Minimum Spanning Tree

Graph Terminology

Node (Vertex, Point)

Graph Terminology

Node (Vertex, Point)

- An identifiable object

Graph Terminology

Node (Vertex, Point)

- An identifiable object
- Examples:

Graph Terminology

Node (Vertex, Point)

- An identifiable object
- Examples: person/user, location, item, etc.

Graph Terminology

Node (Vertex, Point)

- An identifiable object
- Examples: person/user, location, item, etc.

Edge (Connection)

Graph Terminology

Node (Vertex, Point)

- An identifiable object
- Examples: person/user, location, item, etc.

Edge (Connection)

- A relationship between two nodes represented as an **ordered** pair

Graph Terminology

Node (Vertex, Point)

- An identifiable object
- Examples: person/user, location, item, etc.

Edge (Connection)

- A relationship between two nodes represented as an <u>**ordered**</u> pair
- Example:

Graph Terminology

Node (Vertex, Point)

- An identifiable object
- Examples: person/user, location, item, etc.

Edge (Connection)

- A relationship between two nodes represented as an **ordered** pair
- Example: parent/child, road, friendship, etc.

Graph Terminology

Node (Vertex, Point)

- An identifiable object
- Examples: person/user, location, item, etc.

Edge (Connection)

- A relationship between two nodes represented as an **ordered** pair
- Example: parent/child, road, friendship, etc.
- The connected Nodes are called **end points**

Graph Terminology

Node (Vertex, Point)

- An identifiable object
- Examples: person/user, location, item, etc.

Edge (Connection)

- A relationship between two nodes represented as an **ordered** pair
- Example: parent/child, road, friendship, etc.
- The connected Nodes are called **end points**

Graph

Graph Terminology

Node (Vertex, Point)

- An identifiable object
- Examples: person/user, location, item, etc.

Edge (Connection)

- A relationship between two nodes represented as an <u>**ordered**</u> pair
- Example: parent/child, road, friendship, etc.
- The connected Nodes are called **end points**

Graph

- Collection of Nodes and Edges

Edges

Special Edges

Edges

Special Edges

- Loops, multi

Edges

Special Edges

- Loops, multi

Potential Properties

Edges

Special Edges

- Loops, multi

Potential Properties

- Direction

Edges

Special Edges

- Loops, multi

Potential Properties

- Direction
- Weight (cost)

Edge/Node Series

Path

Edge/Node Series

Path

- A series of distinct Nodes

Edge/Node Series

Path

- A series of distinct Nodes
- Adjacent Node pairs are connected by a **distinct** edge
    - From the first node of the series to the second in the case of directed edges

Edge/Node Series

Path

- A series of distinct Nodes
- Adjacent Node pairs are connected by a **<u>distinct</u>** edge
    - From the first node of the series to the second in the case of directed edges
- Sometimes the first Node in the series can be the last Node

Edge/Node Series

Path

- A series of distinct Nodes
- Adjacent Node pairs are connected by a **distinct** edge
    - From the first node of the series to the second in the case of directed edges
- Sometimes the first Node in the series can be the last Node

Cycle

Edge/Node Series

Path

- A series of distinct Nodes
- Adjacent Node pairs are connected by a **<u>distinct</u>** edge
    - From the first node of the series to the second in the case of directed edges
- Sometimes the first Node in the series can be the last Node

Cycle

- Path where the first and last Node are equal

Current Graph Type

Types we will work with (for now)

Current Graph Type

Types we will work with (for now)

- Undirected
  - All edges are undirected

Current Graph Type

Types we will work with (for now)

- Undirected
    - All edges are undirected
- Weighted
    - All edges have a weight

Current Graph Type

Types we will work with (for now)

- Undirected
  - All edges are undirected
- Weighted
  - All edges have a weight
- Connected
  - A graph where a path exists between all pairs of Nodes
  - The Edges’ endpoints’ order does not matter

Current Graph Type

Types we will work with (for now)

- Undirected
  - All edges are undirected
- Weighted
  - All edges have a weight
- Connected
  - A graph where a path exists between all pairs of Nodes
  - The Edges’ endpoints’ order does not matter
- Acyclic
  - A graph with no cycles

Current Graph Type

Types we will work with (for now)

- Undirected
  - All edges are undirected
- Weighted
  - All edges have a weight
- Connected
  - A graph where a path exists between all pairs of Nodes
  - The Edges’ endpoints’ order does not matter
- Acyclic
  - A graph with no cycles
- Tree
  - Undirected, Connected, Acyclic Graph

Problem

We have country. The country has cities. No citizen can move between cities currently. Two-way Highways between cities will be built. Each highway will have some cost based on the clearing and the material cost required. We can assume that the cost of building one highway will not affect the cost of building others.

