PSO 12

MST, Prim/Kruskal, (Backwards) Pattern Matching

Slides @ justin-zhang.com/teaching/CS251

Question 1

(Minimum spanning trees)

1. An edge is called a **light-edge** crossing a cut $\mathcal{C} := (S, V - S)$, if its weight is the minimum of any edge crossing the cut. Show that:

- if an edge (u, v) is contained in some MST, then it is a light-edge crossing some cut of the graph.
- the converse is not true, and give a simple counter-example of a connected graph such that there exists a cut $\mathcal{C} := (S, V - S)$, in which (u, v) is a light-edge crossing the cut $\mathcal{C}$ but does not form a MST of the graph.

2. Show that a graph has a unique MST, if for every cut of the graph, there is a unique light-edge crossing the cut. Show that the converse is not true by giving a counter-example.

3. Let T be an MST of a graph $G = (V, E)$, and let $V^{'}$ be a subset of V. Let $T^{'}$ be the subgraph of T induced by $V^{'}$, and let $G^{'}$ be the subgraph of G induced by $V^{'}$. Show that if $T^{'}$ is connected, then $T^{'}$ is an MST of $G^{'}$.

Question 2

(Prim's & Kruskal's algorithm)

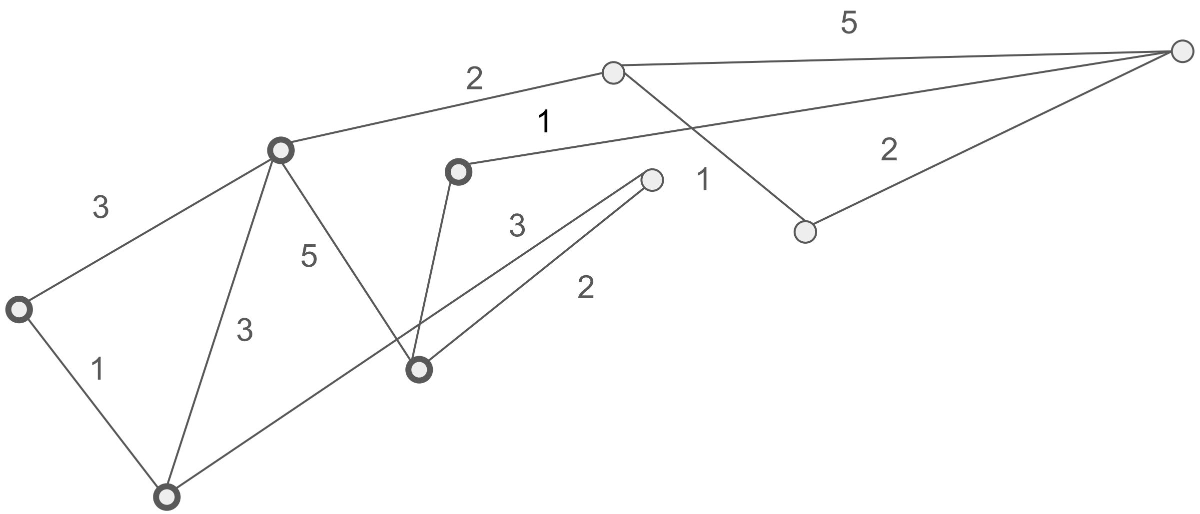
1. Suppose that we represent the graph $G = (V, E)$ as an adjacency-matrix. Give a simple implementation of Prim's algorithm for this case that runs in $O(|V|^2)$ time.

2. Suppose that all edge weights in a graph are integers in the range from 1 to $|V|$. How fast can you make Kruskal's algorithm run?

Question 1

(Minimum spanning trees)

1. An edge is called a **light-edge** crossing a cut $\mathcal{C} := (S, V - S)$, if its weight is the minimum of any edge crossing the cut. Show that:

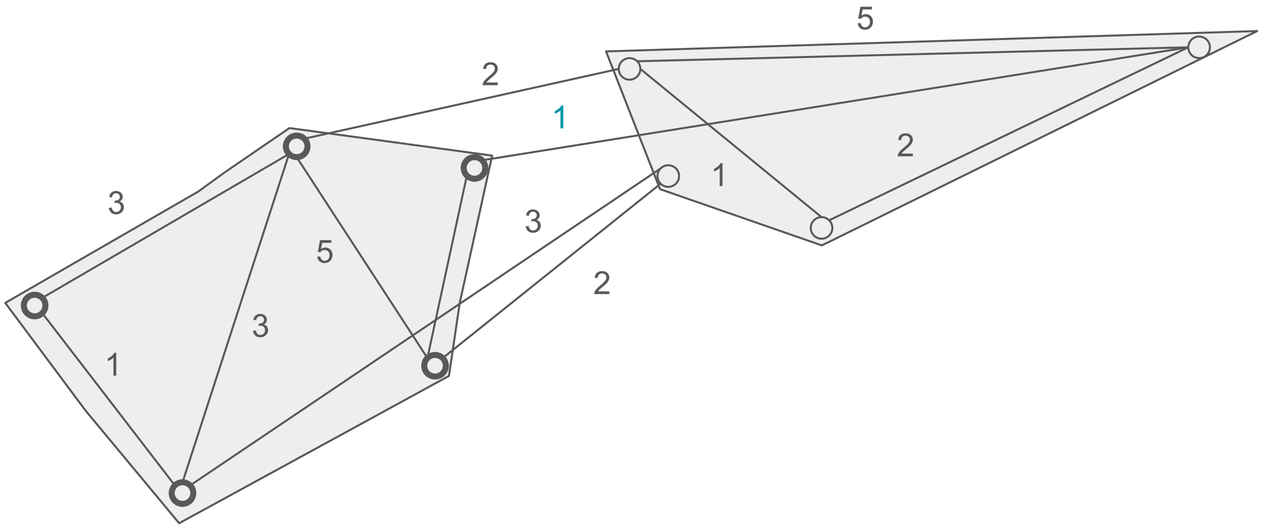


Say I define **C** as

Question 1

(Minimum spanning trees)

1. An edge is called a **light-edge** crossing a cut $\mathcal{C} := (S, V - S)$, if its weight is the minimum of any edge crossing the cut. Show that:

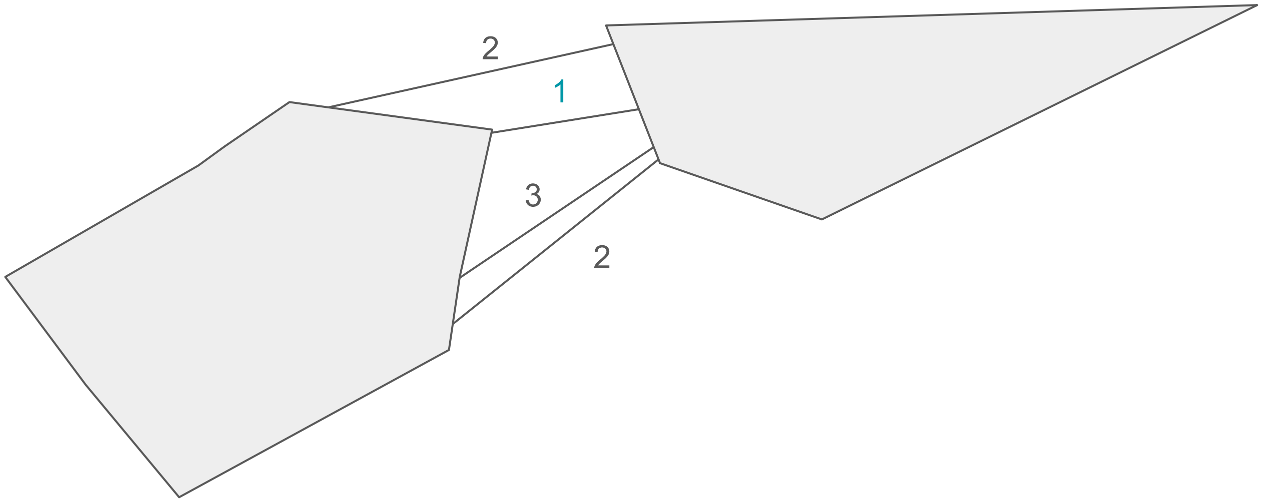


This forms a 'cut'

Question 1

(Minimum spanning trees)

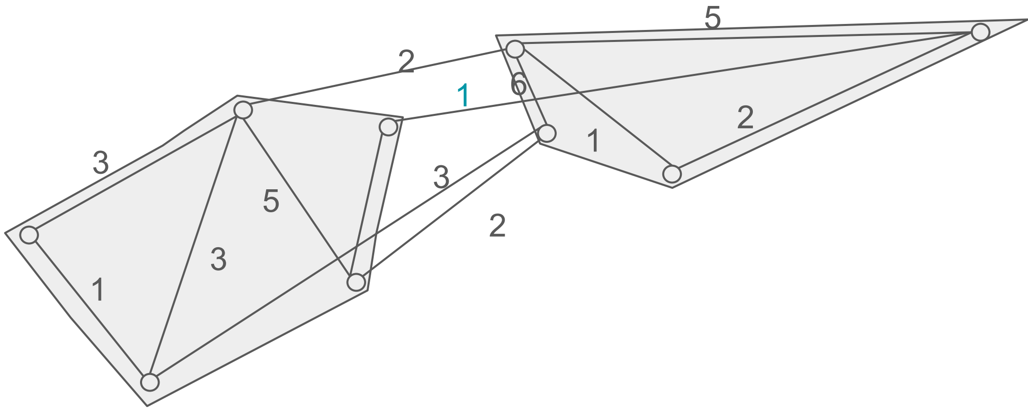
1. An edge is called a **light-edge** crossing a cut $\mathcal{C} := (S, V - S)$, if its weight is the minimum of any edge crossing the cut. Show that:



The light edge of this cut has weight 1

- if an edge (u, v) is contained in some MST, then it is a light-edge crossing some cut of the graph.

