Lecture 16:

Minimum Spanning Trees

Kruskal's Algorithm

Tue, March 12th

Outline For Today

1. Graph Theory Part 1: Trees & Properties of Trees
2. Minimum Spanning Trees
3. Kruskal’s Algorithm
4. Graph Theory Part 2: Cuts & Properties of Cuts
5. Correctness Proof of Kruskal’s Algorithm

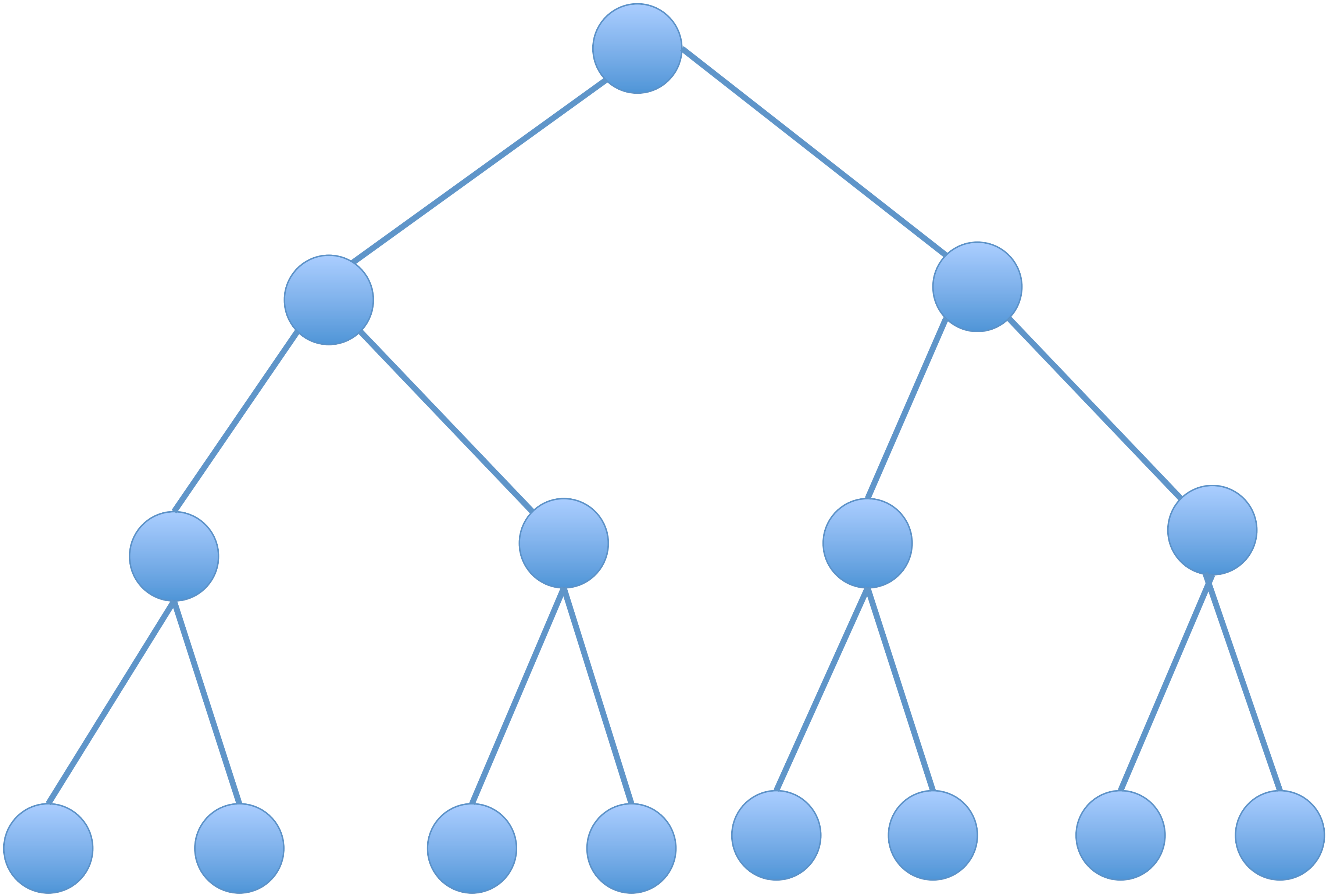
Outline For Today

1. Graph Theory Part 1: Trees & Properties of Trees
2. Minimum Spanning Trees
3. Kruskal's Algorithm
4. Graph Theory Part 2: Cuts & Properties of Cuts
5. Correctness Proof of Kruskal's Algorithm

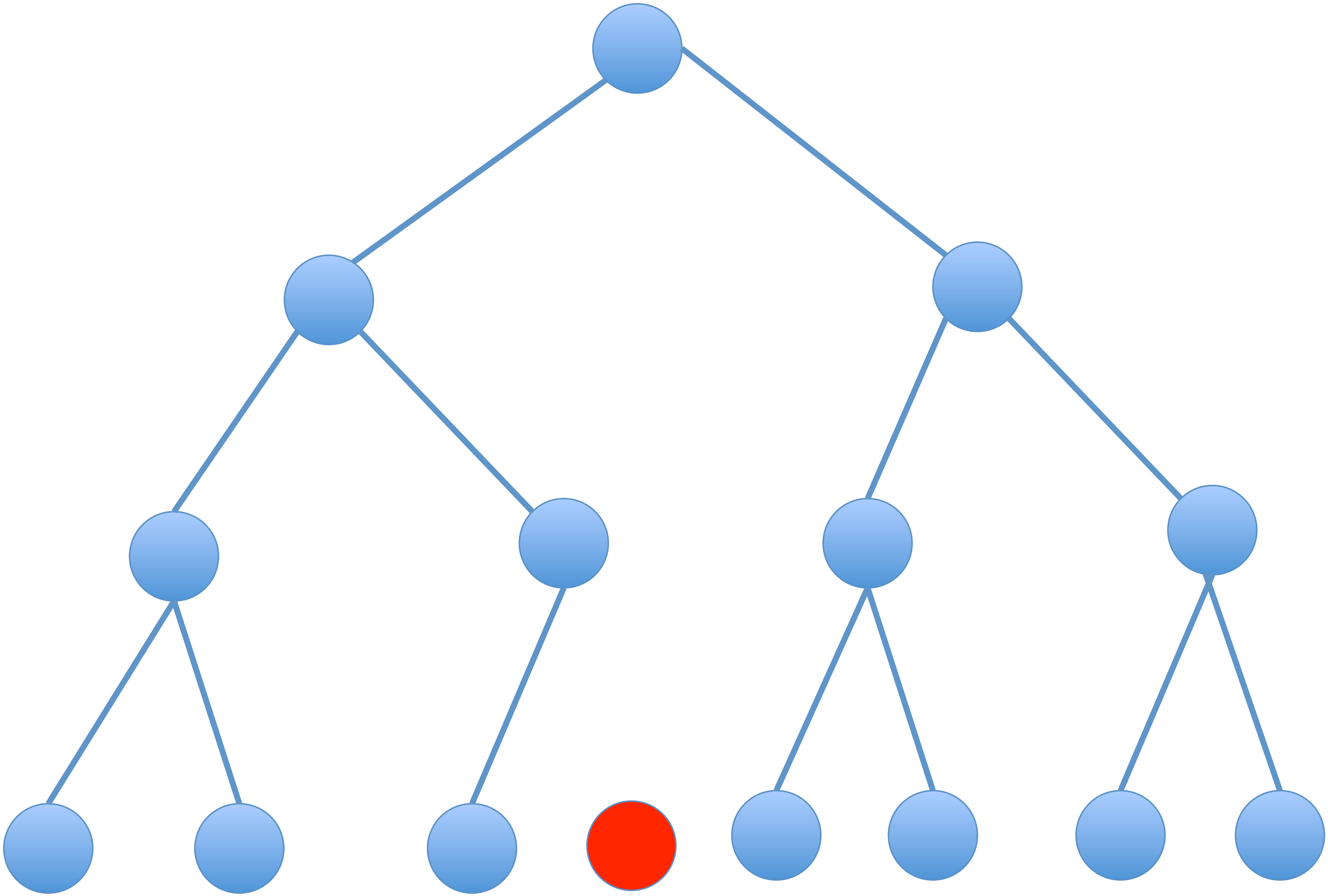
Tree

- Definition: An undirected graph G(V, E) is a tree iff
  1. G is connected
  2. G is acyclic

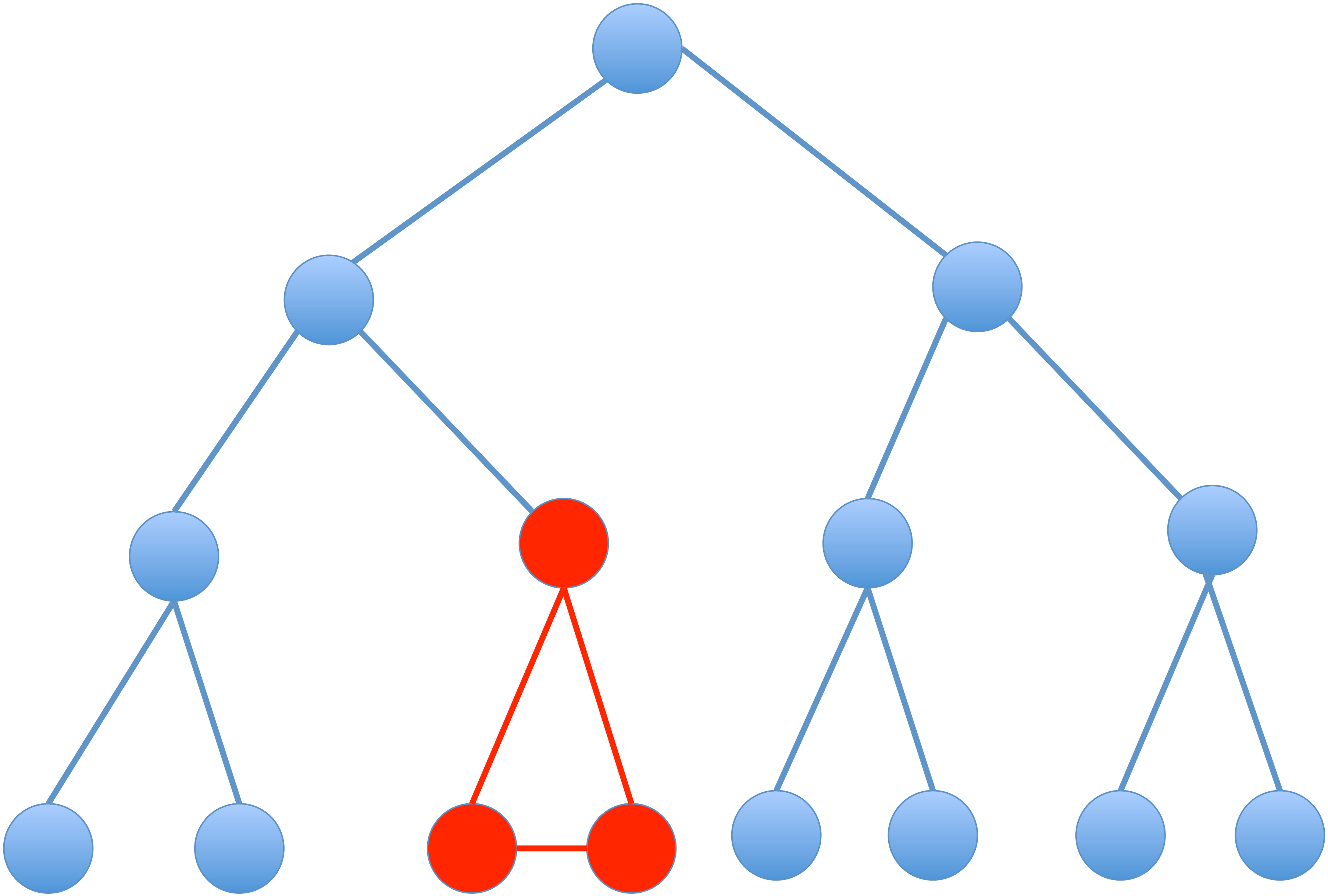
Example: Tree (1)



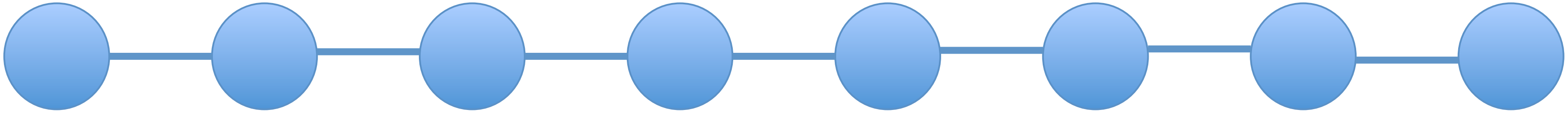
Example: Not a Tree (1)



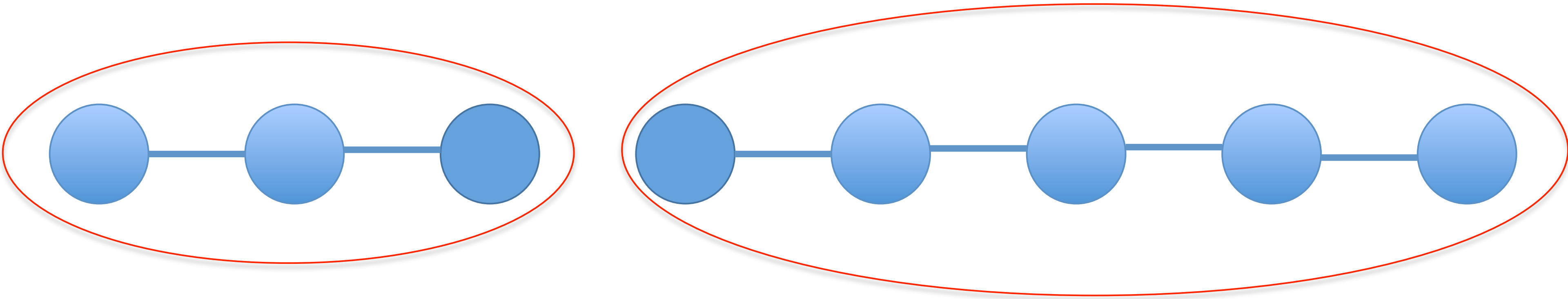
Example: Not a Tree (2)



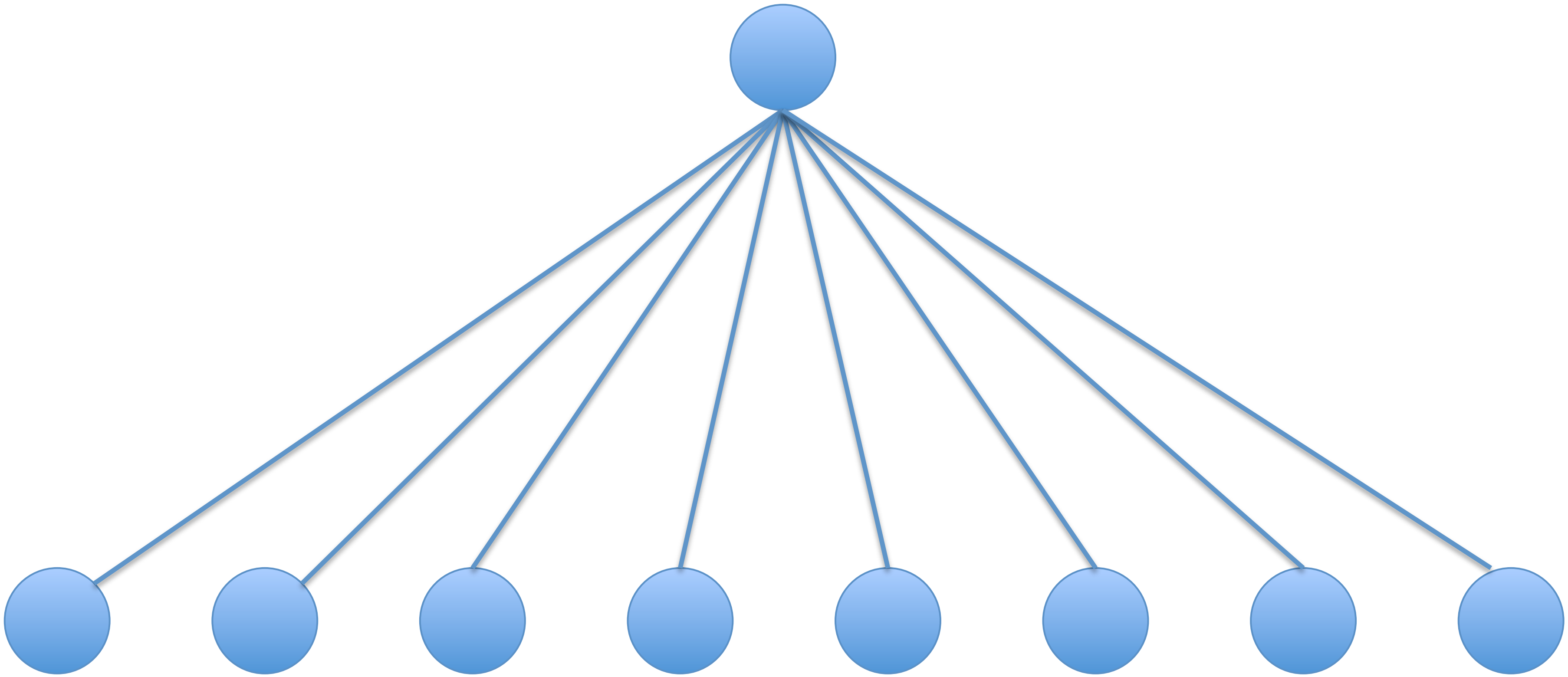
Example: Tree (2)



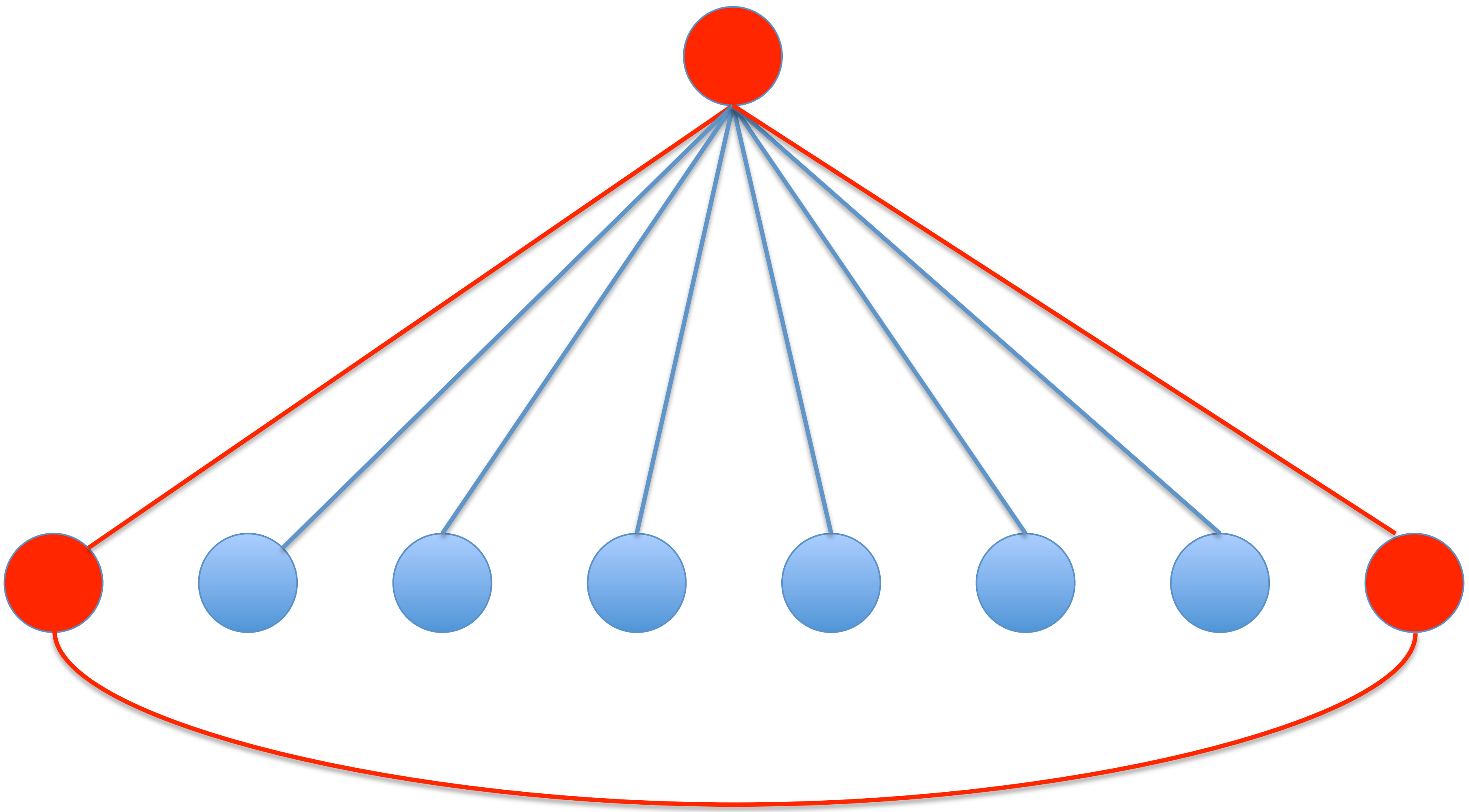
Example: Not a Tree (3)



Example: Tree (3)



Example: Not a Tree (3)



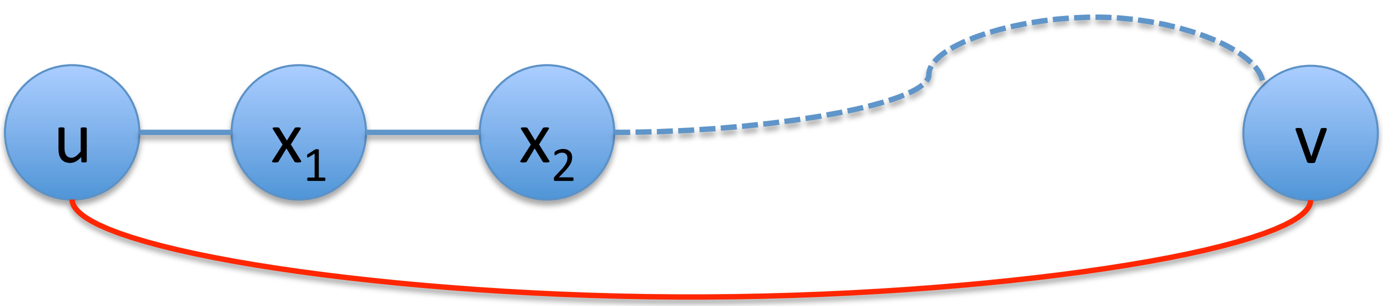
Breaking *a Tree Lemma*

Removing any edge (u, v) from a tree T

disconnects T!

Why?

