

FIT2004

Algorithms and Data Structures

Ian Wern Han Lim
lim.wern.han@monash.edu

Referencing materials by
Nathan Compane, Aamir Cheema, Arun Konagurthu and Lloyd Allison



Faculty of Information Technology, Monash University

COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

This material has been reproduced and communicated to you by or on behalf of Monash University pursuant to Part VB of the Copyright Act 1968 (the Act). The material in this communication may be subject to copyright under the Act. Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act. Do not remove this notice

Ready?

Agenda

- Minimum Spanning Tree (MST)

Agenda

- Minimum Spanning Tree (MST)
 - Prim's algorithm
 - Kruskal's algorithm

Let us begin...

Minimum Spanning Tree

What is it?

Minimum Spanning Tree

What is it?

- A tree

Minimum Spanning Tree

What is it?

- A tree
- Spanning every vertex

Minimum Spanning Tree

What is it?

- A tree
- Spanning every vertex
- Minimum total edges

Minimum Spanning Tree

What is it?

- A tree
- Spanning every vertex
 - Minimum number of edges to connect all vertex? True or False?
 - Maximum number of edges in graph without cycle? True or False?
- Minimum total edges weight

Minimum Spanning Tree

What is it?

- A tree
 - No cycle
 - Undirected
- Spanning every vertex
 - Minimum number of edges to connect all vertex
 - Maximum number of edges in graph without cycle
- Minimum total edges weight



Minimum Spanning Tree

What is it?

- A tree
 - No cycle
 - Undirected
- Spanning every vertex
 - Minimum number of edges to connect all vertex
 - Maximum number of edges in graph without cycle
- Minimum total edges weight

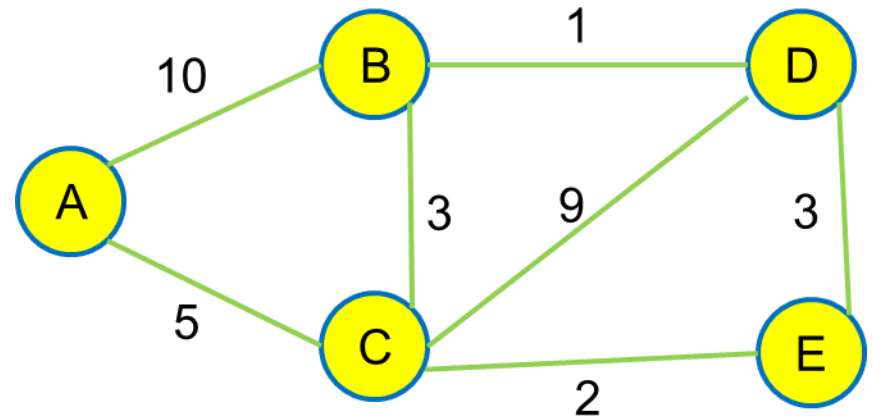
Spanning Tree:

- A **spanning tree** of a general undirected weighted graph G is a tree that **spans** G (i.e., a tree that includes every vertex of G) and is a **subgraph** of G (i.e., every edge in the spanning tree belongs to G).

Minimum Spanning Tree

What is it?

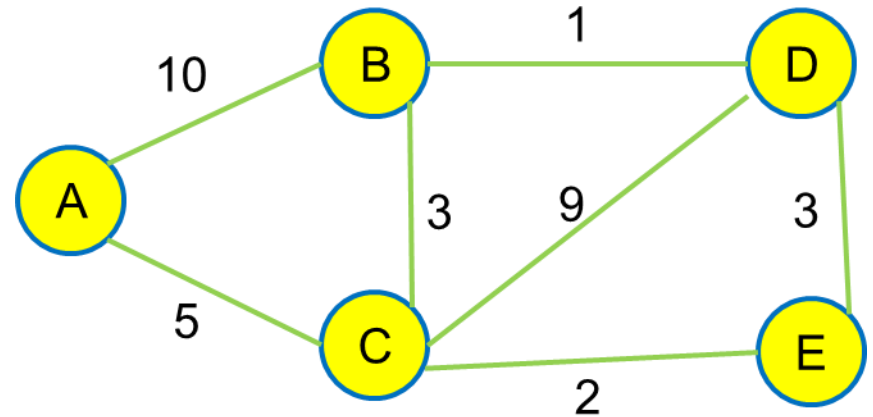
- Let say we have a graph



Minimum Spanning Tree

What is it?

