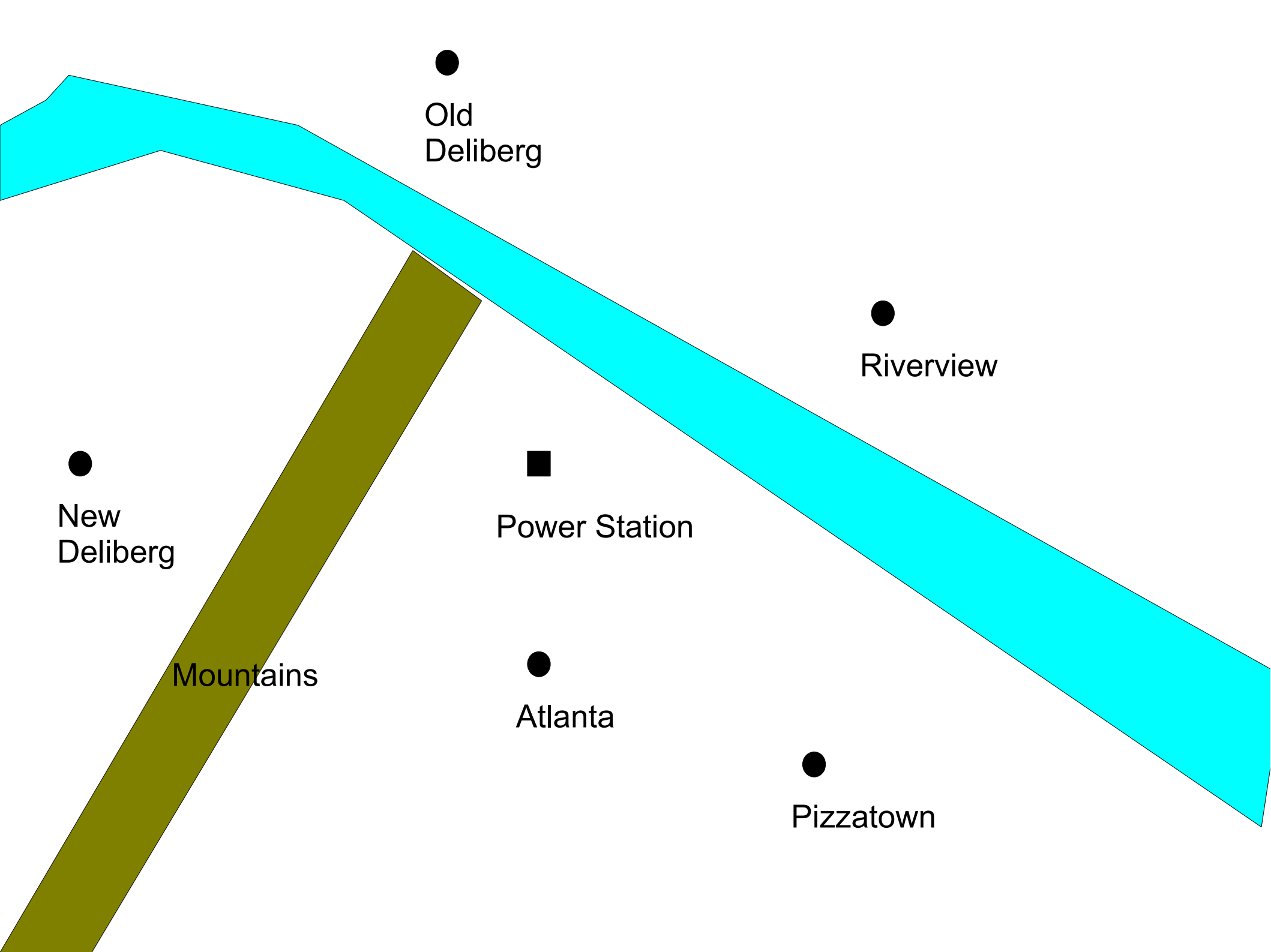
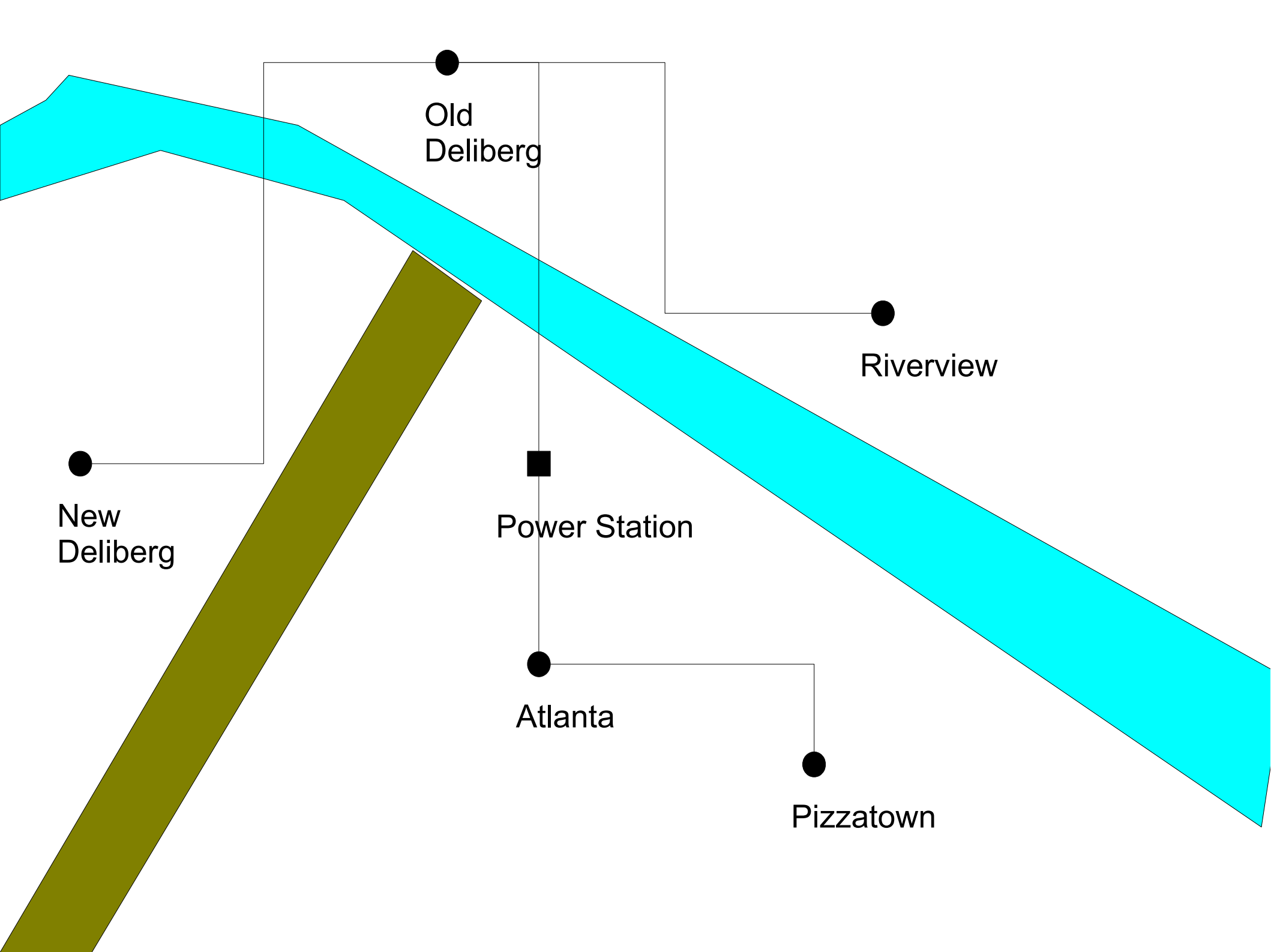
Minimum Spanning Trees

