

Kruskal's MST Algorithm

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

Outline

Topics and Learning Objectives

- Introduce Kruskal's (greedy) algorithms for MSTs
- Discuss disjoint sets

Exercise

- Kruska's exercise

Extra Resources

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

Trick Question for the Day

Which is asymptotically bigger?

$O(m \lg n)$ or $O(m \lg m)$

Minimum-Spanning-Tree Overview

Input: an undirected graph where each edge has an associated **cost**

Output: a minimum-spanning-tree

1. Connects the entire graph as a tree, but
2. Has a minimal cost

Assumptions:

1. The input graph is connected
2. The edges costs are distinct (only necessary/useful for our proof)

Cut Property: if **e** is the cheapest edge crossing a cut, then it must be in the MST

Kruskal's

A **greedy** algorithm for finding the minimum spanning tree

Why are we learning another one?

- Kruskal's will motivate a new data structure: **Union-Find** (disjoint-set)
- It will also let us talk a bit about **clustering**

Can you think of another greedy algorithm for solving MST?

Kruskal's Minimum Spanning Tree Algorithm

Sort E by edge cost

T = empty

Edge-based

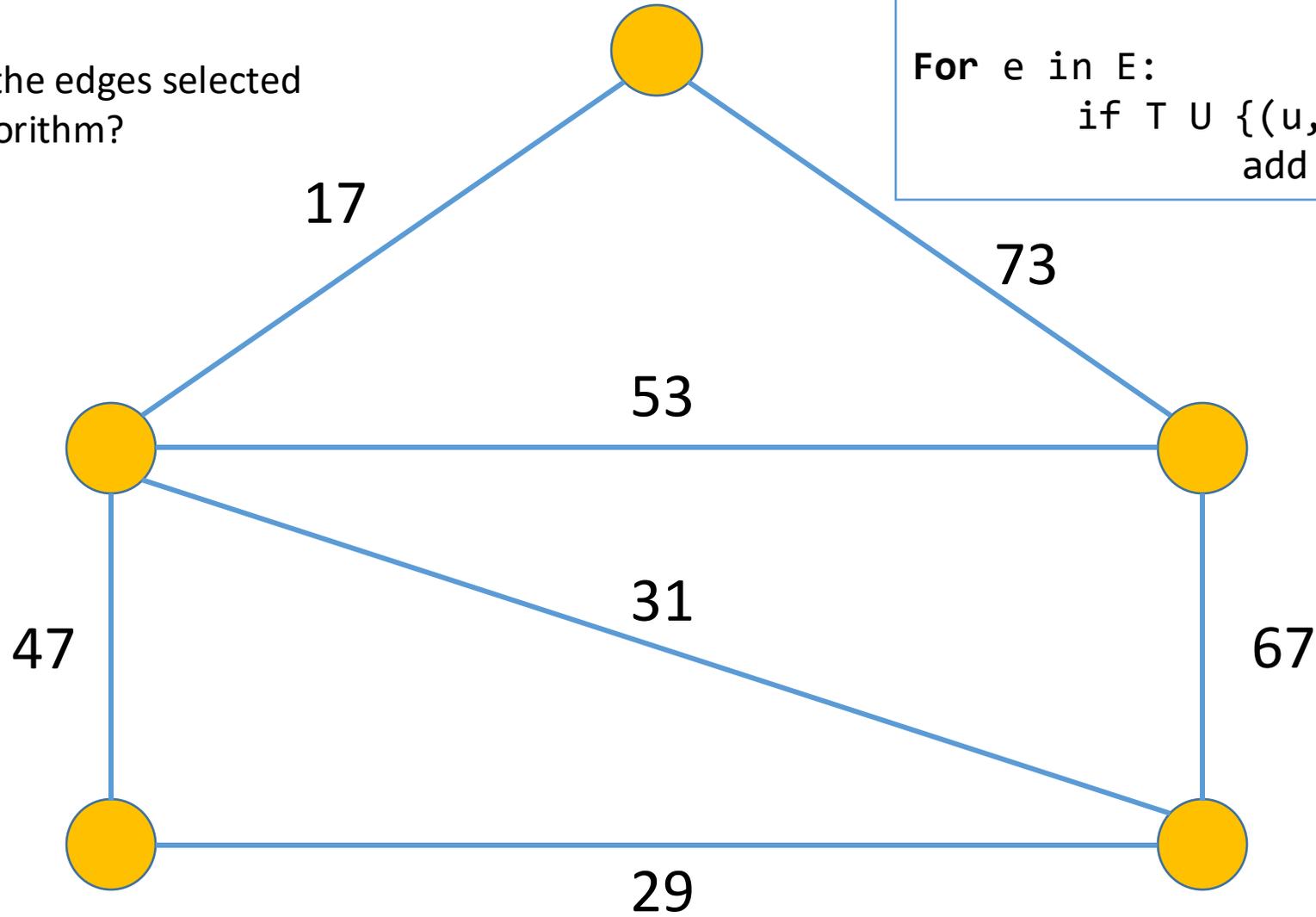
For e in E:

 if T U {(u, v)} has no cycles

 add e to T

Exercise question 1.

1. In what order are the edges selected using Kruskal's Algorithm?



Sort E by edge cost

$T = \text{empty}$

For e in E :

if $T \cup \{(u, v)\}$ has no cycles

add e to T

Proof of Kruskal's Algorithm

Theorem: Kruskal's algorithm is correct (computes the MST)

Let T^* = the output of Kruskal's algorithm

Graph/Cut/Tree Lemmas and Properties

- Empty Cut Lemma: a graph is not connected if there exists a cut (A, B) with zero crossing edges
- 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
- Cut Property: if e is the cheapest edge that crosses the cut (A, B) then it must be in the MST

Proof of Kruskal's Algorithm

Theorem: Kruskal's algorithm is correct (computes the MST)

Let T^* = the output of Kruskal's algorithm

Does Kruskal's output a spanning tree (what are the properties)?

- No cycles
- Connected

Kruskal's Minimum Spanning Tree Algorithm

Sort E by edge cost

$T = \text{empty}$

For e in E :

 if $T \cup \{(u, v)\}$ has no cycles

 add e to T

Can Kruskal's include an edge that creates a cycle?

Kruskal's Minimum Spanning Tree Algorithm

Sort E by edge cost

$T = \text{empty}$

For e in E :

 if $T \cup \{(u, v)\}$ has **no cycles**

 add e to T

Can Kruskal's include an edge that creates a cycle?

Proof of Kruskal's Algorithm

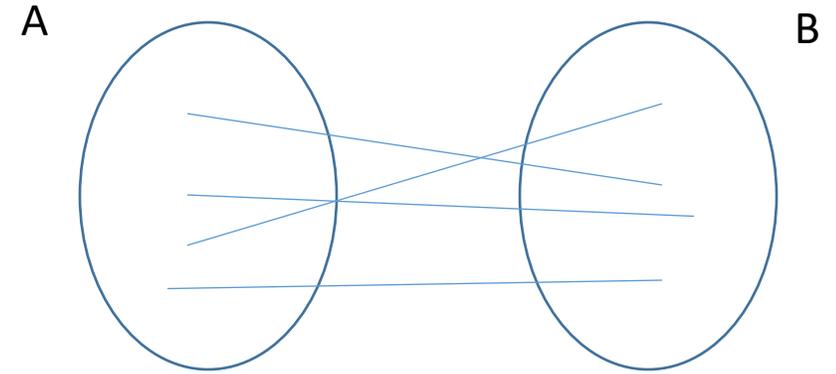
Theorem: Kruskal's algorithm is correct (computes the MST)

Let T^* = the output of Kruskal's algorithm

Does Kruskal's output a spanning tree (what are the properties)?

- No cycles (this is given by the definition of the algorithm)
- Connected

Proof of Kruskal's Algorithm



Proof of Connectivity

- Given the Empty Cut Lemma, we only need to show that Kruskal's produces a tree T^* that crosses every cut.
- Fix a cut (A, B)
- Since G is connected, at least one of its edges crosses the cut (A, B)
- Kruskal's algorithm considers each edge once
- Let's fast-forward to the first time that it encounters an edge crossing (A, B)
- **Claim:** this 1st edge is guaranteed to be in T^*
 - Given the No Cycle Corollary the claim is true
- It is also the **minimum** edge to cross the cut (sorted edges)

Proof of Kruskal's Algorithm

For the second part of the proof, we need to prove that T^* is minimal

- We just finished proving that Kruskal's outputs some spanning tree T^*

Claim: every edge is justified by the Cut Property

- Remember that satisfying the Cut Property implies that we have an MST
- This was very explicit in Prim's Algorithm