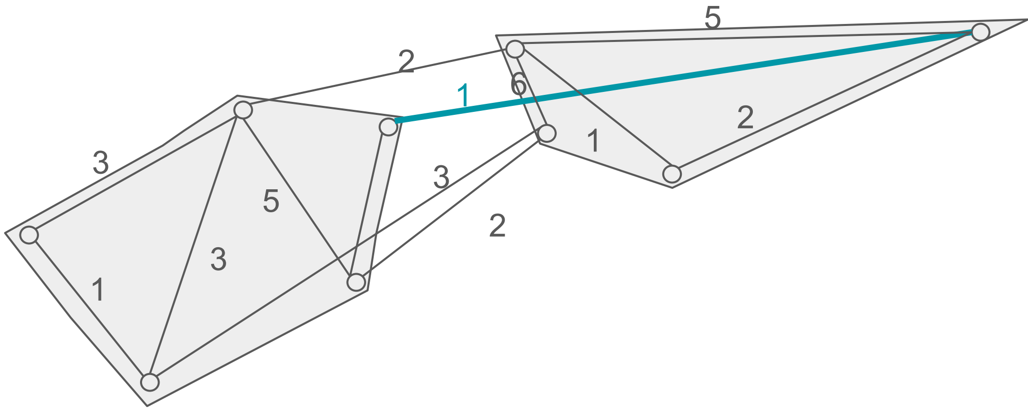
Pf:



- if an edge (u, v) is contained in some MST, then it is a light-edge crossing some cut of the graph.

Pf: AFtSoC e is not in a MST

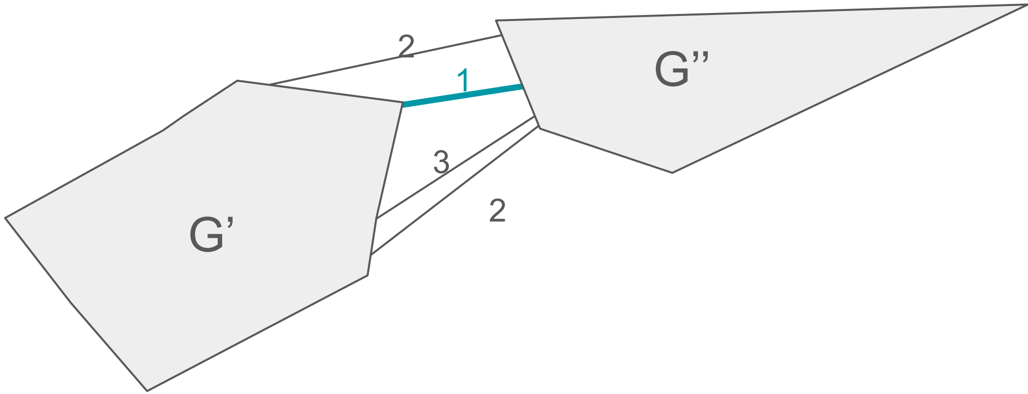
[What happens in the picture?]



- if an edge (u, v) is contained in some MST, then it is a light-edge crossing some cut of the graph.

Pf: AFtSoC e is not in a MST

[What happens in the picture?]

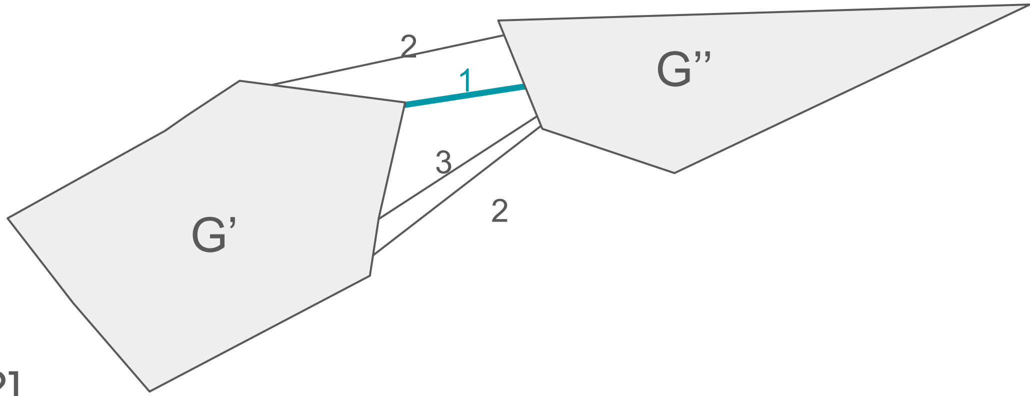


- if an edge (u, v) is contained in some MST, then it is a light-edge crossing some cut of the graph.

Pf: AFtSoC e is not in a MST

In an MST, G' and G'' must be connected.

[How can we get our contradiction?]



Question 1

(Minimum spanning trees)

1. An edge is called a **light-edge** crossing a cut $\mathcal{C} := (S, V - S)$, if its weight is the minimum of any edge crossing the cut. Show that:

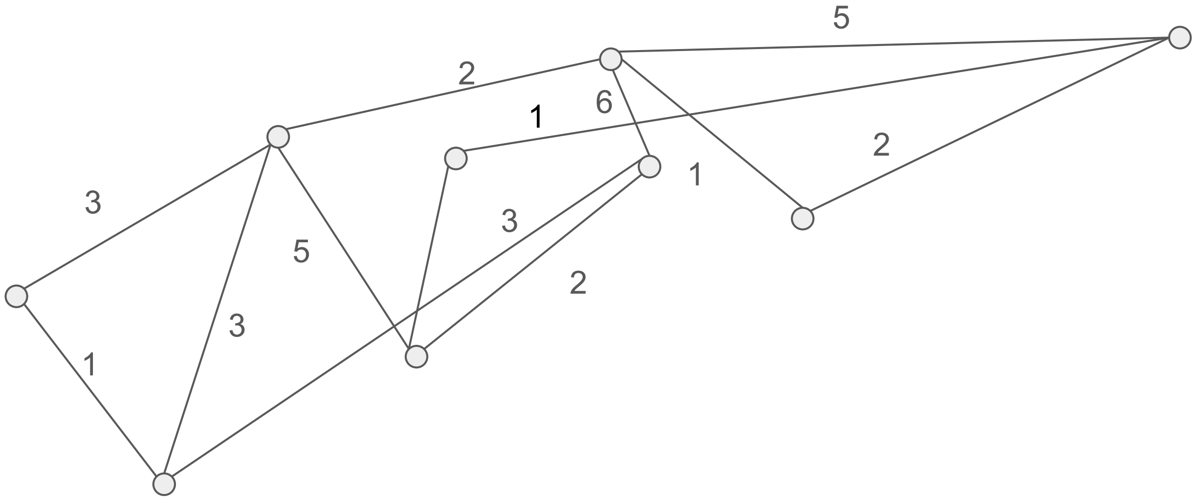
“If e is the light edge of some cut, then it is in *every* MST.”

Show that this is false.

2. Show that a graph has a unique MST, if for every cut of the graph, there is a unique light-edge crossing the cut. Show that the converse is not true by giving a counter-example.