What we want

- Sometimes we wish to find a connected spanning subgraph that has the least cost.

- For example, maybe you are a power company who wants to hook several towns to a new power plant in the cheapest way possible.

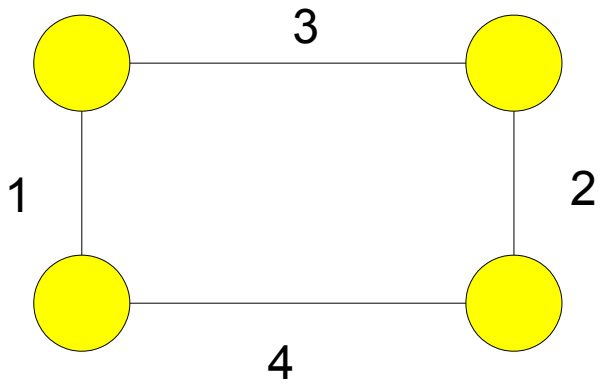

Old
Deliberg
Riverview
New
Deliberg
Power Station
Mountains
Atlanta
Pizzatown


Old
Deliberg
Riverview
New
Deliberg
Power Station
Atlanta
Pizzatown

More specific

- We can use a weighted graph to think about this problem.
- Each vertex represents something we want to connect.
- Each edge represents a potential connection between to vertices
- Each edge's weight is the cost to add that connection
- We want to minimize the sum of the edge weights while still getting a connected spanning subgraph.

Simple Example



Notes on our graph

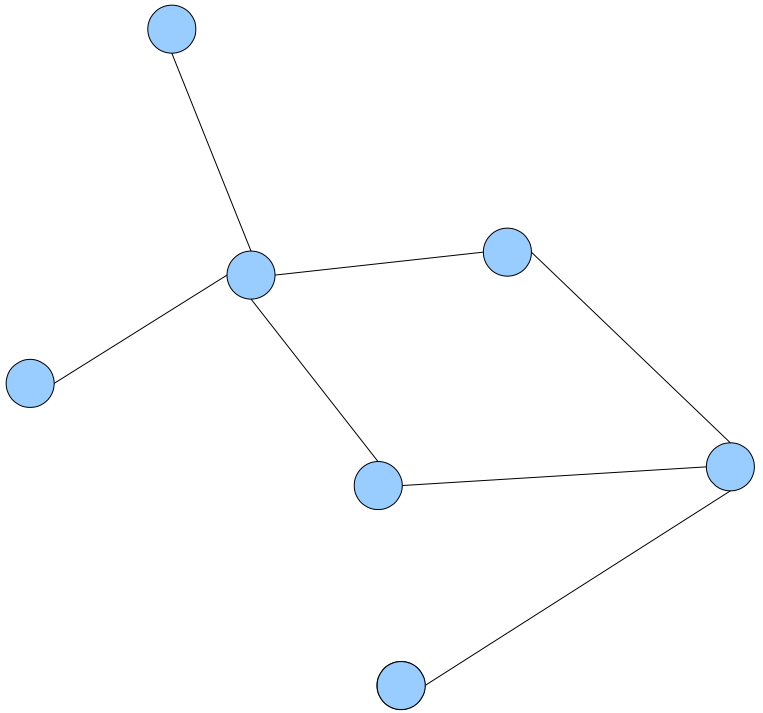
- Note: Our graph must be connect to be able to find a minimum connected spanning subgraph. If it's not connected, we can just find it for each connected component
- We also assume there are no self-loops (which are useless) and the graph is simple (we'll only add the cheapest edge between two vertices)