Problem

We have country. The country has cities. No citizen can move between cities currently. Two-way Highways between cities will be built. Each highway will have some cost based on the clearing and the material cost required. We can assume that the cost of building one highway will not affect the cost of building others.

How do we ensure that we allow a citizen to travel between any pair of cities via the highways?

Problem

We have country. The country has cities. No citizen can move between cities currently. Two-way Highways between cities will be built. Each highway will have some cost based on the clearing and the material cost required. We can assume that the cost of building one highway will not affect the cost of building others.

How do we ensure that we allow a citizen to travel between any pair of cities via the highways?

How do we minimize the cost?

Problem

We have country. The country has cities. No citizen can move between cities currently. Two-way Highways between cities will be built. Each highway will have some cost based on the clearing and the material cost required. We can assume that the cost of building one highway will not affect the cost of building others.

How do we ensure that we allow a citizen to travel between any pair of cities via the highways?

How do we minimize the cost?

Answer: Lazy Builders

Analysis

No Cycles

Analysis

No Cycles

- Costs are all positive

Analysis

No Cycles

- Costs are all positive
- Reduce the cost by removing an edge on a cycle

Analysis

No Cycles

- Costs are all positive
- Reduce the cost by removing an edge on a cycle
- The graph will still be connected

Analysis

No Cycles

- Costs are all positive
- Reduce the cost by removing an edge on a cycle
- The graph will still be connected

Connected Acyclic Undirected Graph

Analysis

No Cycles

- Costs are all positive
- Reduce the cost by removing an edge on a cycle
- The graph will still be connected

Connected Acyclic Undirected Graph (Tree)

Analysis

No Cycles

- Costs are all positive
- Reduce the cost by removing an edge on a cycle
- The graph will still be connected

Connected Acyclic Undirected Graph (Tree)

To find the **Minimum Spanning Tree**

Two Algorithms

Two Algorithms

Prim’s

Two Algorithms

Prim's

- On sparse graphs requires fast incident edge(s) look up AND a Heap

Two Algorithms

Prim’s

- On sparse graphs requires fast incident edge(s) look up AND a Heap

Kruskal’s

Two Algorithms

Prim’s

- On sparse graphs requires fast incident edge(s) look up AND a Heap

Kruskal’s

- Requires a Merge Find Set data structure AND a sort

Prim’s

Initialize our answer with 0
Start with a single node in a set of connected nodes
Until all nodes are in the set do the following
    Find the edge that is smallest leaving our set of connected nodes
    Add the edges weight to our answer update the set of connected nodes
Return the answer

Kruskal’s

Initialize the answer to 0
Have each node in their own group
Sort all the edges
Loop through the sorted edges
    If the endpoints of the current edge are in different groups
        Join the groups
        Add the weight to the answer
Return the answer

Correctness

Exchange Argument

Correctness

Exchange Argument (Proof by Contraction)

Correctness

Exchange Argument (Proof by Contraction)

- Assume that a better answer exists that differs at some point from our answer

Correctness

Exchange Argument (Proof by Contraction)

- Assume that a better answer exists that differs at some point from our answer
- Start with the “best answer” (not our answer obvi.)

Correctness

Exchange Argument (Proof by Contraction)

- Assume that a better answer exists that differs at some point from our answer
- Start with the “best answer” (not our answer obvi.)
- Show that we can use our answer to make an even better answer

Correctness

Exchange Argument (Proof by Contraction)

- Assume that a better answer exists that differs at some point from our answer
- Start with the “best answer” (not our answer obvi.)
- Show that we can use our answer to make an even better answer
- A better answer than the best answer exists ※ (or ⊥)

Prim’s Exchange

It’s best to assume that all edge costs are distinct.

Prim’s Exchange

It’s best to assume that all edge costs are distinct.

Suppose our solution does not find the best answer.

Prim's Exchange

It's best to assume that all edge costs are distinct.

Suppose our solution does not find the best answer.