Suppose each cut has a unique light edge. **WTS**: the graph has a unique MST

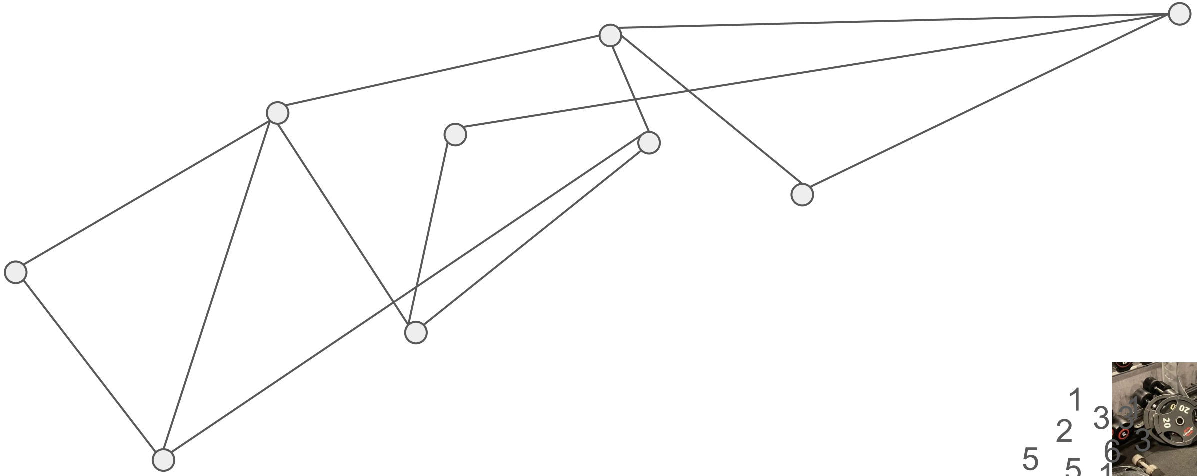
Proof by picture!



2. Show that a graph has a unique MST, if for every cut of the graph, there is a unique light-edge crossing the cut. Show that the converse is not true by giving a counter-example.

Suppose each cut has a unique light edge. **WTS**: the graph has a unique MST

Proof by picture!

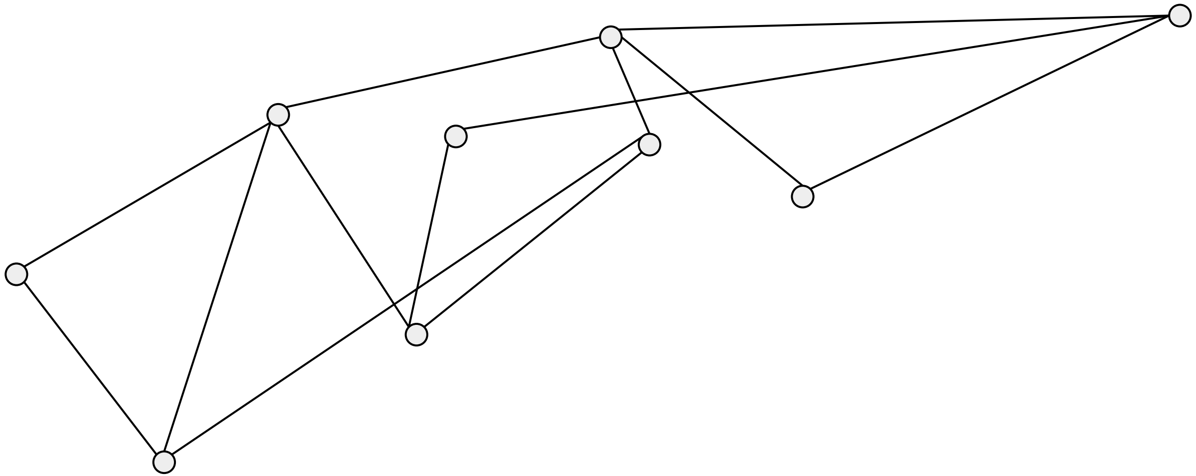


(Me and my bois have taken all the weights off the graph (we need them for our super set))

2. Show that a graph has a unique MST, if for every cut of the graph, there is a unique light-edge crossing the cut. Show that the converse is not true by giving a counter-example.

Suppose each cut has a unique light edge. **WTS**: the graph has a unique MST

Proof by picture!

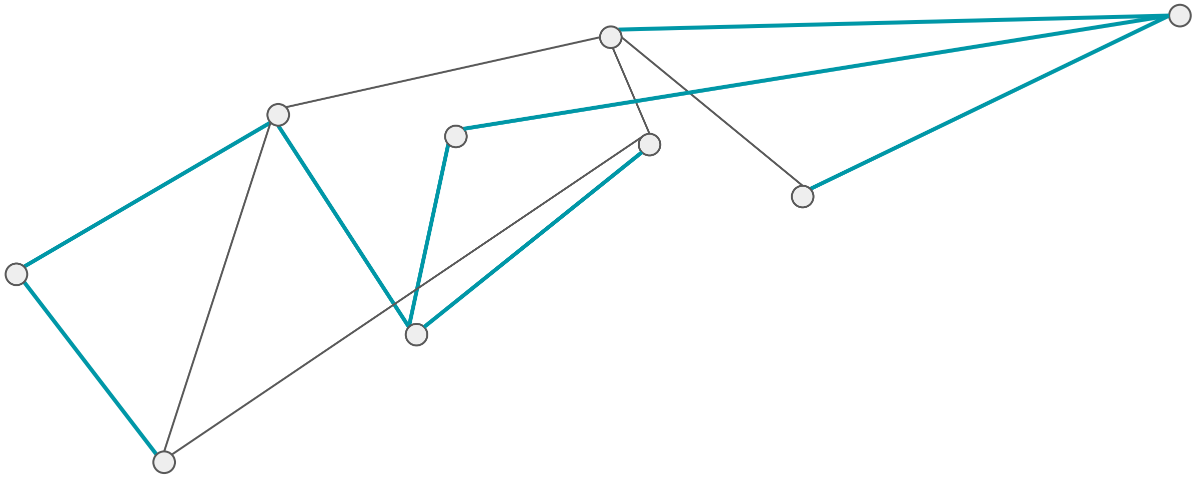


AFtSoC there are two different MSTs T_1 and T_2

2. Show that a graph has a unique MST, if for every cut of the graph, there is a unique light-edge crossing the cut. Show that the converse is not true by giving a counter-example.

Suppose each cut has a unique light edge. **WTS**: the graph has a unique MST

Proof by picture!

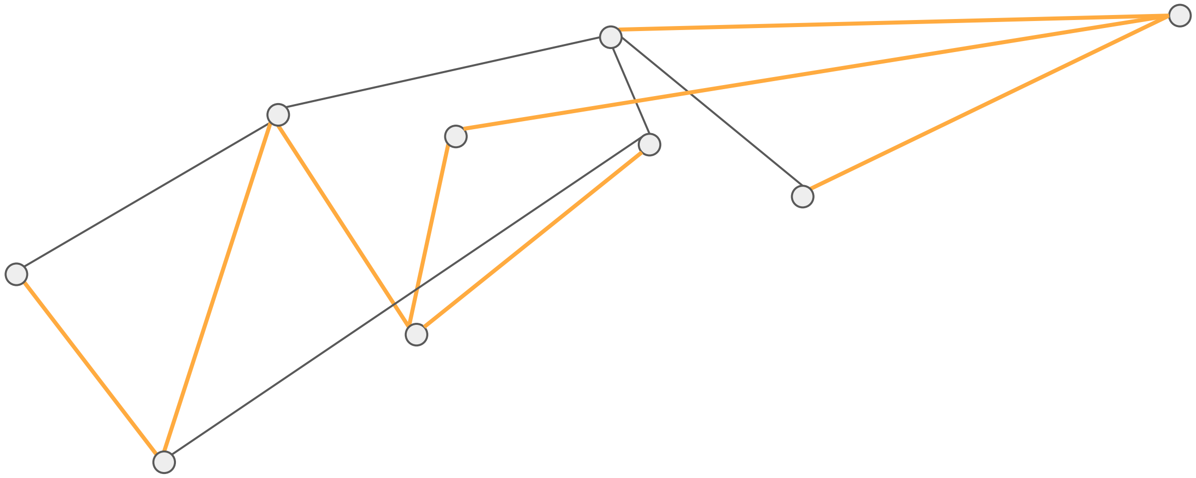


AFtSoC there are two different MSTs T_1 and T_2

2. Show that a graph has a unique MST, if for every cut of the graph, there is a unique light-edge crossing the cut. Show that the converse is not true by giving a counter-example.

Suppose each cut has a unique light edge. **WTS**: the graph has a unique MST

Proof by picture!

