Minimum Spanning Tree

https://cs.pomona.edu/classes/cs140/

Outline

Topics and Learning Objectives

- Discuss spanning tree and minimum spanning trees (MSTs)
- Introduce Prim's algorithms for MSTs
- Prove correctness of Prim's MST Algorithm

Exercise

- MST exercise questions 1 and 2

Extra Resources

- Introduction to Algorithms, 3rd, chapter 23
- Algorithms Illuminated Part 3, Chapter 15

Minimum Spanning Tree

Given a graph, connect all points together as ***cheaply*** *as possible.*

Why are we talking about this?

- It is a fundamental graph problem,
- It has several greedy-based solutions,
- And it has many applications:
  - Clustering
  - Networking
  - Many more

Greedy Solution

Bernard Chazelle (1995) developed a non-greedy algorithm that runs in $O(m\ \alpha(m,n))$.

- Otakar Borůvka in 1926
- Vojtěch Jarník in 1930
  - Rediscovered by Robert Prim in 1957
  - Rediscovered by Edsger Dijkstra in 1959
- Joseph Kruskal in 1956

Blazingly fast algorithm for what you get as output:

- Can run in $O(m \lg n)$
- Remember: it takes $O(n + m)$ just to read the graph!
- There are an exponential number of possible spanning trees

Minimum Spanning Tree

Input: a weighted, undirected graph $G = (V, E)$

- A similar problem can be constructed for directed graphs, and it is then called the optimal branching problem
- Each edge e has a cost c_e
- **Costs can be negative**

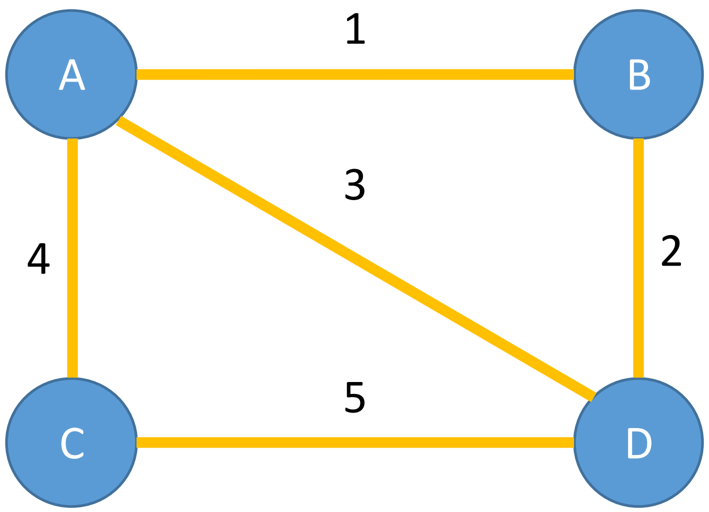
Output: the minimum cost tree T that spans all vertices

- Calculate cost as the sum of all edge costs
- What does it mean to **span** a graph?
- The tree T is just a subset of E

Spanning Tree Properties

1. The spanning tree T does not have any cycles
2. The subgraph (V, T) is connected

What is a spanning tree for this graph?

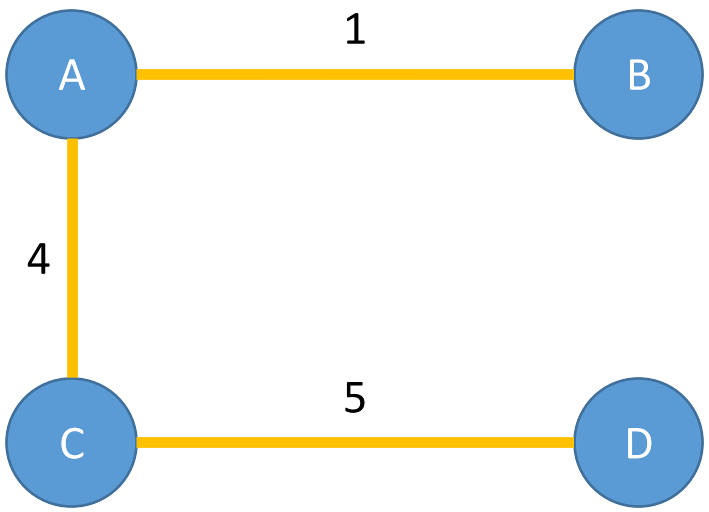


Spanning Tree Properties

1. The spanning tree T does not have any cycles
2. The subgraph (V, T) is connected

What is a spanning tree for this graph?

This is **not** the **minimum** spanning tree



Our MST Problem Assumptions

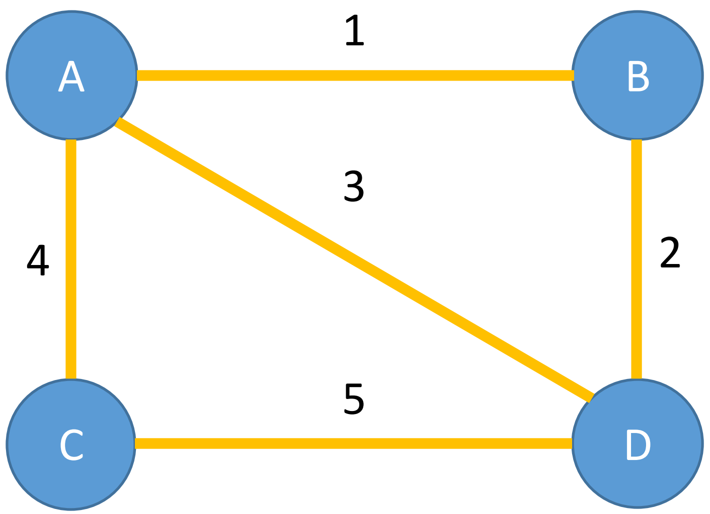
1. The input graph is connected
   - This is easy to check. How?
   - Otherwise we're looking at the minimum spanning forest problem

2. Edge costs are distinct
   - All mentioned algorithms are correct with ties, but
   - It makes our correctness proof much easier if we assume no ties

Prim's Algorithm (aka Jarník's or Dijkstra's)

- A greedy algorithm that finds an MST for a weighted, undirected graph.
- It finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized.

What is a good criteria for finding the minimum spanning tree?

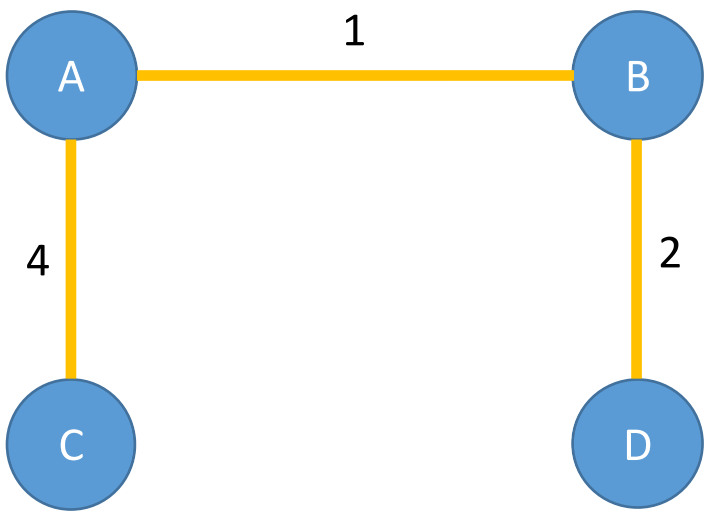


What is the minimum spanning tree for this graph?

Prim's Algorithm

- A greedy algorithm that finds an MST for a weighted, undirected graph.
- It finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized.

What is a good criteria for finding the minimum spanning tree?



What is the minimum spanning tree for this graph?

```
FUNCTION Prims(G, start_vertex)
   found = {start_vertex}
   mst = {}
   mst_cost = 0

   WHILE found.size != G.vertices.size

      min_weight, min_edge = INFINITY, NONE
      FOR v IN found
         FOR vOther, weight IN G.edges[v]
            IF vOther NOT IN found
               IF weight < min_weight
                  min_weight = weight
                  min_edge = (v, vOther)

      found.add(min_edge[1])
      mst.add(min_edge)
      mst_cost = mst_cost + min_weight

   RETURN mst, mst_cost
```

```
FUNCTION Prims(G, start_vertex)
    found = {start_vertex}
    mst = {}
    mst_cost = 0

    WHILE found.size != G.vertices.size

        min_weight, min_edge = INFINITY, NONE
        FOR v IN found
            FOR vOther, weight IN G.edges[v]
                IF vOther NOT IN found
                    IF weight < min_weight
                        min_weight = weight
                        min_edge = (v, vOther)

        found.add(min_edge[1])
        mst.add(min_edge)
        mst_cost = mst_cost + min_weight

    RETURN mst, mst_cost
```

How does this compare with Dijkstra's Algorithm?

Each iteration:

Extend MST in cheapest manner possible

Proof of Prim's

Theorem: *Prim's algorithm always computes the (or a) MST when given a connected graph.*

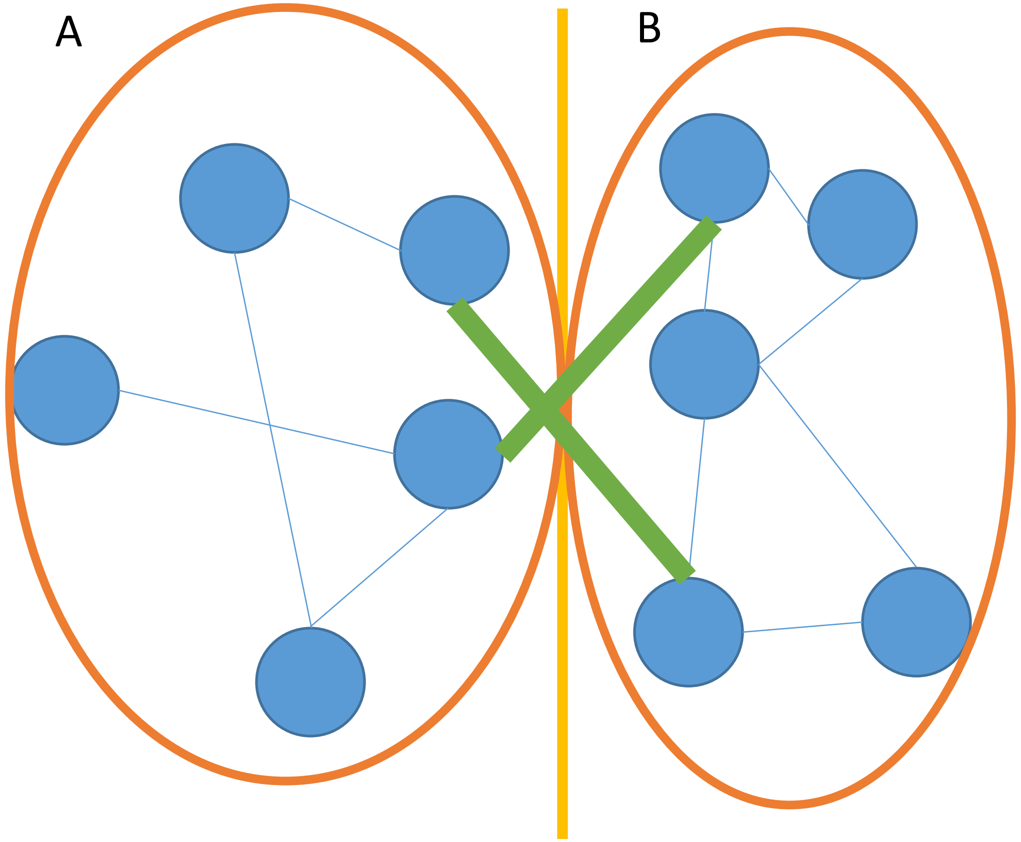
Need to prove two things:

1. That Prim's algorithm creates a spanning tree T*
2. And that T* is the **minimum** spanning tree

We need to define a few things before we conduct the proof

Graph Cuts

A *cut* *of any graph G = (V, E) is a partition of V into two non-empty groups*

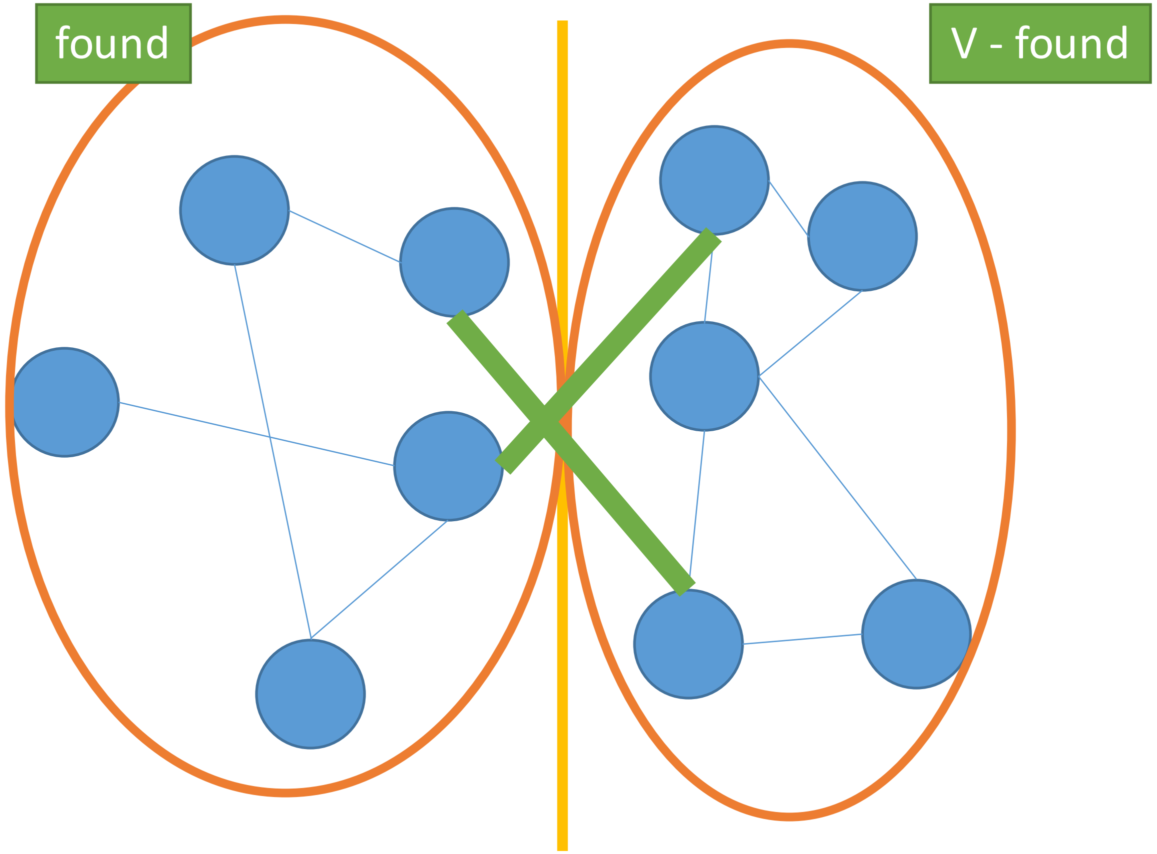


For a graph with n vertices, how many possible cuts are there?

a. O(n)
b. $O(n^2)$
c. $O(2^n)$
d. $O(n^n)$

Graph Cuts

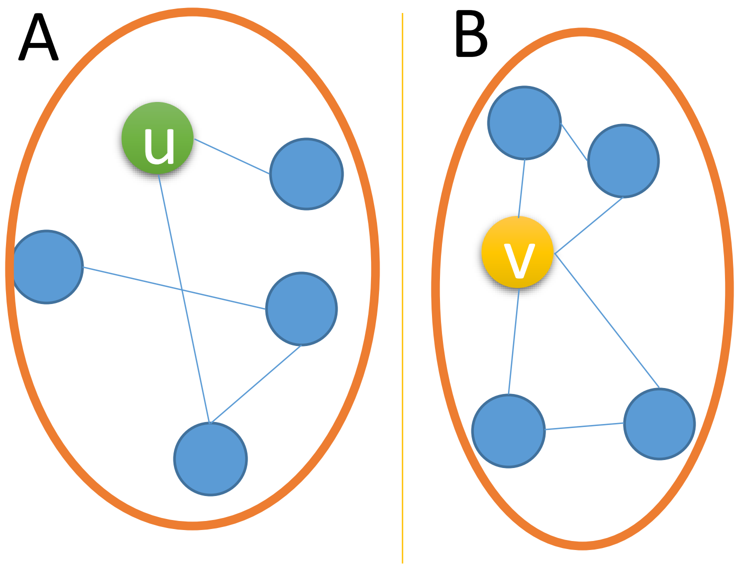
A cut of any graph G = (V, E) is a partition of V into two non-empty groups



For a graph with n vertices, how many possible cuts are there?

a. $O(n)$
b. $O(n^2)$
c. $O(2^n)$
d. $O(n^n)$

Lemma 1: Empty Cuts



*<u>Empty Cut Lemma</u>: a graph is **not connected** if there exists a cut (A, B) with zero crossing edges.*

<u>Proof A:</u>

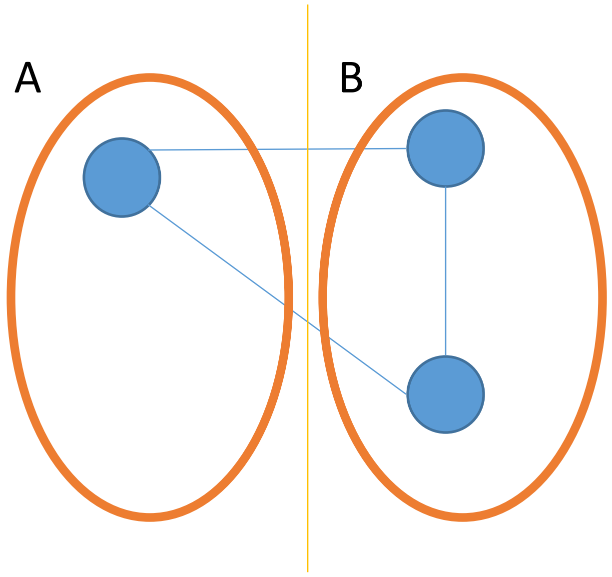
- Assume we have a cut with zero crossing edges
- Pick any u in A and v in B
- There is no path from u to v
- Thus the graph is not connected

<u>Proof B:</u>

- Assume the graph is not connected
- Suppose G has no path from u to v
- Put all vertices reachable from u into A
- Put all other vertices in B
- Thus, no edges cross the cut

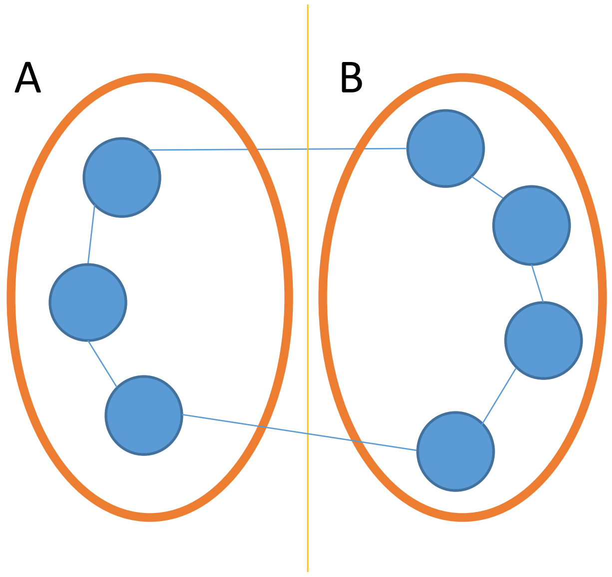
Lemma 2: Double-Crossings

<u>double-crossing Lemma</u>: suppose the cycle C has an edge crossing the cut (A, B). Then, there must be at least one more edge in C that crosses the cut.



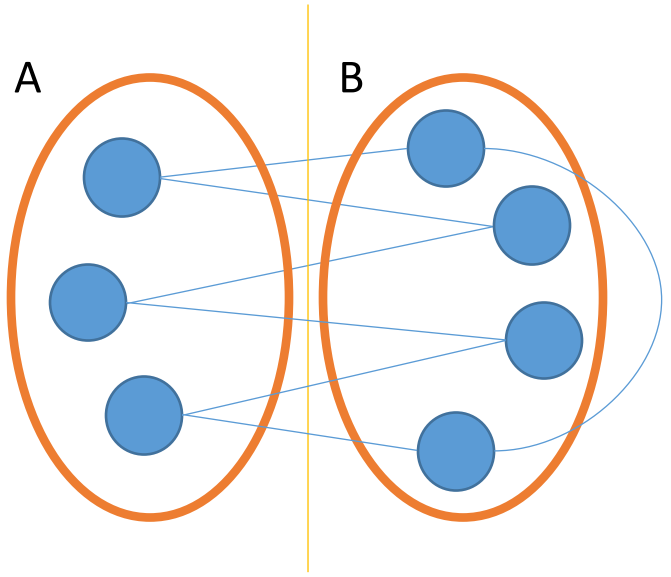
Lemma 2: Double-Crossings

double-crossing Lemma: suppose the cycle C has an edge crossing the cut (A, B). Then, there must be at least one more edge in C that crosses the cut.