No u⤳v path can exist! Assume it did…



Contradicts acyclicity of T!

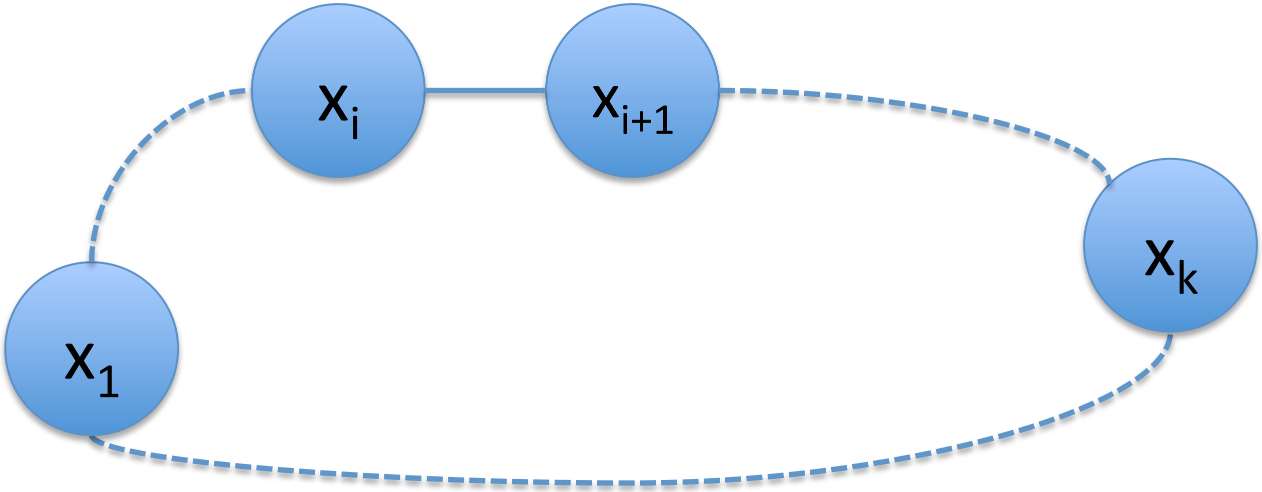
Reverse is Also True

Let G be a connected graph and assume removing any edge from G disconnects it.

Then G is acyclic and hence a tree.

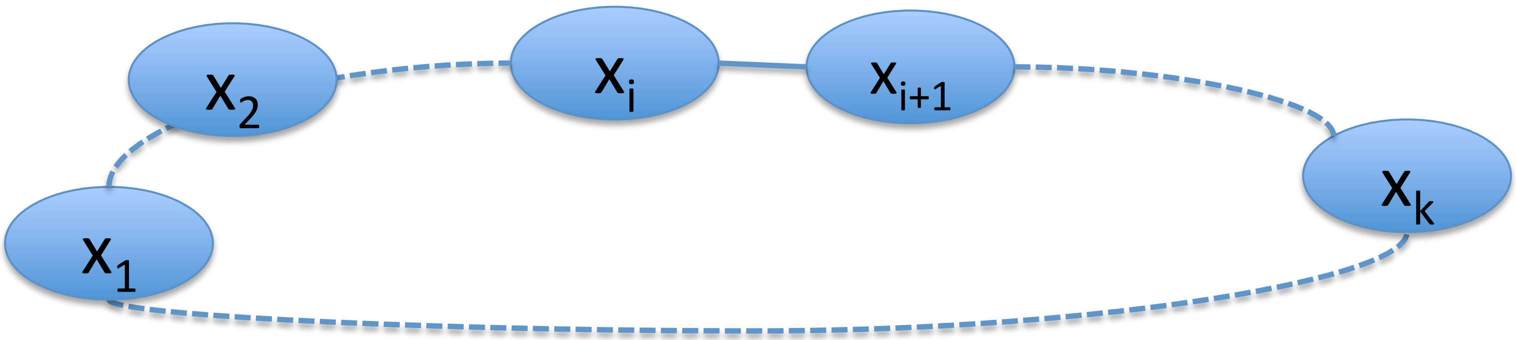
Why?

Claim: G cannot contain a cycle C

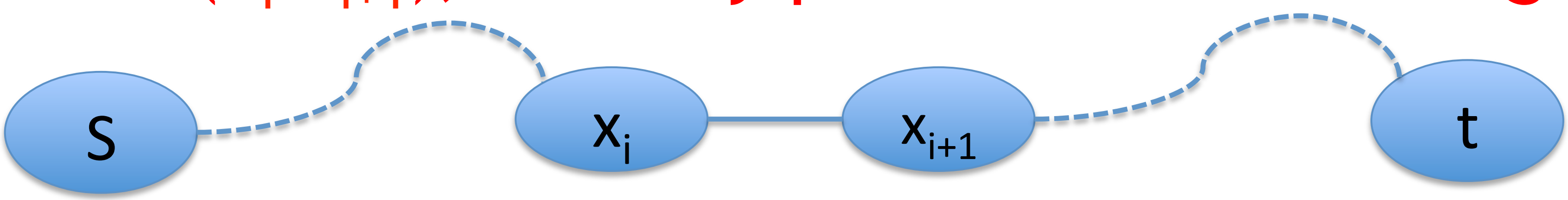


Proof That G Cannot Contain a Cycle C

***Breaking a Cycle Lemma**: Removing **any edge** from a cycle cannot disconnect a connected graph!*

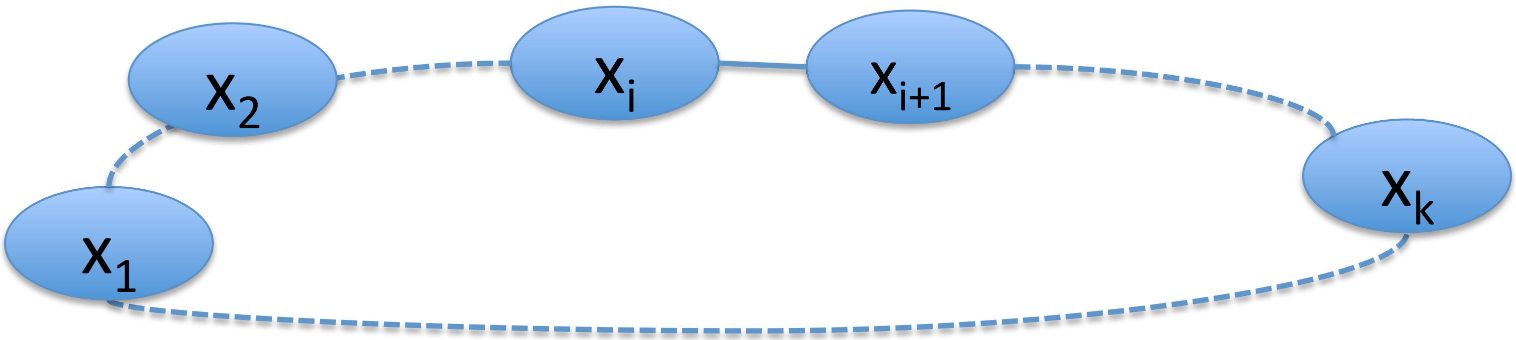


Take (x_i x_{i+1}), and any path P that was using it

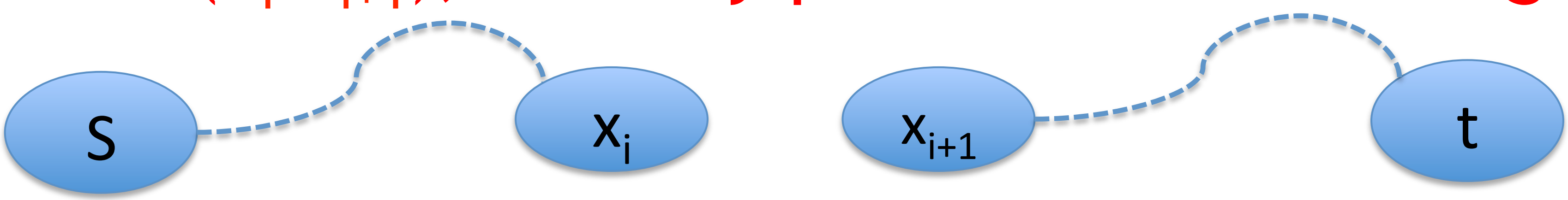


Proof That G Cannot Contain a Cycle C

***Breaking a Cycle Lemma**: Removing **any edge** from a cycle cannot disconnect a connected graph!*

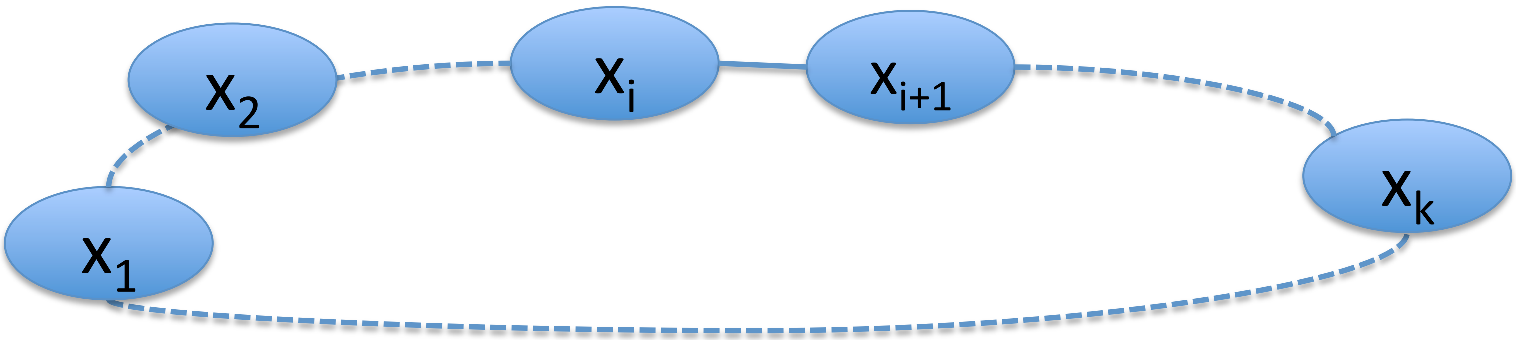


Take (x_i x_{i+1}), and any path P that was using it

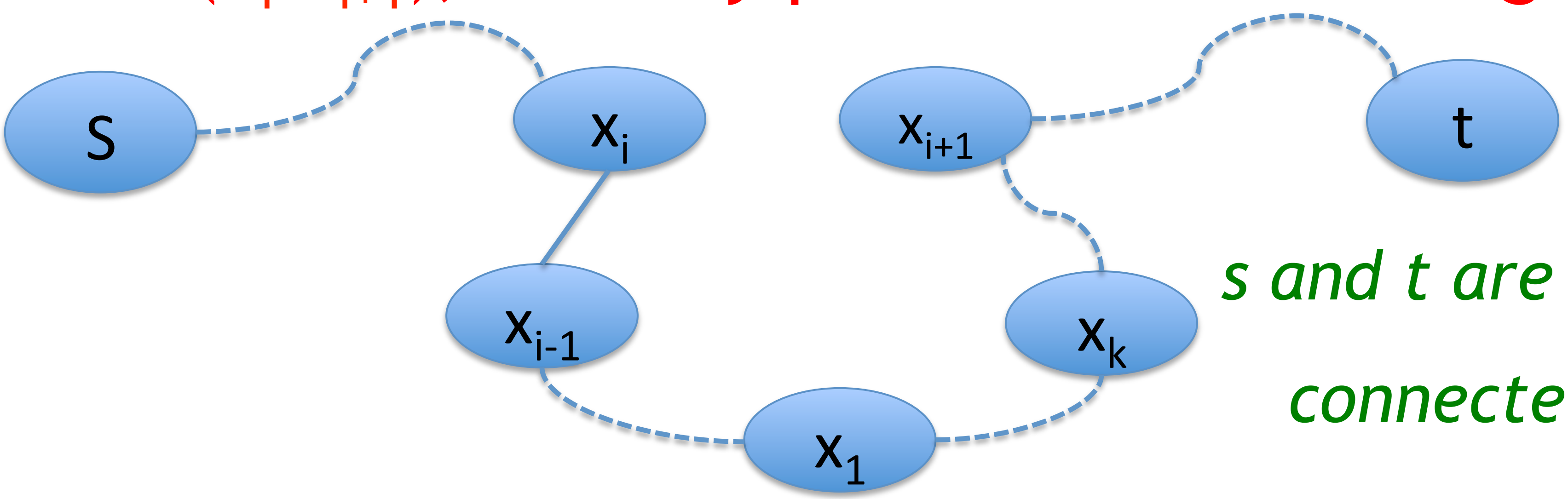


Proof That G Cannot Contain a Cycle C

Breaking a Cycle Lemma*: Removing* ***any edge*** *from a cycle cannot disconnect a connected graph!*



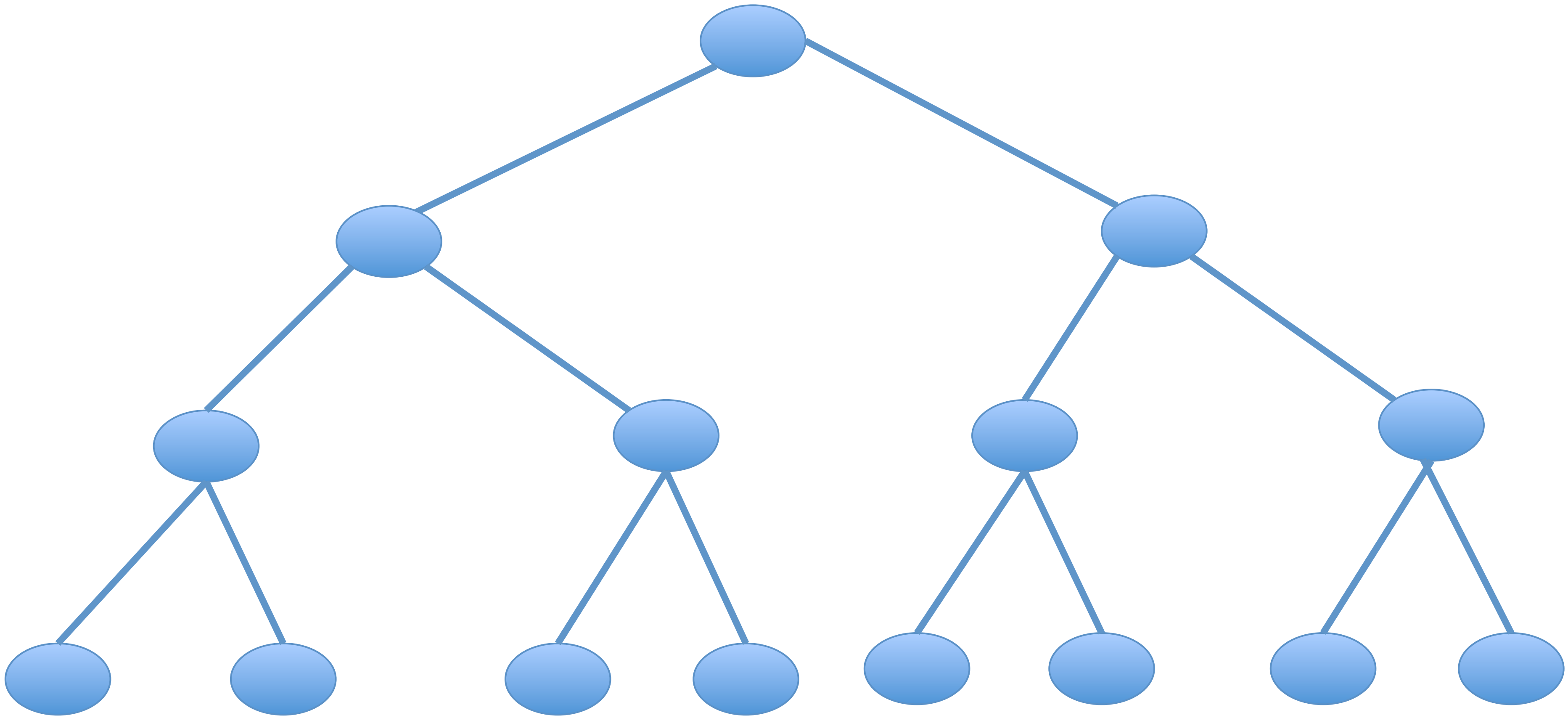
Take $(x_i\ x_{i+1})$, and any path P that was using it



s and t are still connected!

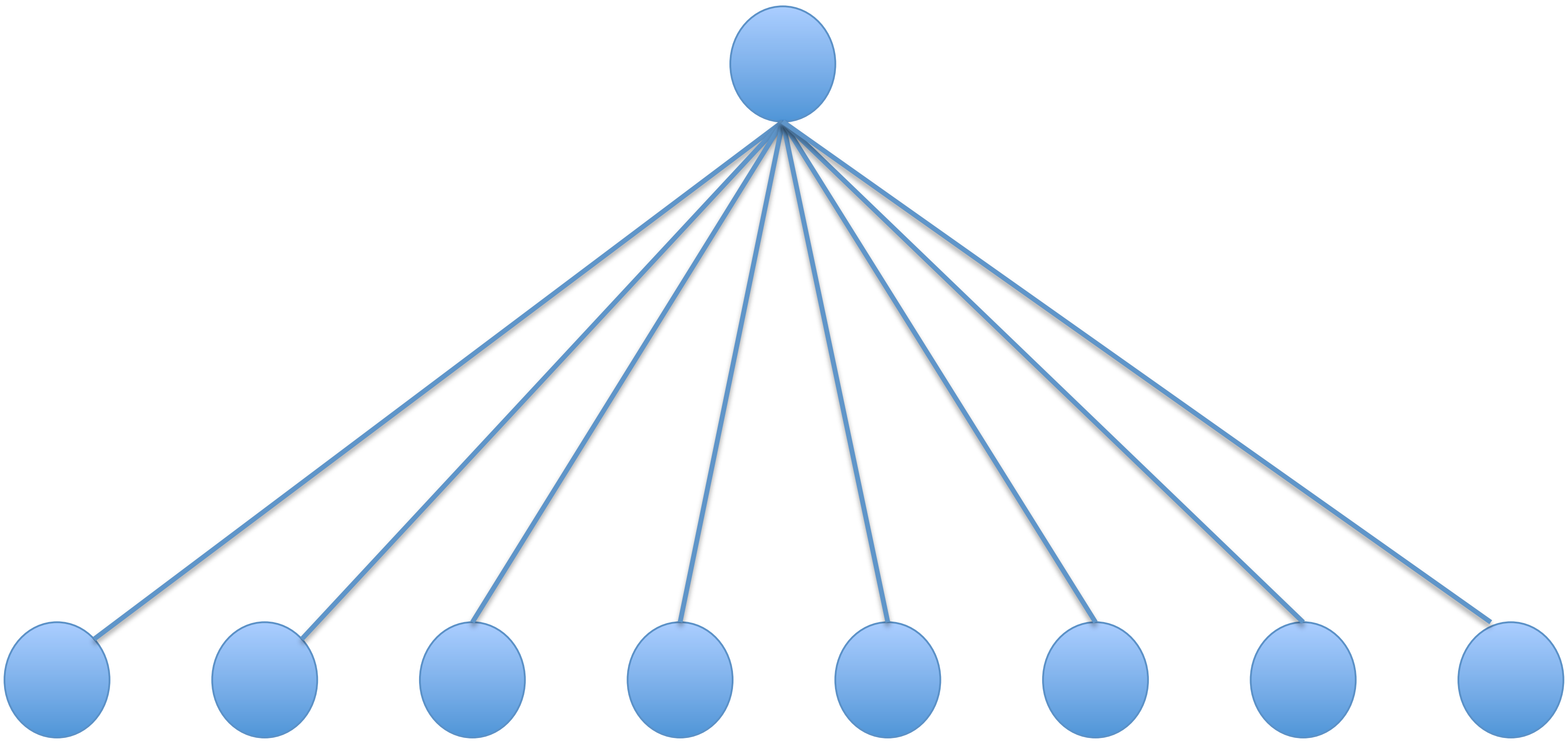
Cycle Creation Lemma

Adding any edge (u, v) to a tree T creates a cycle!



Cycle Creation Lemma

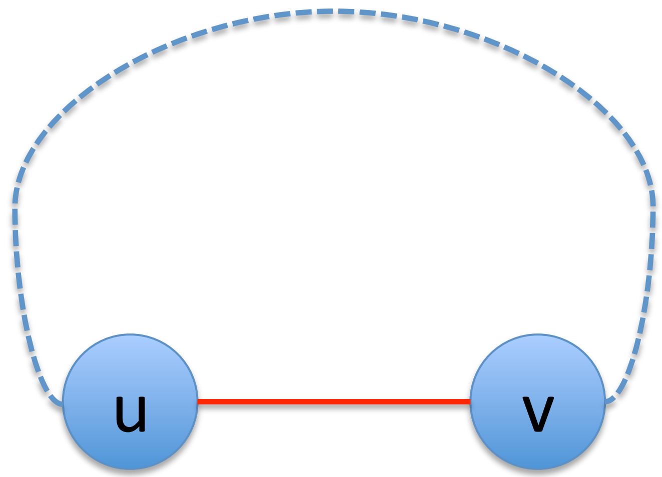
Adding any edge (u, v) to a tree T creates a cycle!



Cycle Creation Lemma

Adding any edge (u, v) to a tree T creates a cycle!

Proof: B/c T is connected, ∃ path P from u to v.



adding (u, v) closes the cycle.

Theorems

Do all by induction as exercise.

1) Every tree of n vertices contains exactly n-1 edges!

2) Every n-1 acyclic set of edges among n nodes is a tree => i.e., they connect V!

3) Every n-1 set of edges that connects n vertices is a tree => i.e., they are acyclic!