Prim's Minimum Spanning Tree Algorithm

$X = \{s\}$

$T = \text{empty}$

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

Proof of Kruskal's Algorithm

Proving that we can use the Cut Property

- Consider each iteration where edge (u, v) is added to T^*
- Since $T^* \cup \{(u, v)\}$ has no cycle, T^* currently has no $u \rightarrow v$ path
- Thus, there must be a cut (A, B) separating u and v . For example:
 - All findable from u in A given current T^*
 - All findable from v in B given current T^*
 - All other vertices can be partitioned arbitrarily
- Hence, (u, v) is the first crossing cut for (A, B)
- Additionally, it must be the cheapest such cut since we sorted the edges
- Finally, the edge (u, v) is justified by the Cut Property

Proof of Kruskal's Algorithm

What have we done?

We proved that Kruskal's outputs a spanning tree

- No cycles by definition
- Connectivity by the Empty Cut Lemma

We then proved that Kruskal's outputs the **minimum** spanning tree

- The Cut Property implies that we are left with the MST
- We showed that Kruskal's uses the Cut Property because the edges are sorted

Running Time and Implementation of Kruskal's

Kruskal's Minimum Spanning Tree Algorithm

Sort E by edge cost

$O(m \lg m)$

$T = \text{empty}$

For e in E :

$O(m)$

 if $T \cup \{(u, v)\}$ has no cycles

 add e to T

How would you detect if adding (u,v) creates a cycle?

Kruskal's Minimum Spanning Tree Algorithm

Sort E by edge cost

$O(m \lg m)$

$T = \text{empty}$

For e in E :

$O(m)$

if $T \cup \{(u, v)\}$ has no cycles

Naively $O(n + m)$

add e to T

$O(m \lg m) + O(m) * O(n+m)$

$O(mn+m^2)$

Kruskal's Minimum Spanning Tree Algorithm

Sort E by edge cost

$T = \text{empty}$

For e in E :

if $T \cup \{(u, v)\}$ has no cycles

 add e to T

What can we change (should we change) to do better?

Kruskal's Minimum Spanning Tree Algorithm

Sort E by edge cost

$T = \text{empty}$

For e in E :

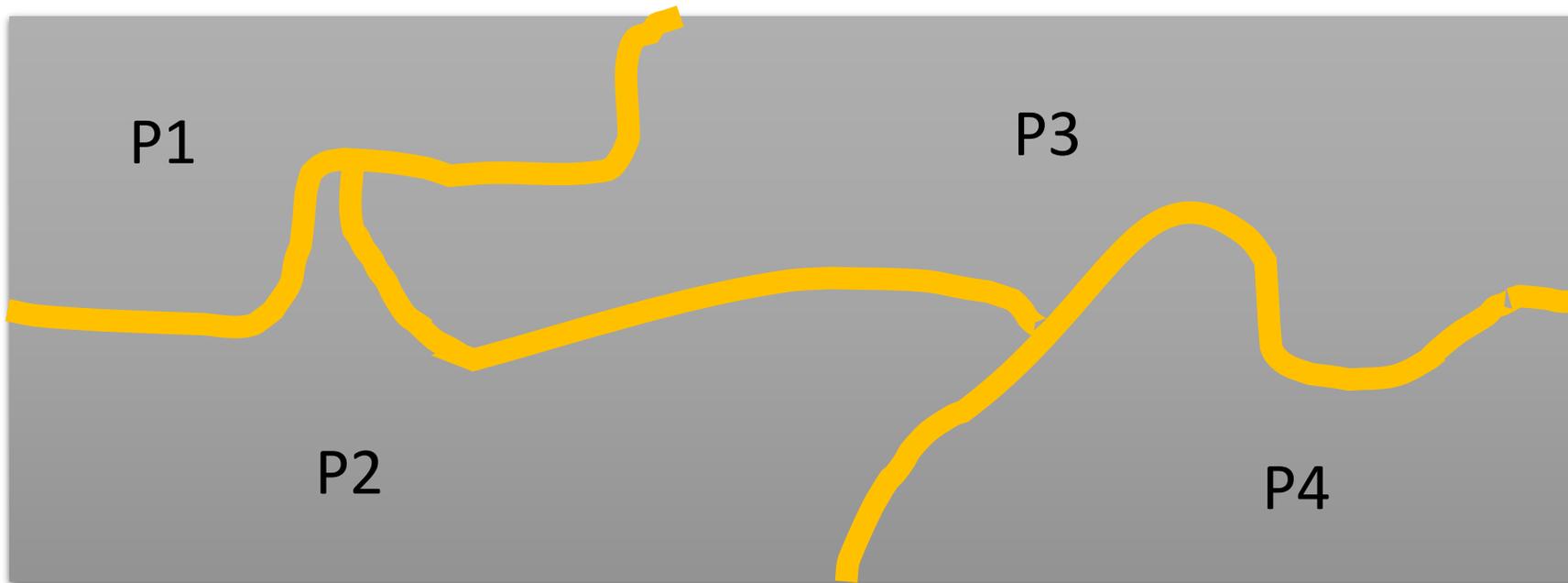
 if $T \cup \{(u, v)\}$ has **no cycles**