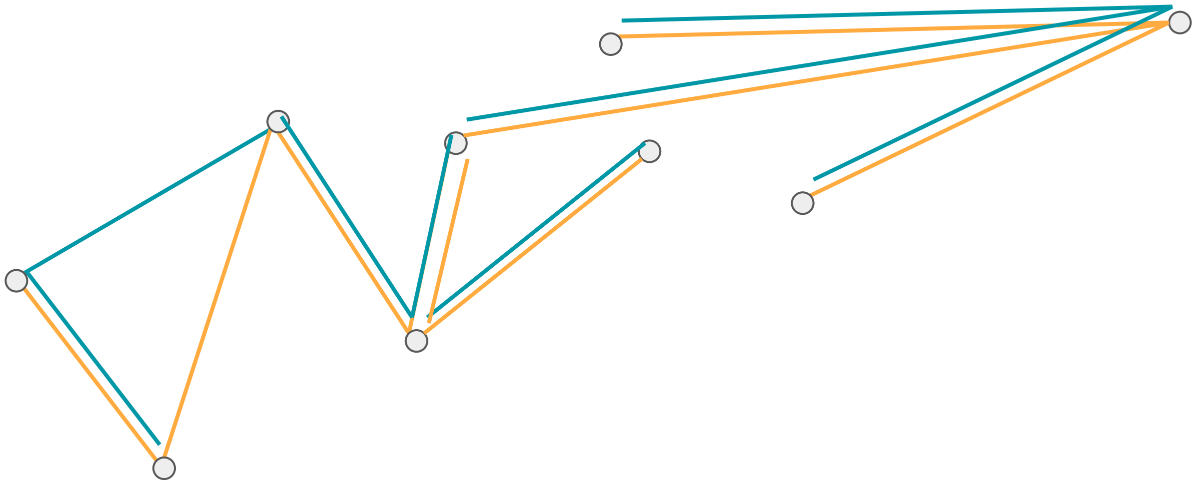


AFtSoC there are two different MSTs T_1 and T_2

2. Show that a graph has a unique MST, if for every cut of the graph, there is a unique light-edge crossing the cut. Show that the converse is not true by giving a counter-example.

Suppose each cut has a unique light edge. **WTS**: the graph has a unique MST

Proof by picture!

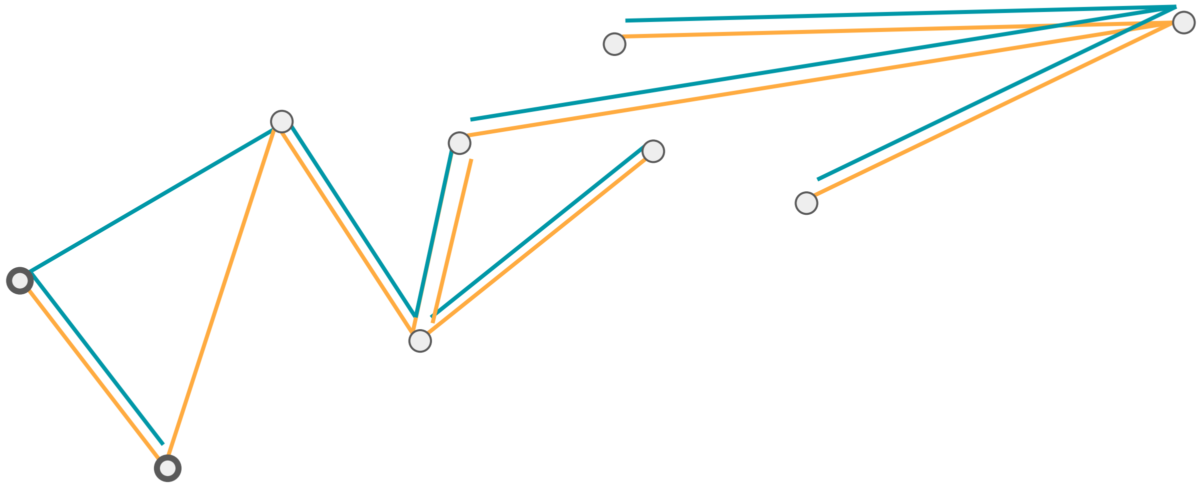


T_1 and T_2 differ on some edges e_1,e_2

2. Show that a graph has a unique MST, if for every cut of the graph, there is a unique light-edge crossing the cut. Show that the converse is not true by giving a counter-example.

Suppose each cut has a unique light edge. **WTS**: the graph has a unique MST

Proof by picture!

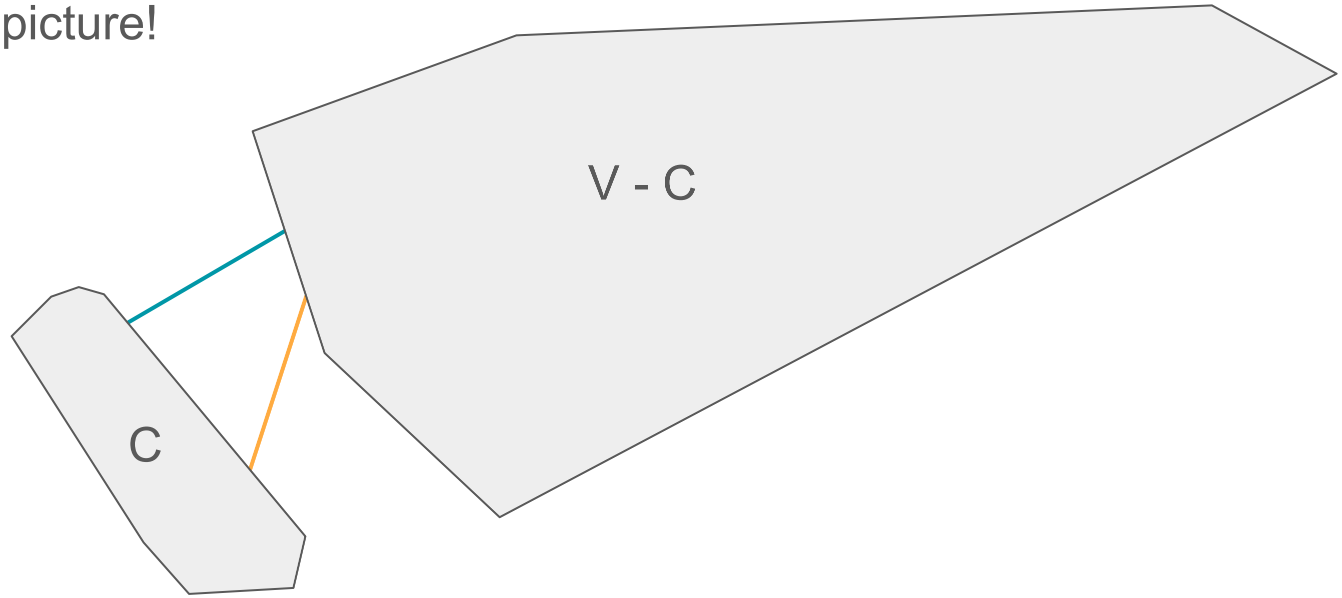


T_1 and T_2 differ on some edges e_1,e_2. Consider cut **C** defined above.

2. Show that a graph has a unique MST, if for every cut of the graph, there is a unique light-edge crossing the cut. Show that the converse is not true by giving a counter-example.

Suppose each cut has a unique light edge. **WTS**: the graph has a unique MST

Proof by picture!

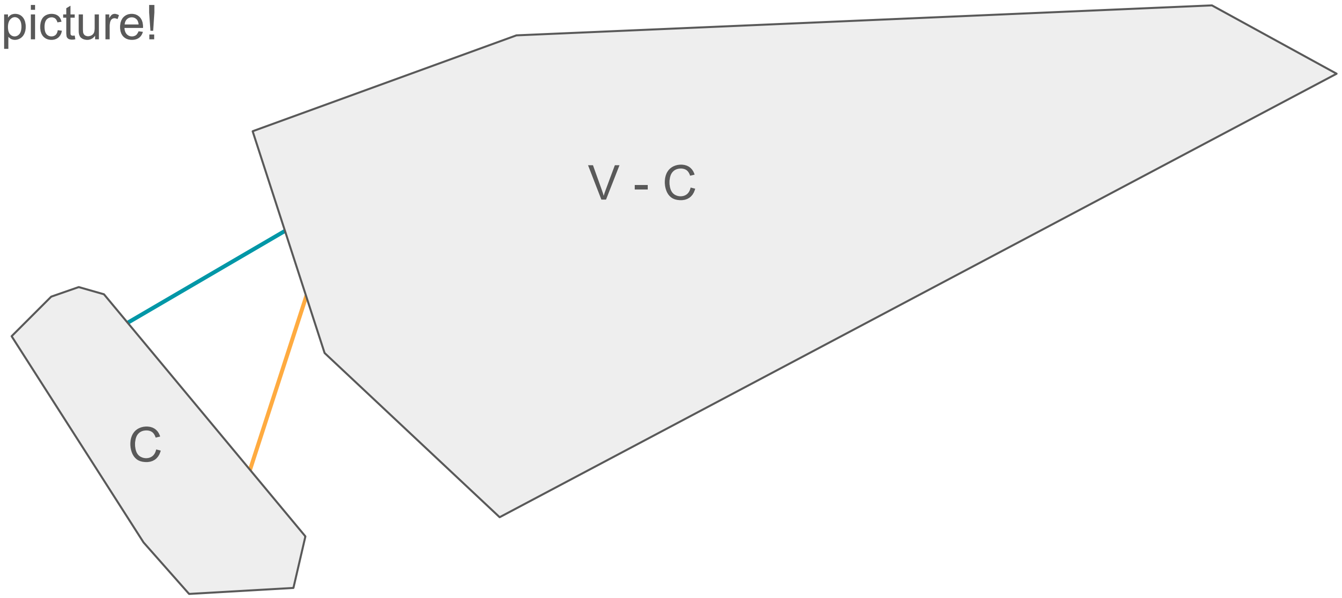


T_1 and T_2 differ on some edges e_1,e_2. Consider cut **C** defined above.