Tree

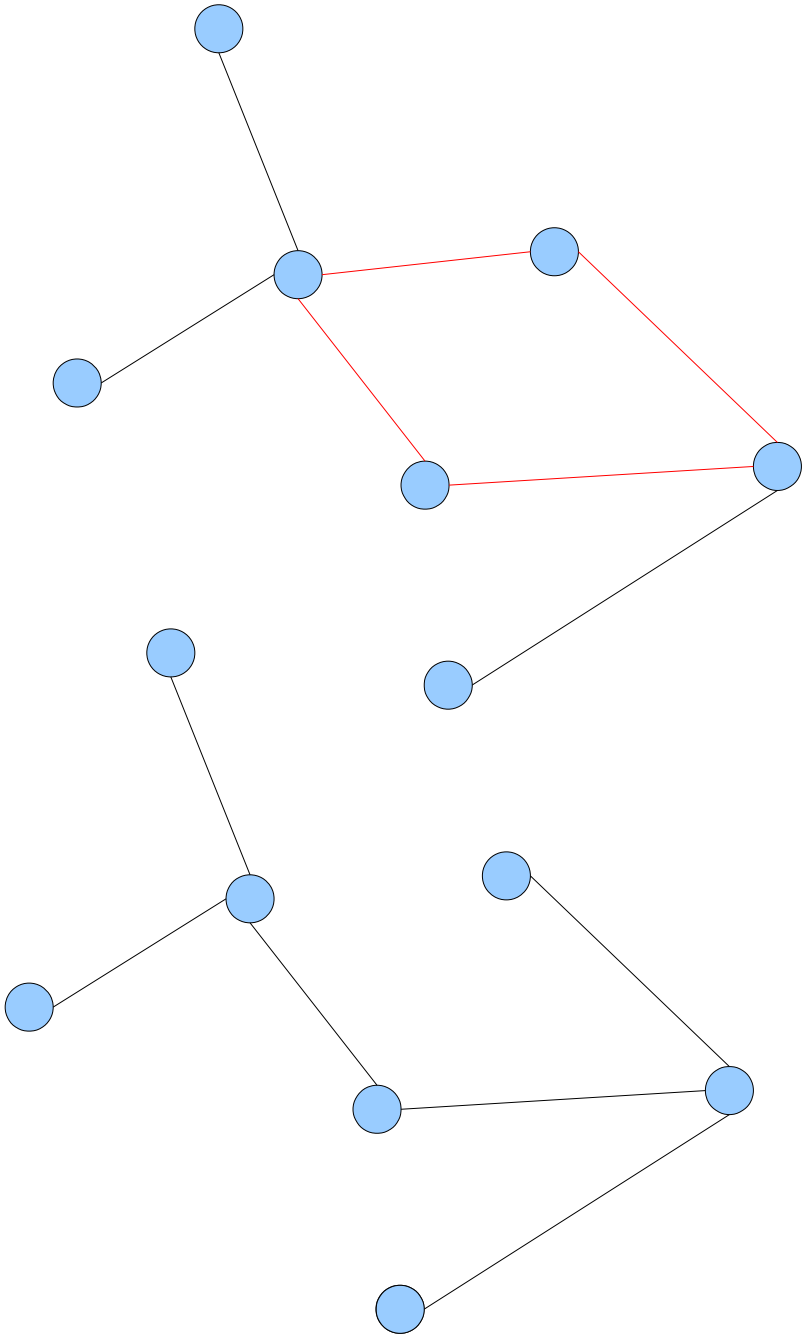
- If all our edge-weights are positive, then the cheapest way to connect the graph is with a spanning tree.

Why Tree

- Let's say we have found the cheapest way to make the graph connected, and it is not a tree.



- Since it is not a tree, it contains some cycle.
- Since all edges in the graph are nonnegative, removing an edge from this cycle will not increase the total cost. Also, removing an edge from this graph does not effect connectivity.

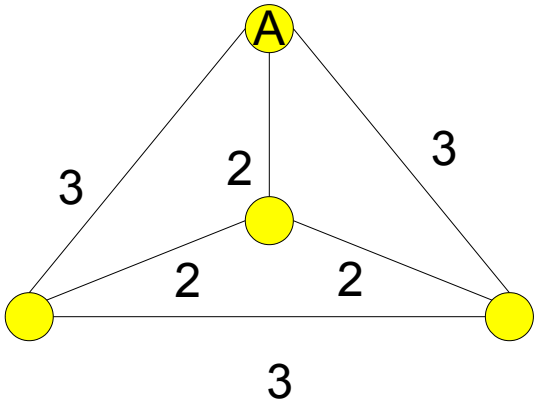


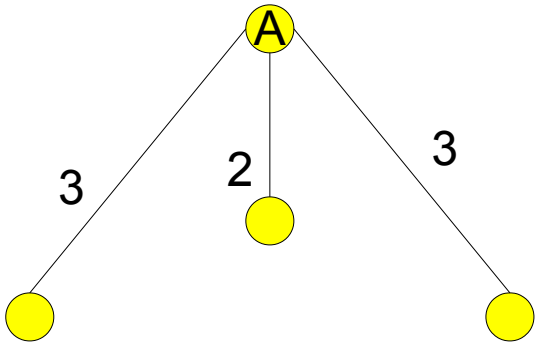
- We just do this until all of the cycles are gone, and so we are left with a fully connect graph with no cycles, and so is a tree.
- Also, we never increased the cost in our procedure, so the tree is also a cheapest way to connect the vertices.
- Since the tree spans the graph, and is a minimal way to create such a graph, we call such a tree a Minimal Spanning Tree (MST)