 add e to T

What can we change (should we change) to do better?

The Union-Find Data Structure

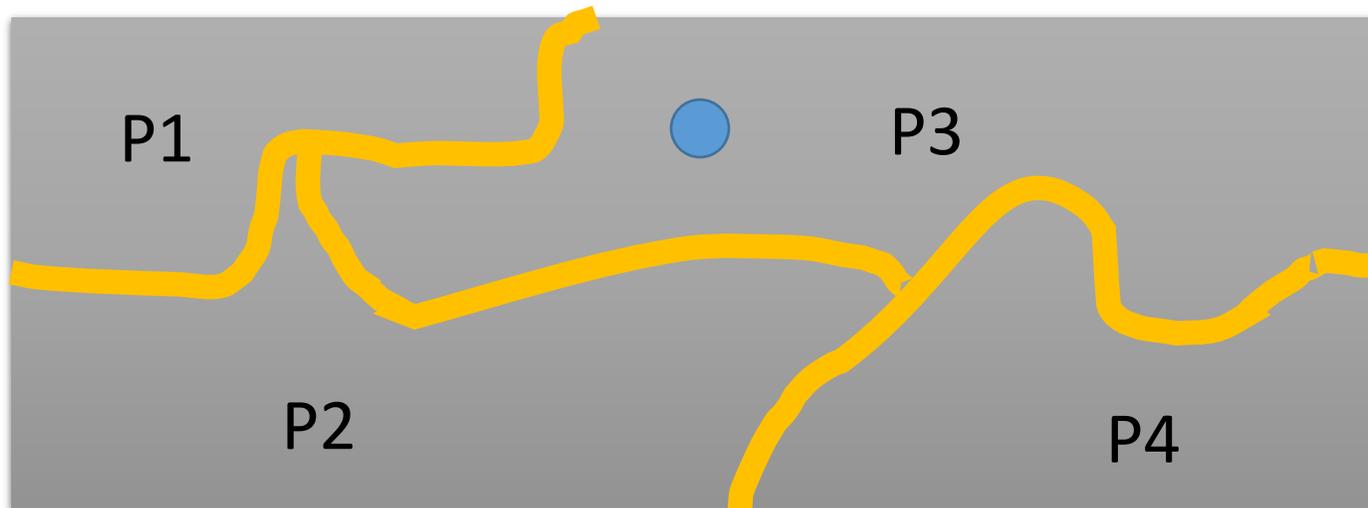
- Also known as the disjoint-set data structure
- Used to maintain a partition of objects



Union-Find

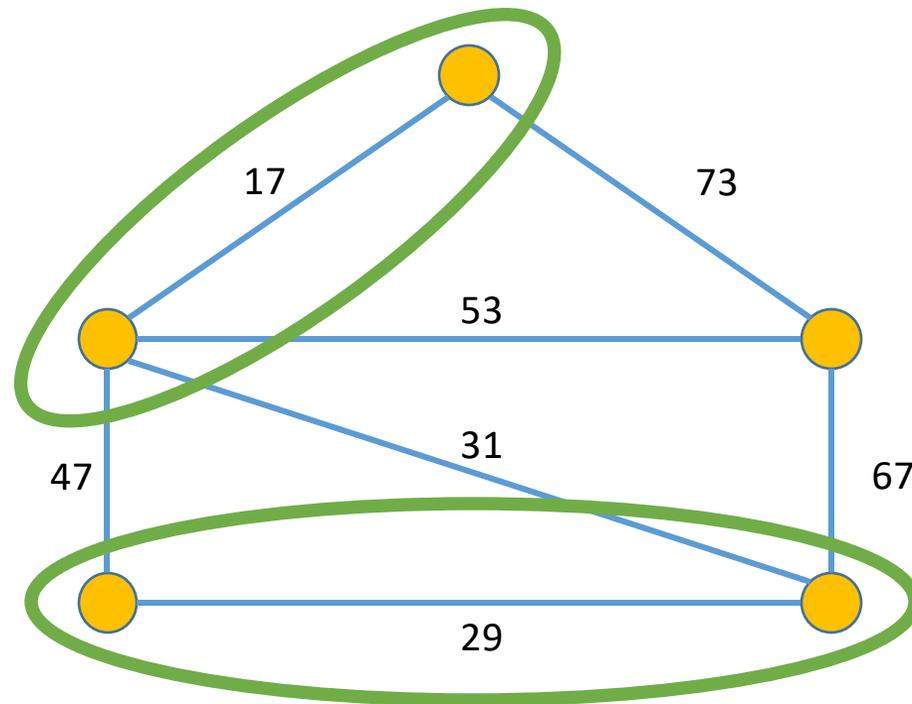
Operations:

- Find(x): return the name of the group to which x belongs
- Union(P_i, P_j): merge the two partitions into a single partition



How does this help us with Kruskal's?

- What do we store in the data structure?
- What makes a partition?



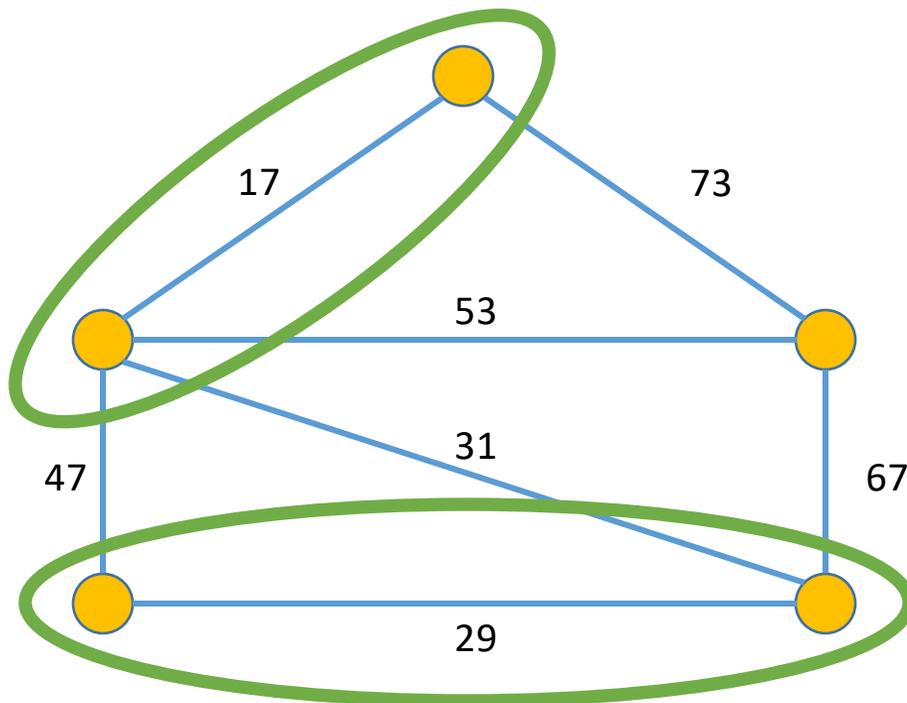
Motivation

- Speed up the way in which we check for cycles.
- How would you implement the Union-Find data structure?
- → Augment each vertex to include another piece of information: the name of its **leader**
 - Or use a separate data structure (**what kind? → what operations matter?**)
- Invariant: each vertex knows its leader
- How long does it take to check for a cycle now?

Checking for cycles

- Given an edge (u, v) , we can check if u and v are in the same partition in constant time $O(1)$.

$Find(u) == Find(v)$?