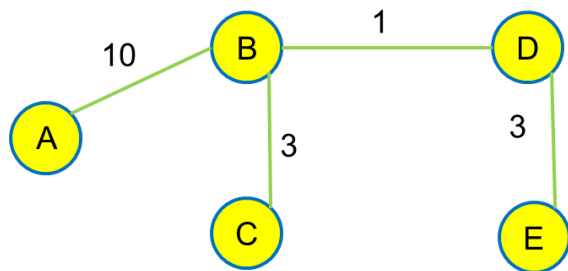
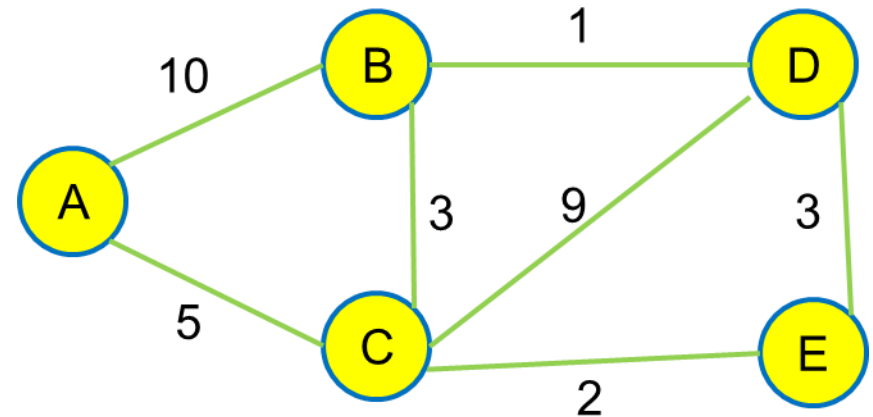
- Let say we have a graph
 - Can you form spanning trees?



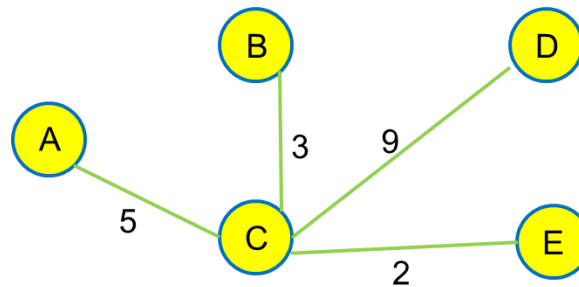
Minimum Spanning Tree

What is it?

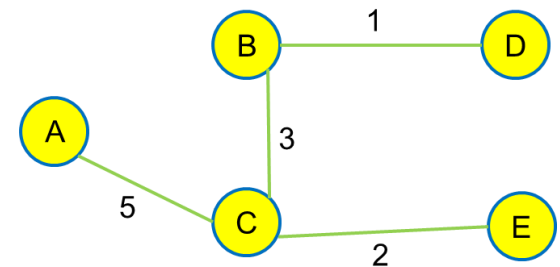
- Let say we have a graph
 - Can you form spanning trees?



Spanning Tree 1



Spanning Tree 2

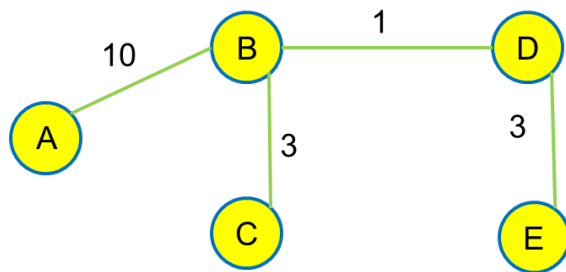
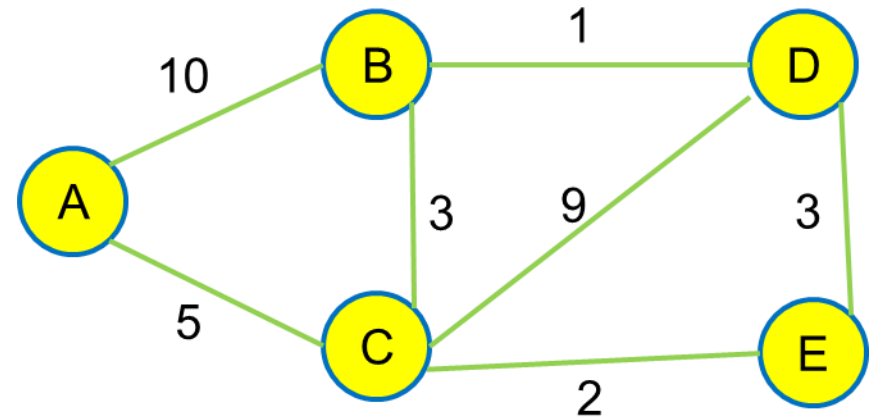


Spanning Tree 3

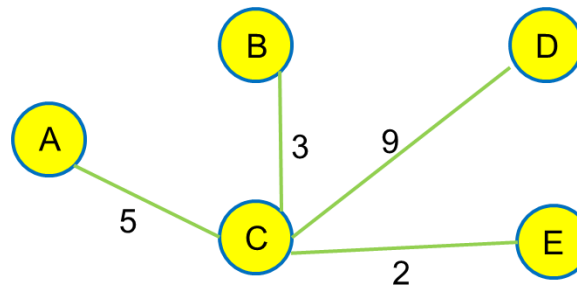
Minimum Spanning Tree

What is it?