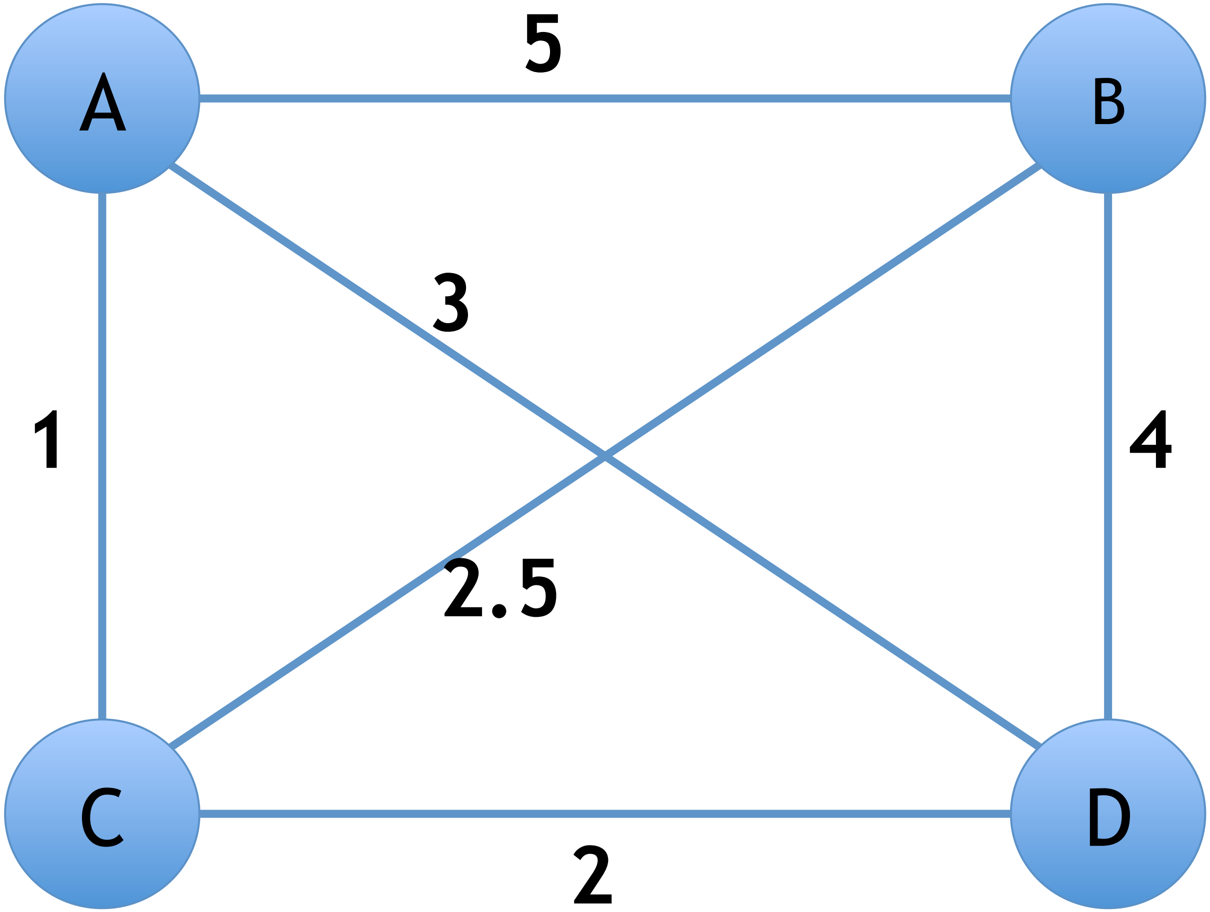
Outline For Today

1. Graph Theory Part 1: Trees & Properties of Trees
2. Minimum Spanning Trees
3. Kruskal's Algorithm
4. Graph Theory Part 2: Cuts & Properties of Cuts
5. Correctness Proof of Kruskal's Algorithm

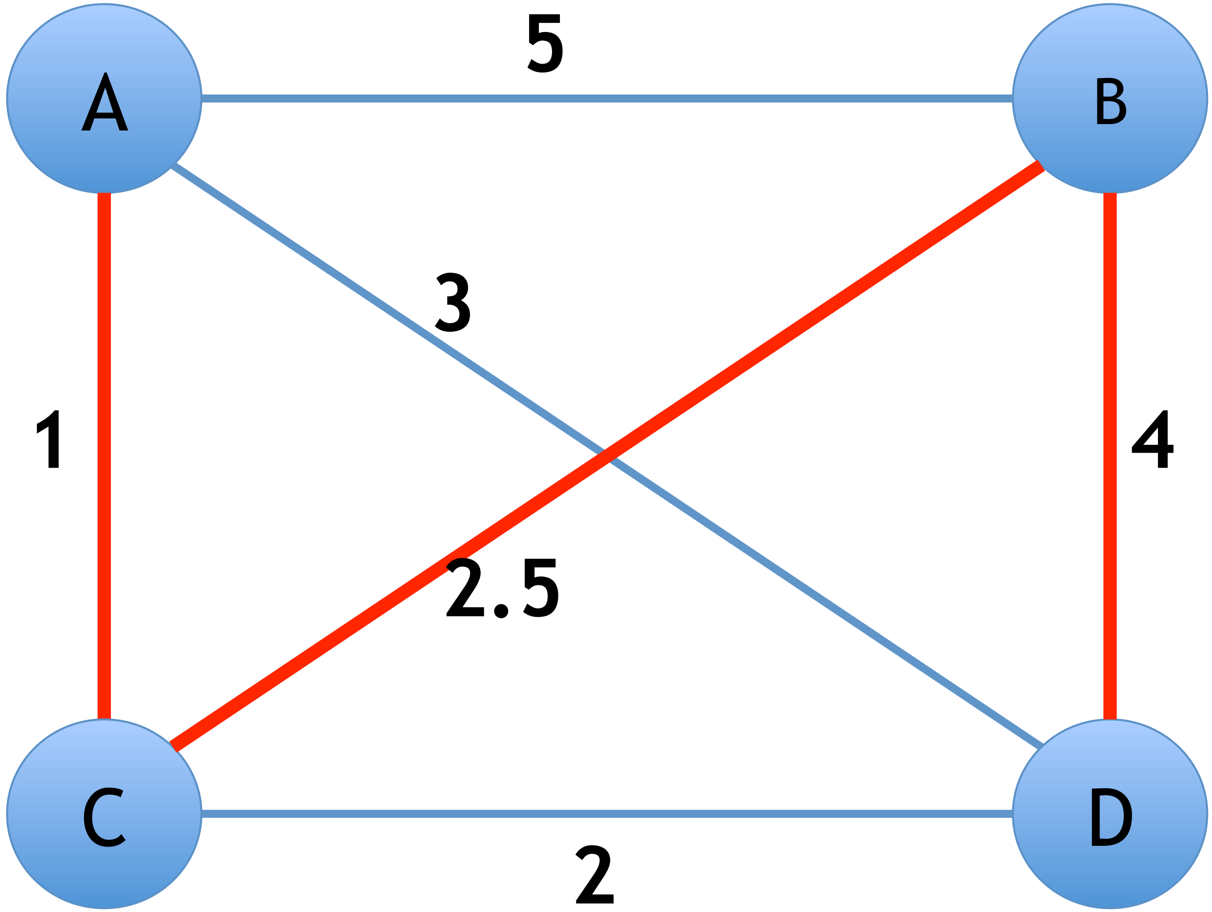
Minimum Spanning Tree

- Input: undirected & connected G(V, E) and arbitrary edge weights
- Output: A tree T* of V such that w(T*) ≤ any other T of V
  - w(T) = sum of the weights of all n-1 edges in T
  - “spanning” tree of G(V, E) means T* has to connect all of V
- Assumptions:
  1. G is connected (minor)
  2. Edge weights are distinct (all of our algorithms work w/o this assumption)

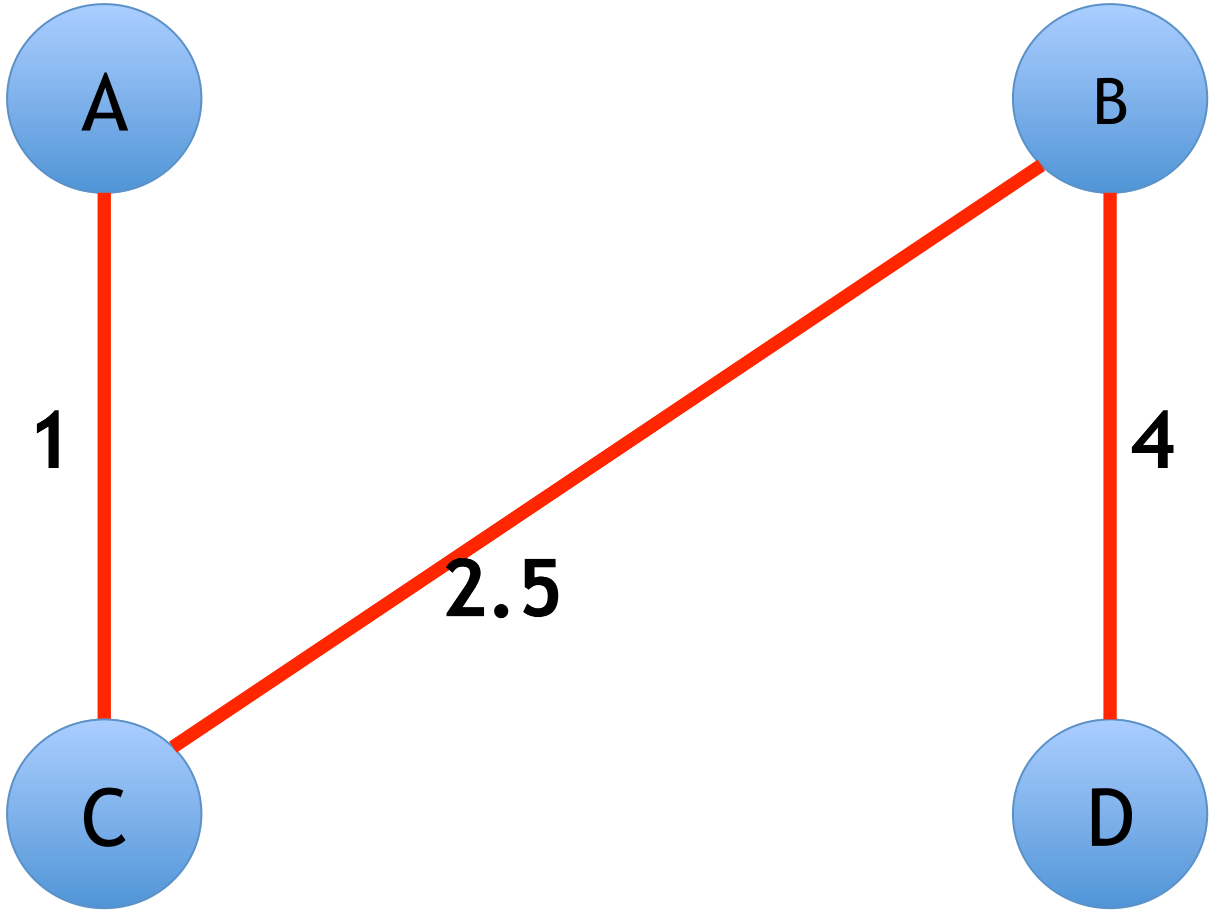
Spanning Tree Example



Spanning Tree Example

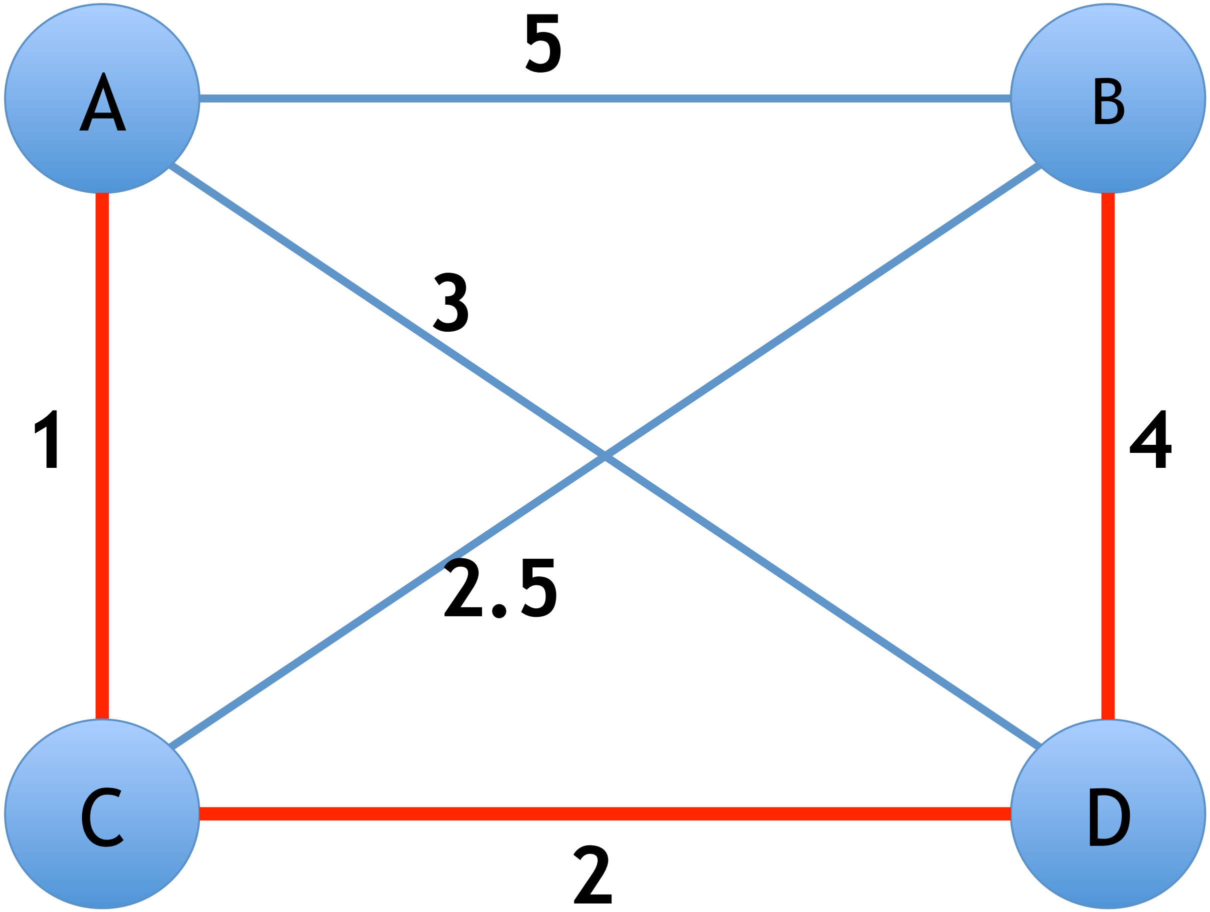


Spanning Tree Example

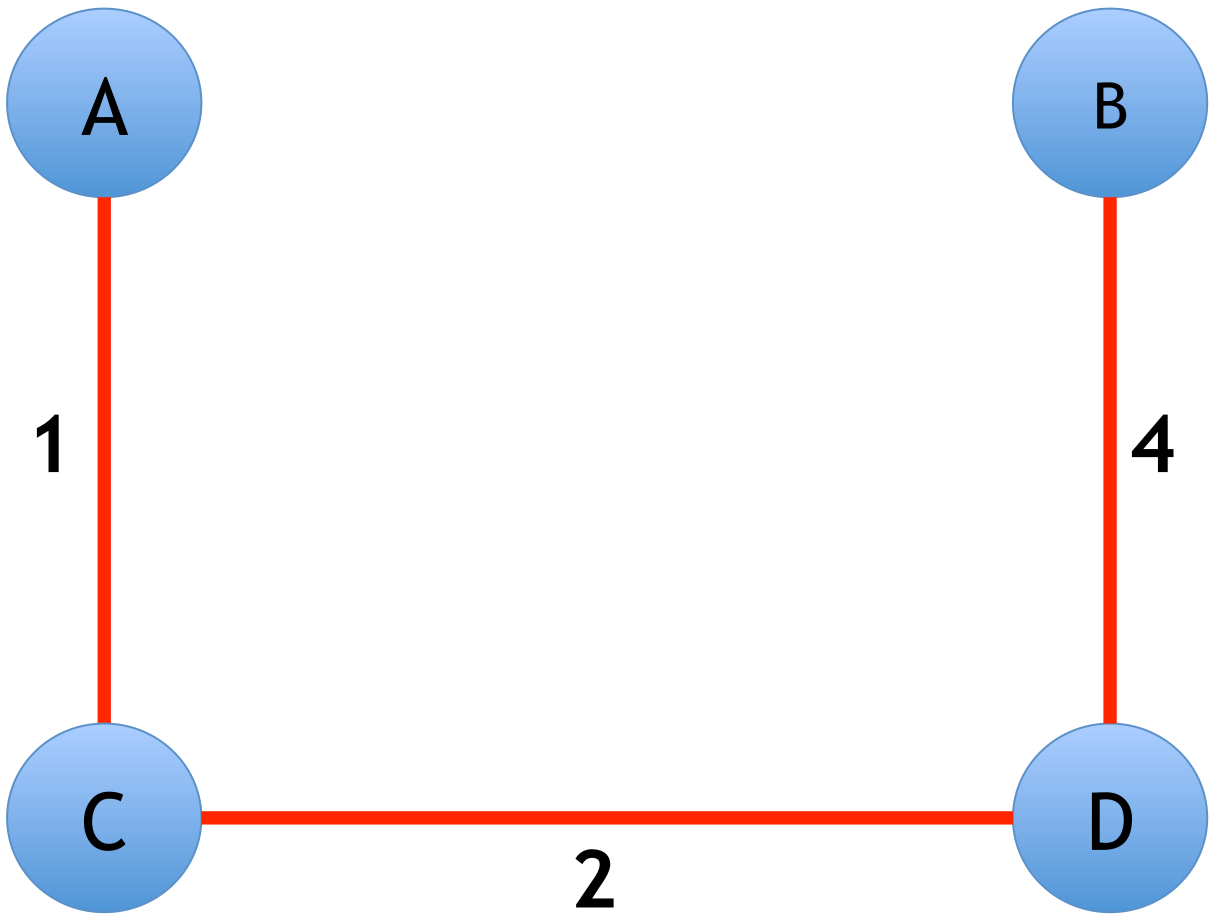


$W((A,C), (C,B), (B,D)) = 1 + 2.5 + 4 = 7.5$

Spanning Tree Example

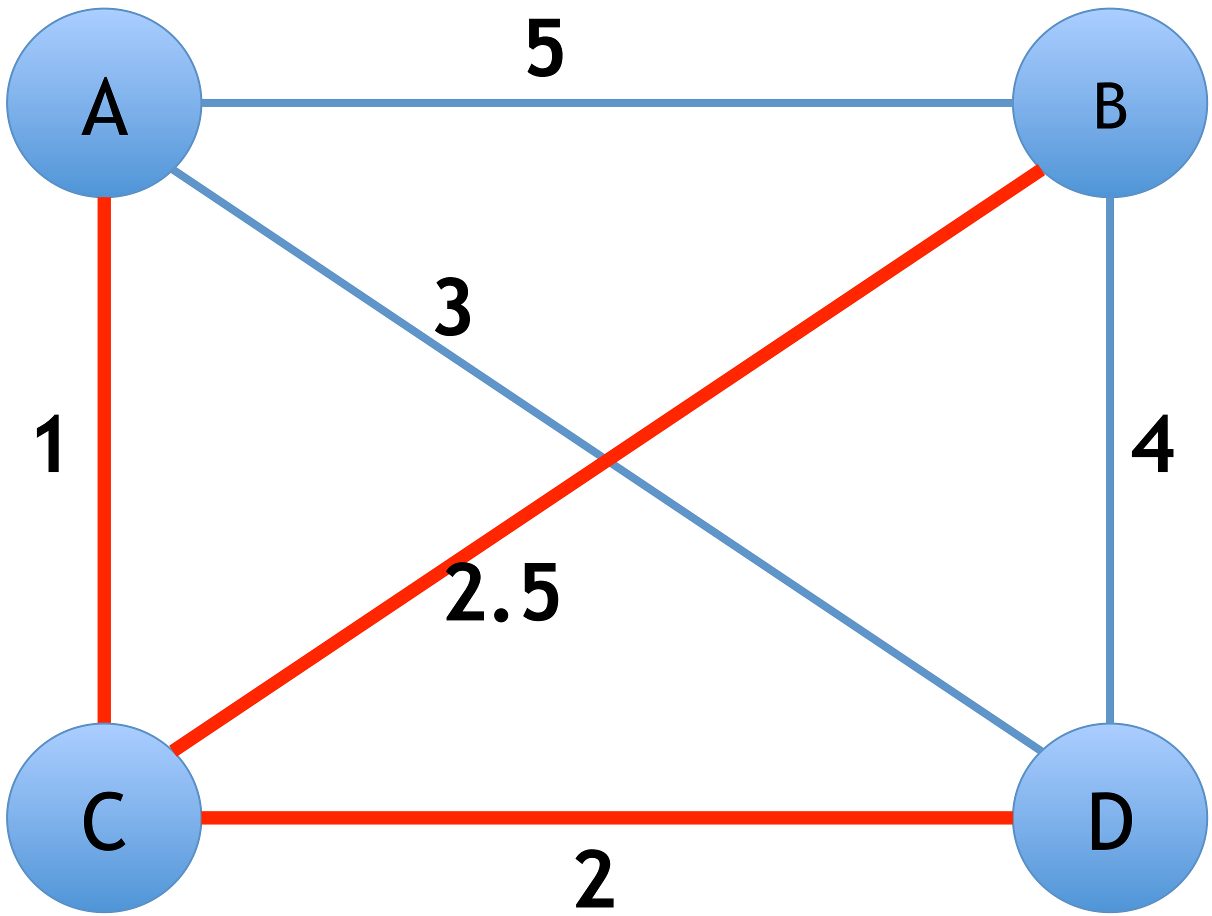


Spanning Tree Example

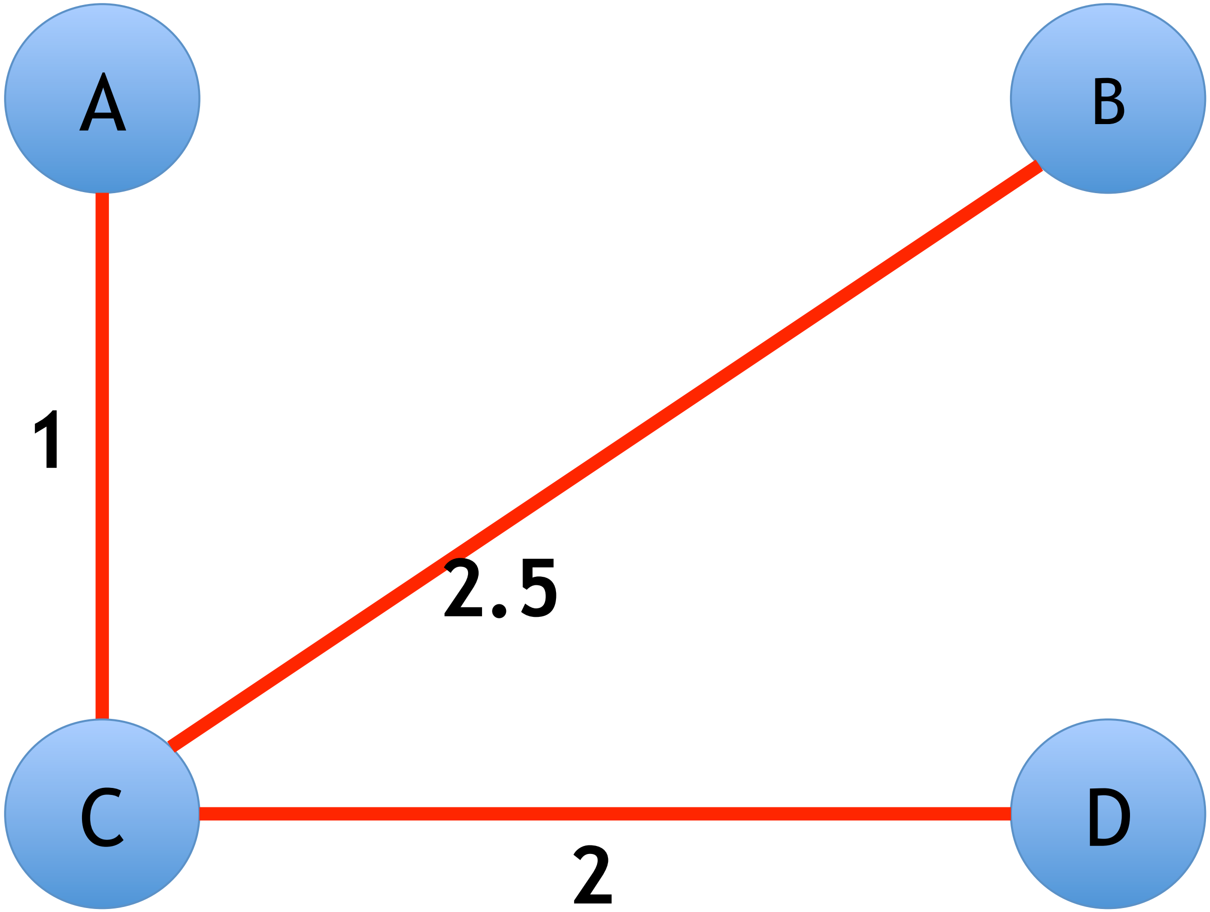


W((A,C), (C,D), (B,D)) = 1 + 2 + 4 = 7

Spanning Tree Example



Spanning Tree Example



W((A,C), (C,D), (C,B)) = 1 + 2 + 2.5 = 5.5

MST Applications

- Designing all kinds of networks:
  - datacenter networks
  - road networks
  - phone networks
- Circuit Design
- Clustering
- Image Segmentation