MST vs SPT

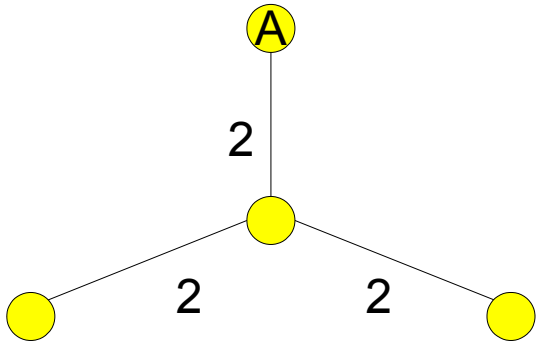
- In a minimum spanning tree we are worried about the total cost of the tree, not the cost of any paths

MST vs SPT





Shortest path tree from A
Total Cost: 8
Total Cost of Paths from A:
3+3+2=8



Minimum Spanning tree
Total Cost: 6
Total of Paths from A:
2+4+4=10

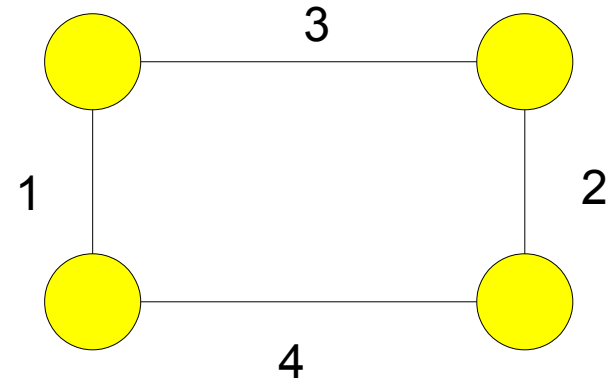
Main Concerns

- When finding the MST, our main concerns are:
  - Making sure we get a tree, which means no cycles
  - Making sure the edge weights are minimized.

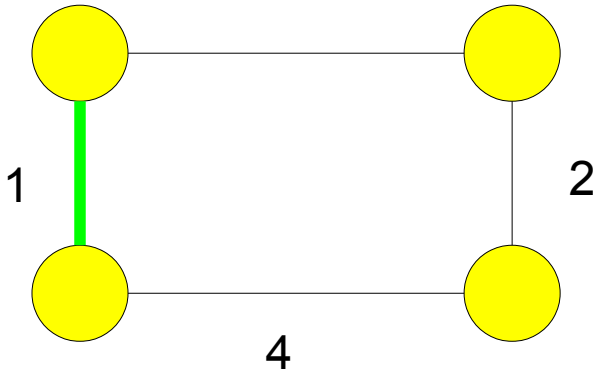
Kruskal's Algorithm

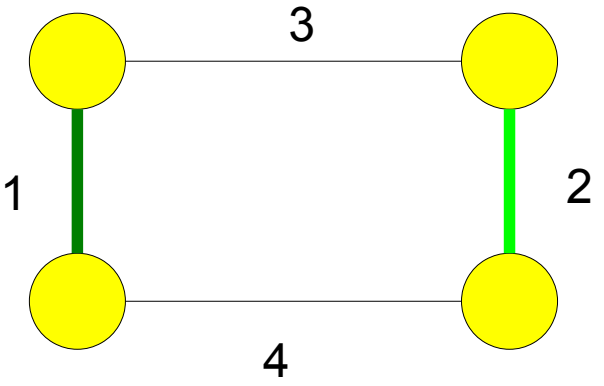
- One way to find a MST is via Kruskal's algorithm:
- Take the smallest edge that does not induce a cycle, and insert it into our subgraph.
- Do this until all nodes are connected
- A naive way to make sure an edge does not induce a cycle is by using DFS or BFS from one of the edge's vertices, and seeing if we reach the other. If we do, adding that edge would create a cycle.