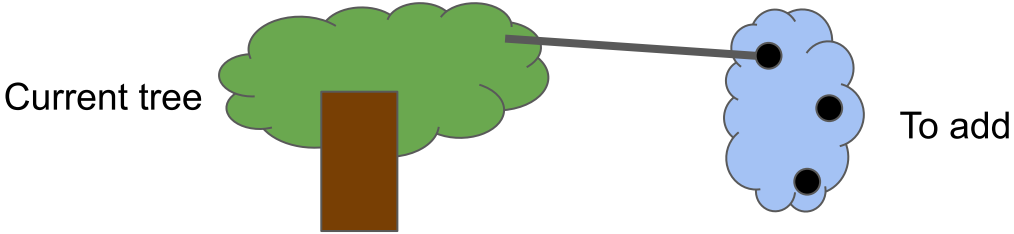
There is some smallest edge (**our edge**) we chose that is not in the best answer. :(

Prim’s Exchange

It’s best to assume that all edge costs are distinct.

Suppose our solution does not find the best answer.

There is some smallest edge (**our edge**) we chose that is not in the best answer. :(



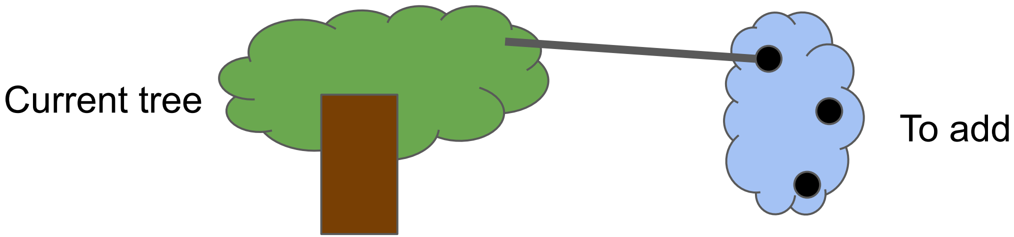
Prim’s Exchange

It’s best to assume that all edge costs are distinct.

Suppose our solution does not find the best answer.

There is some smallest edge (**our edge**) we chose that is not in the best answer. :(

The best answer chose some other path from the tree to that node not in the built up tree.



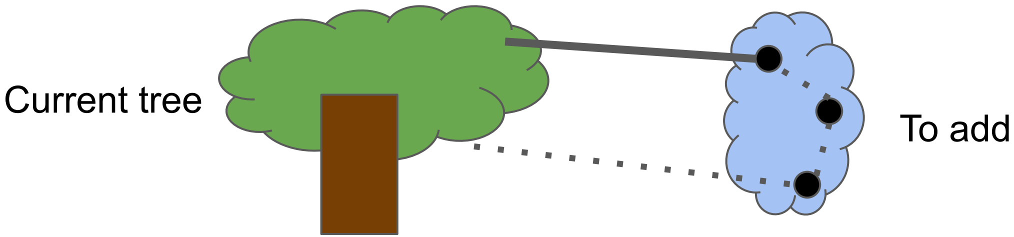
Prim's Exchange

It's best to assume that all edge costs are distinct.

Suppose our solution does not find the best answer.

There is some smallest edge (**our edge**) we chose that is not in the best answer. :(

The best answer chose some other path from the tree to that node not in the built up tree.



Prim's Exchange

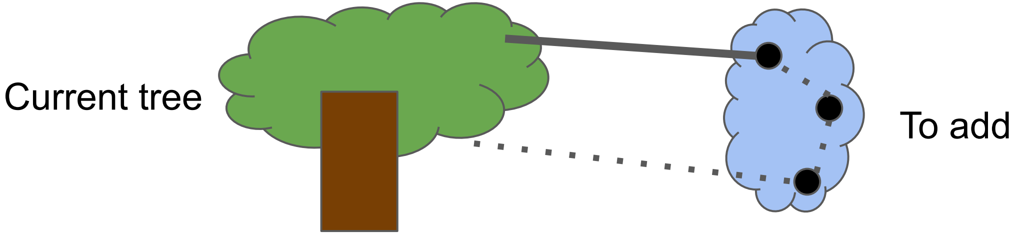
It's best to assume that all edge costs are distinct.

Suppose our solution does not find the best answer.

There is some smallest edge (**our edge**) we chose that is not in the best answer. :(

The best answer chose some other path from the tree to that node not in the built up tree.

Our edge added to the "best answer" would create a cycle.



Prim's Exchange

It's best to assume that all edge costs are distinct.