2. Show that a graph has a unique MST, if for every cut of the graph, there is a unique light-edge crossing the cut. Show that the converse is not true by giving a counter-example.

Suppose each cut has a unique light edge. **WTS**: the graph has a unique MST

Proof by picture!

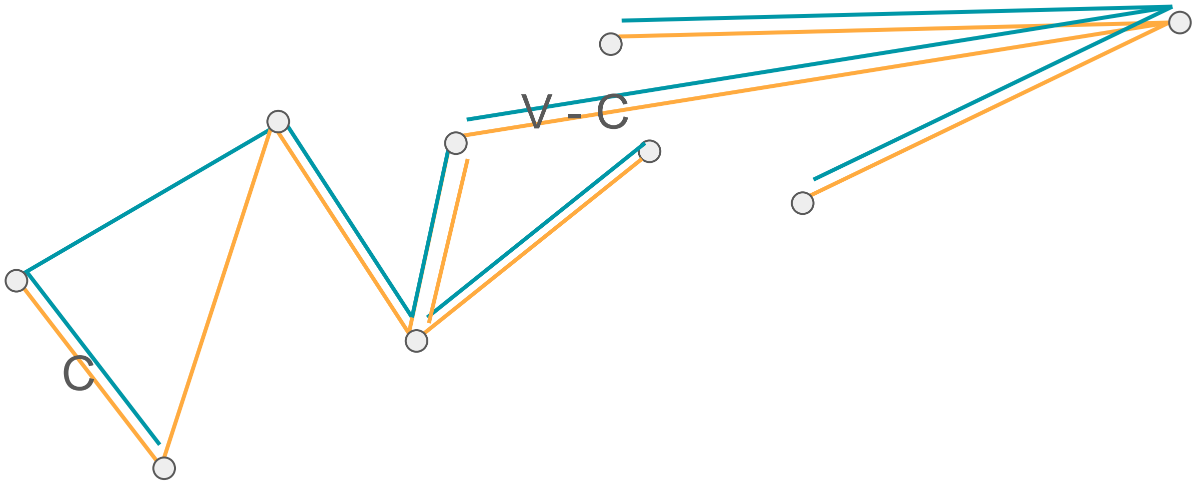


By our assumption, say e_1 is our unique light edge in cut C i.e., $wt(e_1) < wt(e_2)$

2. Show that a graph has a unique MST, if for every cut of the graph, there is a unique light-edge crossing the cut. Show that the converse is not true by giving a counter-example.

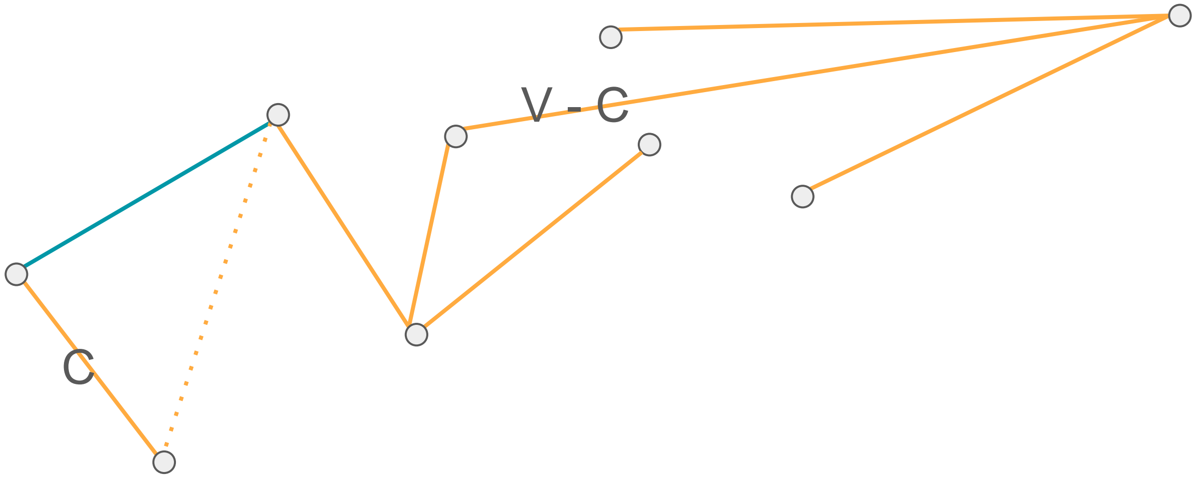
Suppose each cut has a unique light edge. **WTS**: the graph has a unique MST

Proof by picture!



But if wt(e_1) < wt(e_2), then we can lower the weight of MST T_2 by taking e_1 instead of e_2

2. Show that a graph has a unique MST, if for every cut of the graph, there is a unique light-edge crossing the cut. Show that the converse is not true by giving a counter-example.



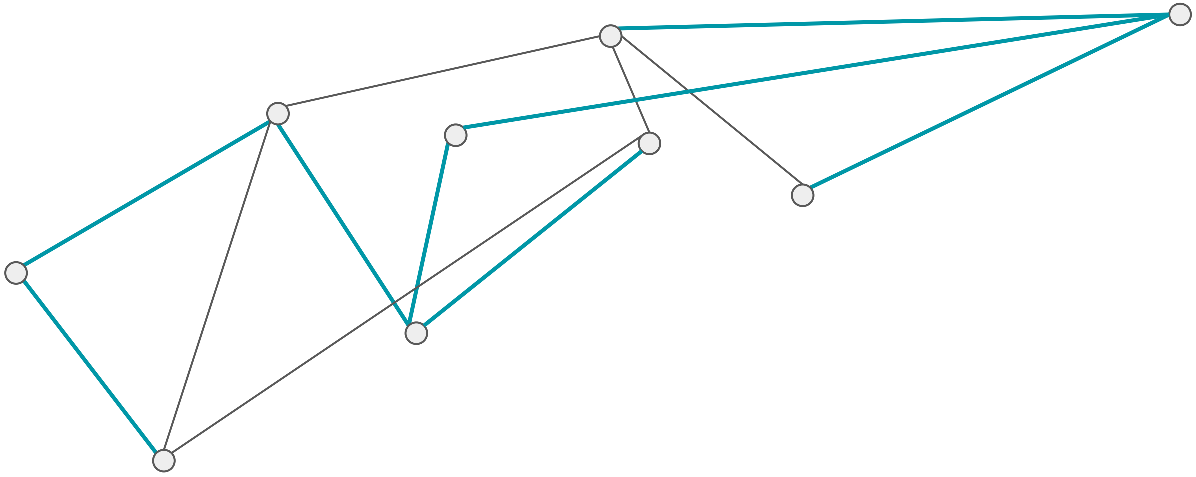
But if wt(e_1) < wt(e_2), then we can lower the weight of MST T_2 by taking e_1 instead of e_2

2. Show that a graph has a unique MST, if for every cut of the graph, there is a unique light-edge crossing the cut. Show that the converse is not true by giving a counter-example.

Time for the counter example

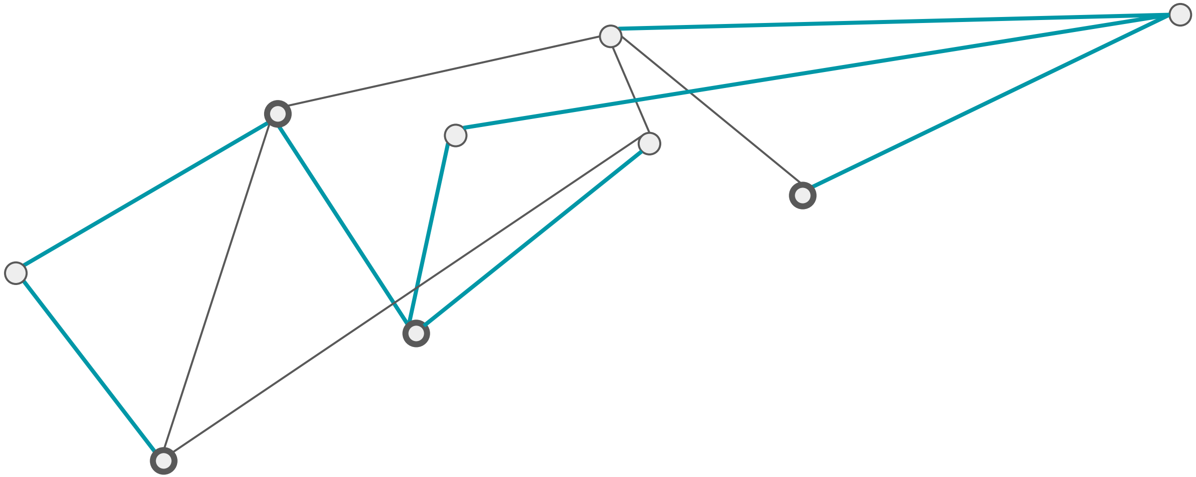
3. Let T be an MST of a graph $G = (V, E)$, and let V' be a subset of V. Let T' be the subgraph of T induced by V', and let G' be the subgraph of G induced by V'. Show that if T' is connected, then T' is an MST of G'.

