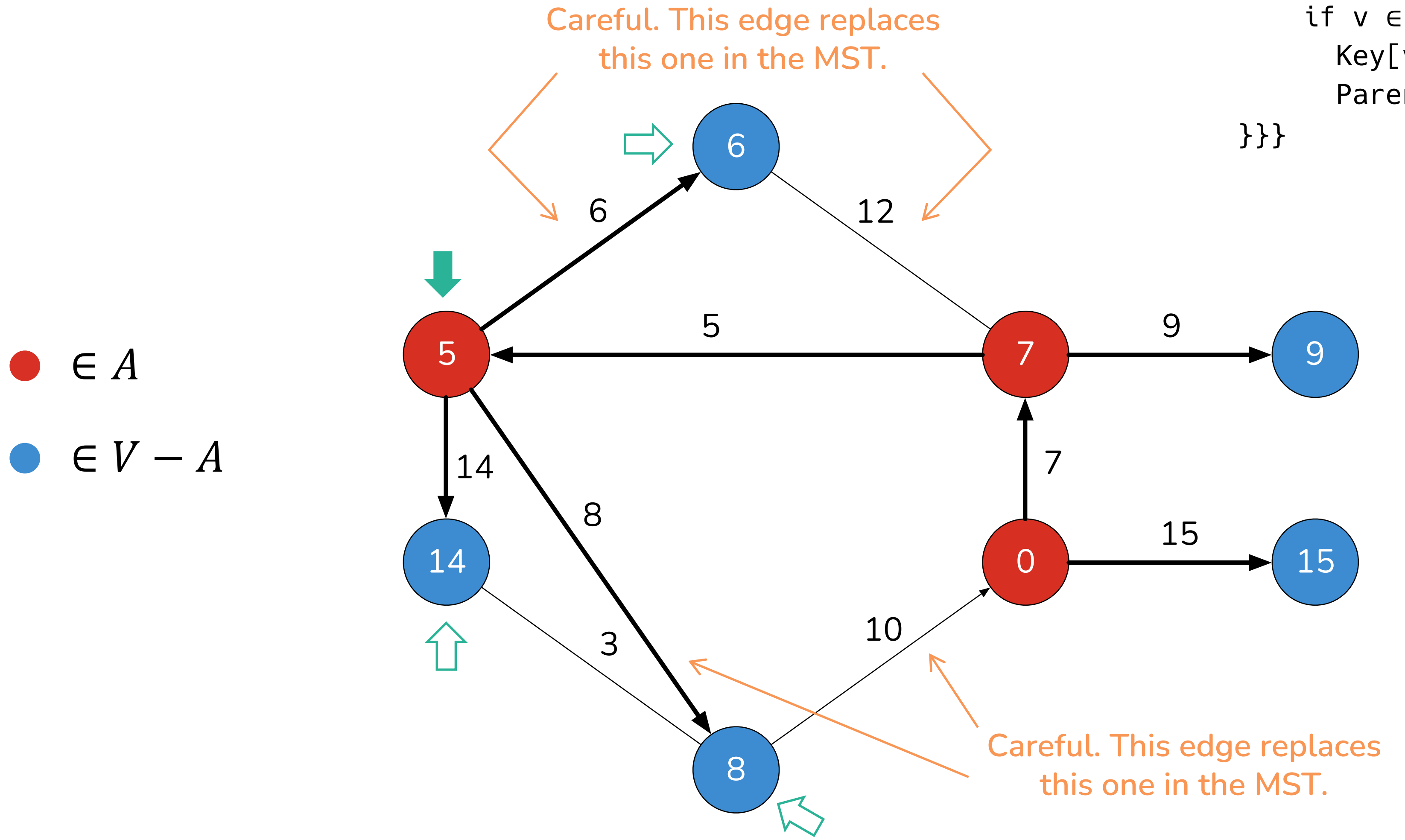
…continued

```
while Q ≠ empty {
  u = DequeueMin(Q)
  for each v ∈ Adjacent[u] {
    if v ∈ Q and w(u,v) < Key[v] then {
      Key[v] = w(u,v)
      Parent[v] = u
}}}
```