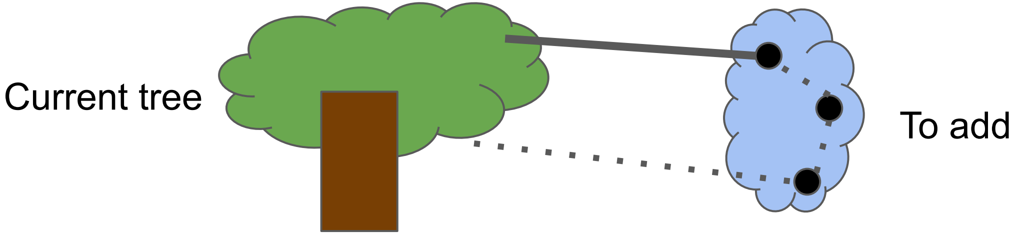
Suppose our solution does not find the best answer.

There is some smallest edge (**our edge**) we chose that is not in the best answer. :(

The best answer chose some other path from the tree to that node not in the built up tree.

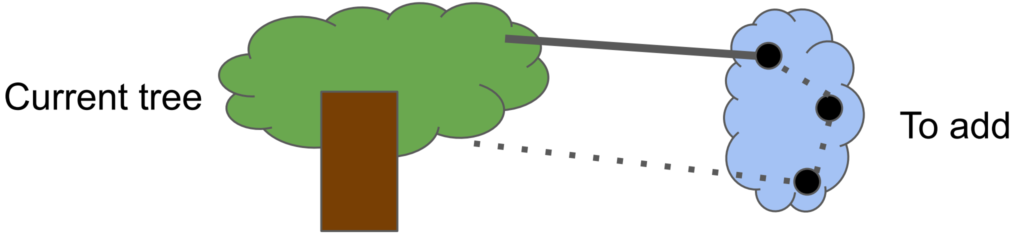
Our edge added to the "best answer" would create a cycle.

The other edge connecting to the set that formed the cycle will be larger than **our edge**



Prim’s Exchange (cont.)

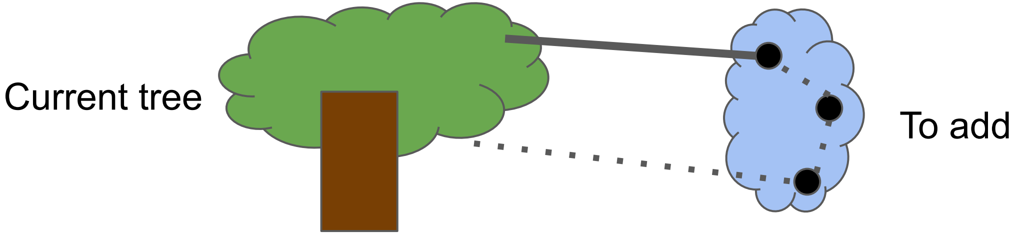
Removing the other edge will remove the cycle leaving making the graph a tree, but the overall cost would decrease.



Prim’s Exchange (cont.)

Removing the other edge will remove the cycle leaving making the graph a tree, but the overall cost would decrease.

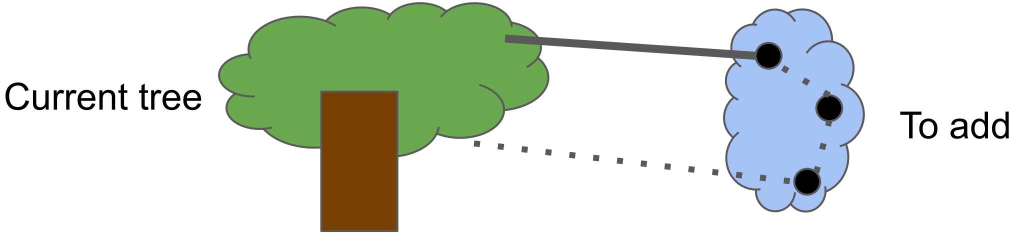
We found a better answer!



Prim’s Exchange (cont.)

Removing the other edge will remove the cycle leaving making the graph a tree, but the overall cost would decrease.

We found a better answer! (Contradiction)



Kruskal’s Exchange

Again it’s best to assume that all edge costs are distinct.

Kruskal’s Exchange

Again it’s best to assume that all edge costs are distinct.

Suppose our solution does not find the best answer.

Kruskal’s Exchange

Again it’s best to assume that all edge costs are distinct.

Suppose our solution does not find the best answer.

There is some smallest edge (**our edge**) we chose that is not in the best answer.

Kruskal's Exchange

Again it's best to assume that all edge costs are distinct.