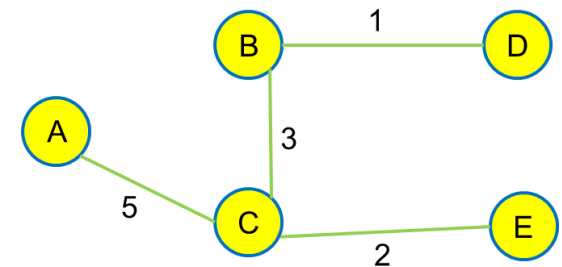
- Let say we have a graph
 - Can you form spanning trees?
 - Which is the minimum?



Spanning Tree 1



Spanning Tree 2

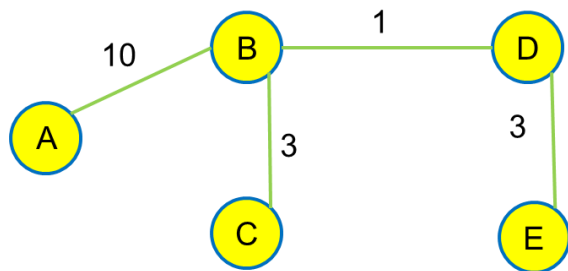
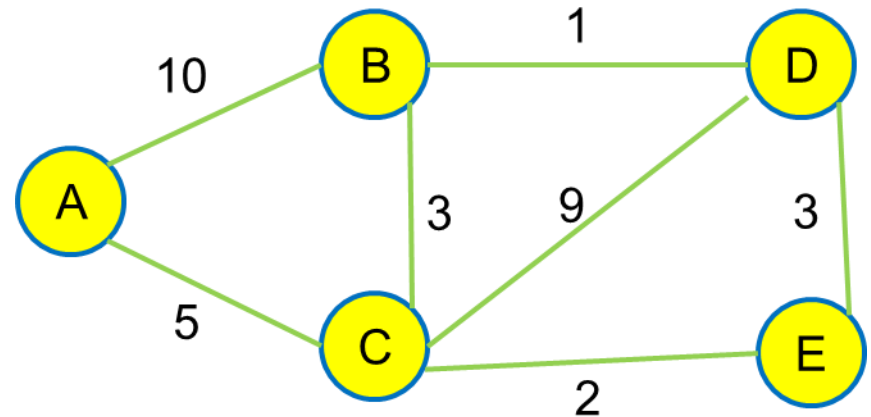


Spanning Tree 3

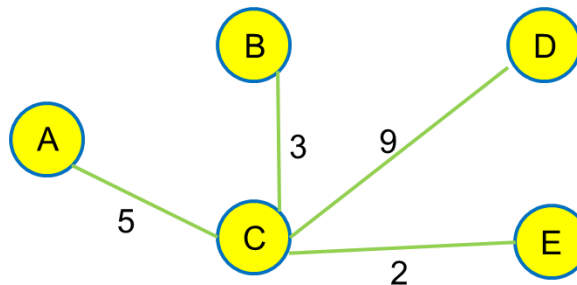
Minimum Spanning Tree

What is it?

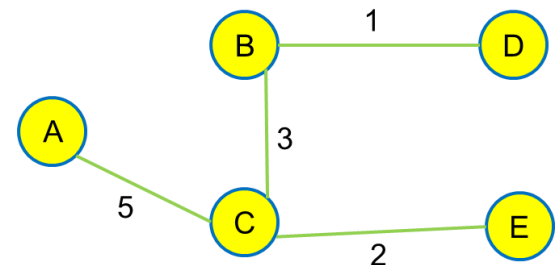
- Let say we have a graph
 - Can you form spanning trees?
 - Which is the minimum?
 - Tree 1 = $10 + 1 + 3 + 3$
 - Tree 2 = $5 + 3 + 9 + 2$
 - Tree 3 = $5 + 3 + 2 + 1$



Spanning Tree 1



Spanning Tree 2

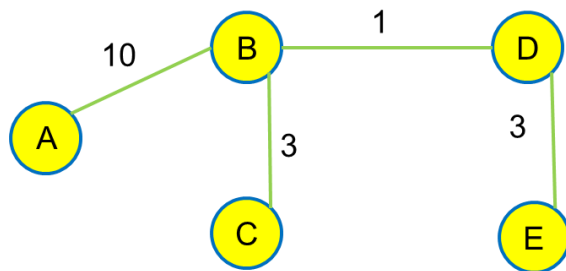
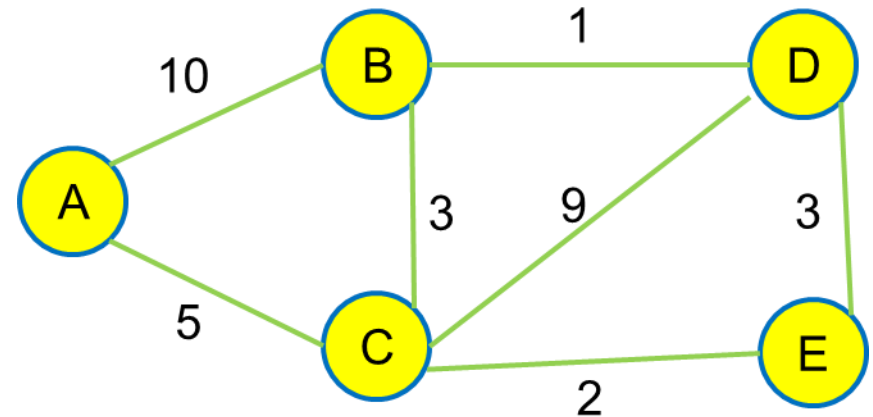