Lemma 2: Double-Crossings

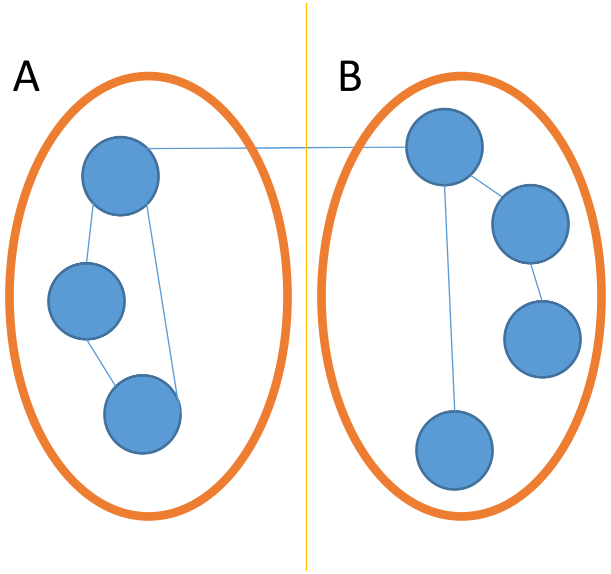
<u>double-crossing Lemma</u>: suppose the cycle C has an edge crossing the cut (A, B). Then, there must be at least one more edge in C that crosses the cut.



Lemma 2: Double-Crossings

double-crossing Lemma: suppose the cycle C has an edge crossing the cut (A, B). Then, there must be at least one more edge in C that crosses the cut.

No Cycle Corollary: if e is the only edge crossing some cut (A, B), then it is not in any cycle.



Proof of Prim’s

Theorem: *Prim’s algorithm always computes the (or a) MST when given a connected graph.*

Need to prove two things:

1. That Prim’s algorithm creates a spanning tree T*
2. And that T* is the **minimum** spanning tree

We’ll use graph cuts, the double-crossing lemma, and the no-cycle lemma in this proof.

Proof of Prim's

Theorem: *Prim's algorithm always computes the (or a) MST when given a connected graph.*

Need to prove two things:

1. That Prim's algorithm creates a spanning tree T*
2. And that T* is the **minimum** spanning tree

We'll use graph cuts, the double-crossing lemma, and the no-cycle lemma in this proof.

Claim 1: Prim's outputs a spanning tree

1. Prim's algorithm maintains the invariant that mst <u>spans</u> found

```
FUNCTION Prims(G, start_vertex)
   found = {start_vertex}
   mst = {}
   mst_cost = 0
   WHILE found.size != G.vertices.size
      min_weight, min_edge = INFINITY, NONE
      FOR v IN found
         FOR vOther, weight IN G.edges[v]
            IF vOther NOT IN found
               IF weight < min_weight
                  min_weight = weight
                  min_edge = (v, vOther)
      found.add(min_edge[1])
      mst.add(min_edge)
      mst_cost = mst_cost + min_weight
   RETURN mst, mst_cost
```

Claim 1: Prim's outputs a spanning tree

1. Prim's algorithm maintains the invariant that T spans X

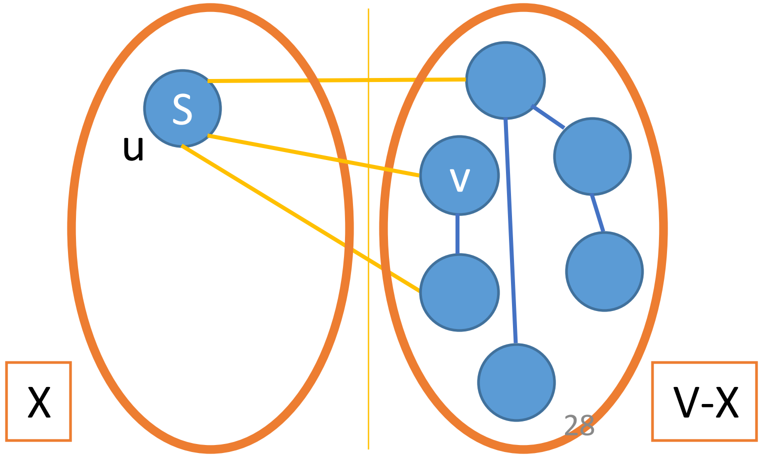
Simplified Pseudocode for Prim's Algorithm

```
X = {s}    // list of found nodes
T = empty  // edges that belong to MST

while X is not V:
    let e = (u, v) be the cheapest edge of E
        with u in X and v not in X
    add e to T
    add v to X
```

Simplified Pseudocode for Prim's Algorithm

```
X = {s}     // list of found nodes
T = empty   // edges that belong to MST

while X is not V:
    let e = (u, v) be the cheapest edge of E
        with u in X and v not in X
    add e to T
    add v to X
```

1. Prim's algorithm maintains the invariant that T spans X

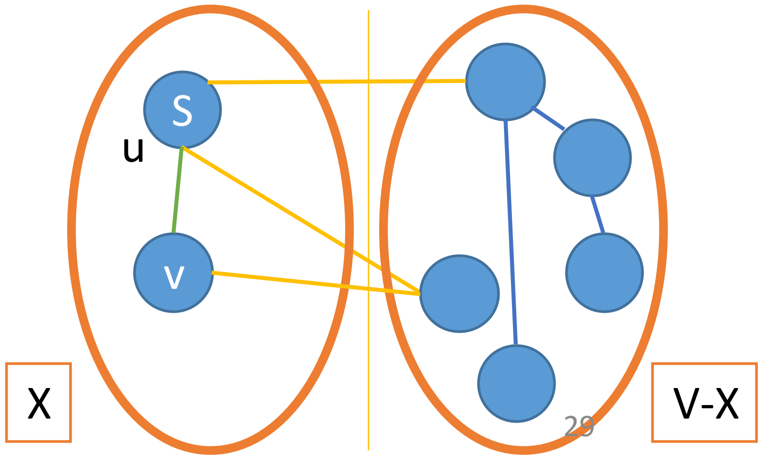


Assume the graph is connected.

Simplified Pseudocode for Prim's Algorithm

```
X = {s}    // list of found nodes
T = empty  // edges that belong to MST

while X is not V:
    let e = (u, v) be the cheapest edge of E
        with u in X and v not in X
    add e to T
    add v to X
```

1. Prim's algorithm maintains the invariant that T spans X

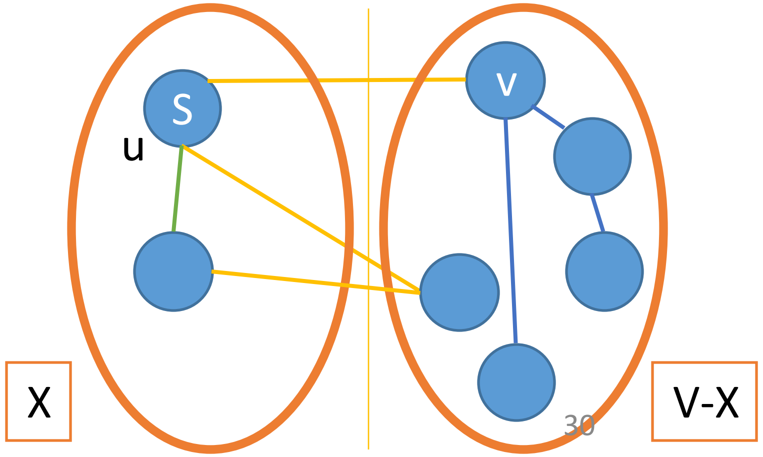


Simplified Pseudocode for Prim's Algorithm

```
X = {s}    // list of found nodes
T = empty // edges that belong to MST
```

1. Prim's algorithm maintains the invariant that T spans X

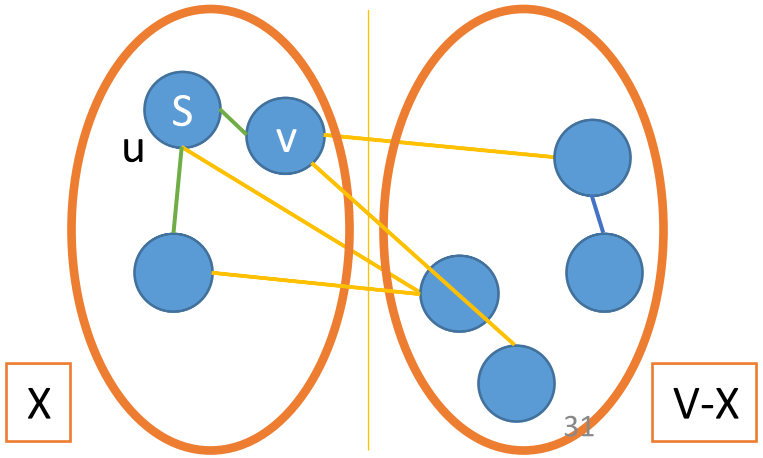
```
while X is not V:
    let e = (u, v) be the cheapest edge of E
        with u in X and v not in X
    add e to T
    add v to X
```



Simplified Pseudocode for Prim's Algorithm

```
X = {s}    // list of found nodes
T = empty  // edges that belong to MST

while X is not V:
    let e = (u, v) be the cheapest edge of E
        with u in X and v not in X
    add e to T
    add v to X
```

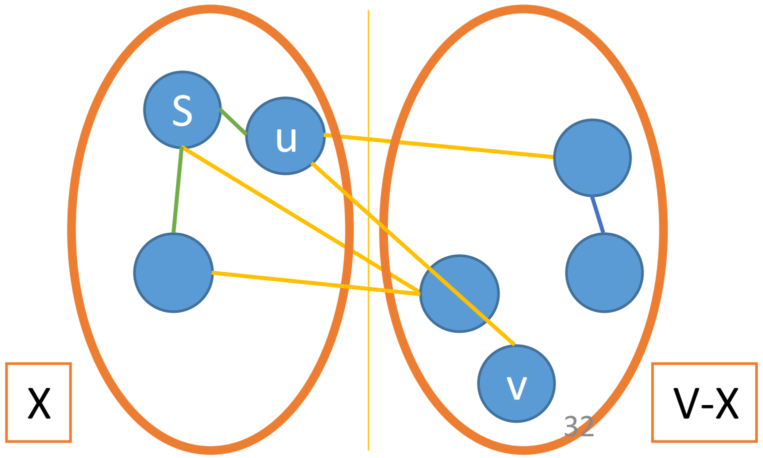
1. Prim's algorithm maintains the invariant that T spans X