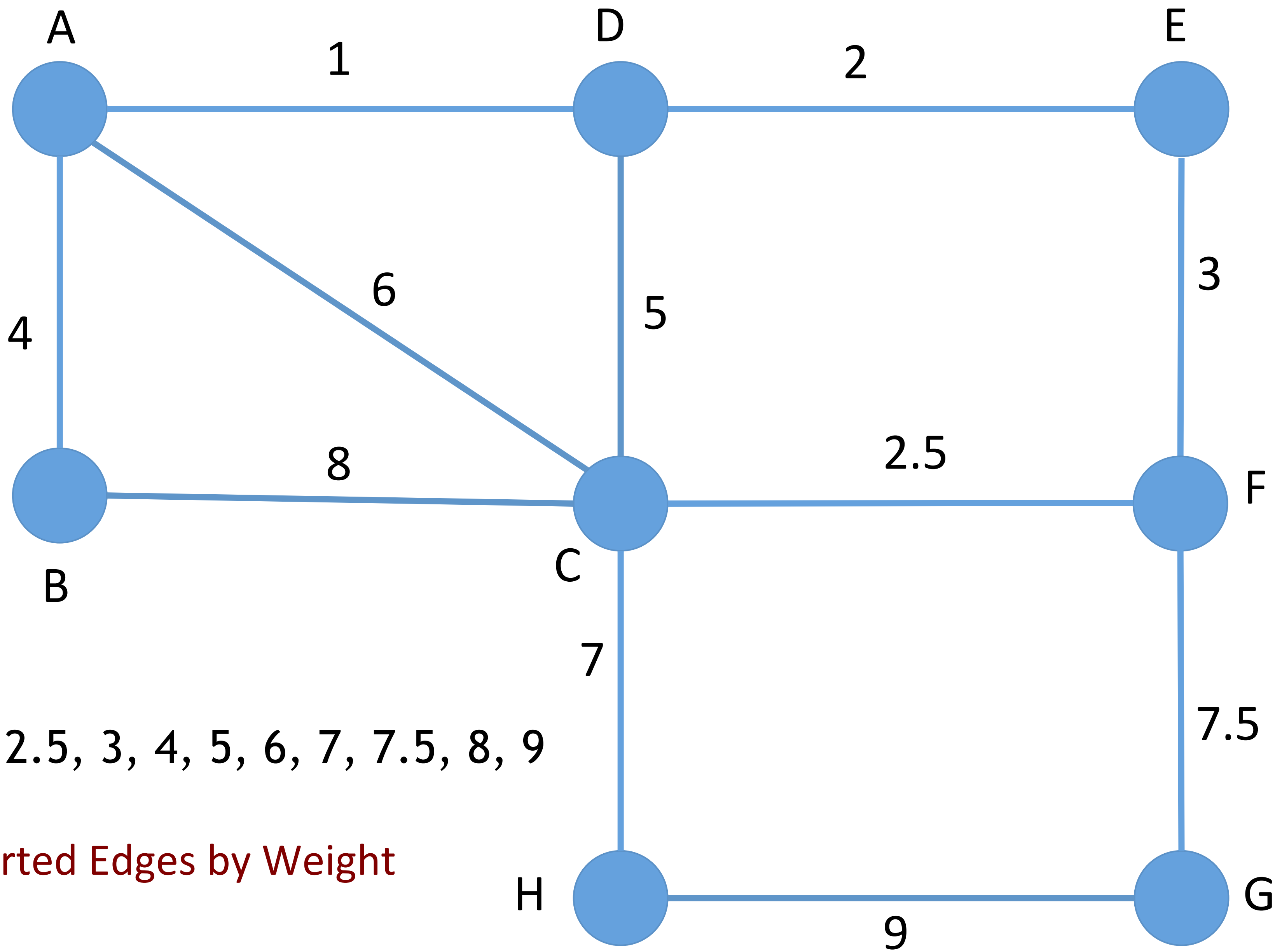
…

> 110 years old problem. Can find publications from 1906.

Outline For Today

1. Graph Theory Part 1: Trees & Properties of Trees
2. Minimum Spanning Trees
3. Kruskal's Algorithm
4. Graph Theory Part 2: Cuts & Properties of Cuts
5. Correctness Proof of Kruskal's Algorithm