Spanning Tree 3

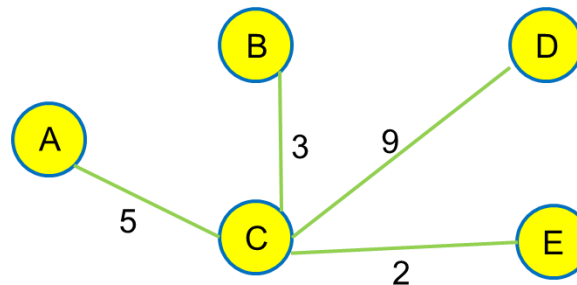
Minimum Spanning Tree

What is it?

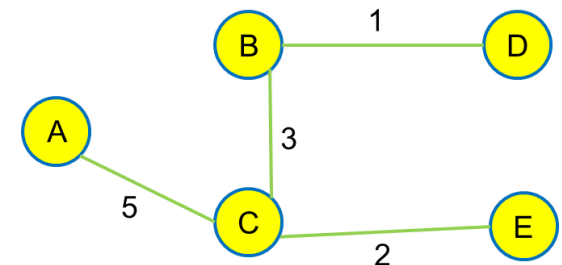
- Let say we have a graph
 - Can you form spanning trees?
 - Which is the minimum?
 - Tree 1 = $10 + 1 + 3 + 3 = 17$
 - Tree 2 = $5 + 3 + 9 + 2 = 19$
 - Tree 3 = $5 + 3 + 2 + 1 = 11$



Spanning Tree 1



Spanning Tree 2



Spanning Tree 3

Minimum Spanning Tree

What is it?

- Let say we have a graph
 - Can you form spanning trees?
 - Which is the minimum?

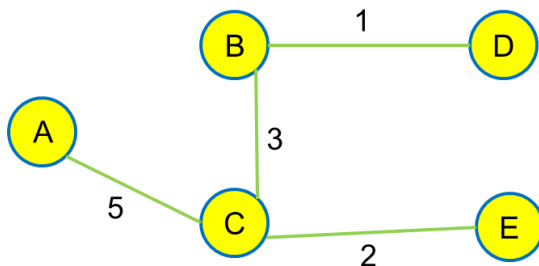
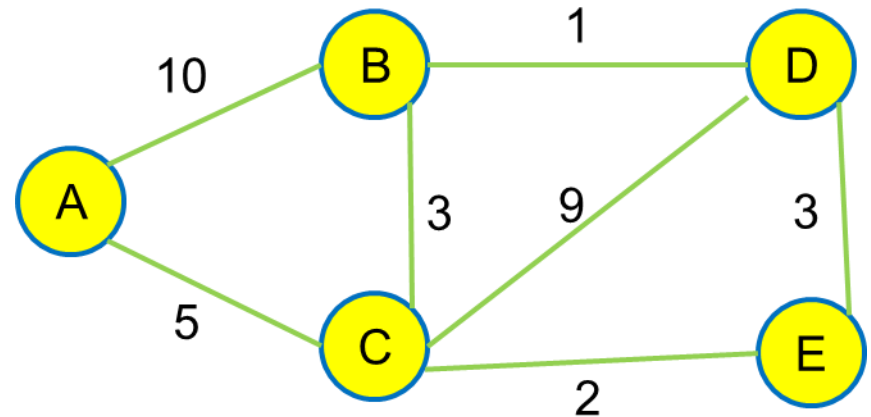
- Tree 1 = $10 + 1 + 3 + 3 = 17$

- Tree 2 = $5 + 3 + 9 + 2 = 19$

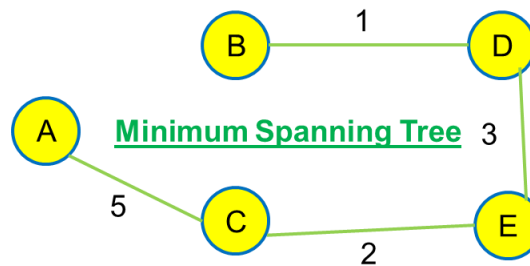
- Tree 3 = $5 + 3 + 2 + 1 = 11$

- Tree 4 = $5 + 2 + 3 + 1 = 11$

} Not unique



Spanning Tree 3



Spanning Tree 4

Questions?