Suppose our solution does not find the best answer.

There is some smallest edge (**our edge**) we chose that is not in the best answer.

It's a little different now because we now have more groups of nodes to consider.

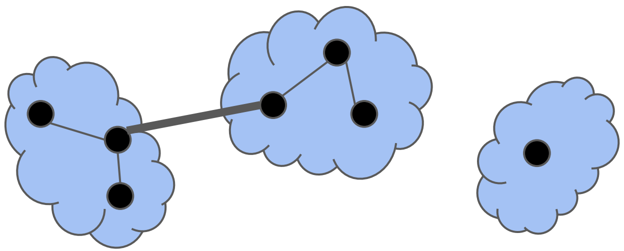
Kruskal’s Exchange

Again it’s best to assume that all edge costs are distinct.

Suppose our solution does not find the best answer.

There is some smallest edge (**our edge**) we chose that is not in the best answer.

It’s a little different now because we now have more groups of nodes to consider.



Kruskal’s Exchange

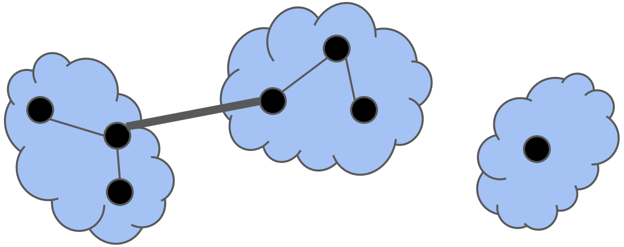
Again it’s best to assume that all edge costs are distinct.

Suppose our solution does not find the best answer.

There is some smallest edge (**our edge**) we chose that is not in the best answer.

It’s a little different now because we now have more groups of nodes to consider.

These two groups need to be connected eventually.



Kruskal’s Exchange

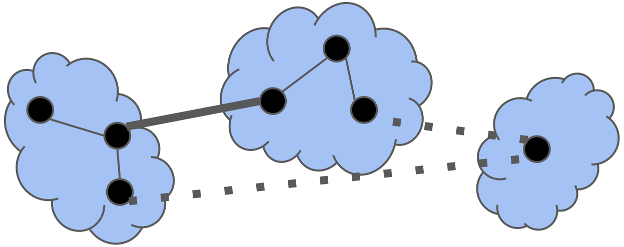
Again it’s best to assume that all edge costs are distinct.

Suppose our solution does not find the best answer.

There is some smallest edge (**our edge**) we chose that is not in the best answer.

It’s a little different now because we now have more groups of nodes to consider.

These two groups need to be connected eventually.



Kruskal's Exchange

Again it's best to assume that all edge costs are distinct.

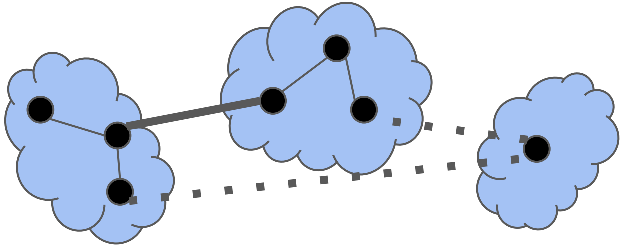
Suppose our solution does not find the best answer.

There is some smallest edge (**our edge**) we chose that is not in the best answer.

It's a little different now because we now have more groups of nodes to consider.

These two groups need to be connected eventually.

A cycle will be formed by adding **our edge**.



Kruskal’s Exchange

Again it’s best to assume that all edge costs are distinct.

Suppose our solution does not find the best answer.

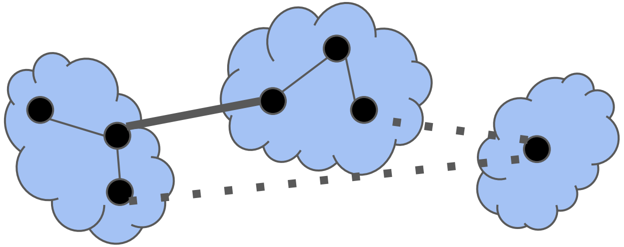
There is some smallest edge (**our edge**) we chose that is not in the best answer.

It’s a little different now because we now have more groups of nodes to consider.

These two groups need to be connected eventually.

A cycle will be formed by adding **our edge**.

By removing one of the other edges and adding **our edge** we form a better solution.



Similar Problems

Minimum Spanning Arborescence

Steiner Tree** (general version is hard)