Simple Example

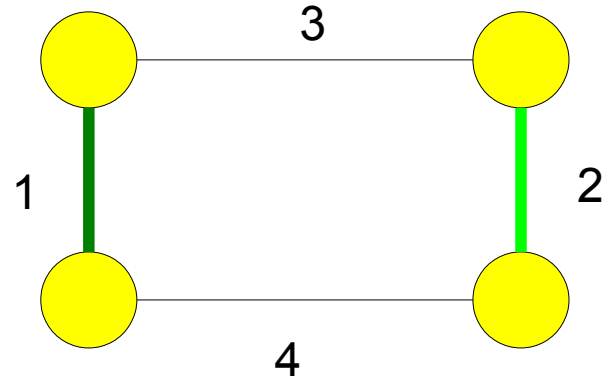


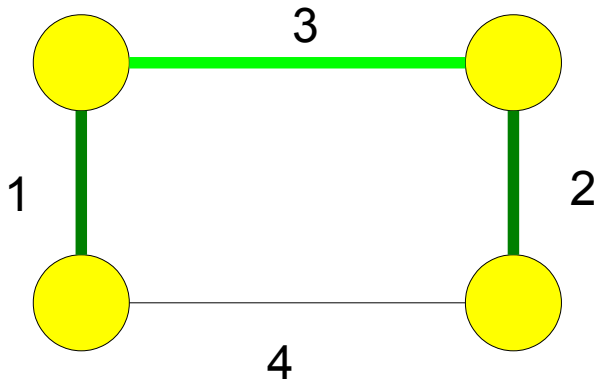
3





Simple Example





Greedy

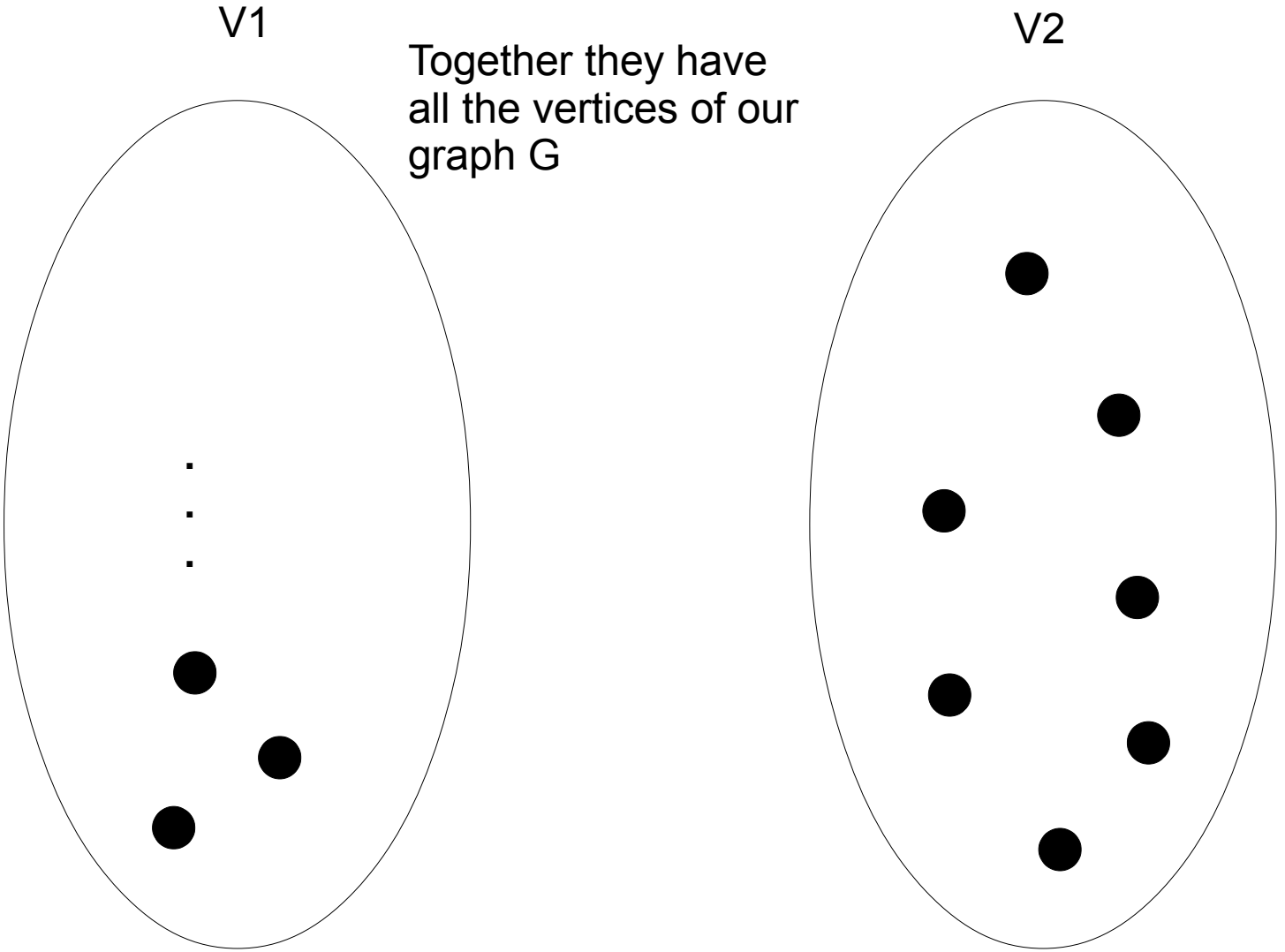
- This is a greedy algorithm. The basic idea is to find the valid edge with the smallest weight and add it to the tree.

Ensuring Minimality

- How do we know this simple, greedy algorithm forms a MST?
- First we must look at what is know as the Cut Property.