Minimum Spanning Tree

How to build it?

- Prim's
- Kruskal's

Minimum Spanning Tree

How to build it?

- Prim's
 - Growing of tree
- Kruskal's
 - Merging of trees

Minimum Spanning Tree

How to build it?

- Prim's
 - Growing of tree
- Kruskal's
 - Merging of trees
- Both are greedy

Minimum Spanning Tree

How to build it?

- Prim's
 - Growing of tree
- Kruskal's
 - Merging of trees
- Both are greedy
 - Choose local optimal
 - Believe to get global optimal

Minimum Spanning Tree

How to build it?

- Prim's
 - Growing of tree
- Kruskal's
 - Merging of trees
- Both are greedy
 - Choose local optimal
 - Believe to get global optimal
 - We will learn to prove it later

Minimum Spanning Tree

How to build it?

- Prim's
 - Growing of tree
 - Very similar to Dijkstra's. Can be known a Prim-Dijkstra
- Kruskal's
 - Merging of trees
- Both are greedy
 - Choose local optimal
 - Believe to get global optimal
 - We will learn to prove it later

Minimum Spanning Tree

How to build it?

- Prim's
 - Growing of tree
 - Very similar to Dijkstra's. Can be known a Prim-Dijkstra
Instead of nearest vertex from source, it is nearest vertex from tree!
- Kruskal's
 - Merging of trees
- Both are greedy
 - Choose local optimal
 - Believe to get global optimal
 - We will learn to prove it later

Questions?

Prim's Algorithm

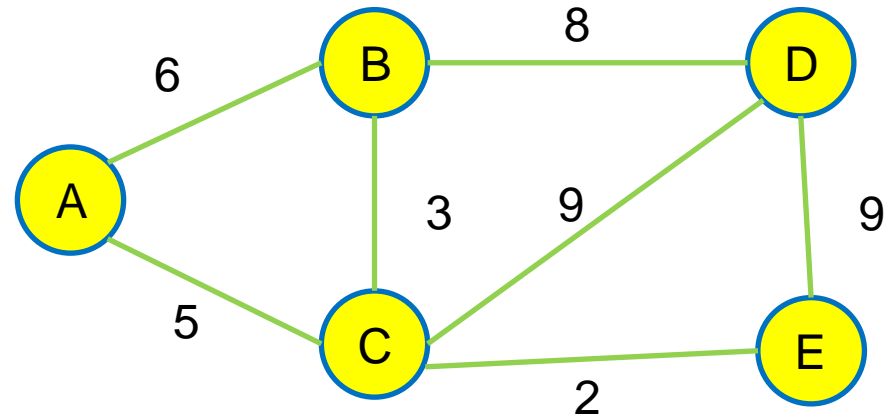
Growing of MST

- Very similar to Dijkstra
 - Choosing nearest vertex to tree

Prim's Algorithm

Growing of MST

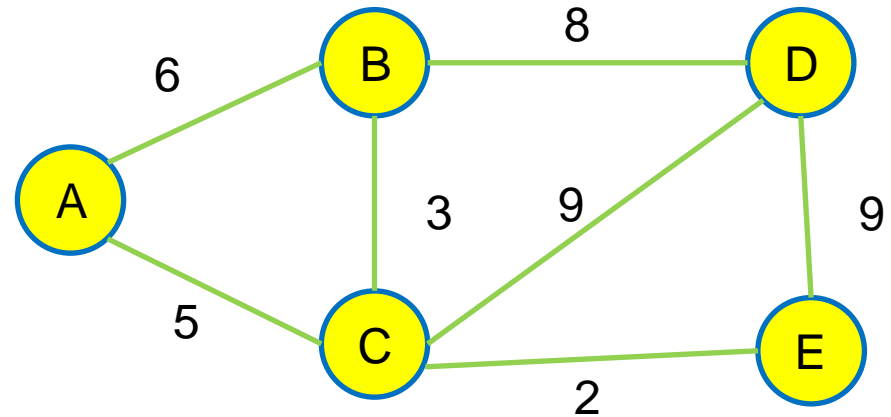
- Very similar to Dijkstra
 - Choosing nearest vertex to tree
 - Let us try it out



Prim's Algorithm

Growing of MST

- Very similar to Dijkstra
 - Choosing nearest vertex to tree
 - Let us try it out