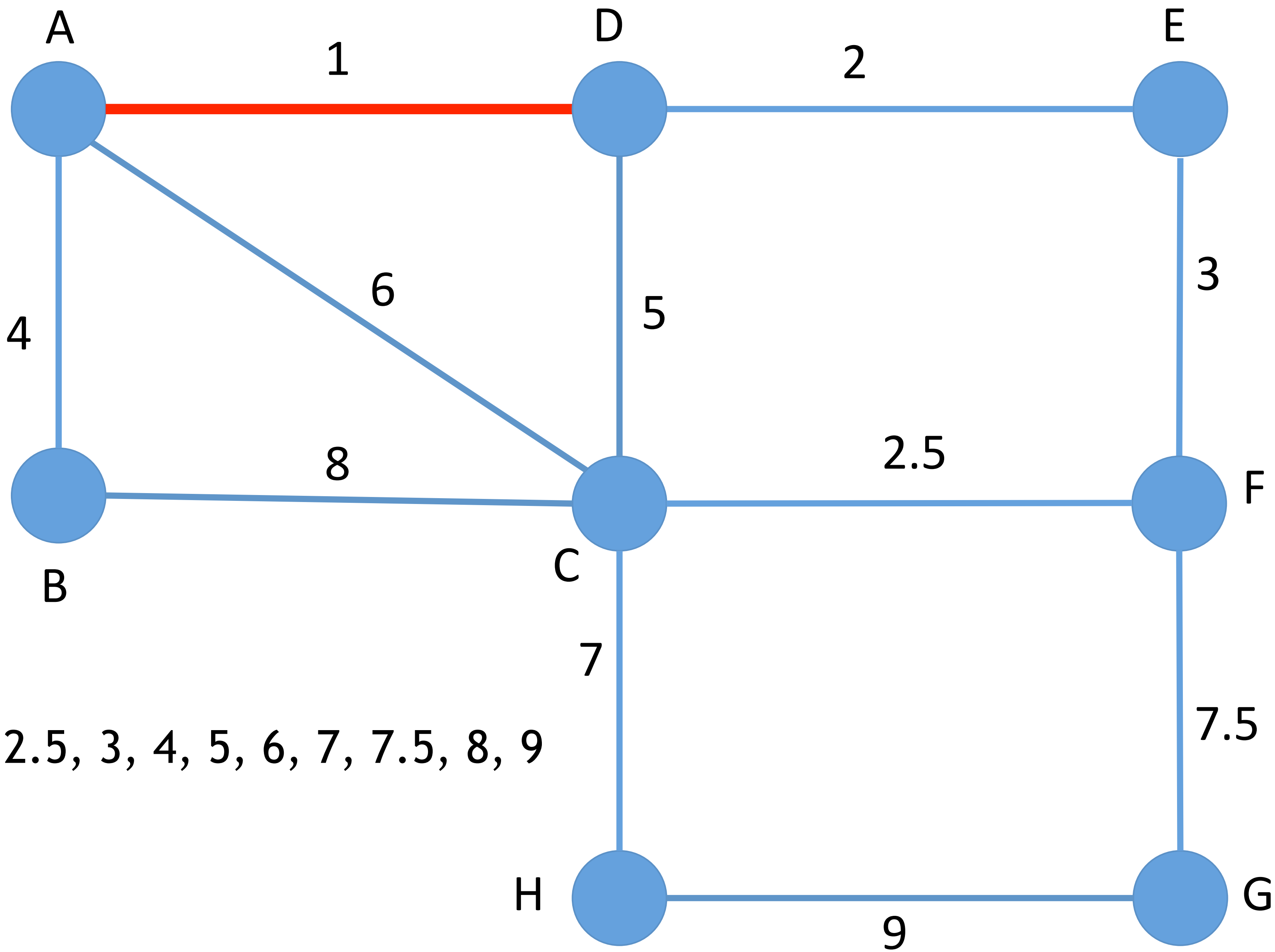
Kruskal’s Algorithm Simulation



1, 2, 2.5, 3, 4, 5, 6, 7, 7.5, 8, 9

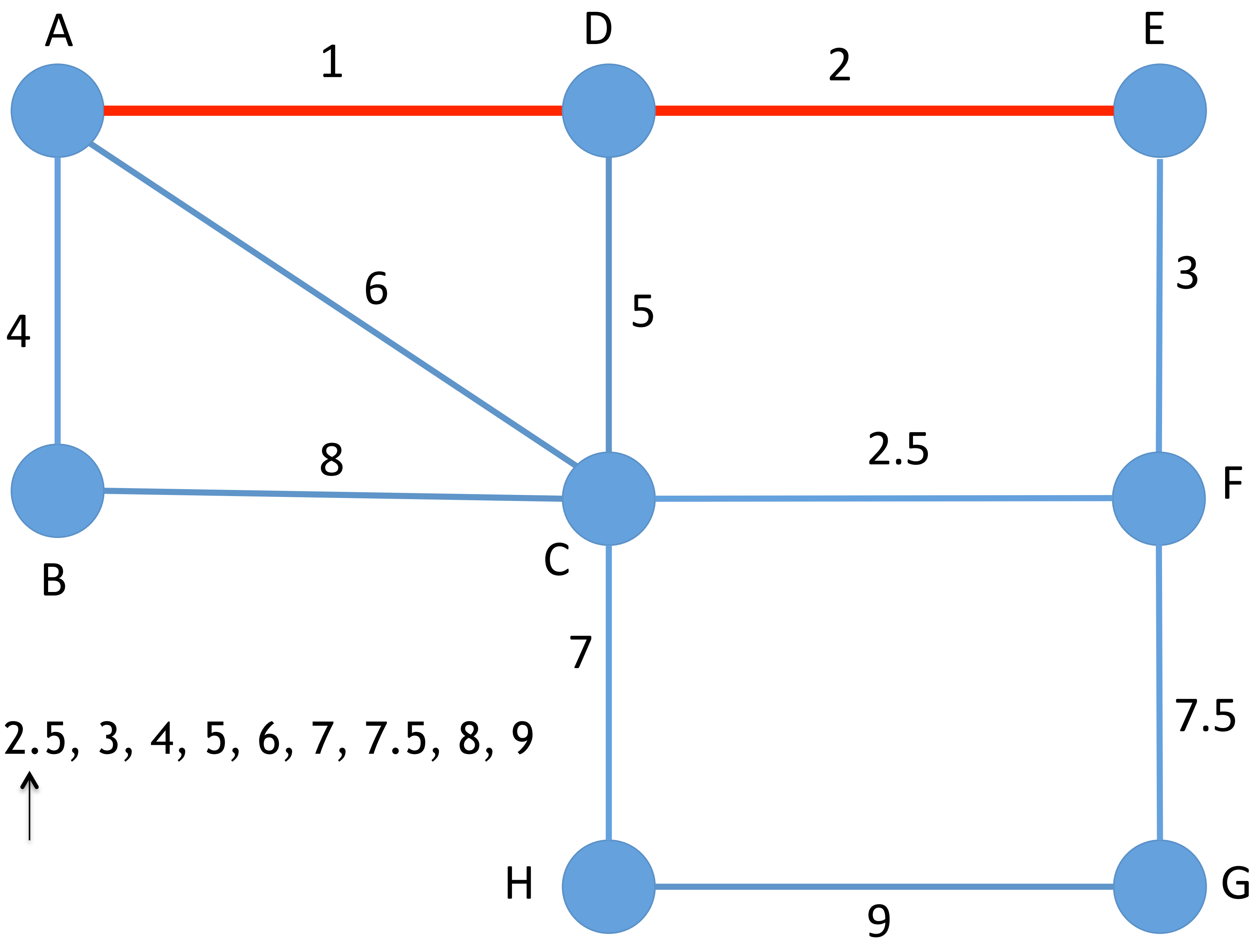
Sorted Edges by Weight

Kruskal's Algorithm Simulation



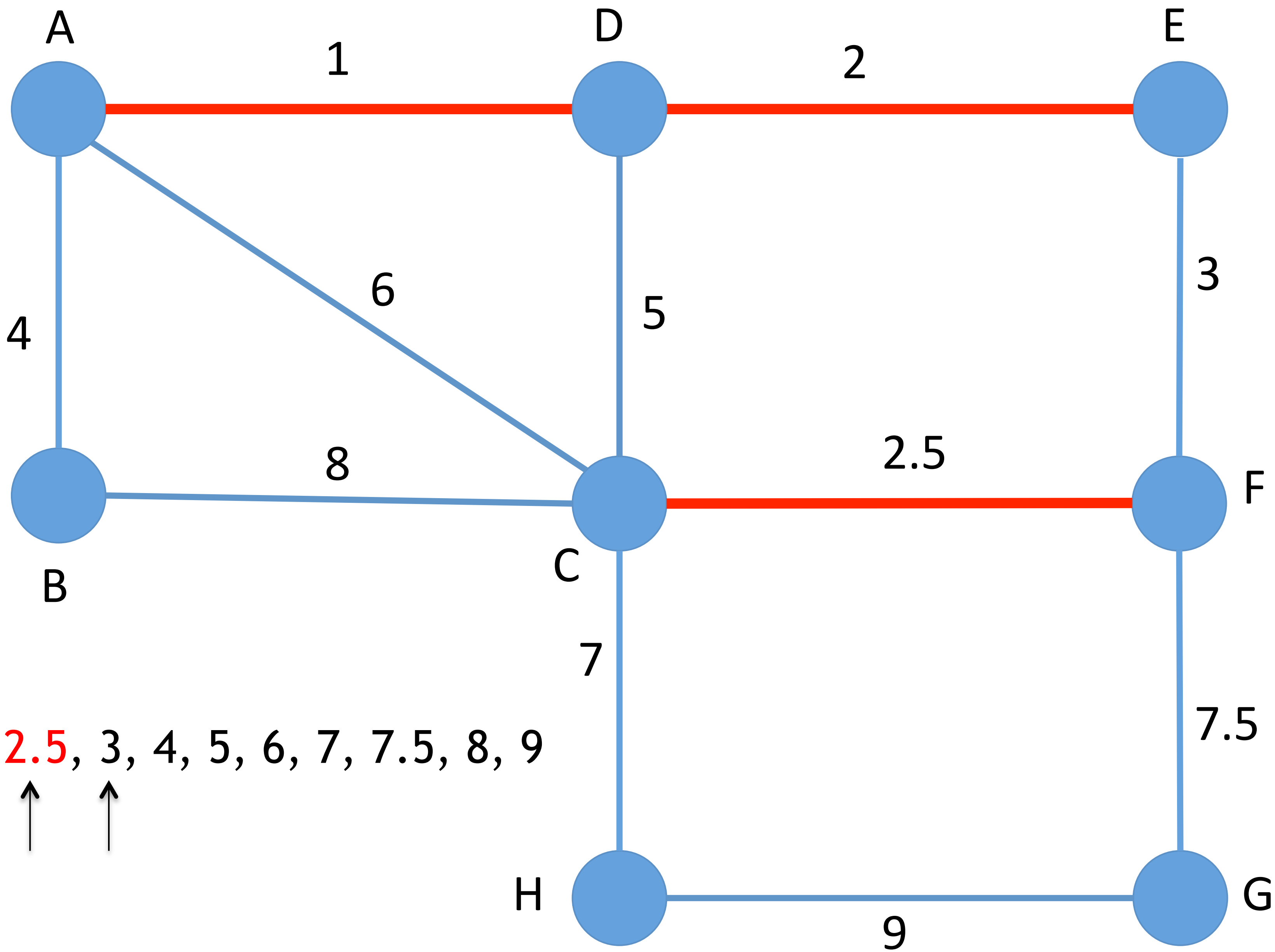
1, 2, 2.5, 3, 4, 5, 6, 7, 7.5, 8, 9

Kruskal's Algorithm Simulation



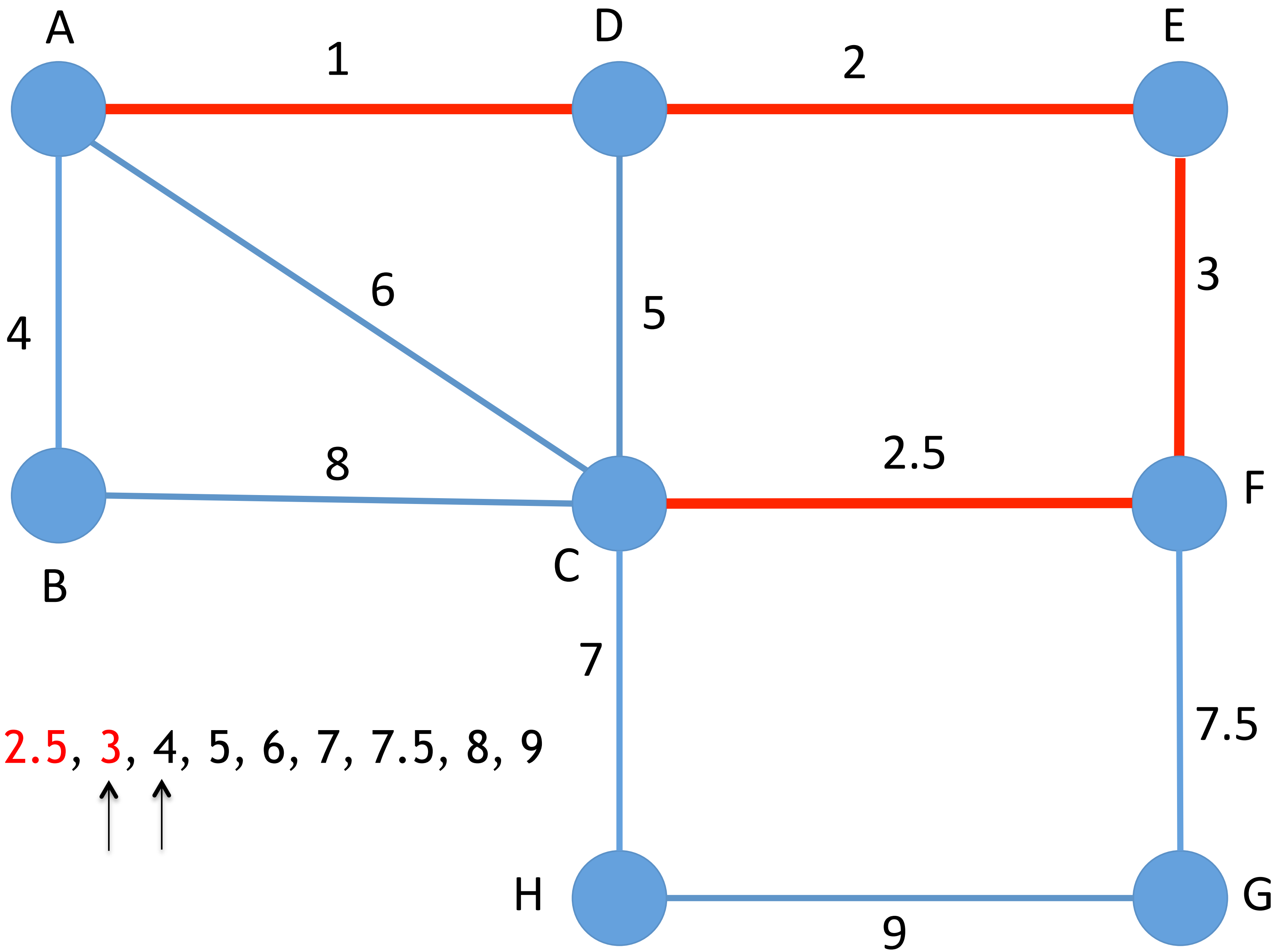
1, 2, 2.5, 3, 4, 5, 6, 7, 7.5, 8, 9

Kruskal's Algorithm Simulation



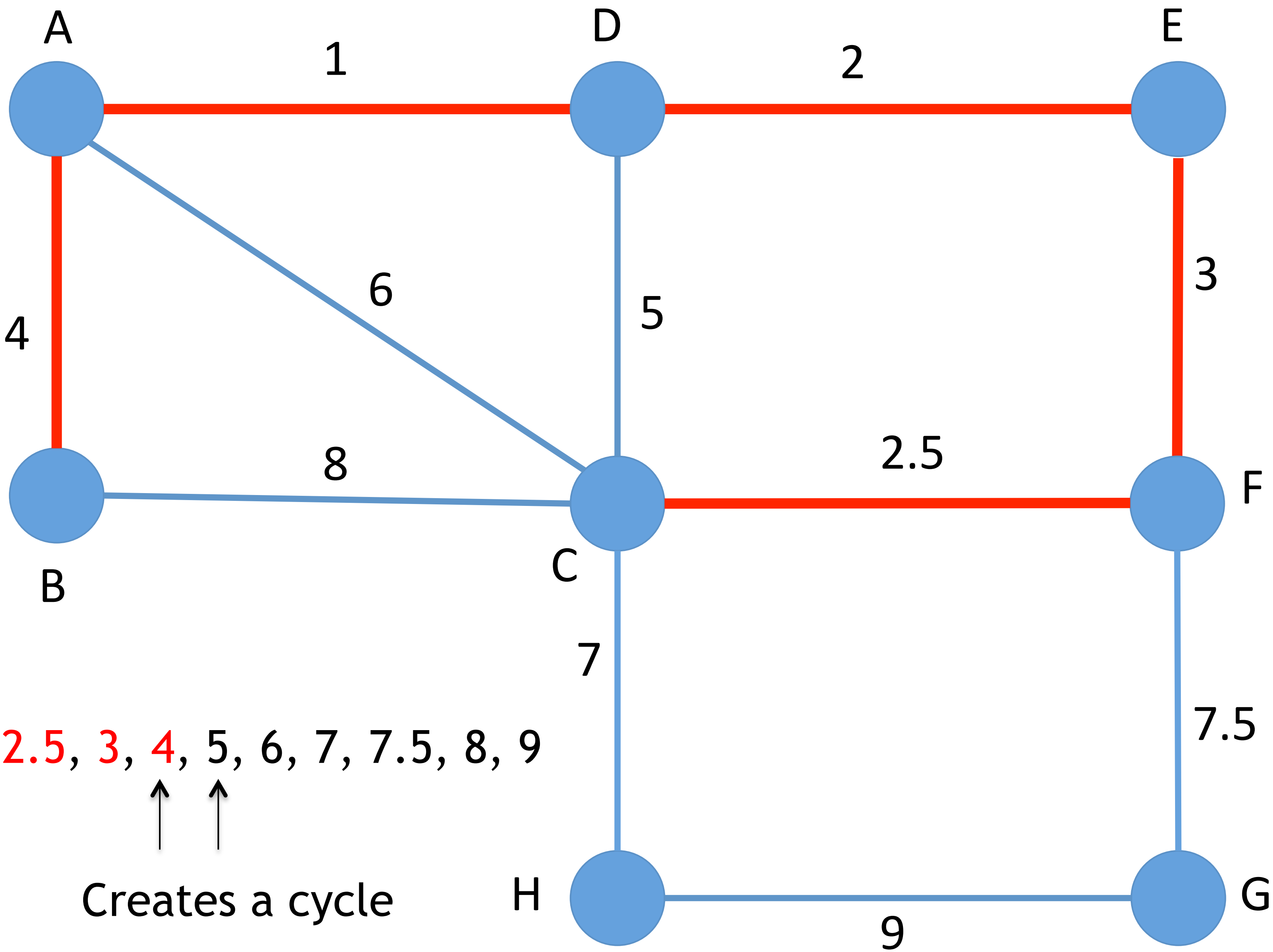
1, 2, 2.5, 3, 4, 5, 6, 7, 7.5, 8, 9

Kruskal's Algorithm Simulation



1, 2, 2.5, 3, 4, 5, 6, 7, 7.5, 8, 9

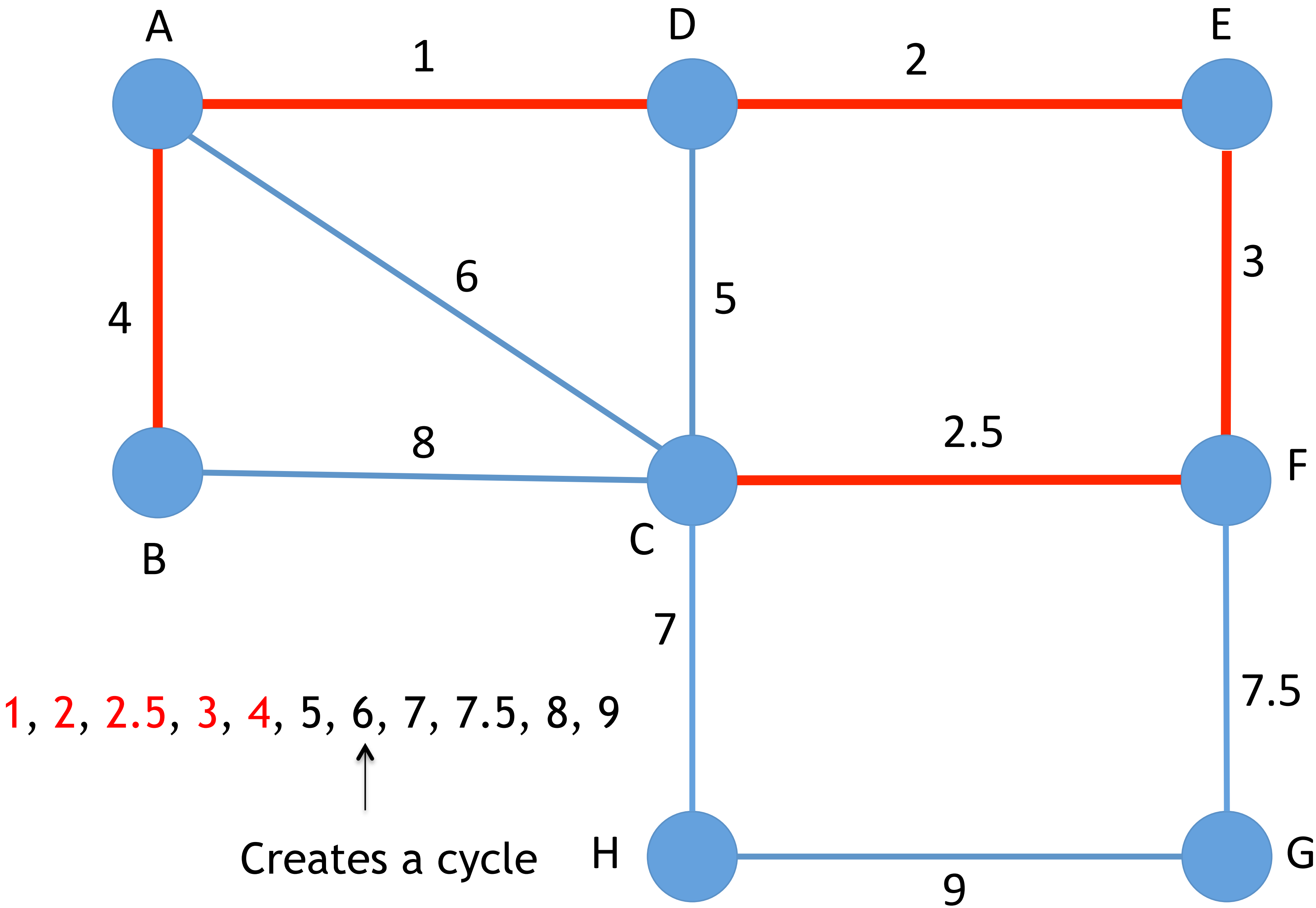
Kruskal's Algorithm Simulation



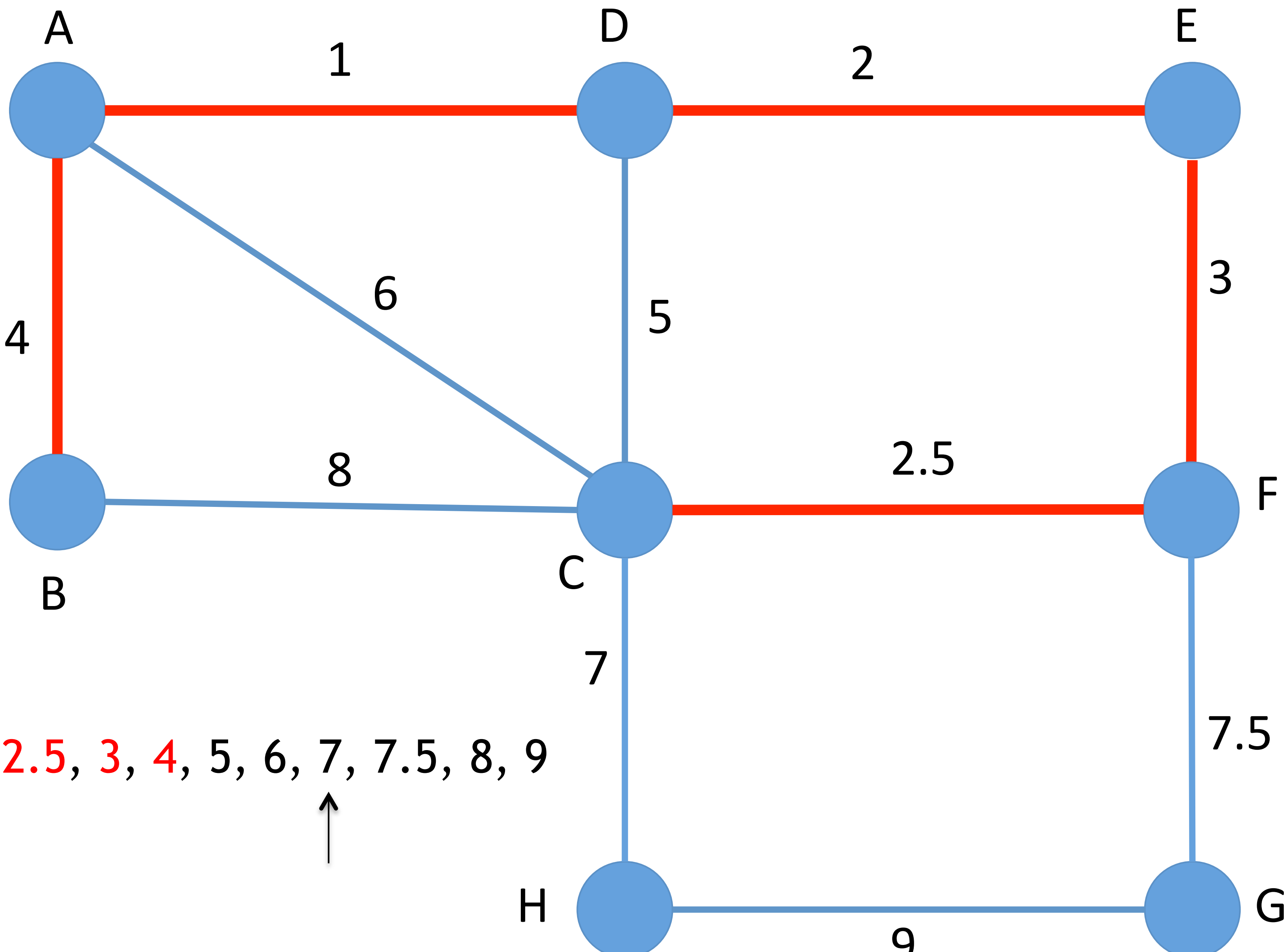
1, 2, 2.5, 3, 4, 5, 6, 7, 7.5, 8, 9

Creates a cycle

Kruskal's Algorithm Simulation

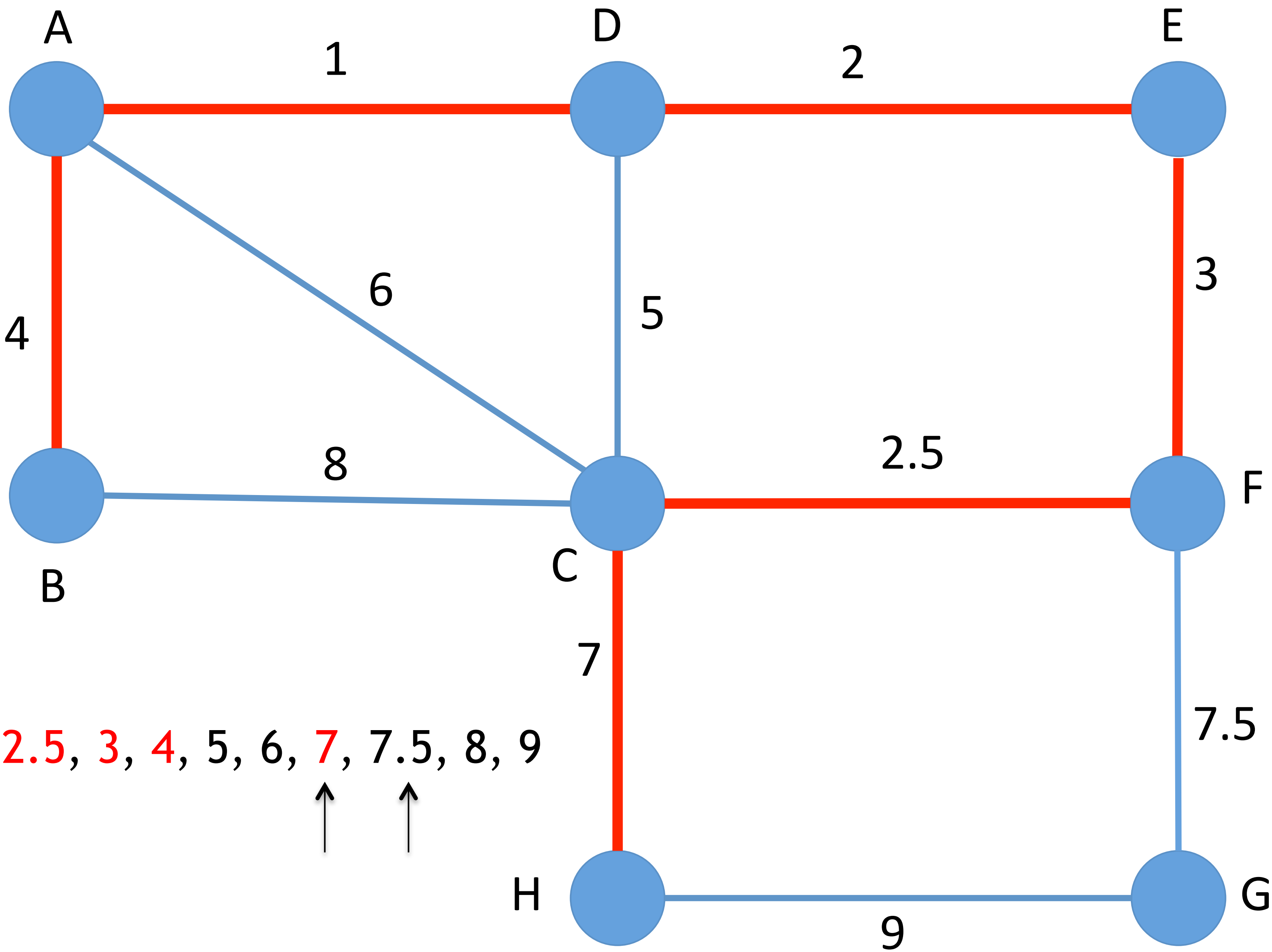


Kruskal's Algorithm Simulation



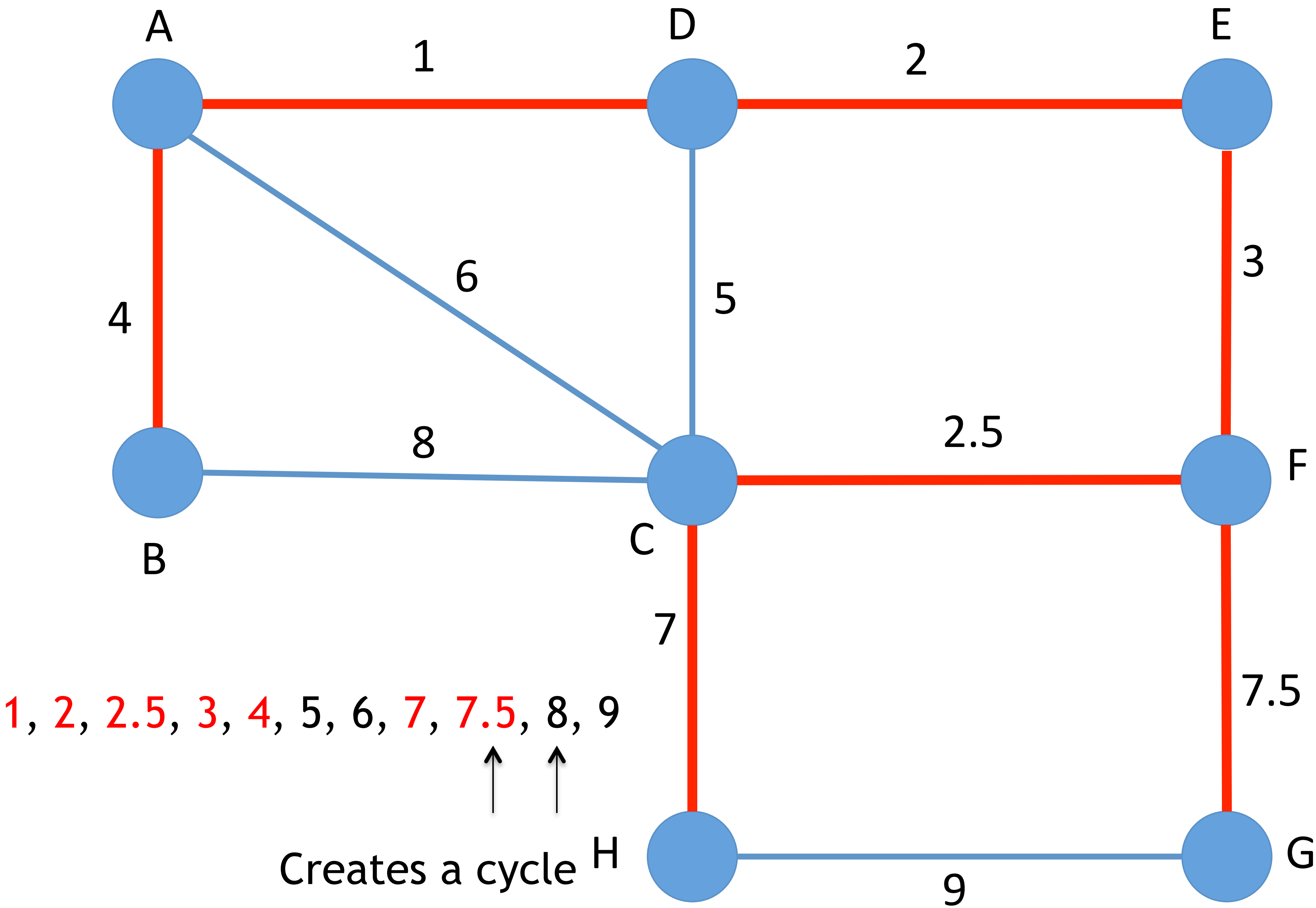
1, 2, 2.5, 3, 4, 5, 6, 7, 7.5, 8, 9

Kruskal's Algorithm Simulation

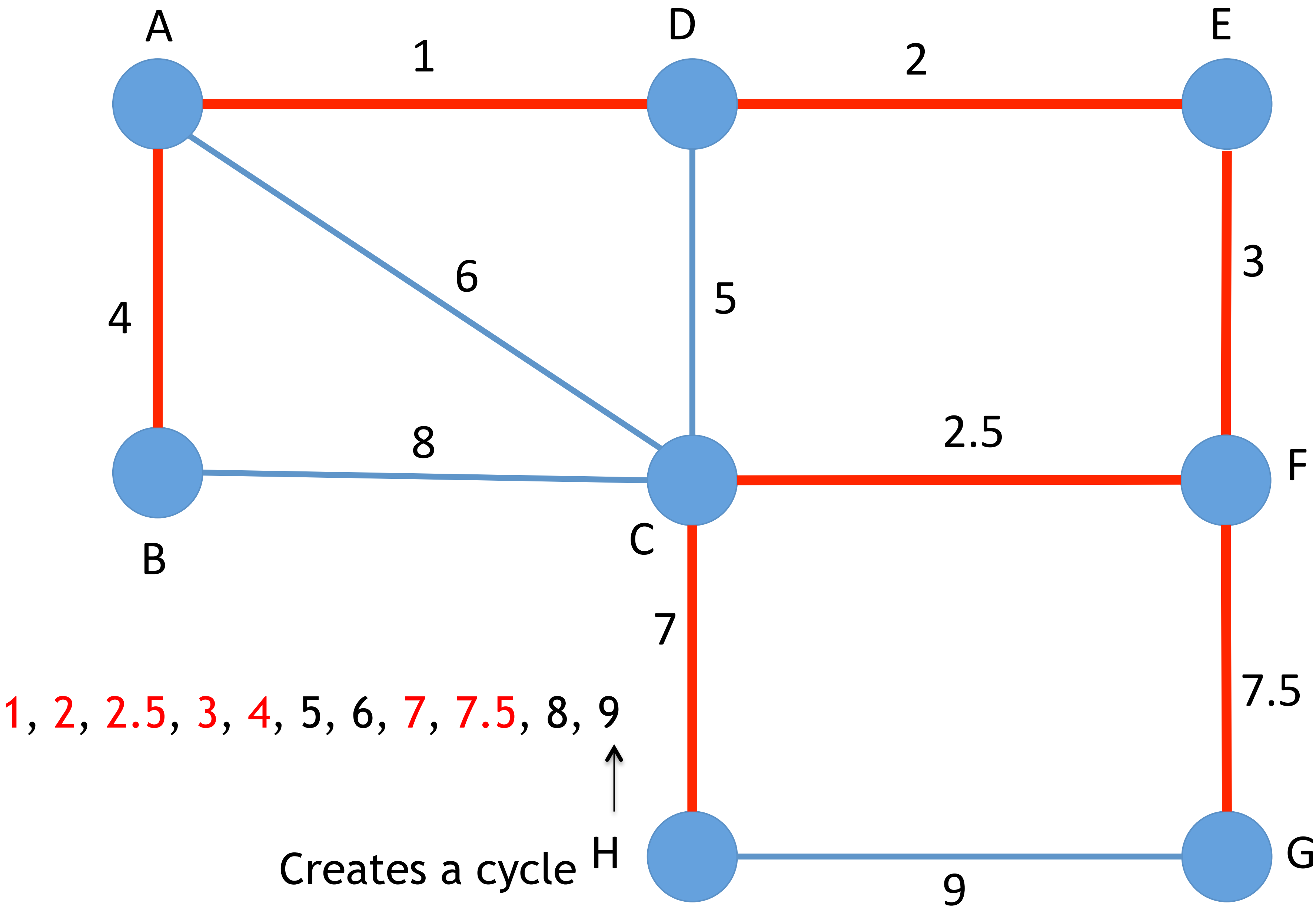


1, 2, 2.5, 3, 4, 5, 6, 7, 7.5, 8, 9

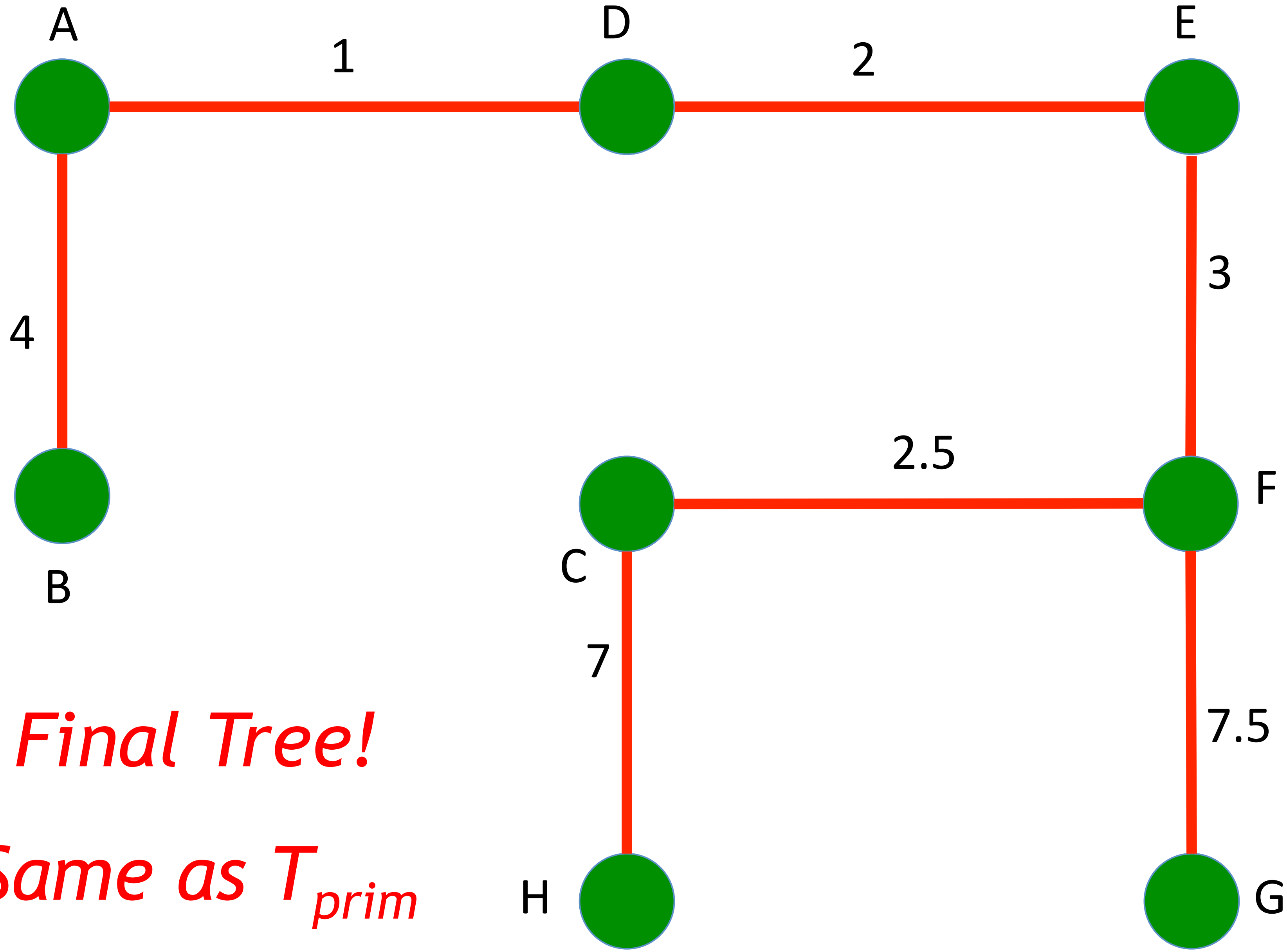
Kruskal's Algorithm Simulation



Kruskal’s Algorithm Simulation



Kruskal's Algorithm Simulation



Final Tree!