Let this be the graph G and mst T



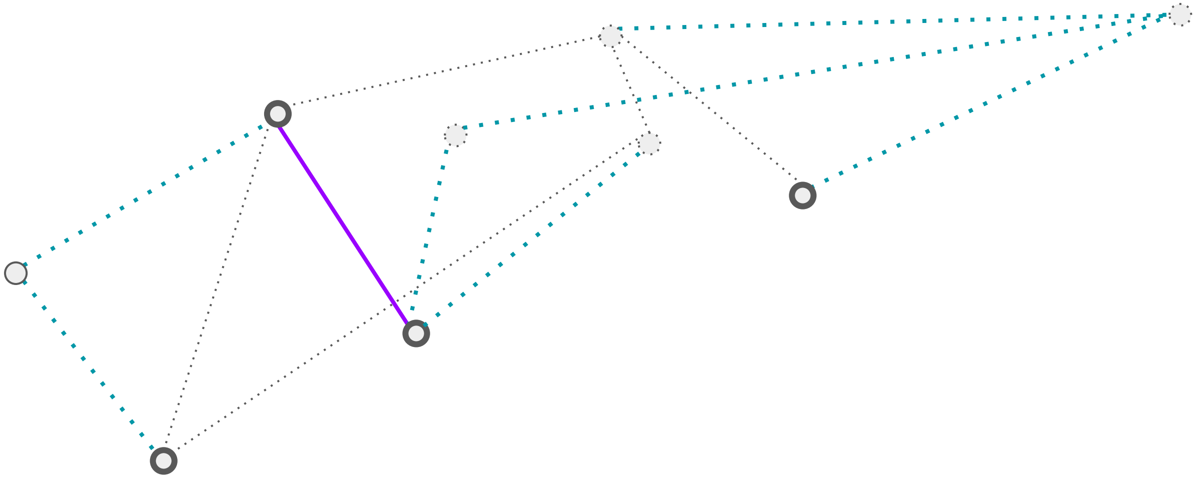
3. Let T be an MST of a graph $G = (V, E)$, and let V' be a subset of V. Let T' be the subgraph of T induced by V', and let G' be the subgraph of G induced by V'. Show that if T' is connected, then T' is an MST of G'.

Let this be the graph G and mst T



Suppose we define **V'** as follows

3. Let T be an MST of a graph $G = (V, E)$, and let V' be a subset of V. Let T' be the subgraph of T induced by V', and let G' be the subgraph of G induced by V'. Show that if T' is connected, then T' is an MST of G'.

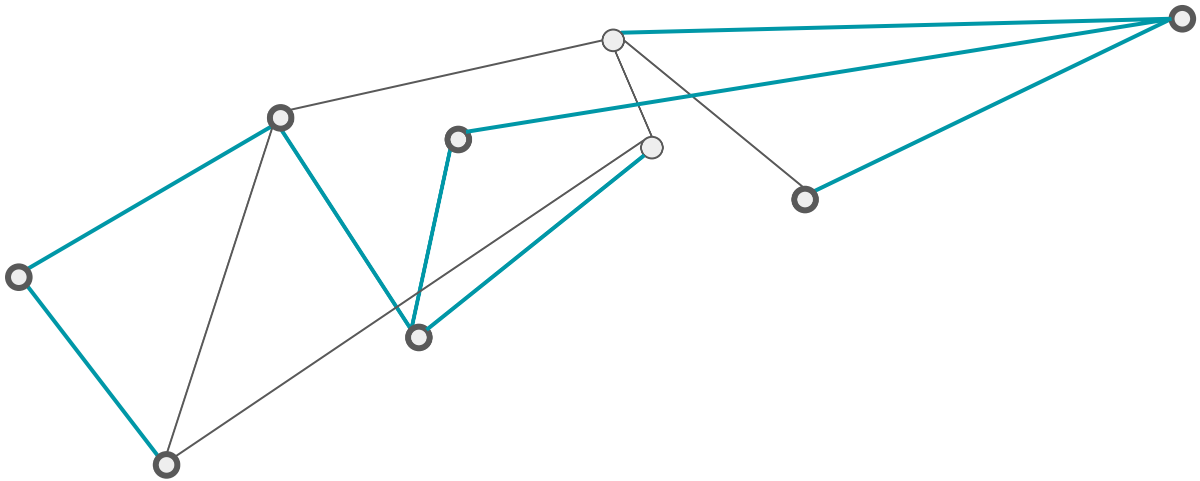


Suppose we define **V'** as follows. This is T', T induced by **V'**

What went wrong? Why isn't a T' MST?

3. Let T be an MST of a graph $G = (V, E)$, and let V' be a subset of V. Let T' be the subgraph of T induced by V', and let G' be the subgraph of G induced by V'. Show that if T' is connected, then T' is an MST of G'.

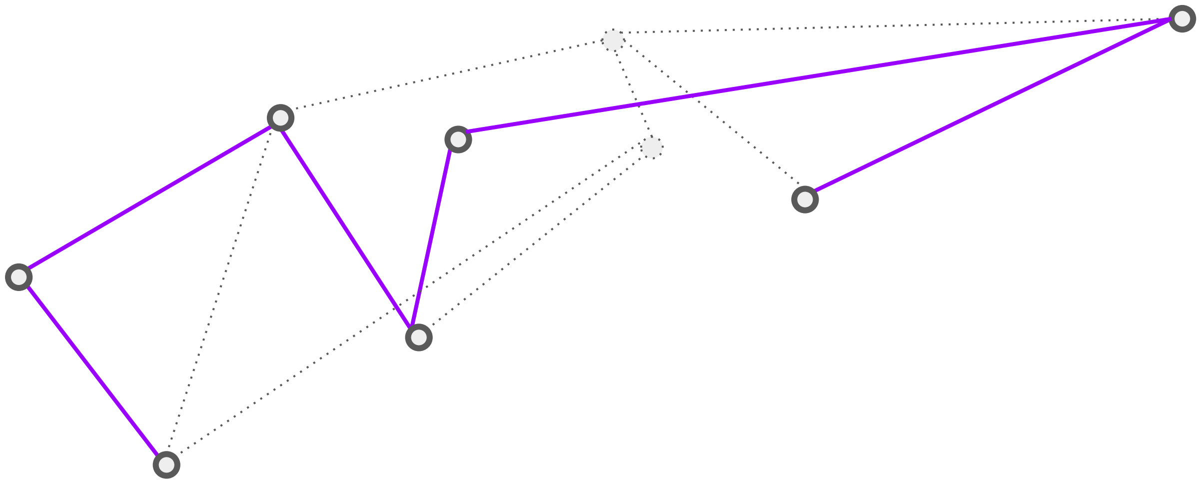
Let this be the graph G and mst T



Suppose we define **V'** as follows

3. Let T be an MST of a graph $G = (V, E)$, and let V' be a subset of V. Let T' be the subgraph of T induced by V', and let G' be the subgraph of G induced by V'. Show that if T' is connected, then T' is an MST of G'.