Simplified Pseudocode for Prim's Algorithm

```
X = {s}     // list of found nodes
T = empty // edges that belong to MST

while X is not V:
    let e = (u, v) be the cheapest edge of E
        with u in X and v not in X
    add e to T
    add v to X
```

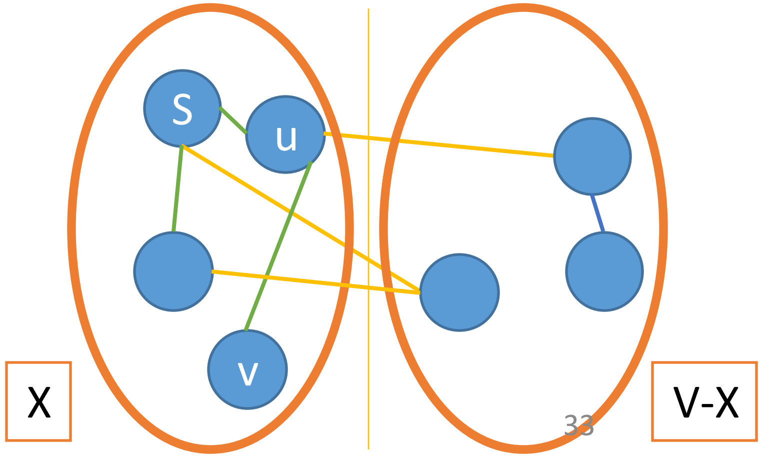
1. Prim's algorithm maintains the invariant that T spans X



Simplified Pseudocode for Prim's Algorithm

```
X = {s}    // list of found nodes
T = empty  // edges that belong to MST

while X is not V:
    let e = (u, v) be the cheapest edge of E
        with u in X and v not in X
    add e to T
    add v to X
```

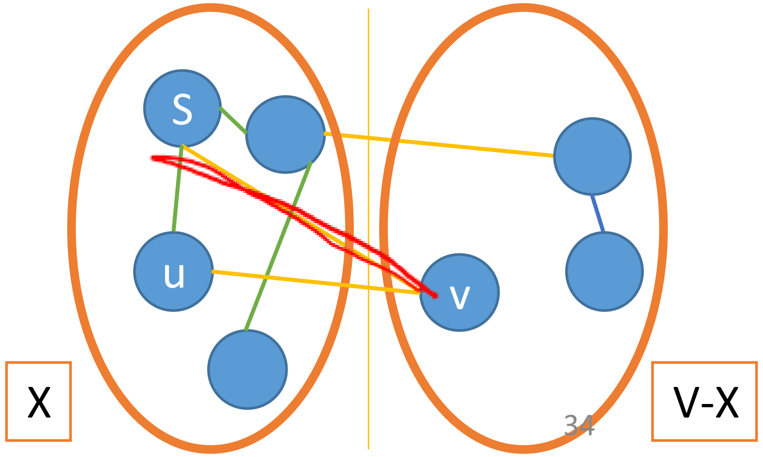
1. Prim's algorithm maintains the invariant that T spans X



Simplified Pseudocode for Prim's Algorithm

```
X = {s}     // list of found nodes
T = empty // edges that belong to MST

while X is not V:
    let e = (u, v) be the cheapest edge of E
        with u in X and v not in X
    add e to T
    add v to X
```

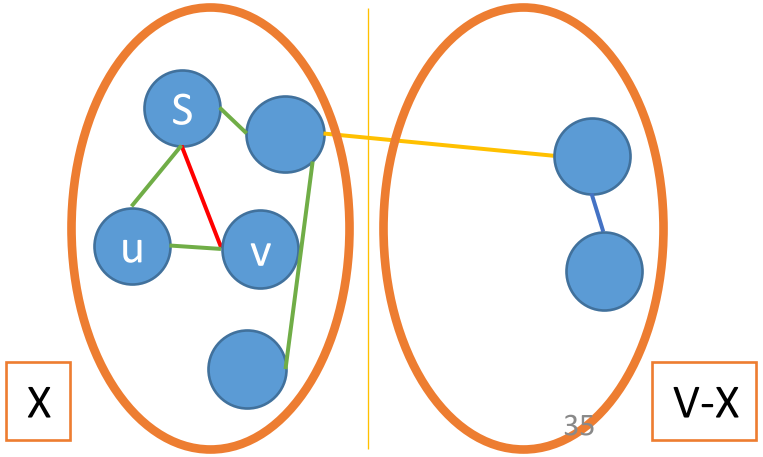
1. Prim's algorithm maintains the invariant that T spans X



Simplified Pseudocode for Prim's Algorithm

```
X = {s}    // list of found nodes
T = empty  // edges that belong to MST

while X is not V:
    let e = (u, v) be the cheapest edge of E
        with u in X and v not in X
    add e to T
    add v to X
```

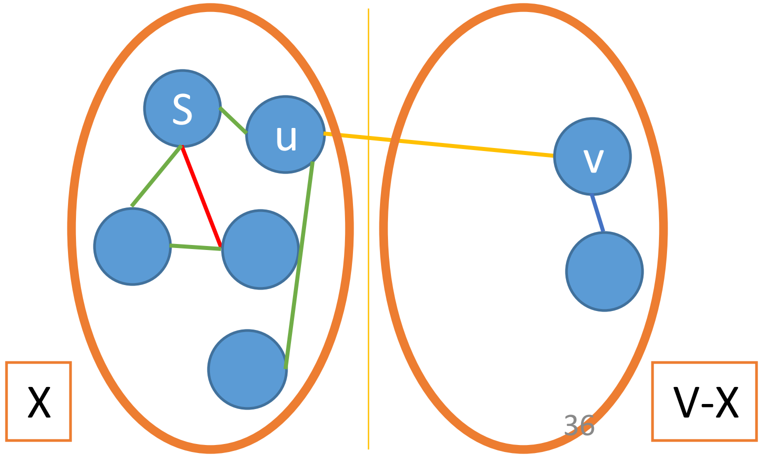
1. Prim's algorithm maintains the invariant that T spans X



Simplified Pseudocode for Prim's Algorithm

```
X = {s}    // list of found nodes
T = empty  // edges that belong to MST

while X is not V:
    let e = (u, v) be the cheapest edge of E
        with u in X and v not in X
    add e to T
    add v to X
```

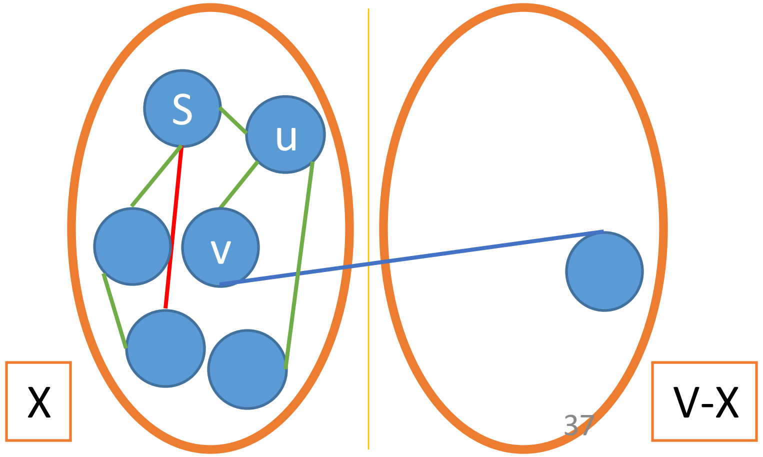
1. Prim's algorithm maintains the invariant that T spans X



Simplified Pseudocode for Prim's Algorithm

```
X = {s}    // list of found nodes
T = empty  // edges that belong to MST

while X is not V:
    let e = (u, v) be the cheapest edge of E
        with u in X and v not in X
    add e to T
    add v to X
```

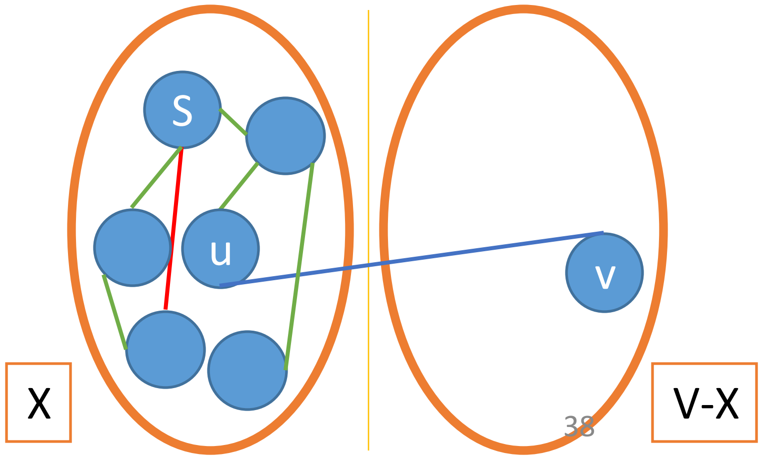
1. Prim's algorithm maintains the invariant that T spans X



Simplified Pseudocode for Prim's Algorithm

```
X = {s}    // list of found nodes
T = empty  // edges that belong to MST

while X is not V:
    let e = (u, v) be the cheapest edge of E
        with u in X and v not in X
    add e to T
    add v to X
```

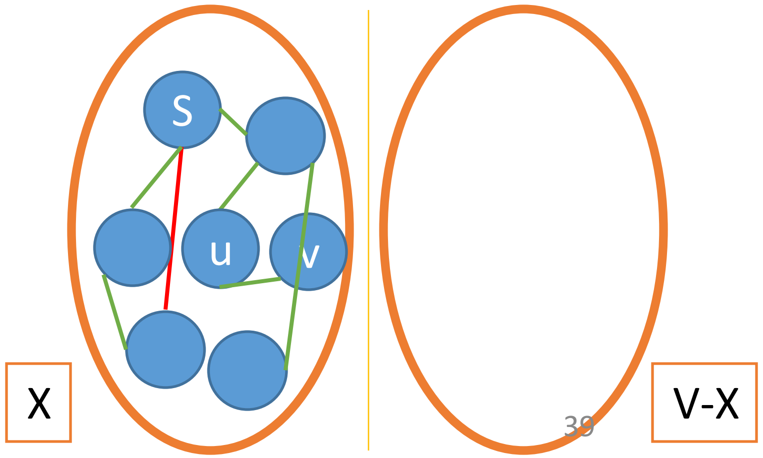
1. Prim's algorithm maintains the invariant that T spans X



Simplified Pseudocode for Prim's Algorithm

```
X = {s}    // list of found nodes
T = empty  // edges that belong to MST

while X is not V:
    let e = (u, v) be the cheapest edge of E
        with u in X and v not in X
    add e to T
    add v to X
```