Same as T_{prim}

Kruskal's Algorithm Pseudocode (1956)

```
procedure kruskal(G(V, E)):
  sort E in order of increasing weights
  rename E so w(e1) < w(e2) < … < w(em)
  T = {} // final tree edges
  for i = 1 to m:
    if T ∪ ei=(u,v) doesn't create cycle
      add ei to T
  return T
```

O(mlog(n))

O(log(n)) by union-find data structure
(Verify as an exercise)
So in total: O(mlog(n))

Total Runtime:
O(mlog(n))

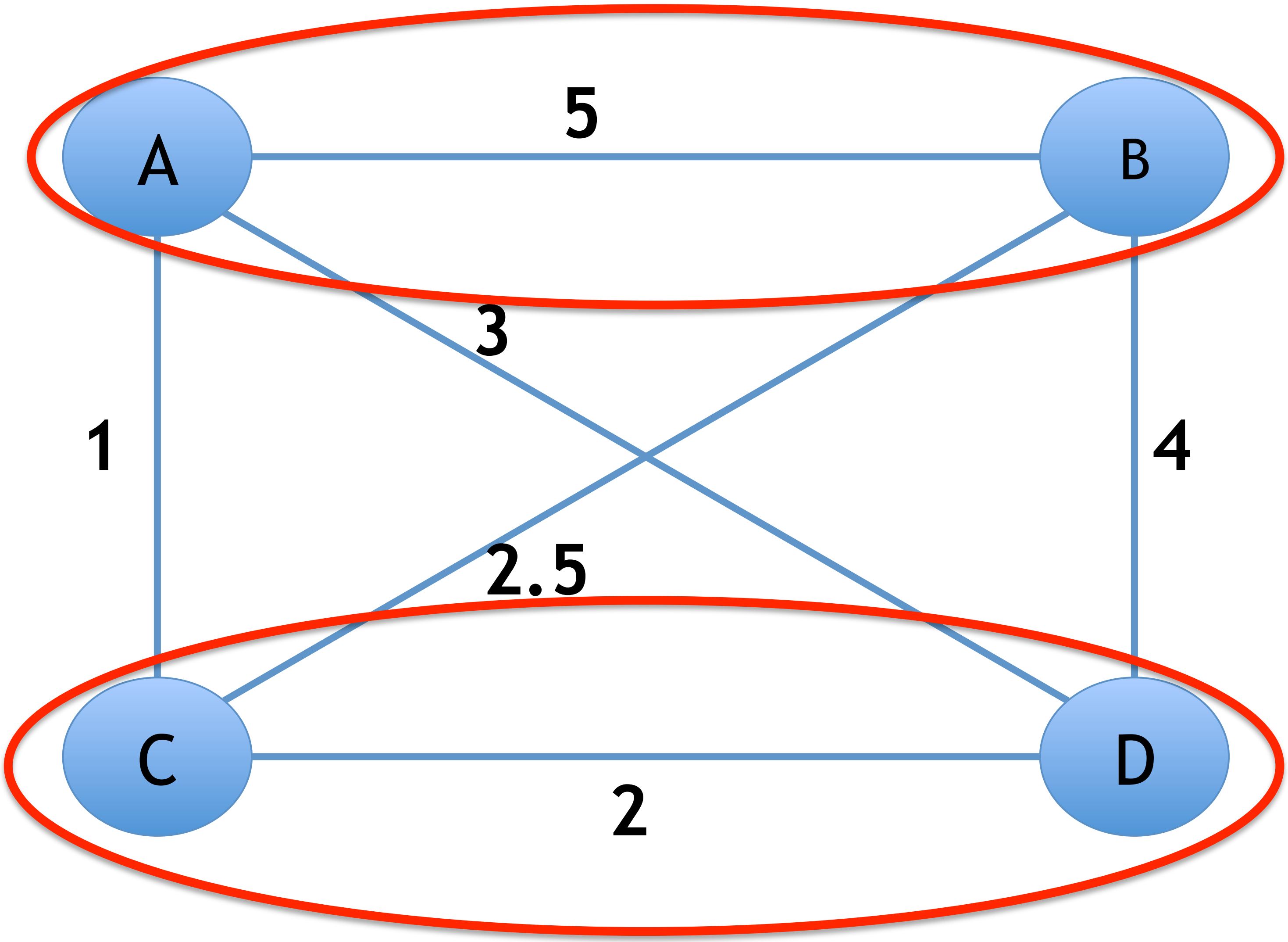
Correctness?

Outline For Today

1. Graph Theory Part 1: Trees & Properties of Trees
2. Minimum Spanning Trees
3. Kruskal's Algorithm
4. Graph Theory Part 2: Cuts & Properties of Cuts
5. Correctness Proof of Kruskal's Algorithm

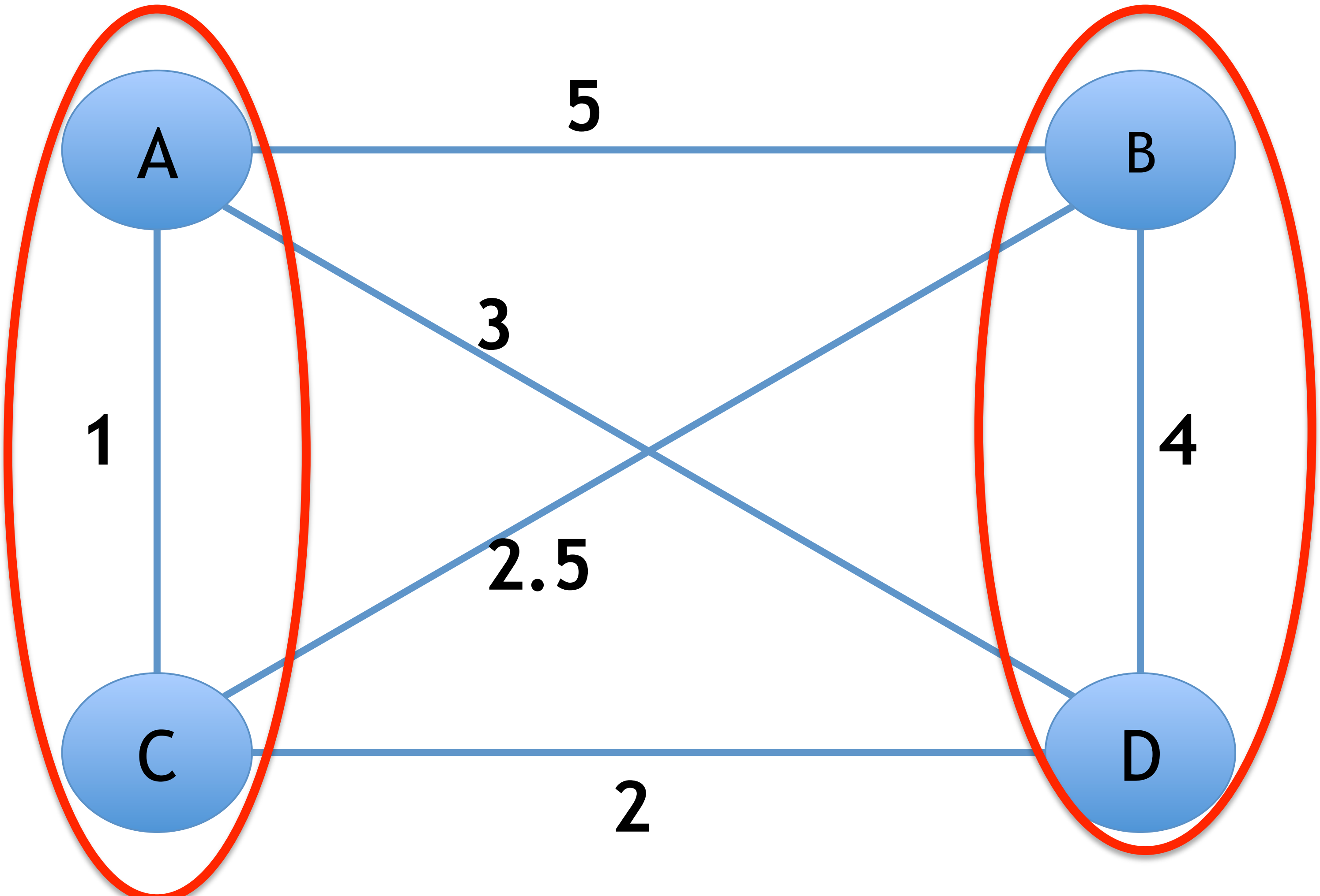
Cuts

- A cut of G(V, E) is a ***slice*** of V into 2 non-empty sets



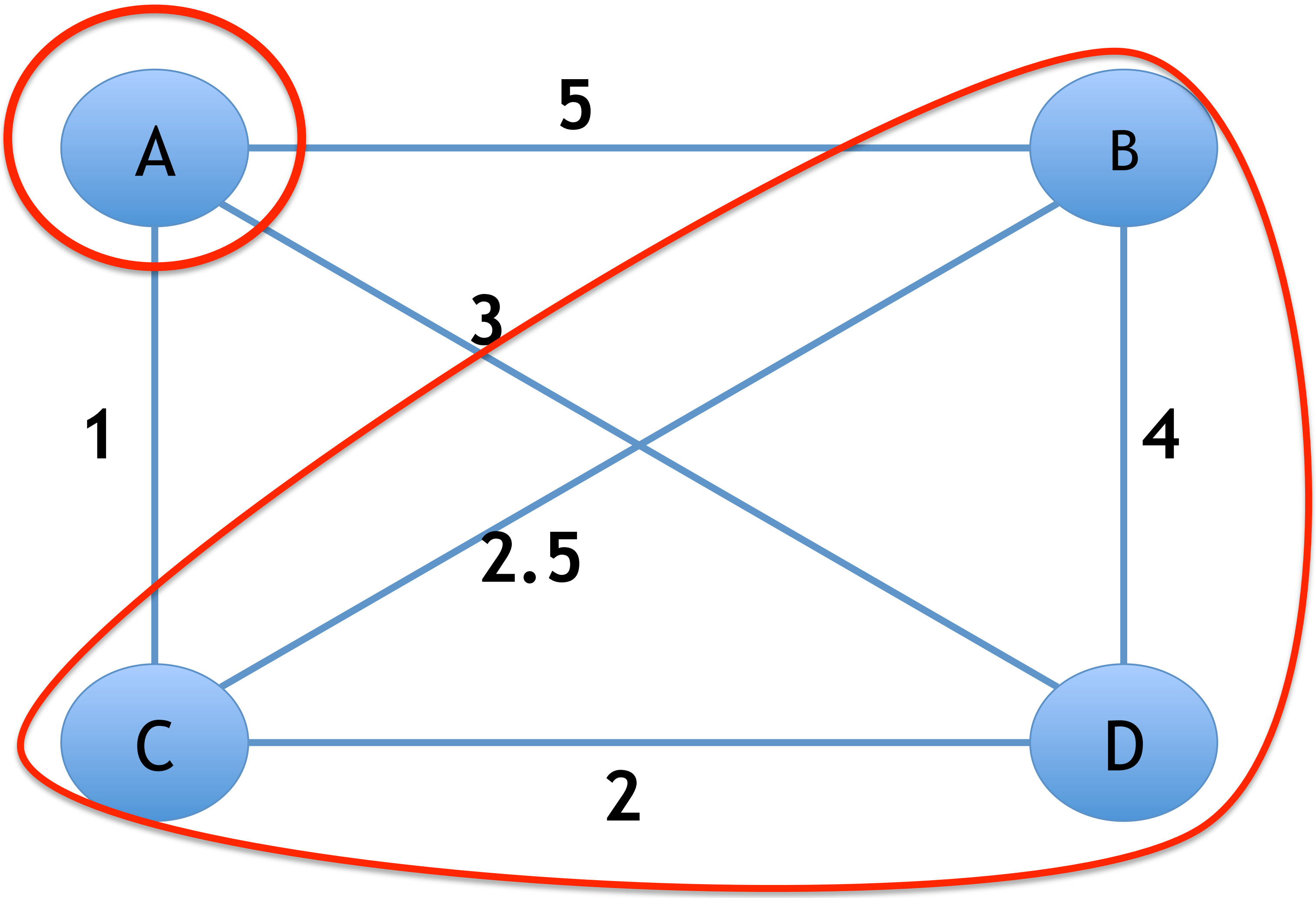
Cuts

- A cut of G(V, E) is a ***slice*** of V into 2 non-empty sets



Cuts

- A cut of G(V, E) is a ***slice*** of V into 2 non-empty sets



Empty Cut Lemma

A graph G(V, E) is disconnected $\Leftrightarrow$ *∃ cut (X, Y) with no crossing edges*

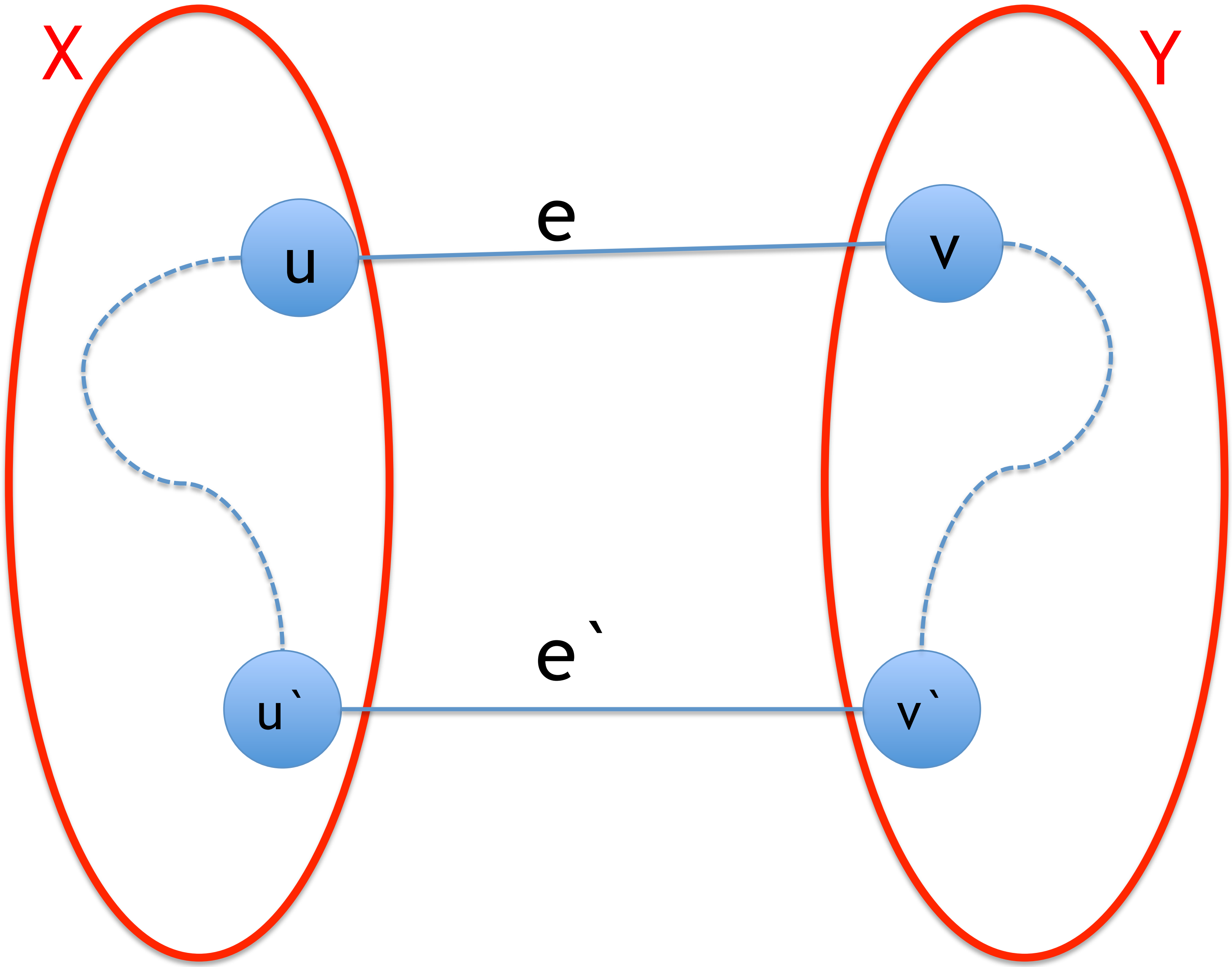
[prove as exercise]

Double Cut Crossing Lemma of Cycles

Suppose a cycle $C \subseteq E$ has an edge e crossing a cut (X, Y)

Claim: Then there is another e` ≠ e of C that also crosses (X, Y)

Double Cut Crossing Lemma of Cycles

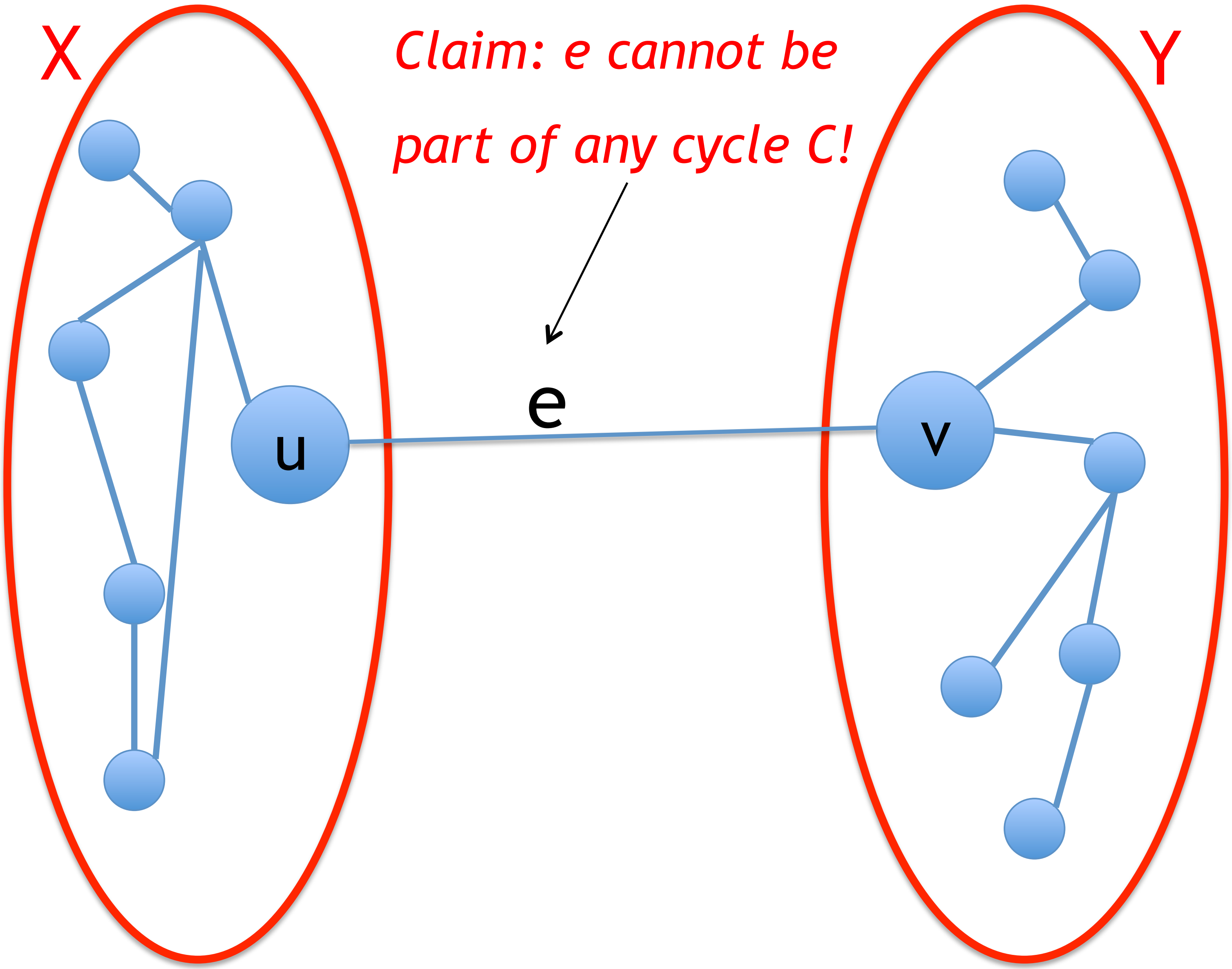


Lonely Cut Edge Corollary

Suppose there is a cut (X, Y) which has only one edge e crossing it

Claim: e cannot be part of any cycle C!