Let this be the graph G and mst T

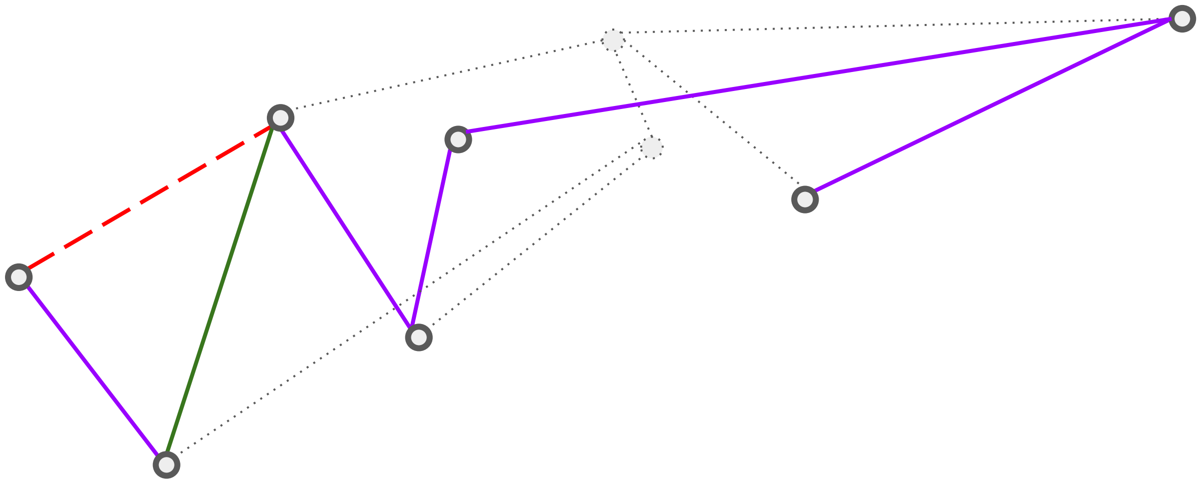


Suppose we define **V'** as follows. This is T', T induced by **V'**

WTS: this is an MST of **V'**

3. Let T be an MST of a graph $G = (V, E)$, and let V' be a subset of V. Let T' be the subgraph of T induced by V', and let G' be the subgraph of G induced by V'. Show that if T' is connected, then T' is an MST of G'.

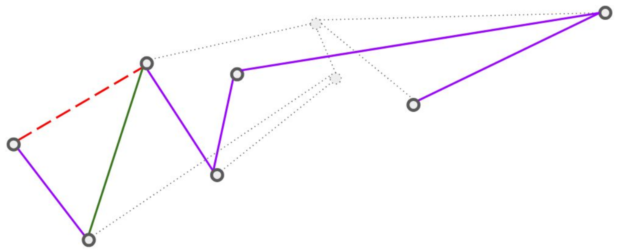
Let this be the graph G and mst T



WTS: this is an MST of **V'**

AFtSoC there is a cheaper tree T'' differing in edges above (added , removed)

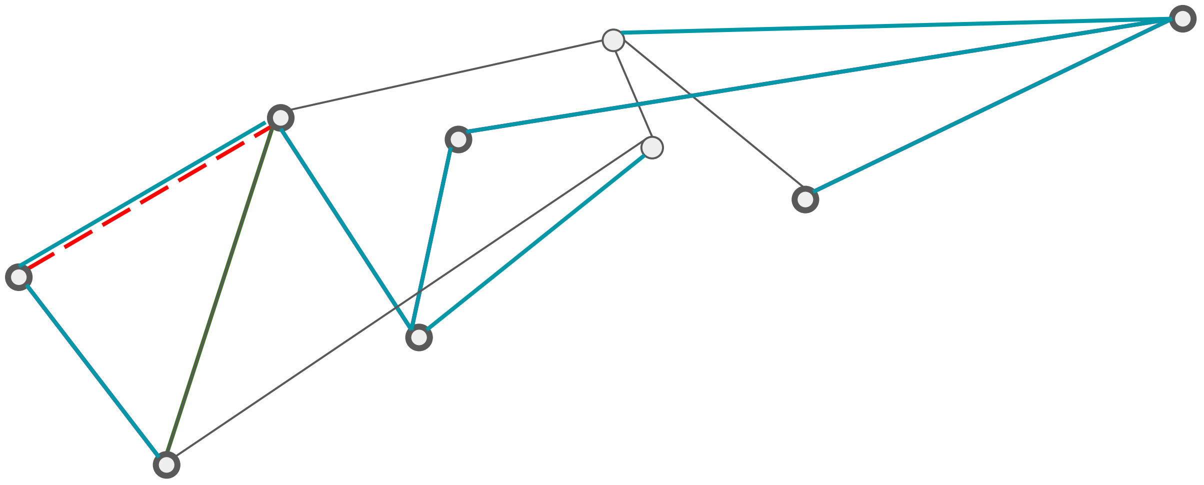
3. Let T be an MST of a graph $G = (V, E)$, and let V' be
induced by V', and let G' be the subgraph of G induced l
is an MST of G'.



WTS: this is an MST of **V'**

AFtSoC there is a cheaper tree T'' differing in edges above (added , removed)

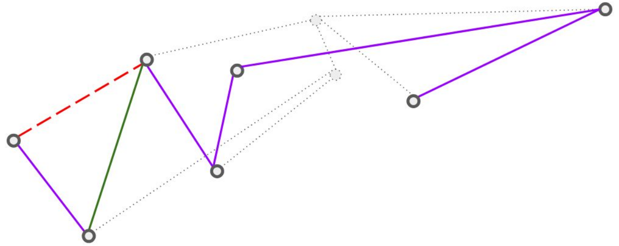
Let this be the graph G and mst T



WTS: this is an MST of **V'**

Back in the original graph we originally had MST T

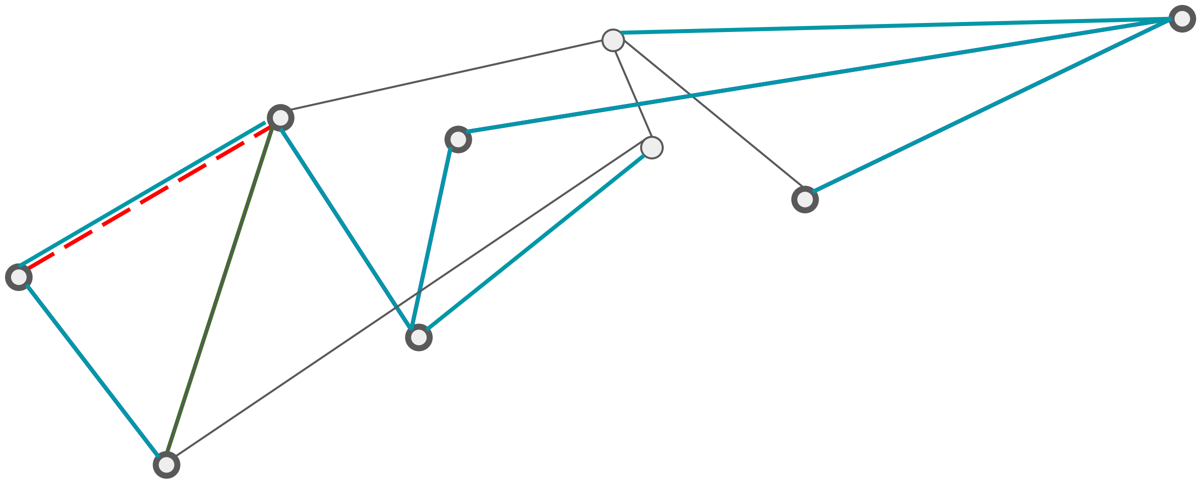
3. Let T be an MST of a graph $G = (V, E)$, and let V' be
induced by V', and let G' be the subgraph of G induced l
is an MST of G'.



WTS: this is an MST of **V’**

AFtSoC there is a cheaper tree T’’ differing in edges above (added , removed)

Let this be the graph G and mst T



WTS: this is an MST of **V’**

Removing the red edge and adding the green edge gives us a cheaper tree

Question 2

(Prim's & Kruskal's algorithm)

1. Suppose that we represent the graph $G = (V, E)$ as an adjacency-matrix. Give a simple implementation of Prim's algorithm for this case that runs in $O(|V|^2)$ time.

2. Suppose that all edge weights in a graph are integers in the range from 1 to $|V|$. How fast can you make Kruskal's algorithm run?

Simple Intuition of Prim's algorithm?

Question 2

(Prim's & Kruskal's algorithm)

1. Suppose that we represent the graph $G = (V, E)$ as an adjacency-matrix. Give a simple implementation of Prim's algorithm for this case that runs in $O(|V|^2)$ time.

Dijkstra

```
algorithm DijkstraShortestPath(G(V,E), s ∈ V)

   let dist:V → ℤ
   let prev:V → V
   let Q be an empty priority queue

   dist[s] ← 0
   for each v ∈ V do
      if v ≠ s then
         dist[v] ← ∞
      end if
      prev[v] ← -1
      Q.add(dist[v], v)
   end for

   while Q is not empty do
      u ← Q.getMin()
      for each w ∈ V adjacent to u still in Q do
         d ← dist[u] + weight(u, w)
         if d < dist[w] then
            dist[w] ← d
            prev[w] ← u
            Q.set(d, w)
         end if
      end for
   end while

   return dist, prev
end algorithm
```

Prim's

Prim's MST

```
algorithm ~~DijkstraShortestPath~~(G(V,E), s ∈ V)

   let dist:V → ℤ
   let prev:V → V
   let Q be an empty priority queue

   dist[s] ← 0
   for each v ∈ V do
      if v ≠ s then
         dist[v] ← ∞
      end if
      prev[v] ← -1
      Q.add(dist[v], v)
   end for

   while Q is not empty do
      u ← Q.getMin()
      for each w ∈ V adjacent to u still in Q do
         d ← ~~dist[u] +~~ weight(u, w)
         if d < dist[w] then
            dist[w] ← d
            prev[w] ← u
            Q.set(d, w)
         end if
      end for
   end while

   return dist, prev
end algorithm
```

Question 2

(Prim's & Kruskal's algorithm)

1. Suppose that we represent the graph $G = (V, E)$ as an adjacency-matrix. Give a simple implementation of Prim's algorithm for this case that runs in $O(|V|^2)$ time.