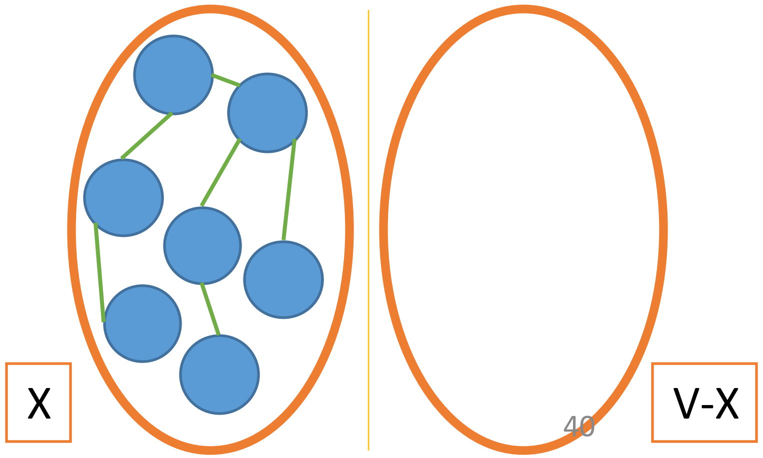
1. Prim's algorithm maintains the invariant that T spans X



Simplified Pseudocode for Prim's Algorithm

```
X = {s}     // list of found nodes
T = empty // edges that belong to MST

while X is not V:
    let e = (u, v) be the cheapest edge of E
        with u in X and v not in X
    add e to T
    add v to X
```

1. Prim's algorithm maintains the invariant that T spans X



Claim 1: Prim's outputs a spanning tree

1. Prim's algorithm maintains the invariant that T spans X
2. The algorithm is guaranteed to terminate with X = V

Simplified Pseudocode for Prim's Algorithm

```
X = {s}    // list of found nodes
T = empty  // edges that belong to MST

while X is not V:
    let e = (u, v) be the cheapest edge of E
        with u in X and v not in X
    add e to T
    add v to X
```

2. The algorithm is guaranteed to terminate with X = V

If the algorithm does not terminate, then by the Empty cut Lemma the input graph must be disconnected.

Claim 1: Prim's outputs a spanning tree

1. Prim's algorithm maintains the invariant that T spans X
2. The algorithm is guaranteed to terminate with X = V
3. The set of edges, T, does not contain any cycles

Simplified Pseudocode for Prim's Algorithm

```
X = {s}     // list of found nodes
T = empty // edges that belong to MST

while X is not V:
    let e = (u, v) be the cheapest edge of E
        with u in X and v not in X
    add e to T
    add v to X
```

3. The set of edges, T, does not contain any cycles

By the No cycle corollary, the addition of e cannot create a cycle (it is the only edge to cross the cut).

Claim 1: Prim's outputs a spanning tree

1. Prim's algorithm maintains the invariant that T spans X

2. The algorithm is guaranteed to terminate with X = V
   - Could anything go wrong here?
   - Under what circumstances cannot we not find an edge to cross the cut (X, V - X)?
   - By the Empty cut Lemma the input graph must be disconnected
   - However, we stated that only connected graphs would be used as inputs

3. The algorithm is guaranteed to create a tree (no cycles)
   - Consider any iteration and our sets X and T
   - Suppose we add an edge e to T
   - The edge e must be the first edge to cross (X, V - X) being added to T
   - By the No cycle corollary, the addition of e cannot create a cycle (only edge to cross the cut)

Claim 1: Prim's outputs a spa… …ee

1. Prim's algorithm maintains the invarian…

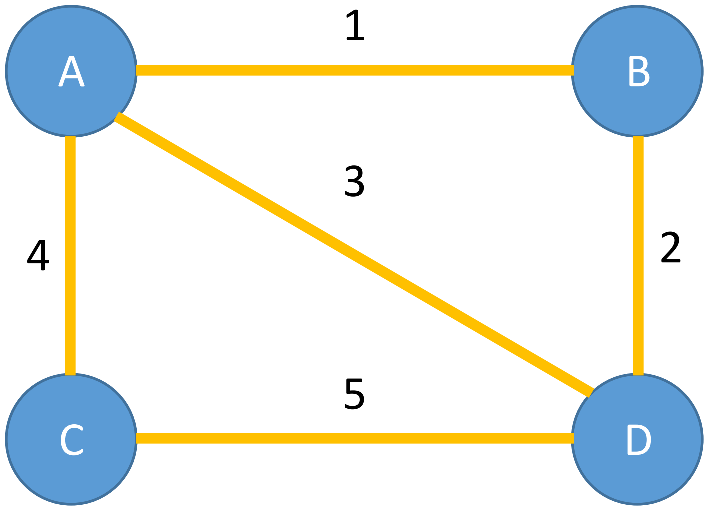
2. The algorithm is guaranteed to te… √
    - Could anythi… …ng here?
    - Under wh… …nno… …to cross the
    - By the Em… …sconnected
    - However, we… …uld be used as inputs

3. The algorithm is g… …tree (no cycles)
    - Consider any iteration
    - Suppose we add an edg…
    - The edge e must be the f… (X, V - X) being added to T
    - By the No cycle corollary, t… e cannot create a cycle (only edge to cross the cut)

Claim 2: Prim's outputs the Minimum ST

Before we can prove that the output is an MST, we need another helper definition

- Consider an edge e of G
- Suppose you can find a cut (A, B) such that e is the cheapest edge of G that crosses (A, B)
- *<u>Cut Property</u>: e belongs to the MST of G*



- ***<u>Assume that this is true! We'll prove it later</u>***

Claim 2: Prim's outputs the MST

- Claim: the Cut Property implies that Prim's algorithm outputs the MST

Simplified Pseudocode for Prim's Algorithm

```
X = {s}     // list of found nodes
T = empty // edges that belong to MST

while X is not V:
    let e = (u, v) be the cheapest edge of G
        with u in X and v not in X
    add e to T
    add v to X
```

Claim: the Cut Property implies that Prim's algorithm outputs the MST

Cut Property: if e is the cheapest edge that crosses the cut (X, V – X) then it must be in the MST.

Claim 2: Prim's outputs the MST

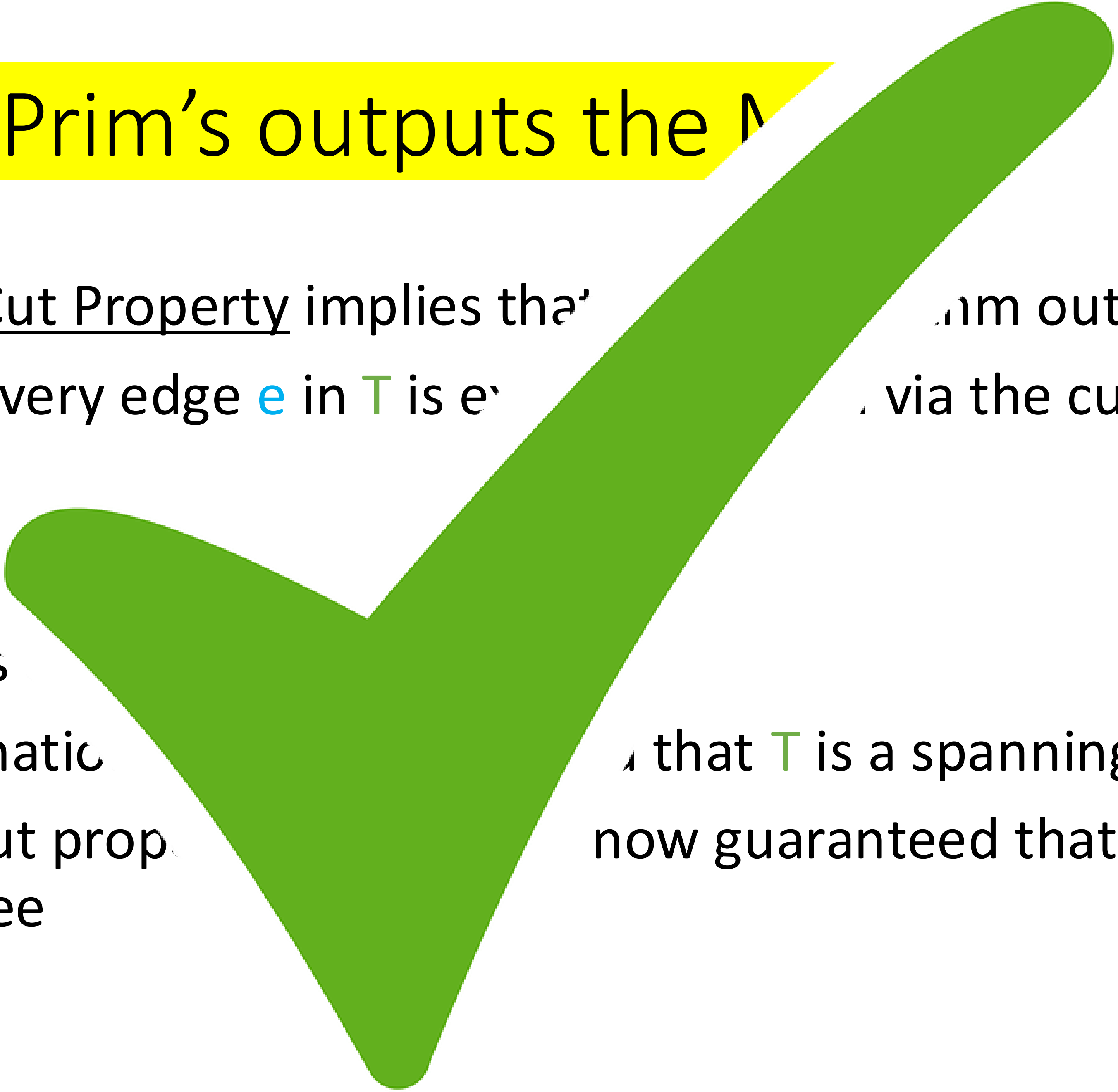
- Claim: the Cut Property implies that Prim's algorithm outputs the MST
- Key point: every edge e in T is explicitly chosen via the cut property

At any given iteration:

- The tree T is a subset of the MST
- After termination, we are guaranteed that T is a spanning tree
- Given the cut property, we are also now guaranteed that T is minimal spanning tree

Claim 2: Prim's outputs the M

- Claim: the Cut Property implies tha ... m outputs the MST
- Key point: every edge e in T is e ... via the cut property



At any giver

- The tree T is
- After terminatio ... that T is a spanning tree
- Given the cut prop ... now guaranteed that T is minimal spanning tree

Proof of Prim's

Theorem: *Prim's algorithm always computes the (or a) MST when given a connected graph.*

Need to prove two things:

1. That Prim's algorithm creates a spanning tree T*
2. And that T* is the **minimum** spanning tree

* Need to prove the cut property!