Lonely Cut Edge Corollary



Outline For Today

1. Graph Theory Part 1: Trees & Properties of Trees
2. Minimum Spanning Trees
3. Kruskal’s Algorithm
4. Graph Theory Part 2: Cuts & Properties of Cuts
5. Correctness Proof of Kruskal’s Algorithm

For Correctness We Need To Prove 2 Things

1. Outputs a Spanning Tree T_{krsk}
2. T_{krsk} is a minimum spanning tree

1: Kruskal Outputs a Spanning Tree (1)

Need to prove T_{krsk} is spanning AND is acyclic

Acyclic is by construction of the algorithm.

Why is T_{krsk} spanning (i.e., connected)?

Recall Empty Cut Lemma:

A graph is not connected iff $\exists$ cut (X, Y) with no crossing edges

$\Rightarrow$ *If all cuts have a crossing edge -> graph is connected!*

1: Kruskal Outputs a Spanning Tree (2)

Consider any cut (X, Y)

Since input G is connected, ∃ edges crossing (X, Y)

Let e be the min-weight edge*

Kruskal inspects all edges

*Consider the time when Kruskal inspects e**

Claim: No edge crossing (X, Y) is in T_{krsk}

And adding e to T_{krsk} cannot create a cycle => Why?*

(because of lonely cut edge corallary)

Therefore Kruskal will add e, so for each cut, there is an edge in T_{krsk} crossing it => T_{krsk} spans V.*

Plan for 2: T_{krsk} is a *Minimum* Spanning Tree

1. "MST Cut Property" of Edges
2. Argue that the MST Cut Property Implies T_{krsk} is an MST

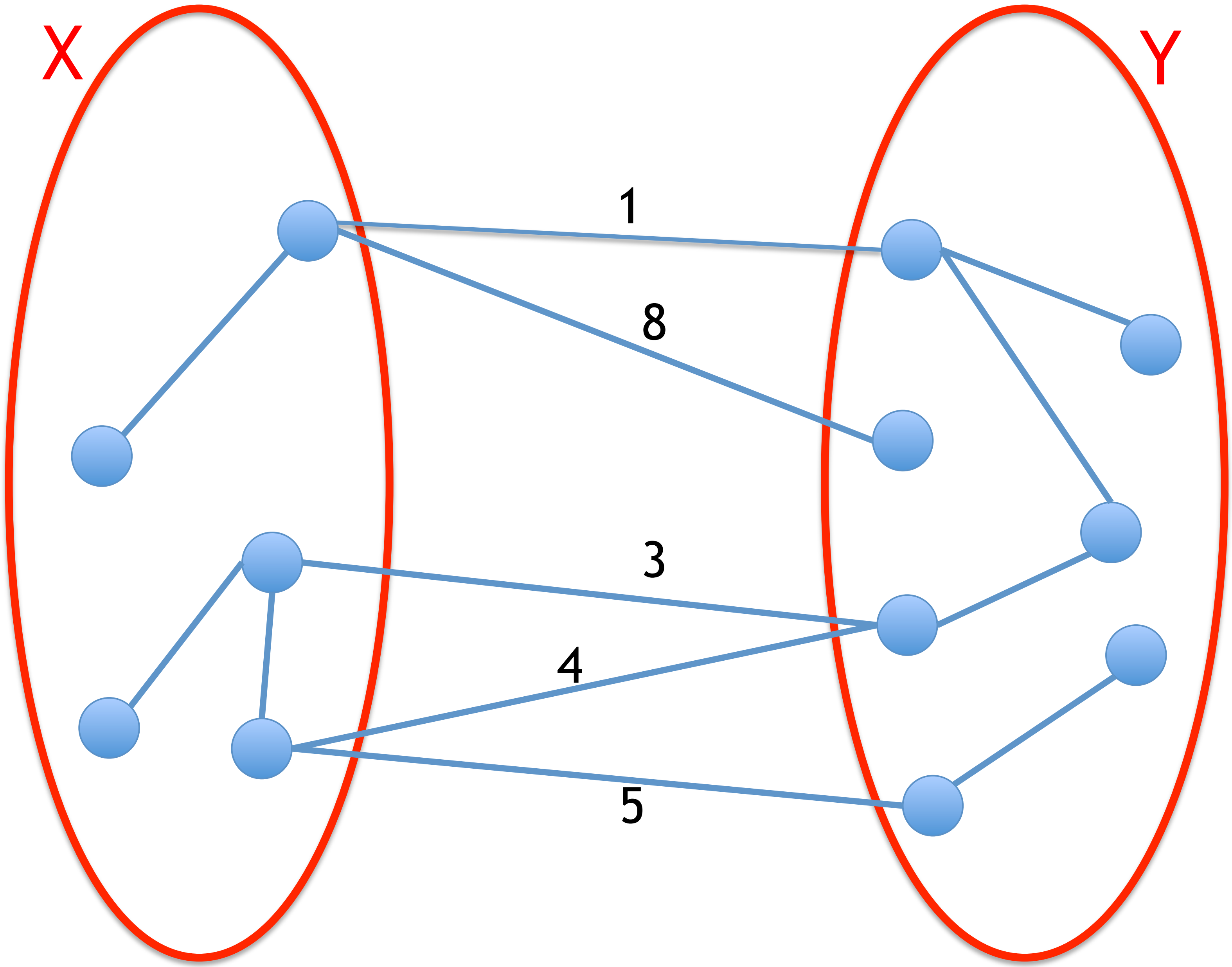
MST Cut Property of Edges

Consider any cut (X, Y) of G, and suppose e is the minimum edge crossing (X, Y)

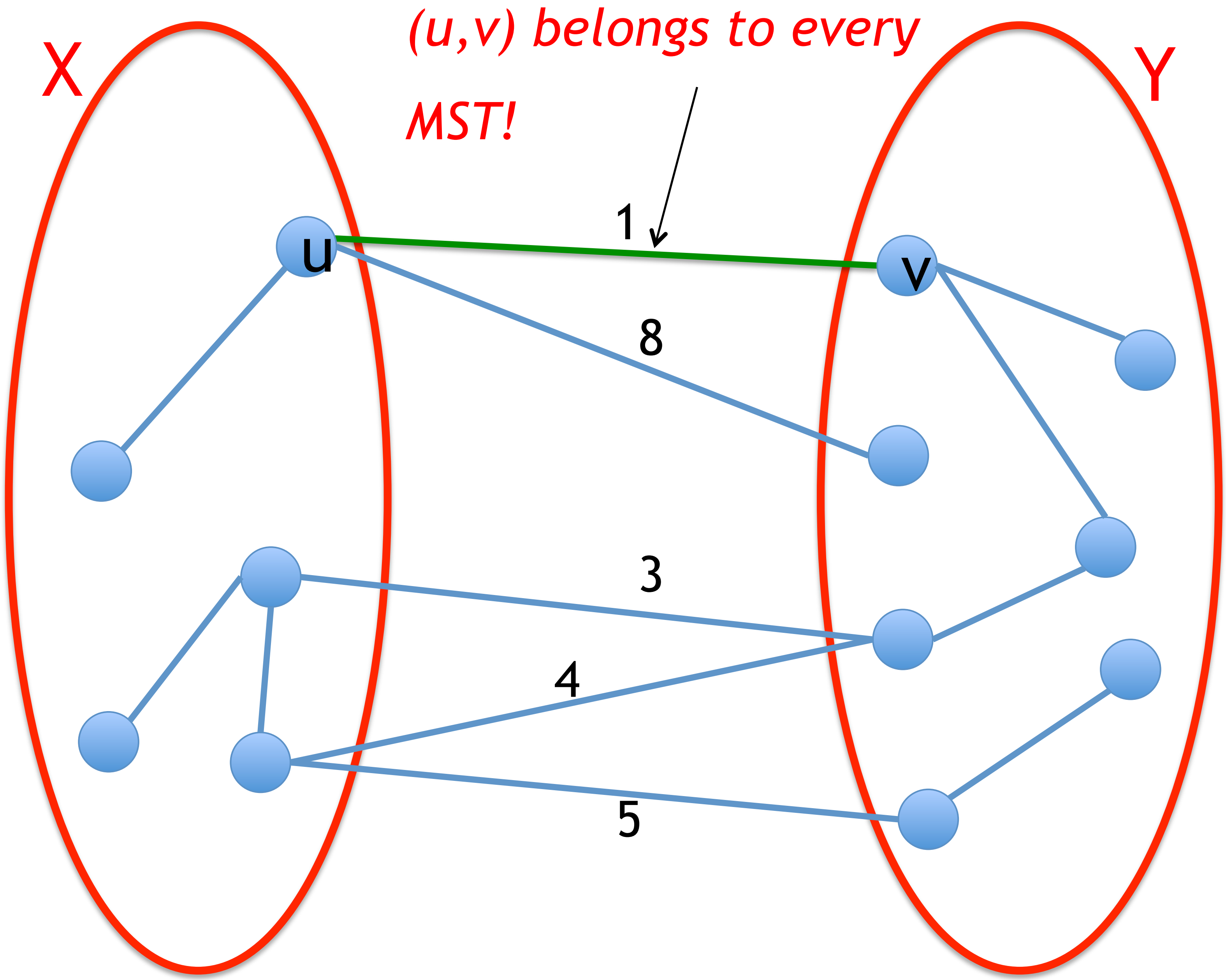
Claim: e belongs to every MST

(assuming edge weights are distinct)

MST Cut Property of Edges



MST Cut Property of Edges



Final Step: MST Cut Property => T_{krsk} is a MST

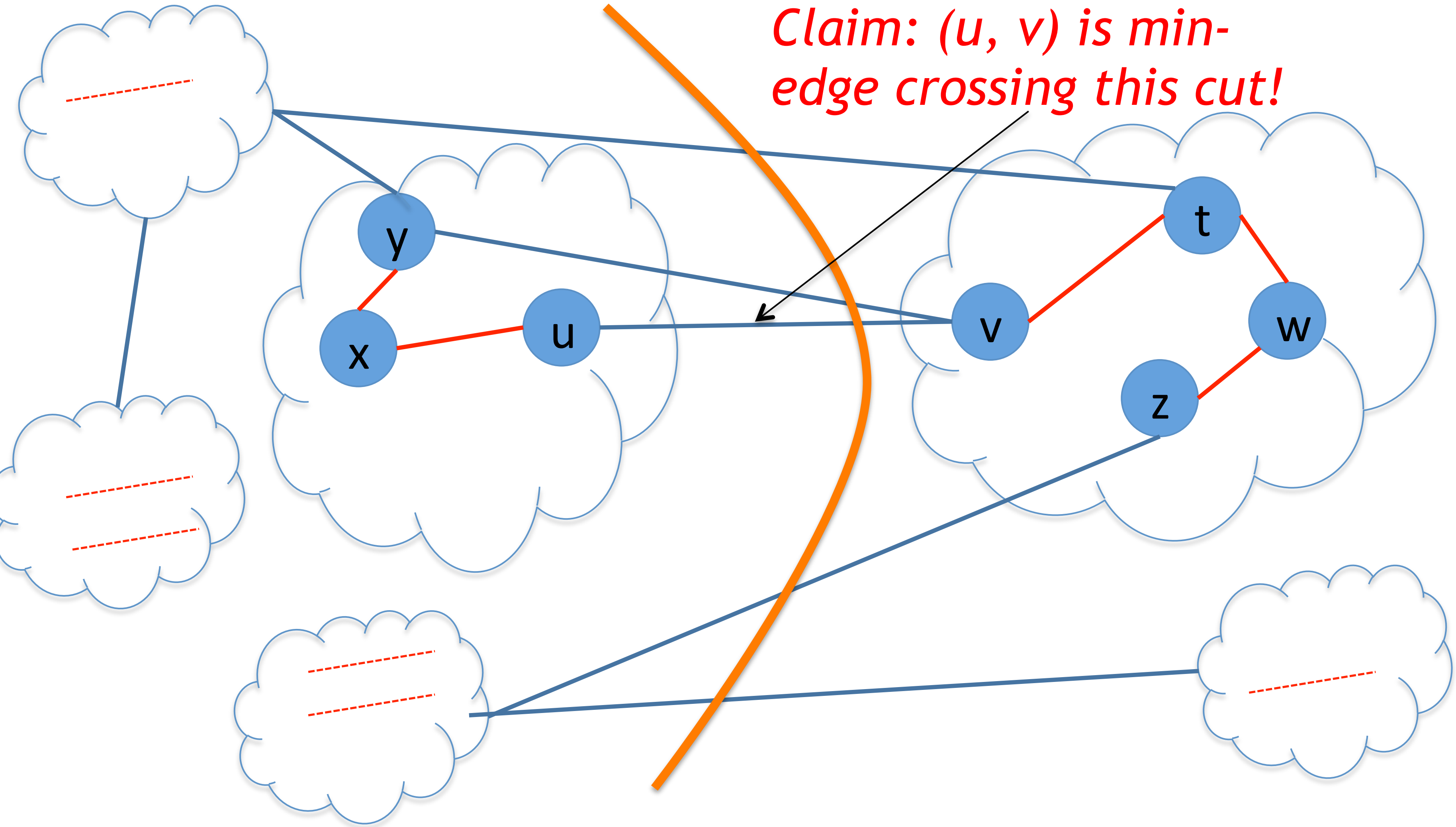
Argue that MST Cut Property Implies T_{krsk} is an MST

In particular we will argue that each edge (u, v) added to T_{krsk} is a min-edge in a cut (X, Y) in G.

MST Cut Property => T_{krsk} is a MST (1)

Let (u, v) be any edge added by Kruskal's Algorithm.

u and v are in different comp. (b/c Kruskal checks for cycles)



MST Cut Property => T_{krsk} is a MST (2)

(u, v) is the min edge crossing the cut b/c Kruskal looks at edges in increasing weights.

By the Cut Property => (u, v) is in every MST.

Therefore all edges added to T_{krsk} is in every MST

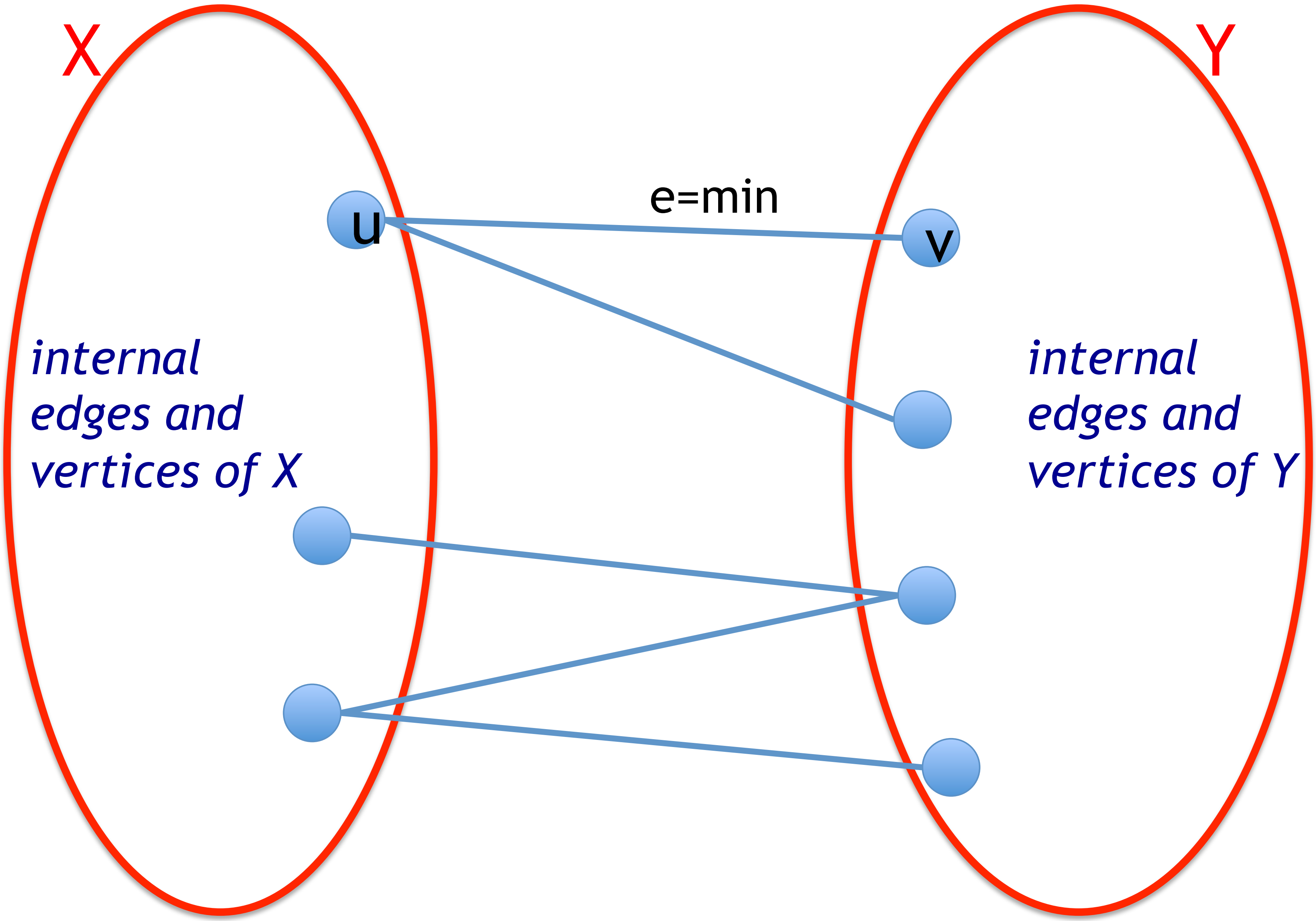
=> T_{krsk} is "the" MST.

Q.E.D.

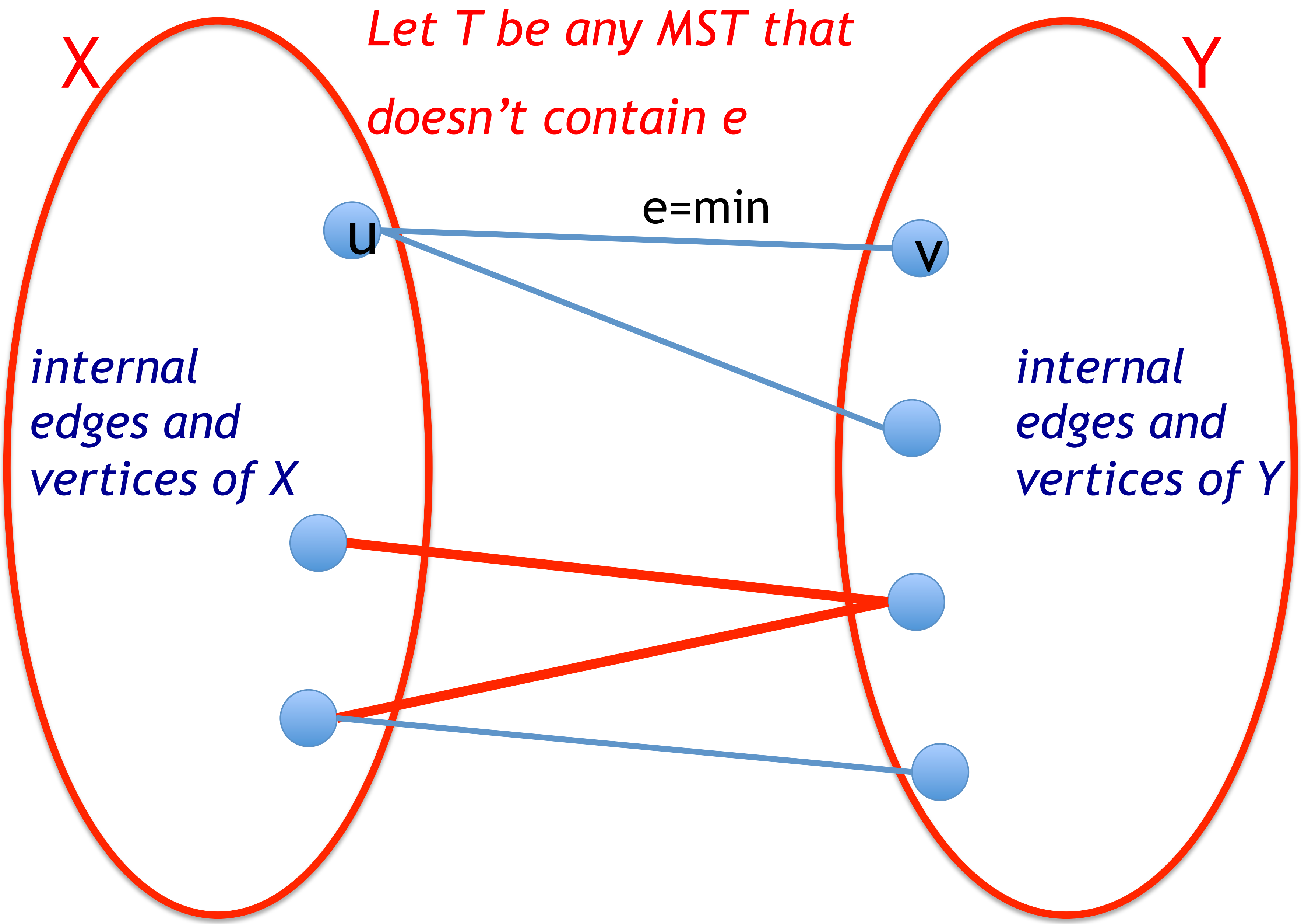
Summary of Kruskal's Correctness

- First proved T_{krsk} is a spanning tree
  - Acyclicity was by definition
  - Connectedness followed from showing that there is an edge of T_{krsk} crossing any cut (X, Y) of G, which by the empty cut lemma implied that T_{krsk} was connected
- Second proved T_{krsk} was a *minimum* spanning tree by arguing that any edge (u, v) if T_{krsk} is a min-edge in some cut (X, Y)
- Note this version of Kruskal's Correctness Proof contains:
  - Exchange Argument: MST Cut Property
  - Greedy Stays Ahead Argument: Kruskal adds n-1 edges and all are justified.