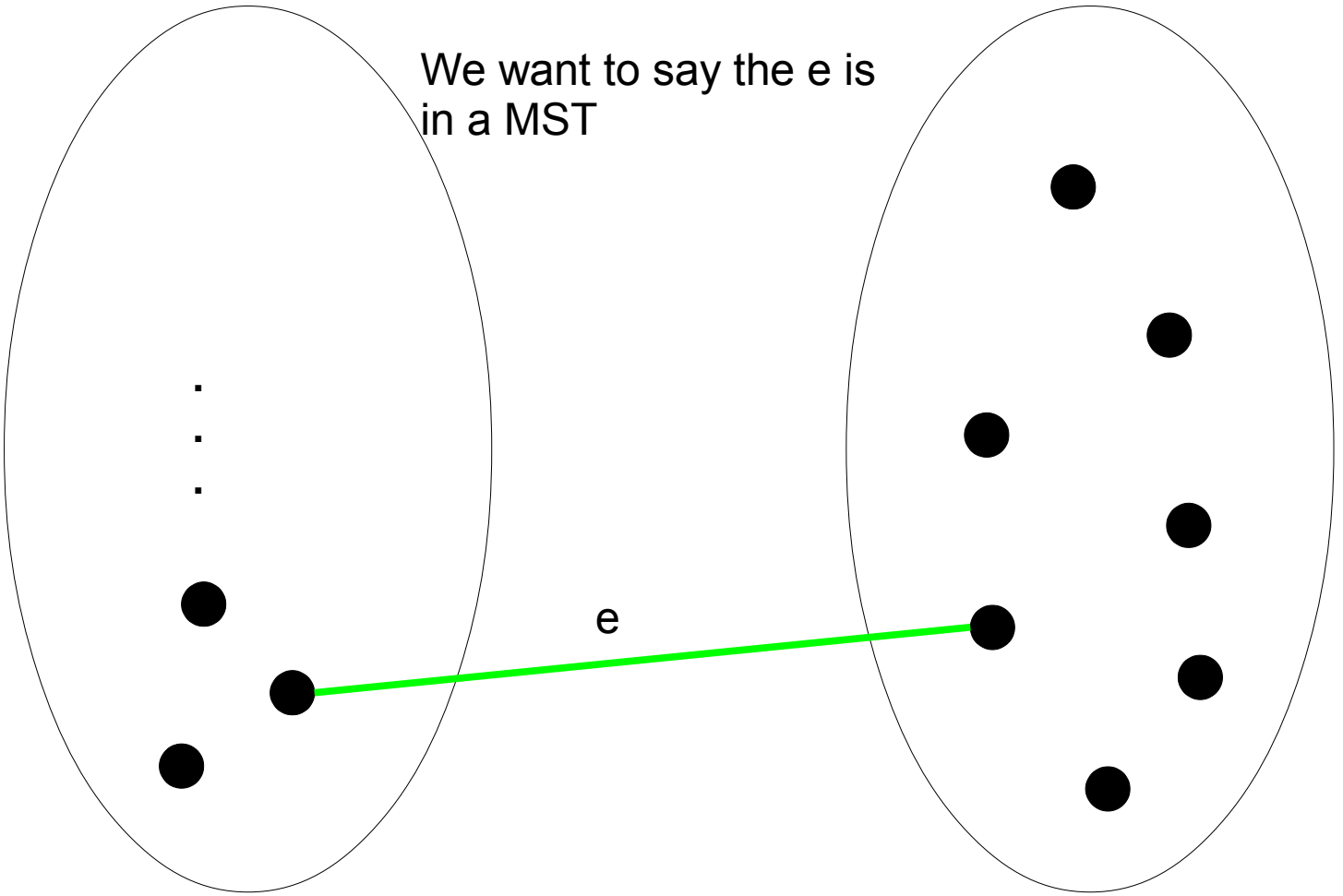
We have two distinct groups of vertices.

Together they have all the vertices of our graph G



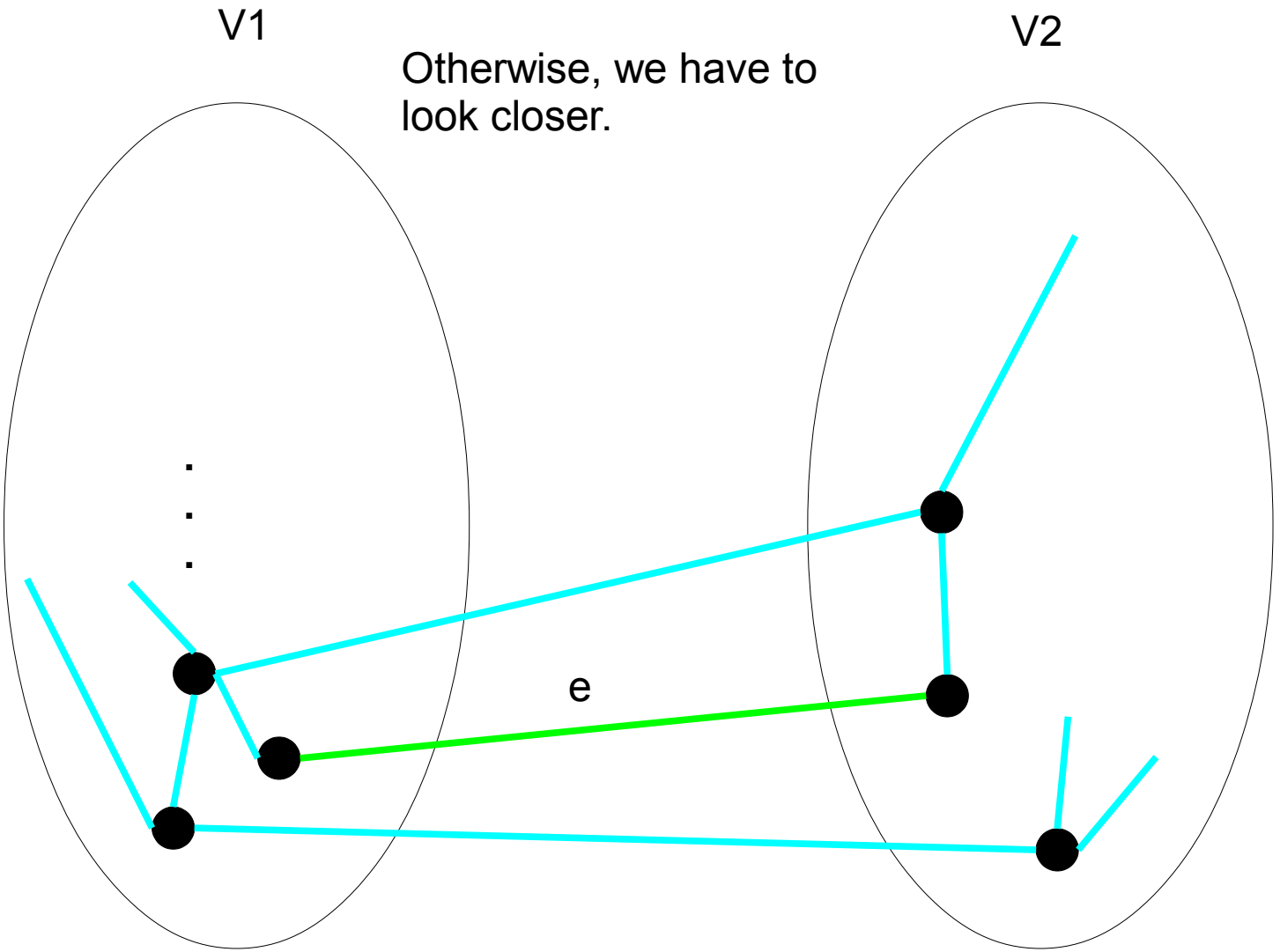
Since our graph is connected, there is a least edge between them.