Prim's MST

```
algorithm ~~DijkstraShortestPath~~(G(V,E), s ∈ V)

   let dist:V → ℤ
   let prev:V → V
   let Q be an empty priority queue

   dist[s] ← 0
   for each v ∈ V do
      if v ≠ s then
         dist[v] ← ∞
      end if
      prev[v] ← -1
      Q.add(dist[v], v)
   end for

   while Q is not empty do
      u ← Q.getMin()
      for each w ∈ V adjacent to u still in Q do
         d ← ~~dist[u] +~~ weight(u, w)
         if d < dist[w] then
            dist[w] ← d
            prev[w] ← u
            Q.set(d, w)
         end if
      end for
   end while

   return dist, prev
end algorithm
```

Pseudocode

//Initialize prev, dist

Let dist[v] = current min. edge to v

while pq is not empty:

  Vertex u <- pq.pop()

  for each edge (u,v):

    if wt(u,v) < dist[v]:

      update dist and pq

What we can do with an adj matrix

Question 2

(Prim's & Kruskal's algorithm)

1. Suppose that we represent the graph $G = (V, E)$ as an adjacency-matrix. Give a simple implementation of Prim's algorithm for this case that runs in $O(|V|^2)$ time.

Prim's MST

```
algorithm ~~DijkstraShortestPath~~(G(V,E), s ∈ V)

   let dist:V → ℤ
   let prev:V → V
   let Q be an empty priority queue

   dist[s] ← 0
   for each v ∈ V do
      if v ≠ s then
         dist[v] ← ∞
      end if
      prev[v] ← -1
      Q.add(dist[v], v)
   end for

   while Q is not empty do
      u ← Q.getMin()
      for each w ∈ V adjacent to u still in Q do
         d ← ~~dist[u] +~~ weight(u, w)
         if d < dist[w] then
            dist[w] ← d
            prev[w] ← u
            Q.set(d, w)
         end if
      end for
   end while

   return dist, prev
end algorithm
```

Pseudocode

//Initialize prev, dist

Let dist[v] = current min. edge to v

while pq is not empty:

    Vertex u <- pq.pop()

    for each edge (u,v):

        if wt(u,v) < dist[v]:

            update dist and pq

What we can do with an adj matrix

Question 2

(Prim's & Kruskal's algorithm)

1. Suppose that we represent the graph $G = (V, E)$ as an adjacency-matrix. Give a simple implementation of Prim's algorithm for this case that runs in $O(|V|^2)$ time.

Prim's MST

```
algorithm ~~DijkstraShortestPath~~(G(V,E), s ∈ V)

   let dist:V → ℤ
   let prev:V → V
   let Q be an empty priority queue

   dist[s] ← 0
   for each v ∈ V do
      if v ≠ s then
         dist[v] ← ∞
      end if
      prev[v] ← -1
      Q.add(dist[v], v)
   end for

   while Q is not empty do
      u ← Q.getMin()
      for each w ∈ V adjacent to u still in Q do
         d ← ~~dist[u] +~~ weight(u, w)
         if d < dist[w] then
            dist[w] ← d
            prev[w] ← u
            Q.set(d, w)
         end if
      end for
   end while

   return dist, prev
end algorithm
```

Pseudocode

//Initialize prev, dist

Let dist[v] = current min. edge to v

while pq is not empty:

    Vertex u <- pq.pop()

    for each edge (u,v):

        if wt(u,v) < dist[v]:

            update dist and pq

What we can do with an adj matrix

What we cannot do (right away)

Question 2

(Prim's & Kruskal's algorithm)

1. Suppose that we represent the graph $G = (V, E)$ as an adjacency-matrix. Give a simple implementation of Prim's algorithm for this case that runs in $O(|V|^2)$ time.

```
//Initialize prev, dist

Let dist[v] = current min. edge to v
    while pq is not empty:

        Vertex u &lt;- pq.pop():

        for each edge (u,v):
            if wt(u,v) < dist[v]:

                update dist and pq
```

```
Prims(G,start):
  //Initialize prev, dist

  Let T = {start}

  ________________:

        ____________________

        ____________________

        ____________________:
          if wt((_,_)) < dist[_]:

              ____________________

              ____________________
```

2. Suppose that all edge weights in a graph are integers in the range from 1 to $|V|$. How fast can you make Kruskal's algorithm run?

Kruskal

- Sort edges by increasing order of their weights // O(?) time
- Run a Union Finding procedure // ~O(|E|) time

The **values** of the edges are bounded by |V|. What's a good sorting algorithm for this?

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	a	a	a	a	a	a
P	b	a	a	a	a	a			

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

<u>Boyer-Moore</u>: Iteratively compare pattern P with target, going backward

T	a	a	a	a	a	**a**	a	a	a
P	b	a	a	a	a	**a**			

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa\cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	a	**a**	a	a	a	a
P	b	a	a	a	**a**	a			

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa\cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	**a**	a	a	a	a	a
P	b	a	a	**a**	a	a			

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa\cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

<u>Boyer-Moore</u>: Iteratively compare pattern P with target, going backward

T	a	a	**a**	a	a	a	a	a	a
P	b	a	**a**	a	a	a			

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	**a**	a	a	a	a	a	a	a
P	b	**a**	a	a	a	a			

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa\cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	**a**	a	a	a	a	a	a	a	a
P	**b**	a	a	a	a	a			

T[0] does not equal P[0]! Next steps..

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	a	a	a	a	a	a
P	b	a	a	a	a	a			

T[0] does not equal P[0]! Next steps.. We mismatched on target a

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa\cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	a	a	a	a	a	a
P	b	a	a	a	a	a			

T[0] does not equal P[0]! Next steps.. We mismatched on target a

The last occurrence of pattern a

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	a	a	a	a	a	a
P	b	a	a	a	a	a			

Move P (to align target a with pattern a) OR (one after target mismatch)

Whichever moves P the *least* amount – in this ex. We move one after mismatch

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa\cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	a	a	a	a	a	a
P		b	a	a	a	a	a		

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	a	a	a	**a**	a	a
P		b	a	a	a	a	**a**		

Fast forward..

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	**a**	a	a	a	a	a	a	a
P		**b**	a	a	a	a	a		

Fast forward.. Same mismatch, jump 1

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	a	a	a	a	a	a
P			b	a	a	a	a	a	

Same thing will happen 1 more time

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

<u>Boyer-Moore</u>: Iteratively compare pattern P with target, going backward

T	a	a	a	a	a	a	a	a	a
P				b	a	a	a	a	a

Total compares:

Same thing will happen 1 more time, and conclude no match

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	a	a	a	**a**	a	a
P		b	a	a	a	a	**a**		

Fast forward..

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	**a**	a	a	a	a	a	a	a
P		**b**	a	a	a	a	a		

Fast forward.. Same mismatch, jump 1

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aaa \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	a	a	a	a	a	a
P			b	a	a	a	a	a	

Same thing will happen 1 more time

Question 3

(Backward pattern matching)

The Boyer-Moore algorithm is based upon backward pattern matching. Let us do a simple review via the following questions:

1. Run Boyer-Moore algorithm in the following worst-case scenario:

$$T := \underbrace{aa a \cdots a}_{9} \quad \text{and} \quad P := baaaaa.$$

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	a	a	a	a	a	a	a	a	a
P				b	a	a	a	a	a

Total compares:

Same thing will happen 1 more time, and conclude no match

A good Boyer-Moore Example

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	o	o	x	**x**	x	x	o	o	o
P	x	x	x	**x**					

A good Boyer-Moore Example

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	o	o	**x**	x	x	x	o	o	o
P	x	x	**x**	x					

A good Boyer-Moore Example

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	o	**o**	x	x	x	x	o	o	o
P	x	**x**	x	x					

Mismatch!

Move P (to align target o with pattern o) OR (one after target mismatch)

Whichever moves P the *least* amount

A good Boyer-Moore Example

Boyer-Moore: Iteratively compare pattern P with target, going backward

T	o	**o**	x	x	x	x	o	o	o
P			x	**x**	x	x			

Mismatch!

Move P (to align target o with pattern o) OR (**one after target mismatch**)

Whichever moves P the *least* amount (Since no o in pattern, latter case**)**