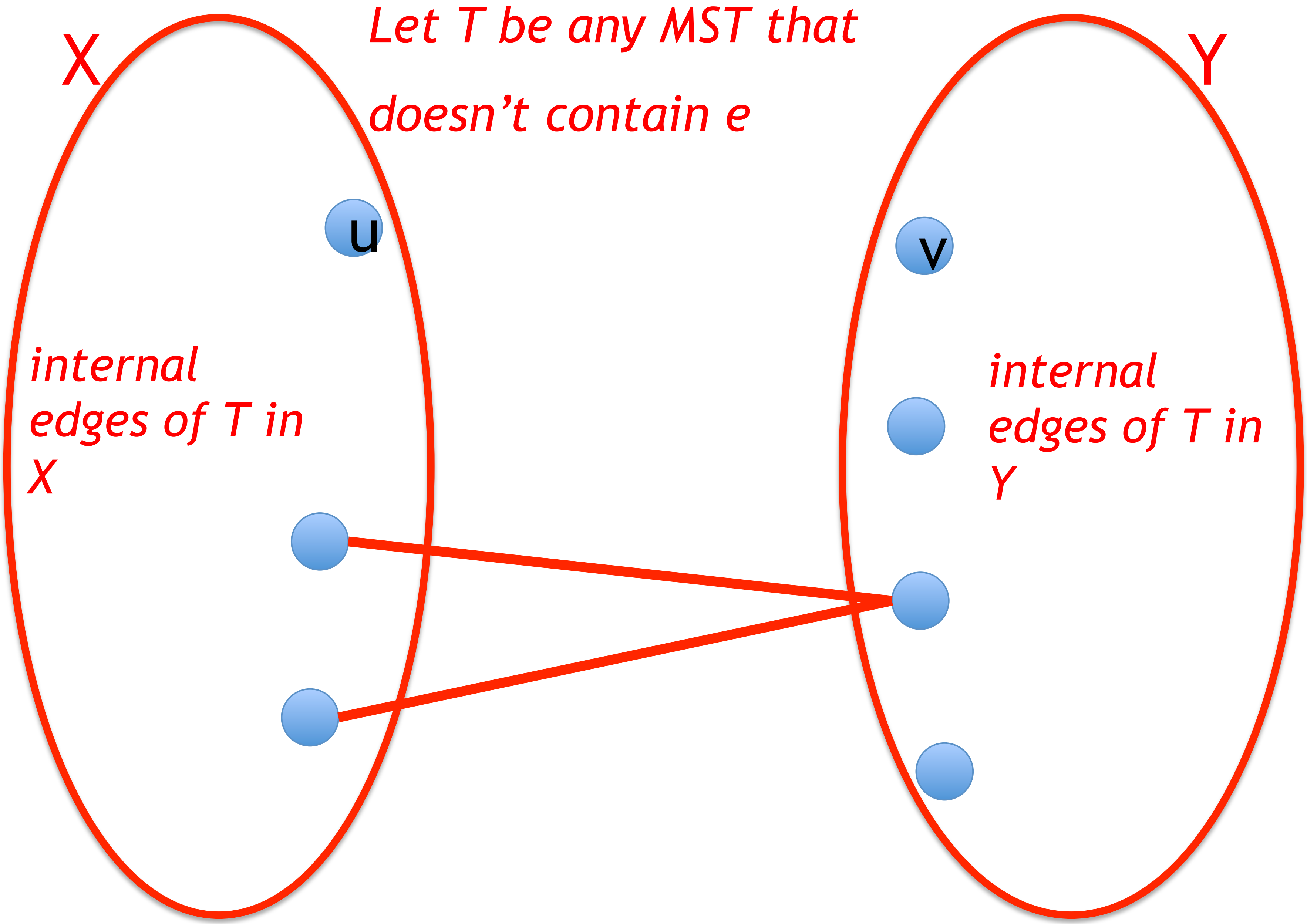
Proof of MST Cut Property (Exchange Argument)



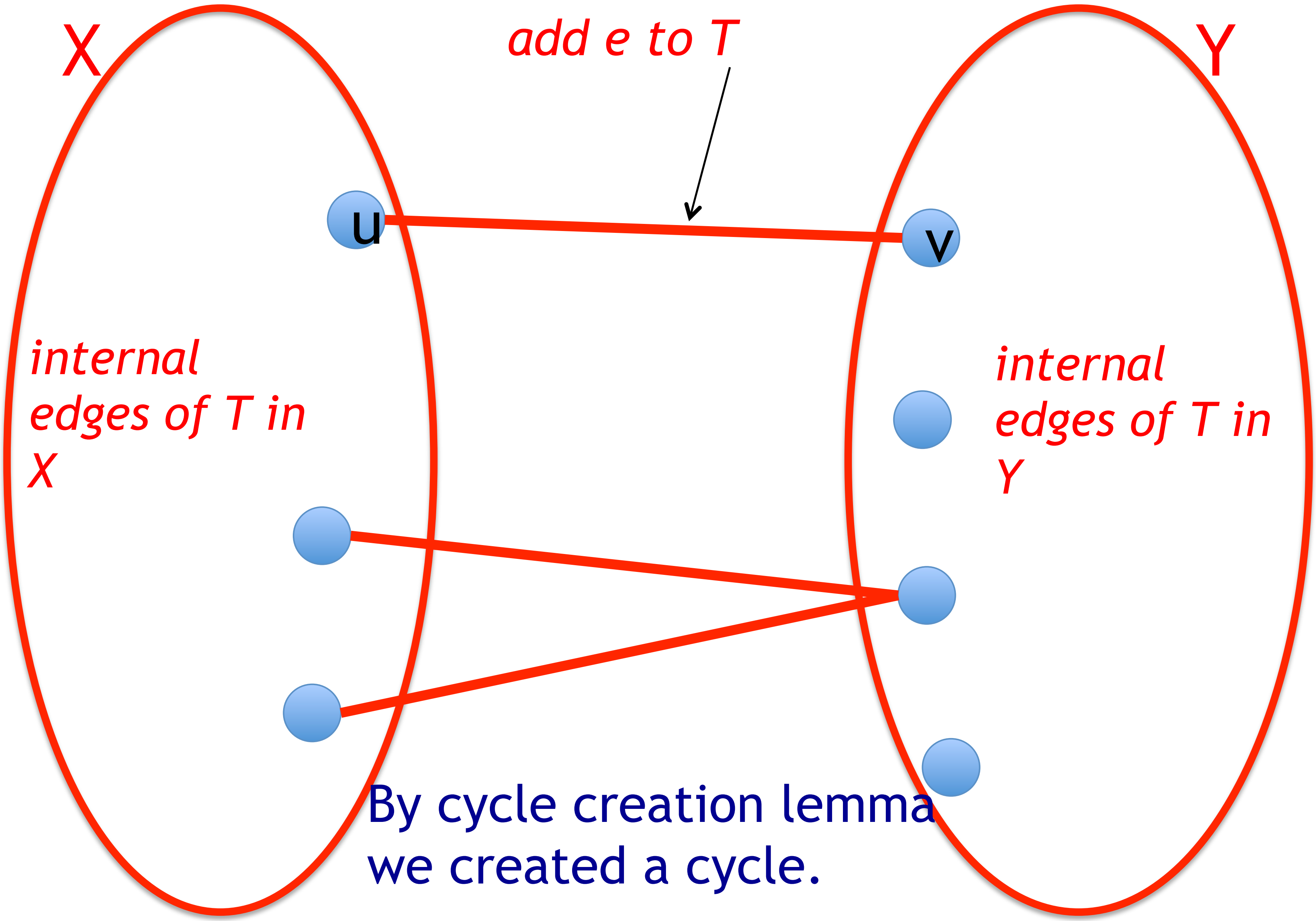
Proof of MST Cut Property (Exchange Argument)



Proof of MST Cut Property (Exchange Argument)

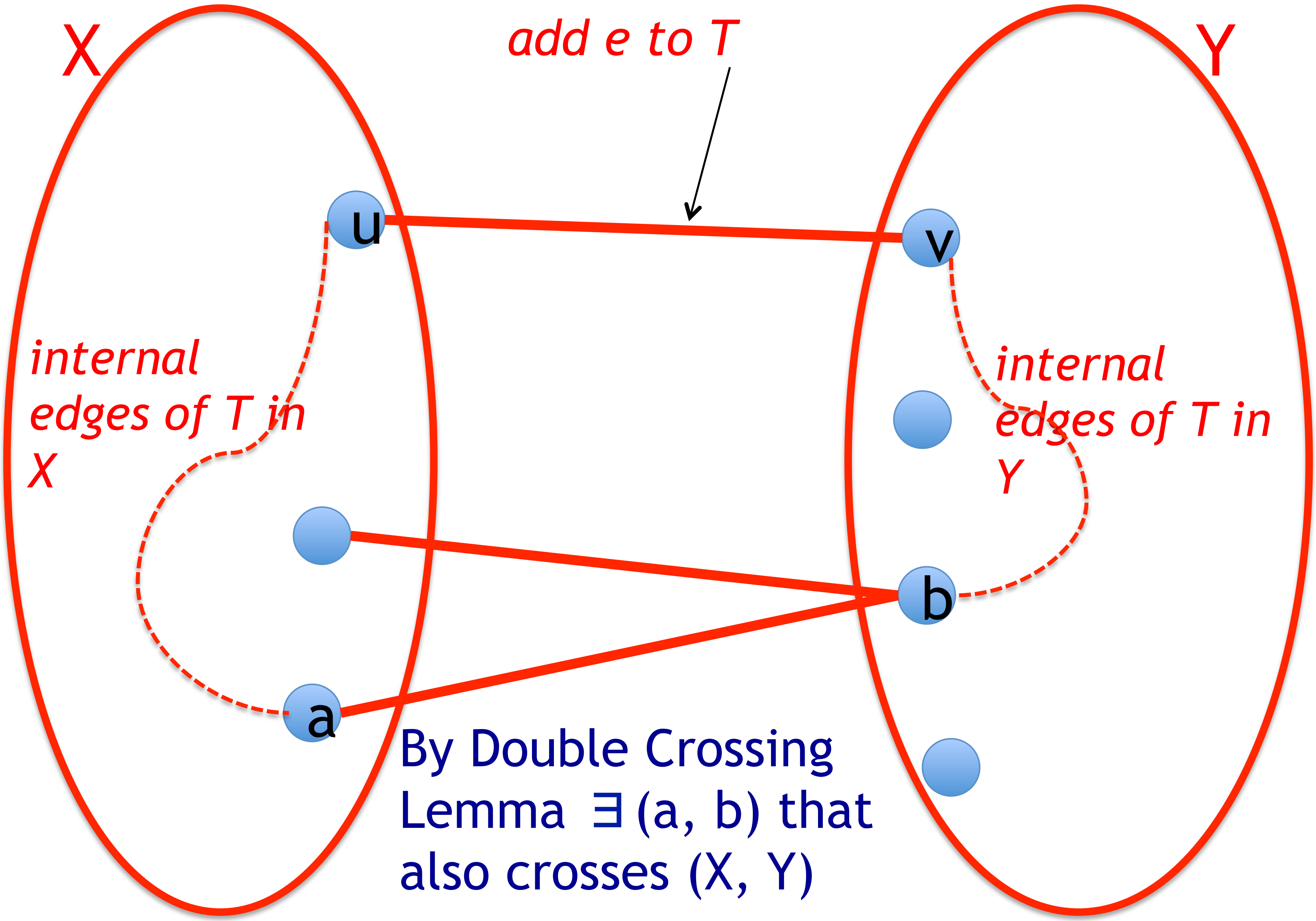


Proof of MST Cut Property (Exchange Argument)

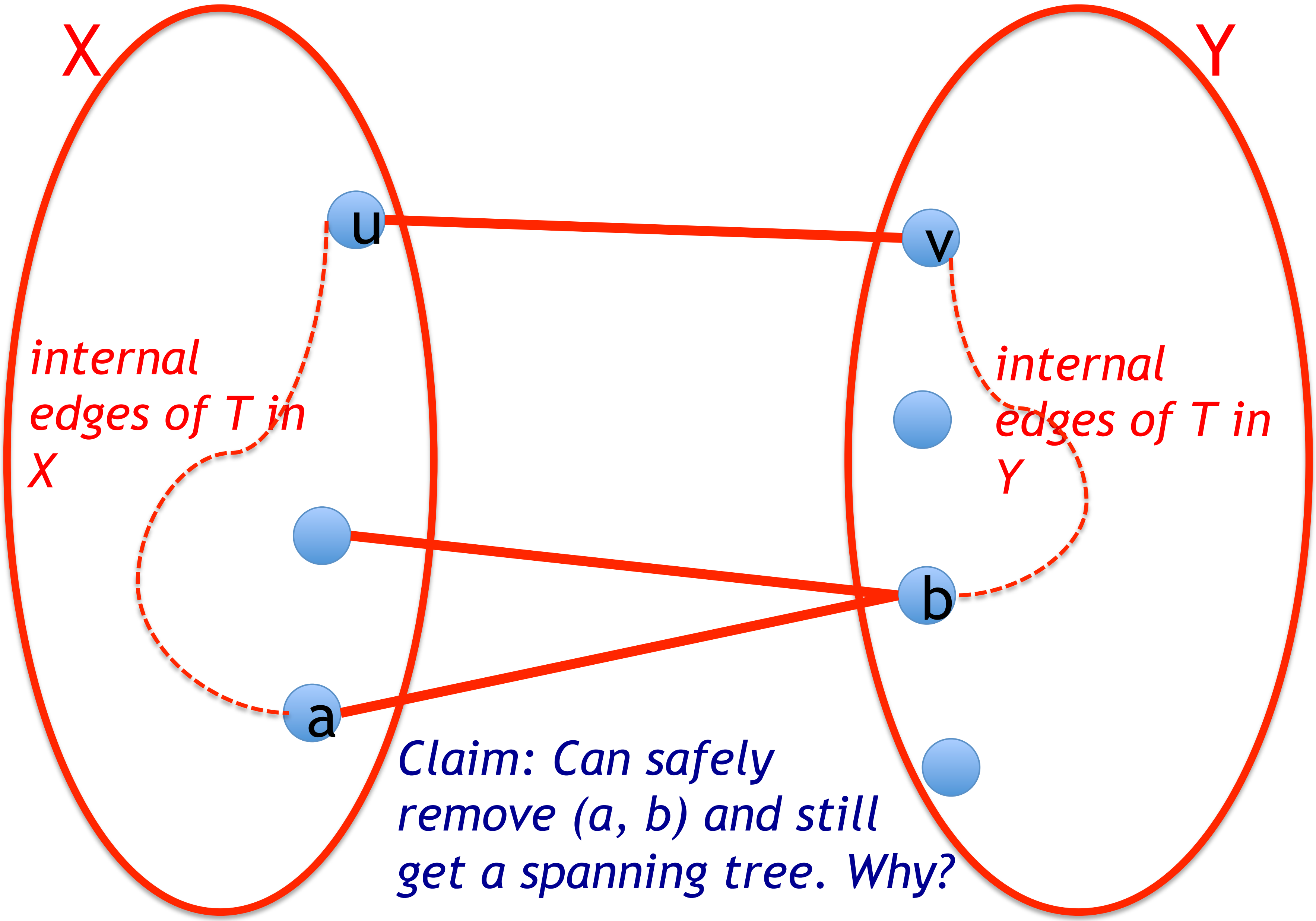


By cycle creation lemma we created a cycle.

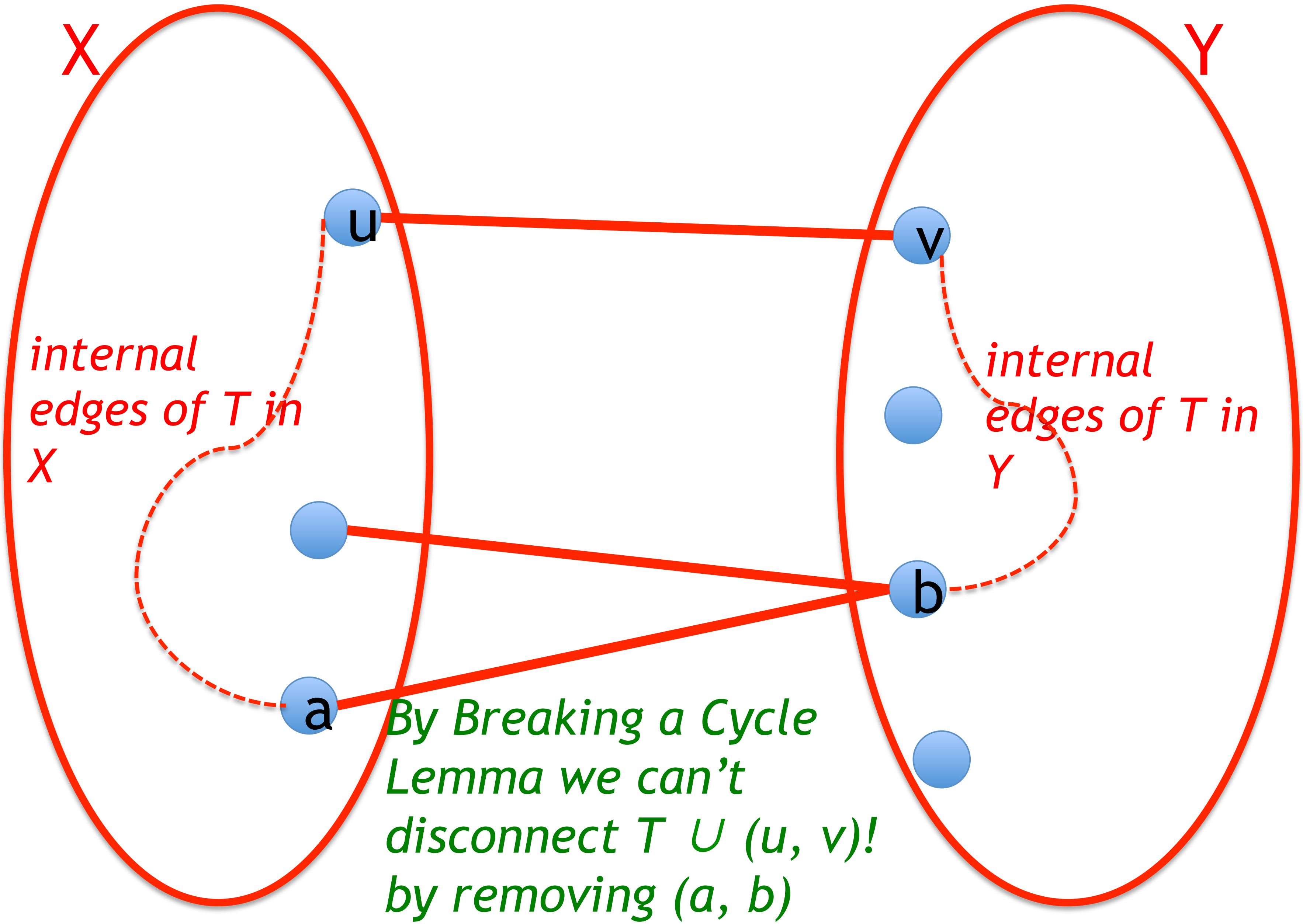
Proof of MST Cut Property (Exchange Argument)



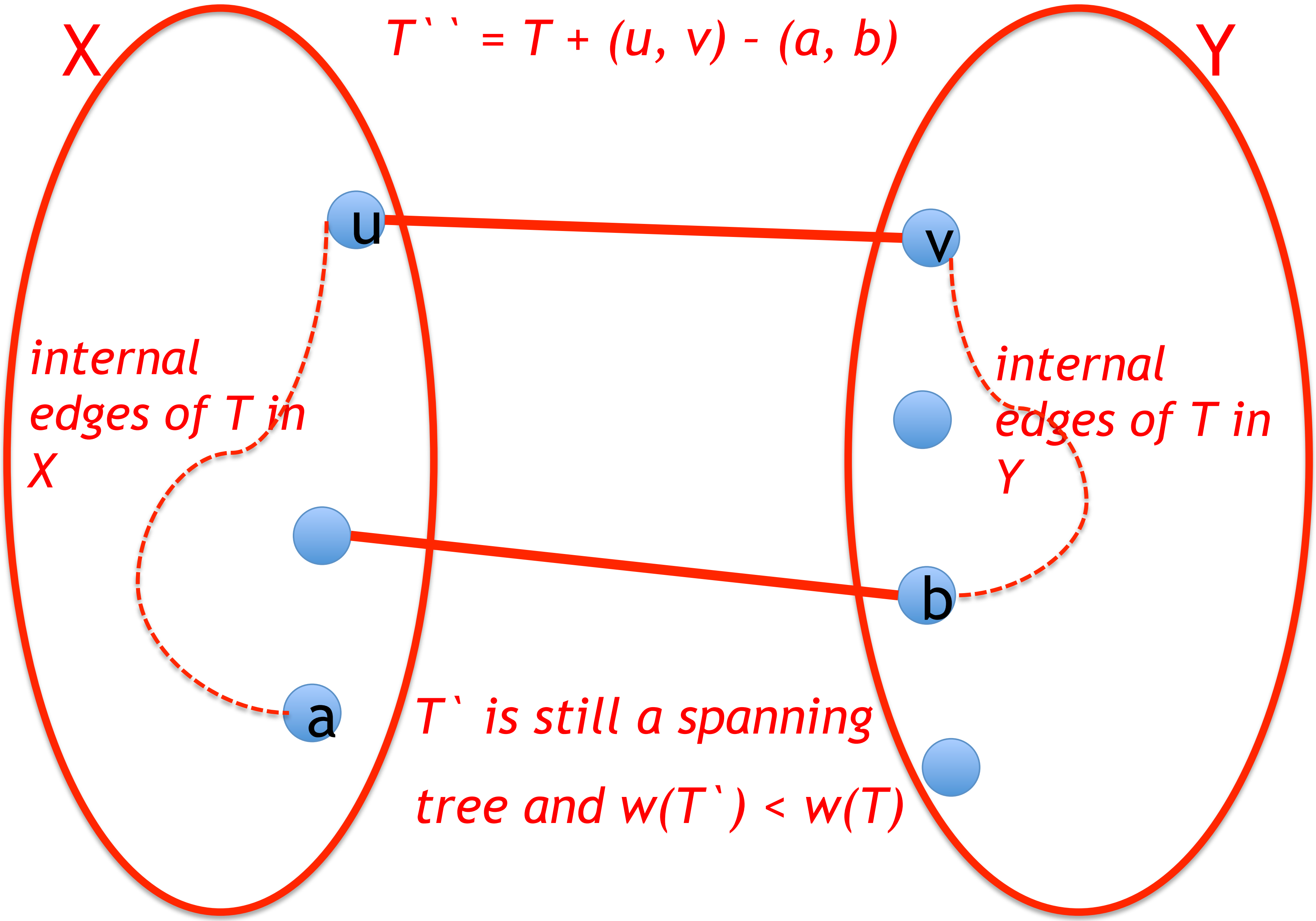
Proof of MST Cut Property (Exchange Argument)



Proof of MST Cut Property (Exchange Argument)



Proof of MST Cut Property (Exchange Argument)



Proof of MST Cut Property (Exchange Argument)

- Started with (X, Y) cut, and e=(u, v): min edge crossing (X, Y)
- Took any minimum spanning tree T that did not include e.
- Added e to T => created a cycle C (by cycle-creation-lemma)
- By DCL, there is another edge e` that crosses (X, Y)
- Removed e` => T`
- By Breaking A Cycle Lemma: T` is still connected
- Therefore w(T`) < w(T) => T was not an MST.

=> Every MST has to include e!

Q.E.D.