We want to say the e is in a MST



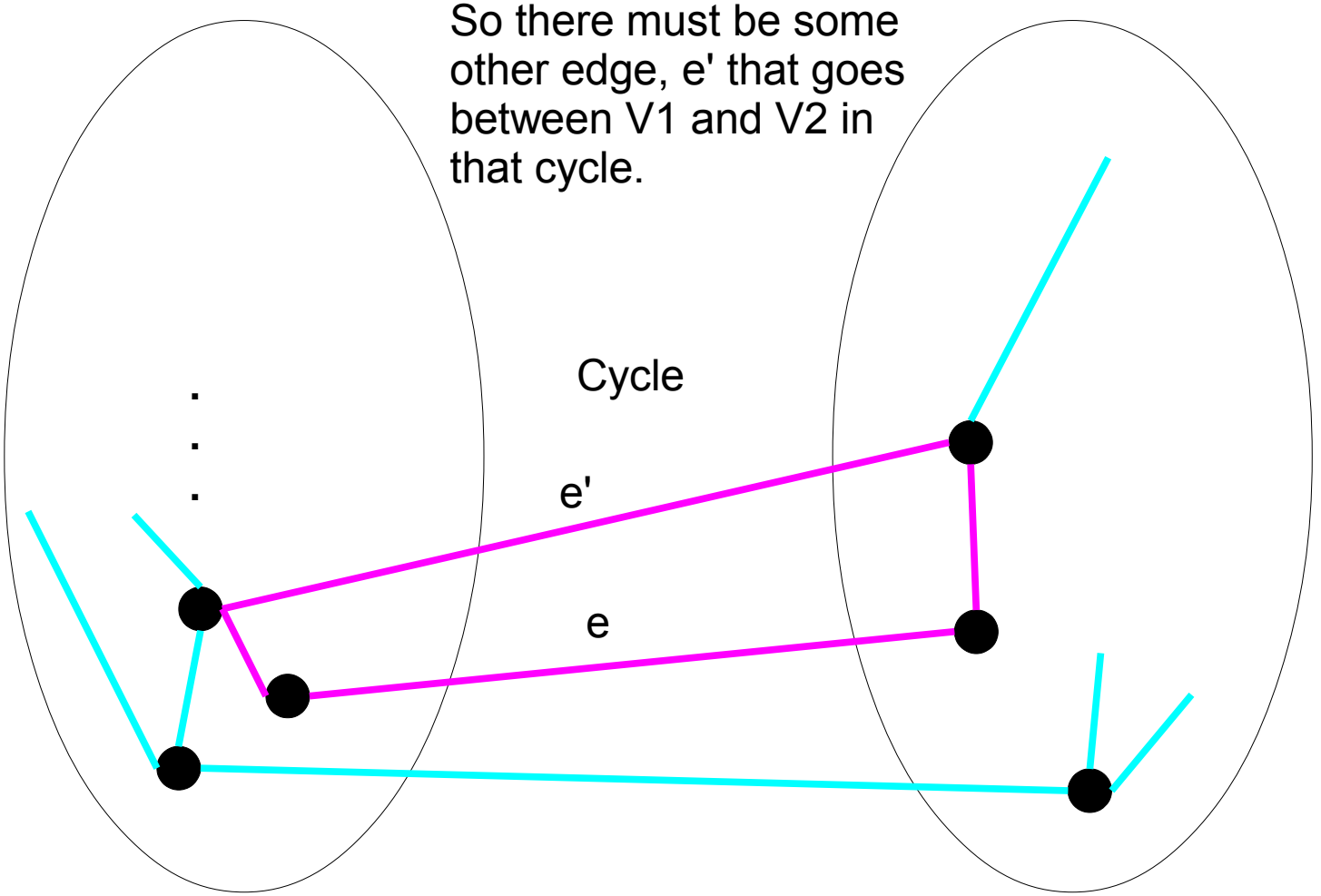
If we have a MST with e, we are done.