Proof of the Cut Property

Assume distinct edge costs

- Here is where our assumption of distinct edge costs is useful.

*Cut Property: if e is the cheapest edge that crosses the cut (X, V – X) then it **must** be in the MST*

We are going to prove this using exchange argument contradiction

Proof of the Cut Property

Claim: Suppose there is an edge e that is the cheapest one to cross a cut (X, V-X), but e is not in the MST T*

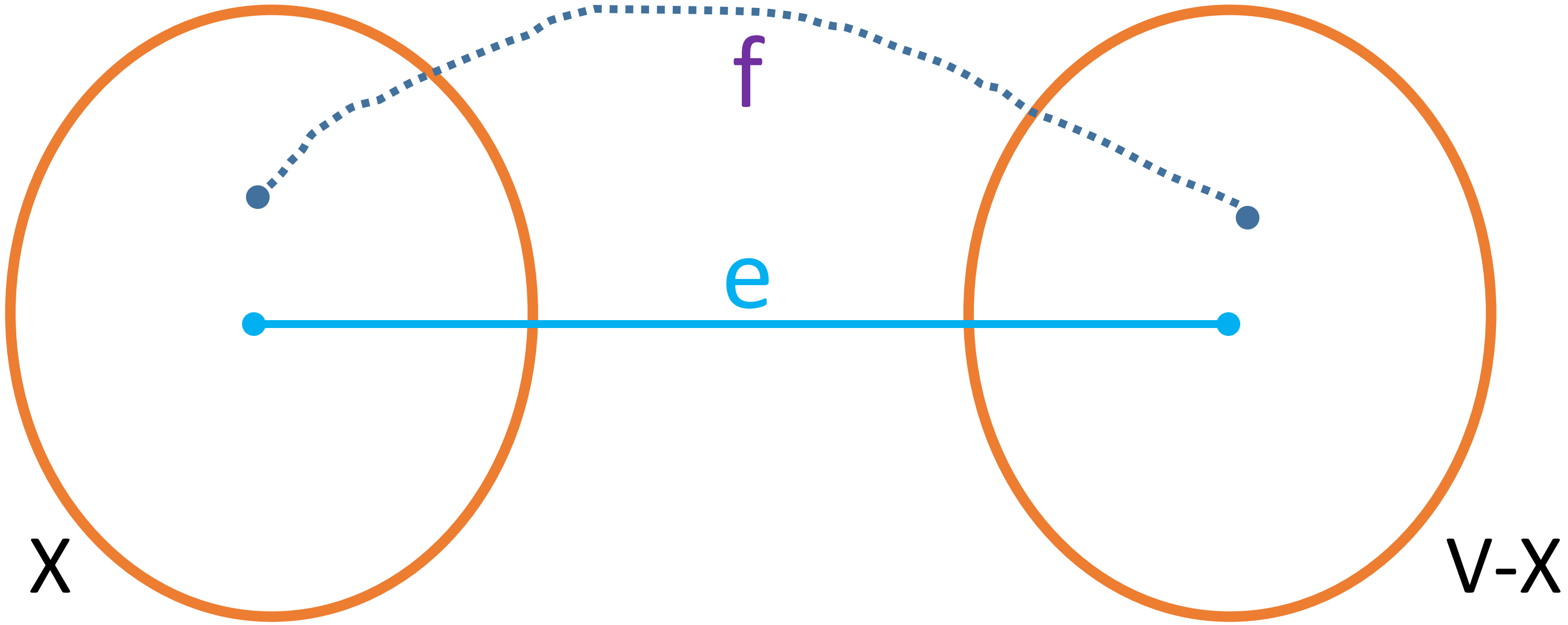
- What are we going to exchange?

Idea: exchange e with another edge in T* to make the cost of T* even cheaper (which would result in a contradiction)

What edge in T* can we swap with e?

Proof of the Cut Property

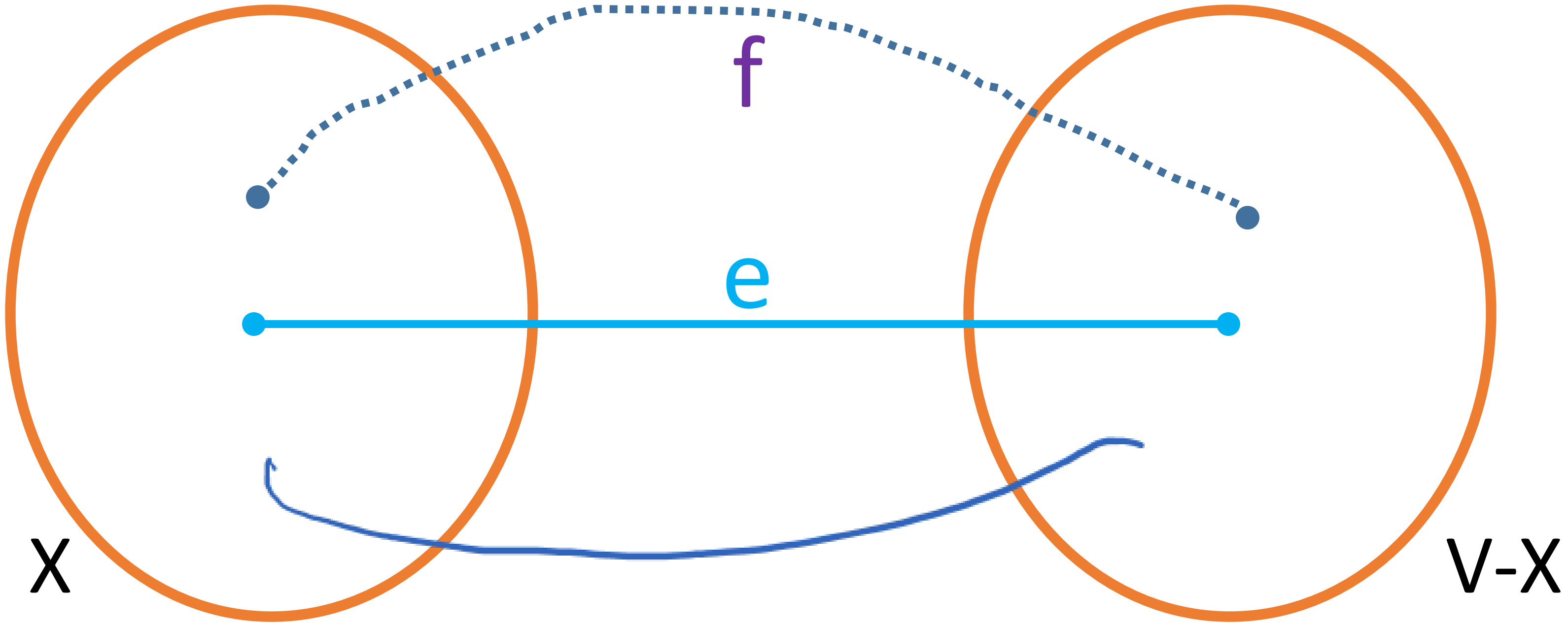
- The edge e is the cheapest to cross (X, V-X)
- MST T* must contain some other edge that crosses (X, V-X), otherwise T* would be disconnected.
- Let's call this other edge f
- Let's try to exchange e and f to get a spanning tree that is cheaper than T*



Proof of the Cut Property

Is T* U {e} – {f} a spanning tree of G?

Yes	No	Only if e is the cheapest edge	Maybe



Proof of the Cut Property

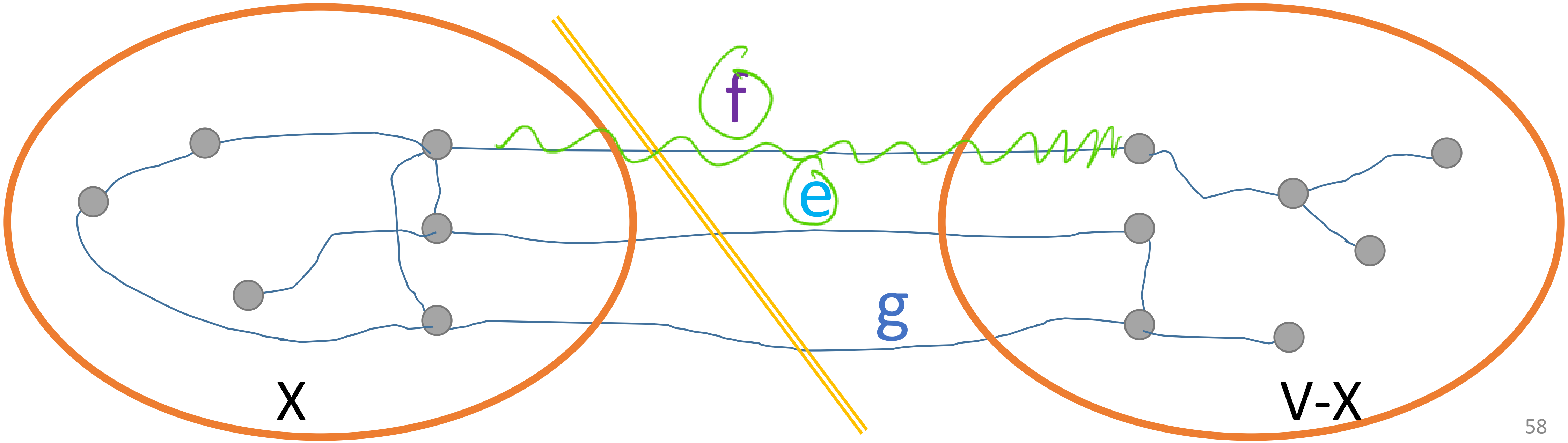
Is T* U {e} – {f} a spanning tree of G?