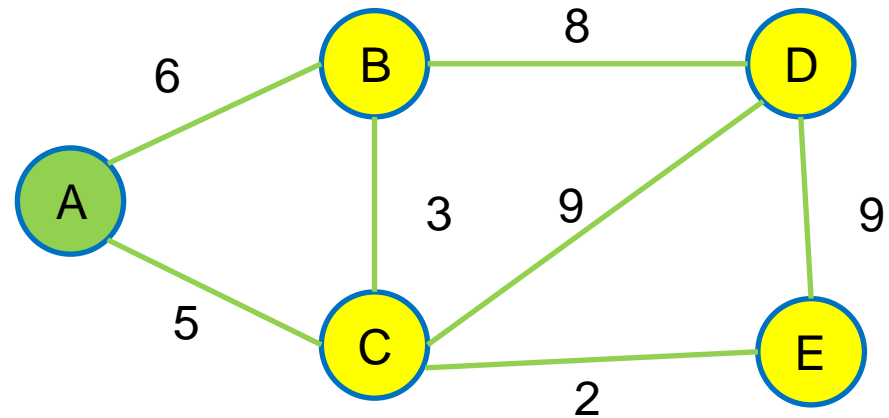


A	B	C	D	E
0	inf	inf	inf	inf

Prim's Algorithm

Growing of MST

- Very similar to Dijkstra
 - Choosing nearest vertex to tree
 - Let us try it out
 - Start from A

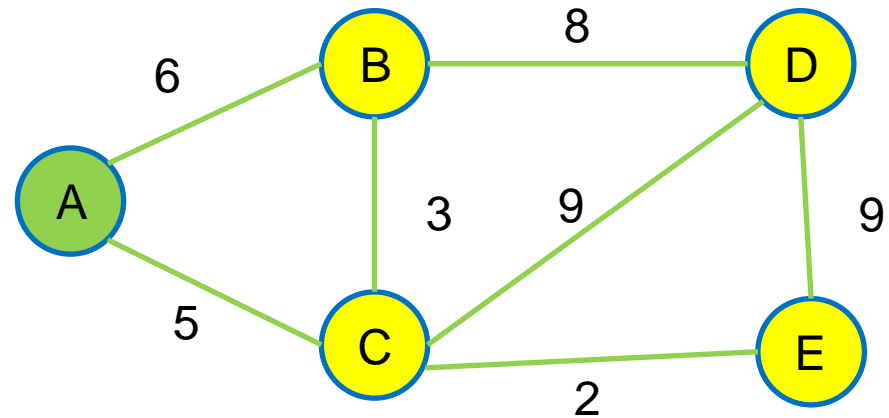


A	B	C	D	E
0	inf	inf	inf	inf

Prim's Algorithm

Growing of MST

- Very similar to Dijkstra
 - Choosing nearest vertex to tree
 - Let us try it out
 - Start from A
 - Update adjacent B and C

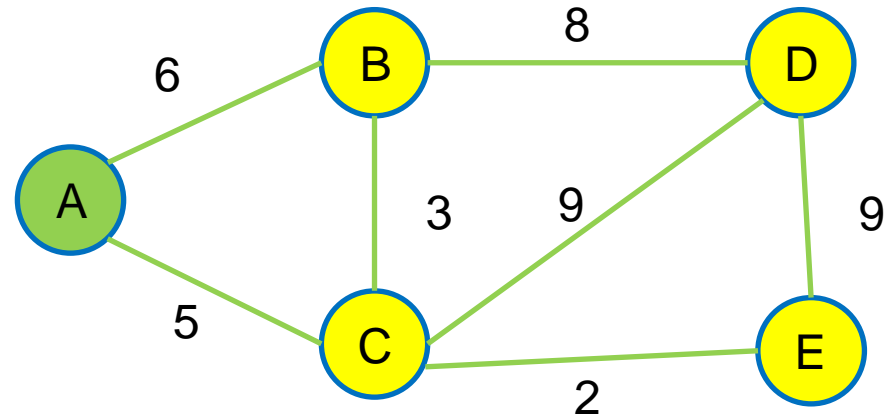


A	B	C	D	E
0	6, A	inf	inf	inf

Prim's Algorithm

Growing of MST

- Very similar to Dijkstra
 - Choosing nearest vertex to tree
 - Let us try it out
 - Start from A
 - Update adjacent B and C

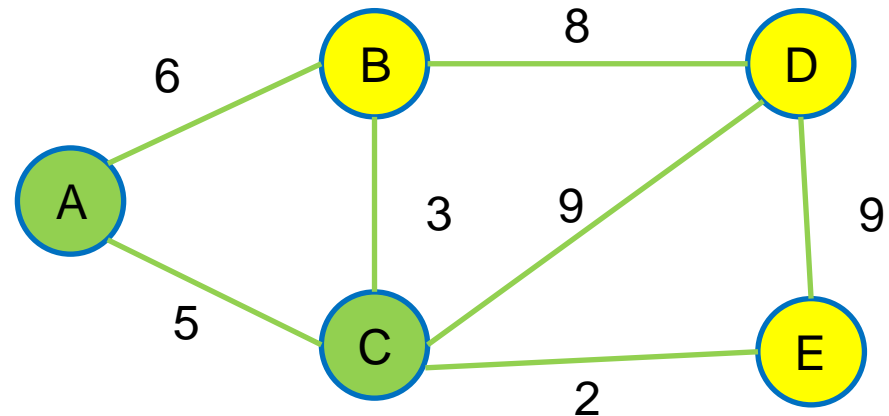


A	B	C	D	E
0	6, A	5, A	inf	inf

Prim's Algorithm

Growing of MST

- Very similar to Dijkstra
 - Choosing nearest vertex to tree
 - Let us try it out
 - Start from A
 - Update adjacent B and C
 - Choose closest C