Otherwise, we have to look closer.



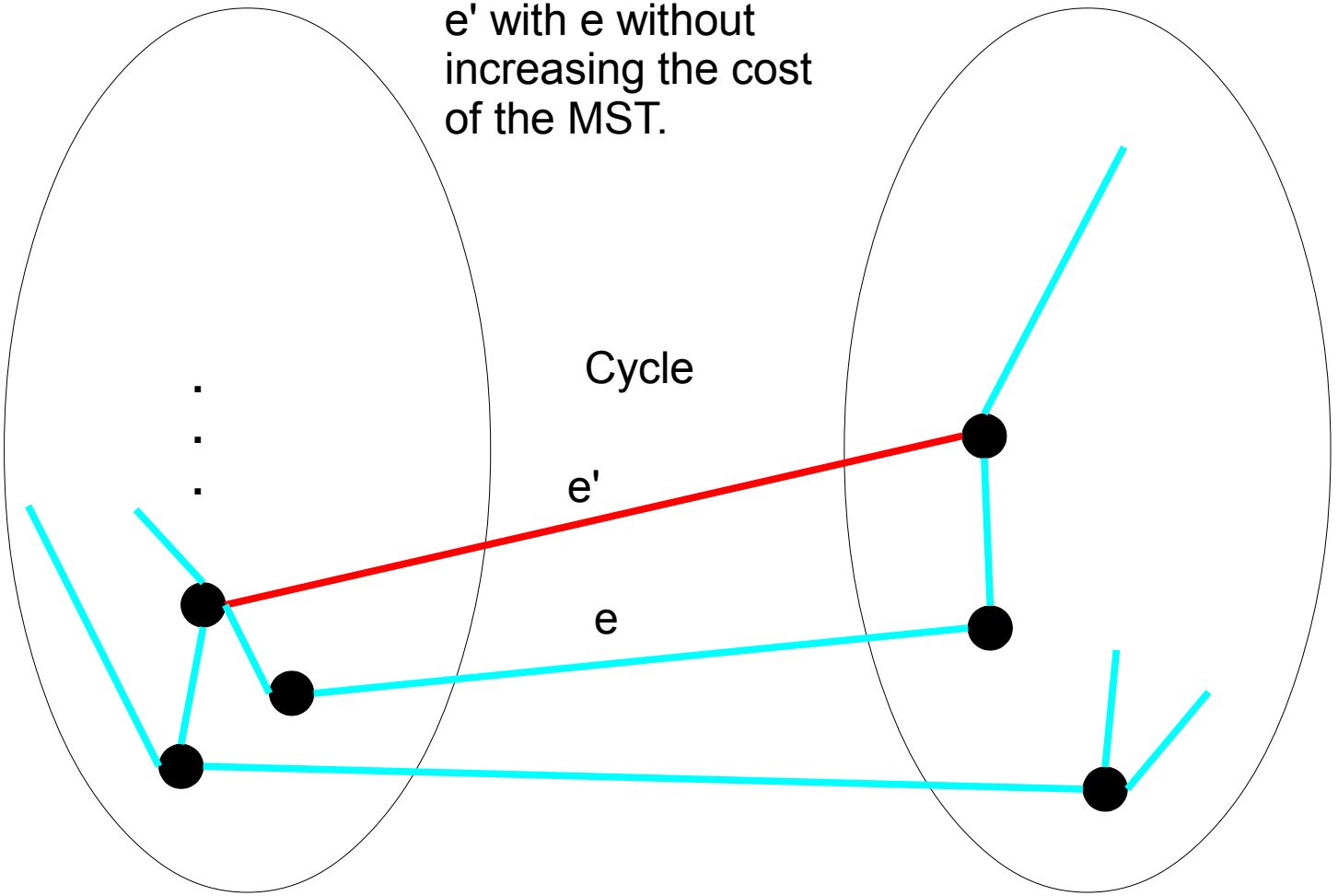
Since we had a MST, adding e creates a cycle.

So there must be some other edge, e' that goes between V1 and V2 in that cycle.

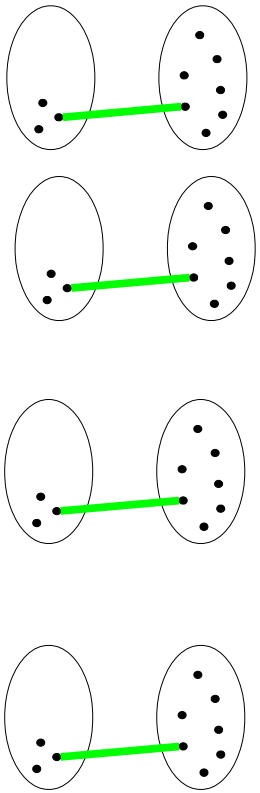


(Since it's a cycle, we have to go from V1 to V2, and then back again)

However, since e is the least edge between V1 and V2, we can replace e' with e without increasing the cost of the MST.



The graph is also connected because removing one edge from a cycle never disconnects the graph.



In Kruskal's algorithm, we adding the least edge e that does not form a cycle.

In other words, if our current connected components are C1, C2, C3...Cn, then e is a least edge between Ck and V-Ck for some connected component.