Yes

No

Only if e is the cheapest edge

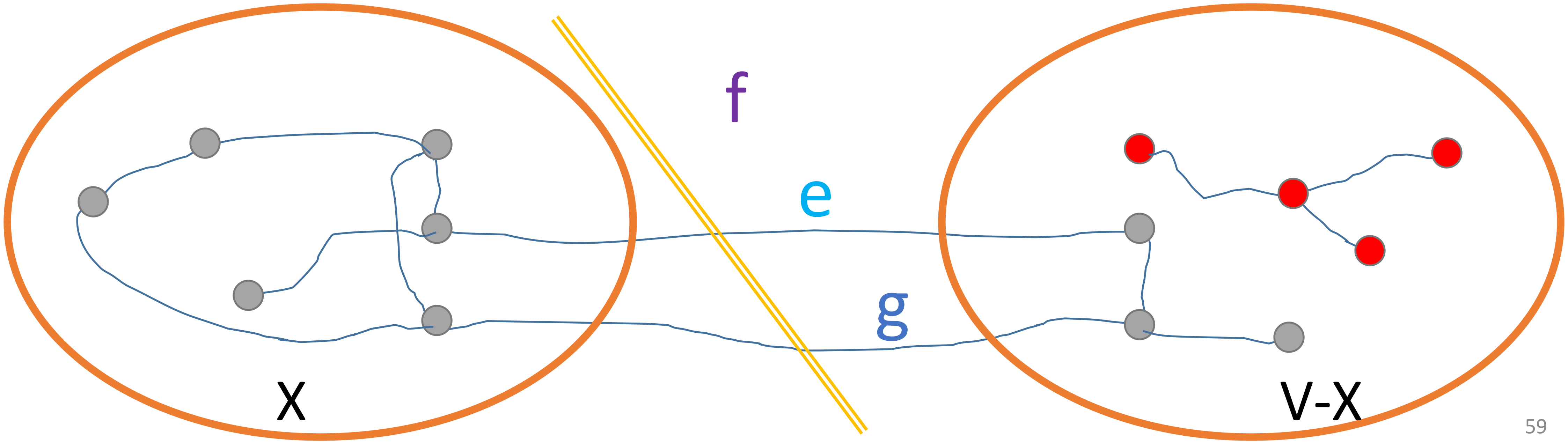
Maybe



Proof of the Cut Property

Is T* U {e} – {f} a spanning tree of G?

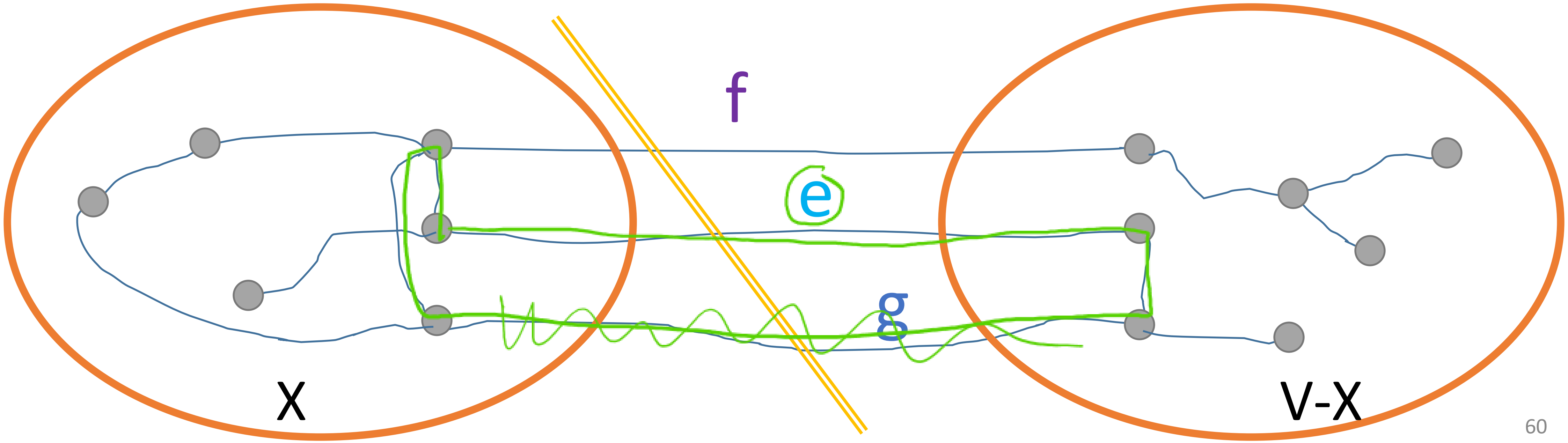
Yes	No	Only if e is the cheapest edge	Maybe



Proof of the Cut Property

Is T* U {e} – {f} a spanning tree of G?

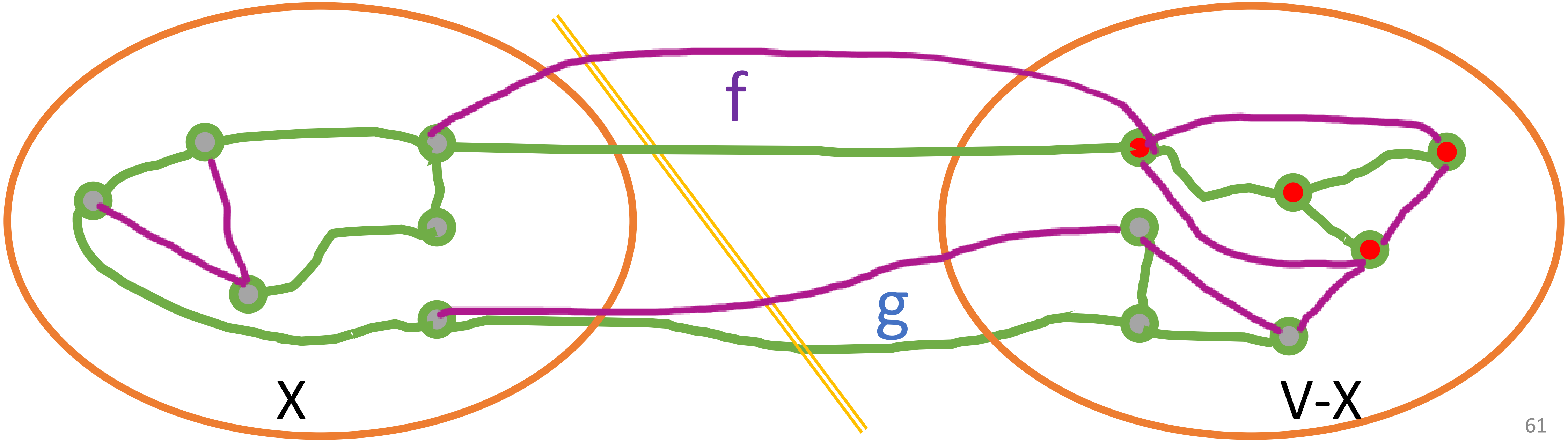
Yes | No | Only if e is the cheapest edge | Maybe



Proof of the Cut Property

Hope: that we can always find a suitable edge e' so that exchanging edges yields a valid spanning tree

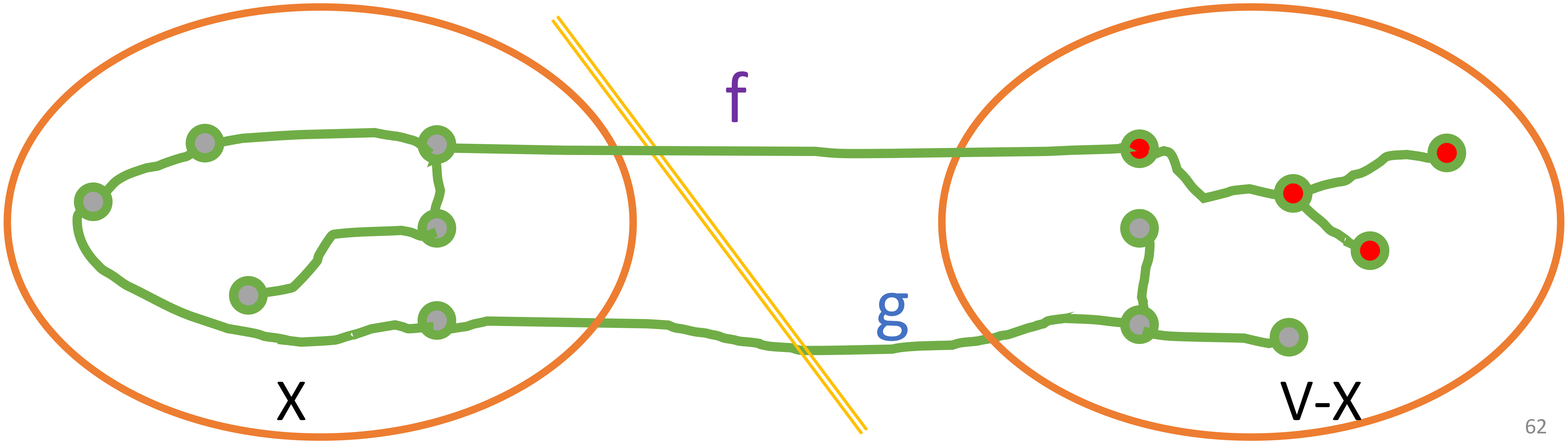
Solid green lines are those that are currently part of T*
Rainbow lines are other edges



Proof of the Cut Property

Hope: that we can always find a suitable edge e’ so that exchanging edges yields a valid spanning tree

Solid green lines are those that are currently part of T*



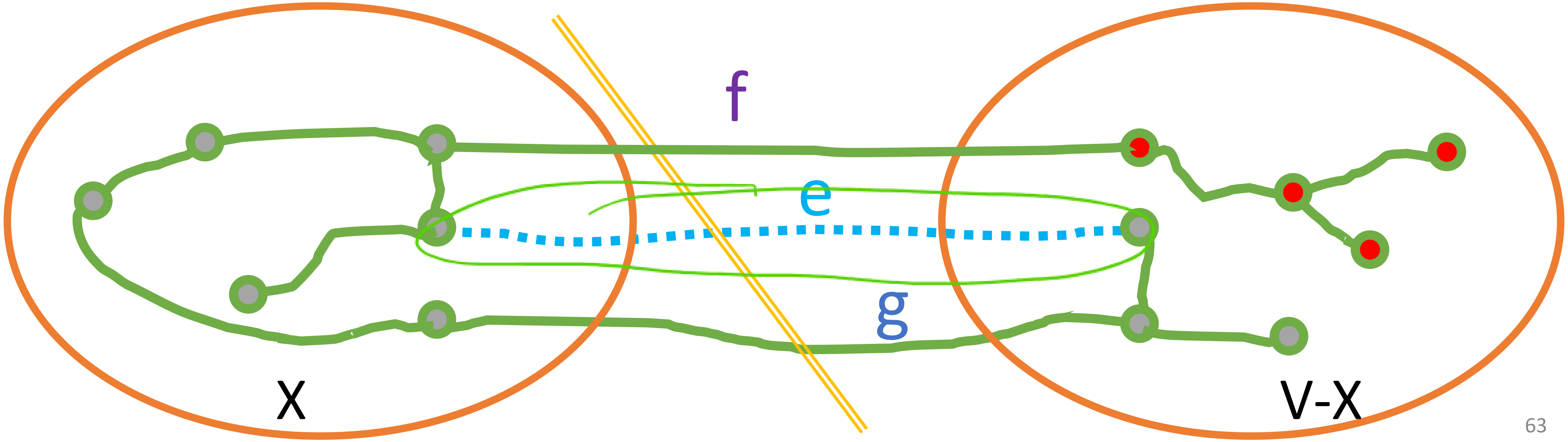
Proof of the Cut Property

Add the edge e.