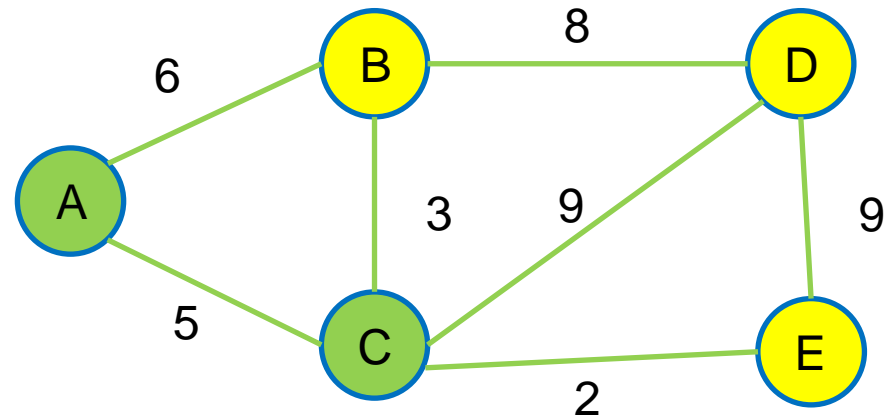


A	B	C	D	E
0	6, A	5, A	inf	inf

Prim's Algorithm

Growing of MST

- Very similar to Dijkstra
 - Choosing nearest vertex to tree
 - Let us try it out
 - Start from A
 - Update adjacent B and C
 - Choose closest C
 - Update adjacent B, D, E

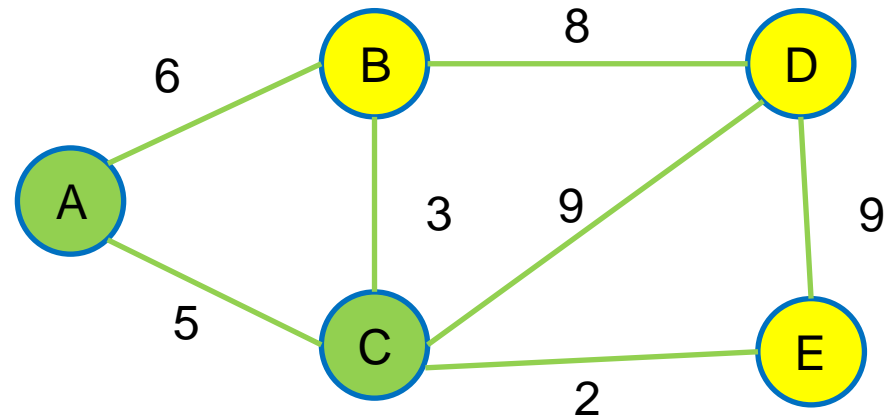


A	B	C	D	E
0	6	5, A	inf	inf

Prim's Algorithm

Growing of MST

- Very similar to Dijkstra
 - Choosing nearest vertex to tree
 - Let us try it out
 - Start from A
 - Update adjacent B and C
 - Choose closest C
 - Update adjacent B, D, E

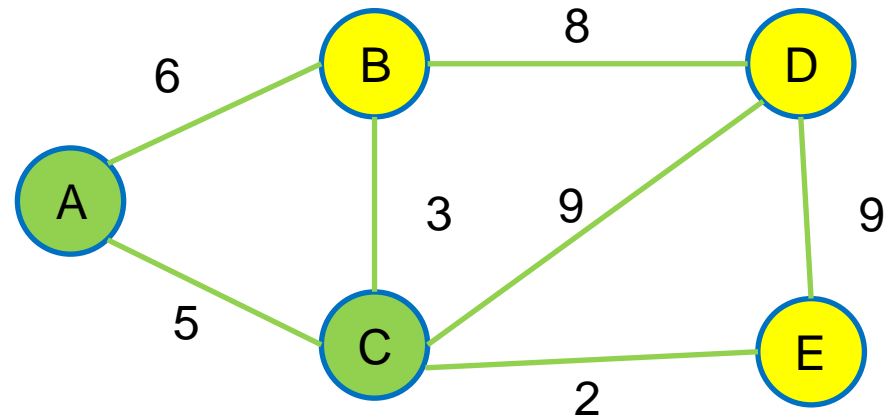


A	B	C	D	E
0	6vs3	5, A	inf	inf

Prim's Algorithm

Growing of MST

- Very similar to Dijkstra
 - Choosing nearest vertex to tree
 - Let us try it out
 - Start from A
 - Update adjacent B and C
 - Choose closest C
 - Update adjacent B, D, E