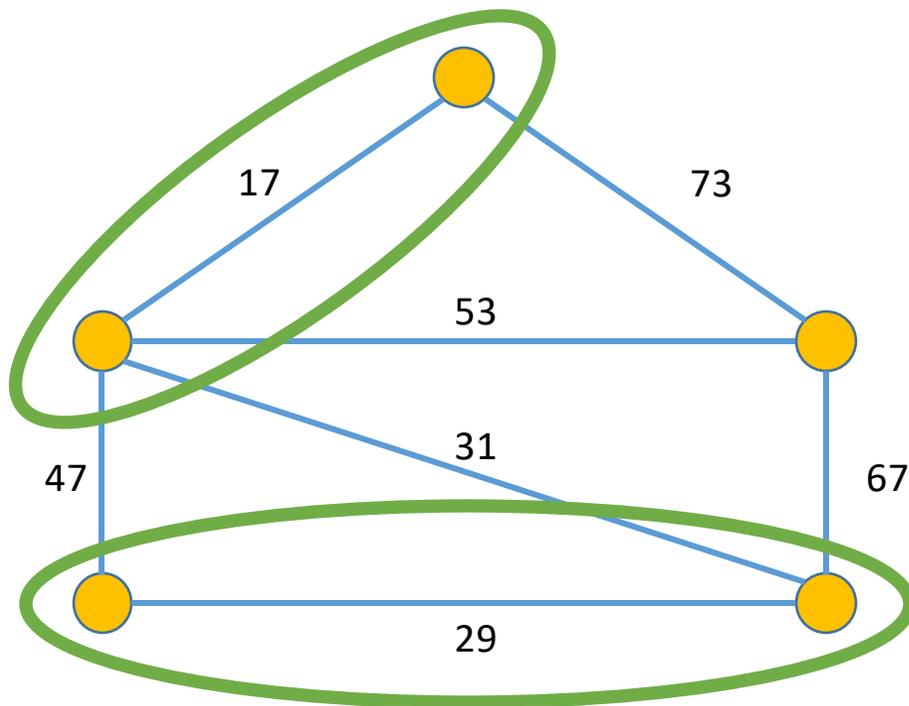


What happens during
the next iteration?

What's the catch?

Maintaining the Invariant

- Invariant: each vertex knows its leader



What is the maximum number of vertex leaders that must be fixed after a union?

Exercise Question 2

Union-Find Data Structure

- Put every element in its own partition
 - Every element has its own leader
- Join partitions by copying the leader of the larger partition elements to all elements of the smaller partition
- You can use an array or hash table to keep track of leaders
- No other information/memory is needed

Kruskal's Minimum Spanning Tree Algorithm

Sort E by edge cost

$O(m \lg m)$

$T = \text{empty}$

For e in E :

$O(m)$

 if $T \cup \{(u, v)\}$ has no cycles

 add e to T

 union

What do we have as a
running time now?

What happened?

Sort E by edge cost

$O(m \lg m)$

T = empty

For e in E:

$O(m)$

if T U {(u, v)} has no cycles

$O(n+m) \rightarrow O(1)$ (checking leaders)

add e to T

We don't do this every iteration

union

$O(1) \rightarrow O(n)$ (updating leaders)

Compared with finding cycles with BFS

Maximum number of leader updates?

How many times can we update the leader of a single vertex?

- We only update the leader of a vertex if we merge it with a bigger partition.
- How many times can we merge a vertex into a bigger partition?
 - (Or: How many times can we double the size of a partition?)
- This is our global view of something happening inside the loop.

```
For e in E:  
  if T U {(u, v)} has no cycles  
    add e to T  
  union
```

Kruskal's Minimum Spanning Tree Algorithm

Sort E by edge cost

$O(m \lg m)$

$T = \text{empty}$

For e in E :

$O(m)$

if $T \cup \{(u, v)\}$ has no cycles

$O(1)$ just for the cycle check

add e to T

$O(n \lg n)$ for Union (not per iteration)

union

Technically this is $O(n \lg n + m \lg m)$

$O(m \lg m)$

+

$O(n \lg n)$

+

$O(m) * O(1)$

=

$O(m \lg m)$

Sort

Union

Loop

Total

Cutting Edge

- Can we do better than $O(m \lg m)$?
- Yes!
- **Average** $O(m)$ using a randomized algorithm (1995)
- We do not actually know if a deterministic $O(m)$ algorithm exists.
- We do have a deterministic algorithm that is $O(m \alpha(n))$
- α is the inverse Ackermann function
- Which is slower than the **Iterated logarithm: \lg^***
 - the number of times the logarithm function must be iteratively applied before the result is less than or equal to 1
- An optimal deterministic algorithm was developed in 2002
- But we do not know the exact asymptotic complexity
- Just that it is between $O(m)$ and $O(m \alpha(n))$

x	$\lg^* x$
$(-\infty, 1]$	0
$(1, 2]$	1
$(2, 4]$	2
$(4, 16]$	3
$(16, 65536]$	4
$(65536, 2^{65536}]$	5