What does adding e do? A tree will always have n-1 edges It creates a cycle that crosses the cut!

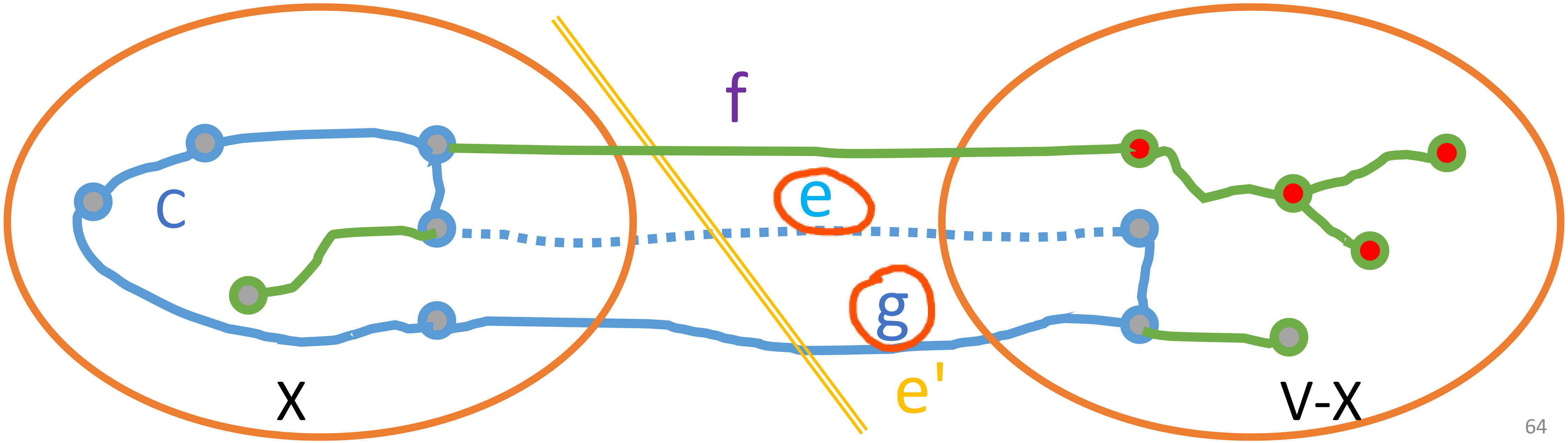
Which one of these edges can we exchange with e?

Solid green lines are those that are currently part of T*



Proof of the Cut Property

- Let C be the cycle created in T* by adding the edge e
- Find all edges that cross (X, V-X)
- By the double-crossing Lemma, there must be an edge e' that crosses (X, V-X)

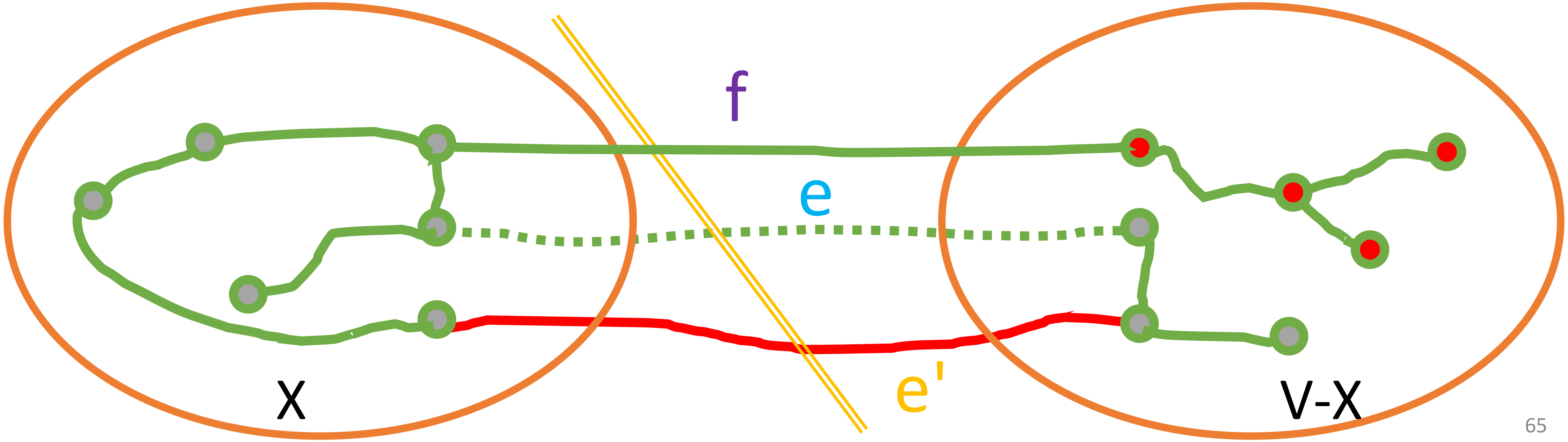


Proof of the Cut Property

- Let T = T* U {e} – {e'}

Exchange

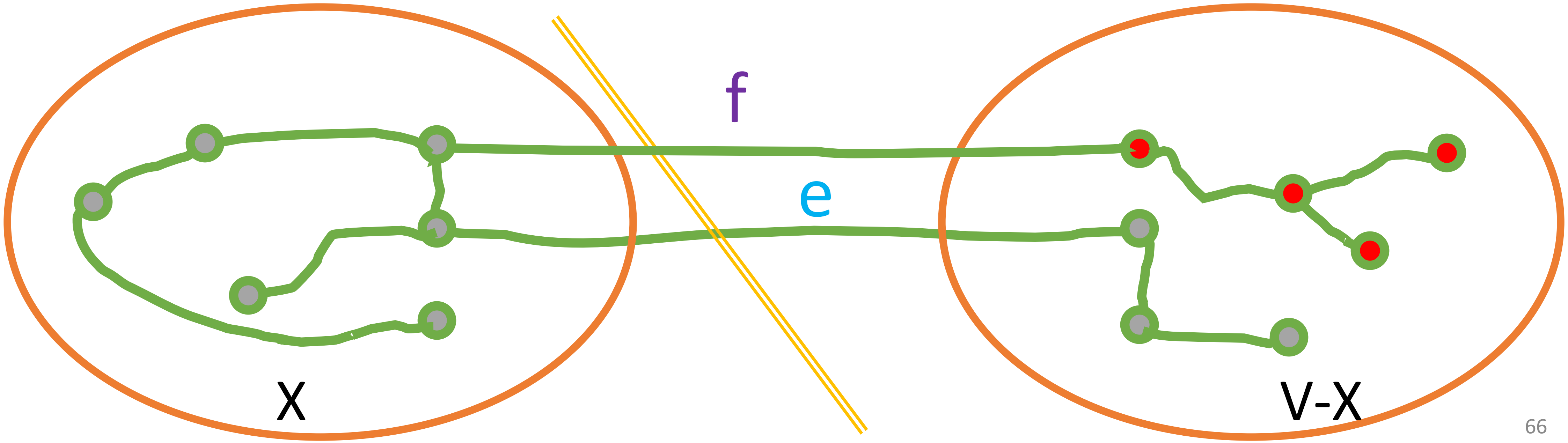
The exchange argument was easier for greedy scheduling since every exchange resulted in a valid schedule



Proof of the Cut Property

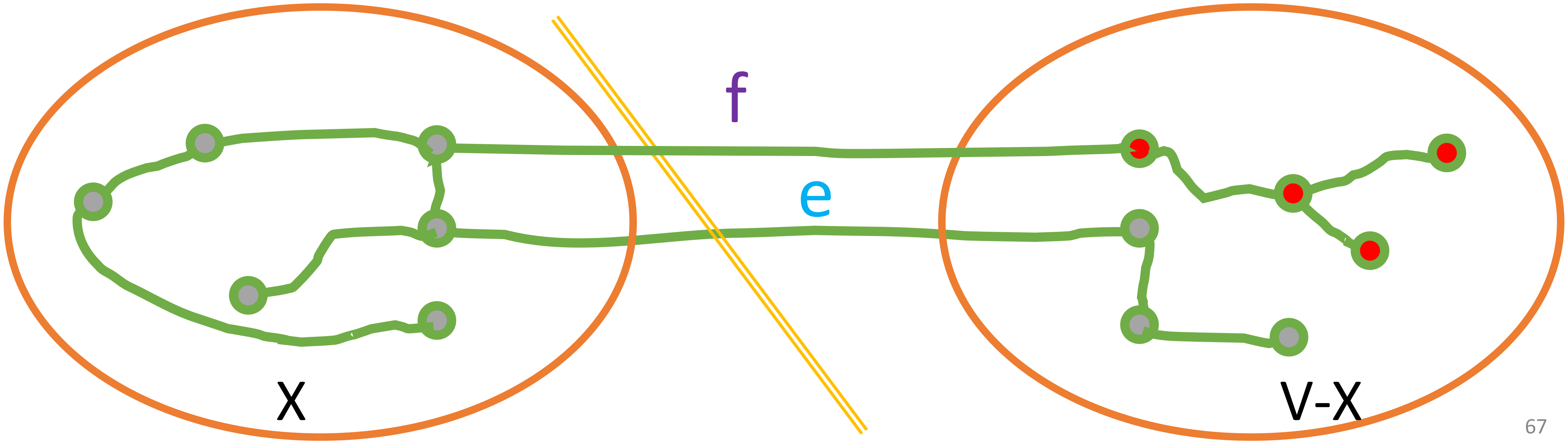
- Let T = T* U {e} – {e'}

Exchange



Proof of the Cut Property

- Let T = T* U {e} – {e'} Exchange
- T is also a spanning tree
- Since $c_e < c_{e'}$ T is a cheaper spanning tree than T* (CONTRADICTION)



Proof of Prim's

Theorem: *Prim's algorithm always computes the (or a) MST when given a connected graph.*

Need to prove two things:

1. That Prim's algorithm creates a spanning tree T*
2. And that T* is the **minimum** spanning tree *

* Need to prove the cut property!

What is the running time of Prim's?

Can we do better than O(mn)?

Can easily get to O(m lg n) using a heap (or faster with a Fibonacci Heap)

```
X = {s}     // list of found nodes
T = empty   // edges that belong to MST

while X is not V:
    let e = (u, v) be the cheapest edge of E
        with u in X and v not in X
    add e to T
    add v to X
```

O(n) for this while loop

O(m) to find cheapest edge that crosses the cut (X, V-X)