...continued

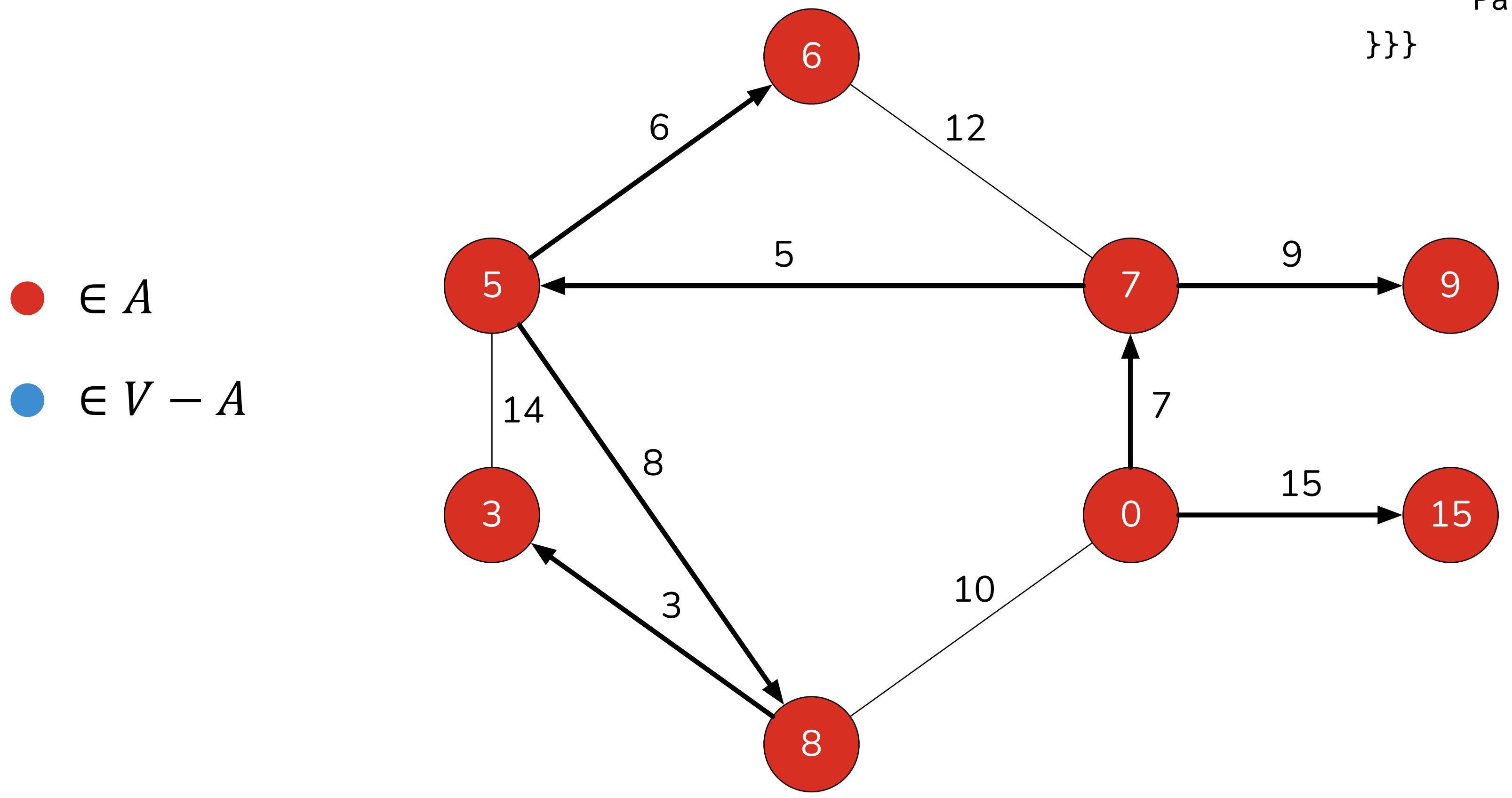
```
while Q ≠ empty {
  u = DequeueMin(Q)
  for each v ∈ Adjacent[u] {
    if v ∈ Q and w(u,v) < Key[v] then {
      Key[v] = w(u,v)
      Parent[v] = u
}}}
```



...continued

- And so on...

```
while Q ≠ empty {
  u = DequeueMin(Q)
  for each v ∈ Adjacent[u] {
    if v ∈ Q and w(u,v) < Key[v] then {
      Key[v] = w(u,v)
      Parent[v] = u
}}}
```



Analysis

```
Q = A
Key[v] = ∞
Key[s] = 0
```

Linear time.

```
while Q ≠ empty {
  u = DequeueMin(Q)
  for each v ∈ Adjacent[u] {
        if v ∈ Q and w(u,v) < Key[v] then {
               Key[v] = w(u,v)
               Parent[v] = u
        }
  }
}
```

Degree(u) times.

|V| times.

...continued

- So, we have |V| DequeueMins.
- And Degree(u) UpdateKeys for |V| times giving O(E) UpdateKeys.
- Total running time then is:
  - O(V)×TimeTo(DequeueMin) + O(E)×TimeTo(UpdateKey).
  - TimeTo() depends on the data structure I use to implement the Queue.

Remember the handshaking lemma:

$$\sum_{v \in V} \text{Degree}(v) = 2 \times |E|$$

Q Data Structure	DequeueMin	UpdateKey	Total
Unsorted Array	O(V)	O(1)	$O(V^2) + O(E) = O(V^2)$
Min Heap	$O(\log_2 V)$	$O(\log_2 V)$	$O(V.\log_2(V) + O(E.\log_2(V)) = O(E.\log_2(V))$

Note: if graph is dense, E approaches V^2 so Unsorted Array is better. If graph is sparse, Min Heap is better.

Further reading

- These notes should be supplemented by:
  - Introduction to Algorithms (Clifford Stein, Thomas H Cormen, Ronald L Rivest, Charles E Leiserson – MIT Press)
  - Also see Eric Demaine: http://videolectures.net/mit6046jf05_leiserson_lec16/
  - Also see previous material on graphs.