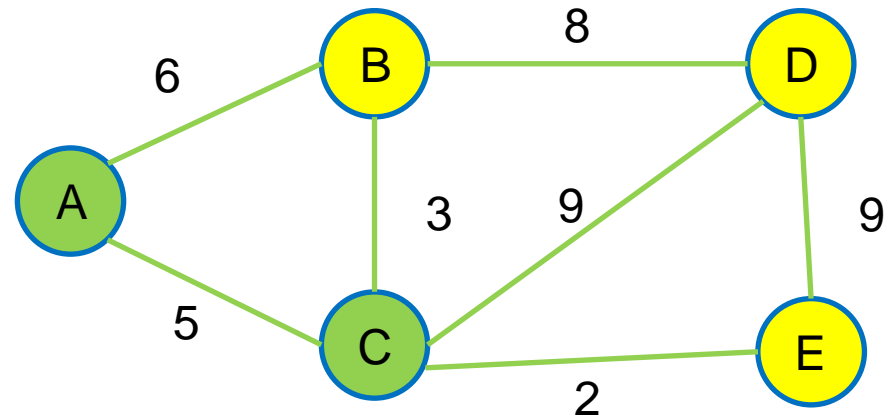


A	B	C	D	E
0	3, C	5, A	inf	inf

Prim's Algorithm

Growing of MST

- Very similar to Dijkstra
 - Choosing nearest vertex to tree
 - Let us try it out
 - Start from A
 - Update adjacent B and C
 - Choose closest C
 - Update adjacent B, D, E

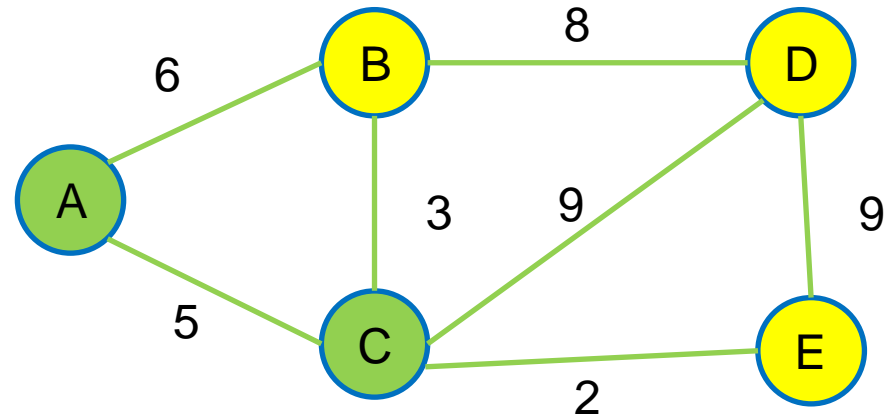


A	B	C	D	E
0	3, C	5, A	9, C	inf

Prim's Algorithm

Growing of MST

- Very similar to Dijkstra
 - Choosing nearest vertex to tree
 - Let us try it out
 - Start from A
 - Update adjacent B and C
 - Choose closest C
 - Update adjacent B, D, E

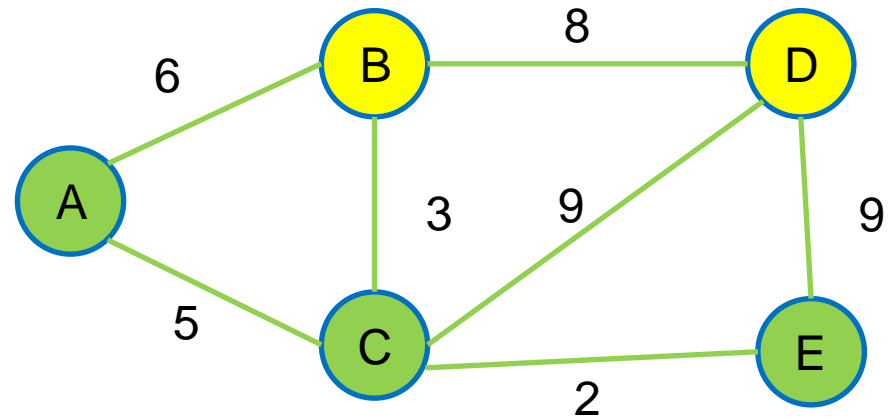


A	B	C	D	E
0	3, C	5, A	9, C	2, C

Prim's Algorithm

Growing of MST

- Very similar to Dijkstra
 - Choosing nearest vertex to tree
 - Let us try it out
 - Start from A
 - Update adjacent B and C
 - Choose closest C
 - Update adjacent B, D, E
 - Choose closest E

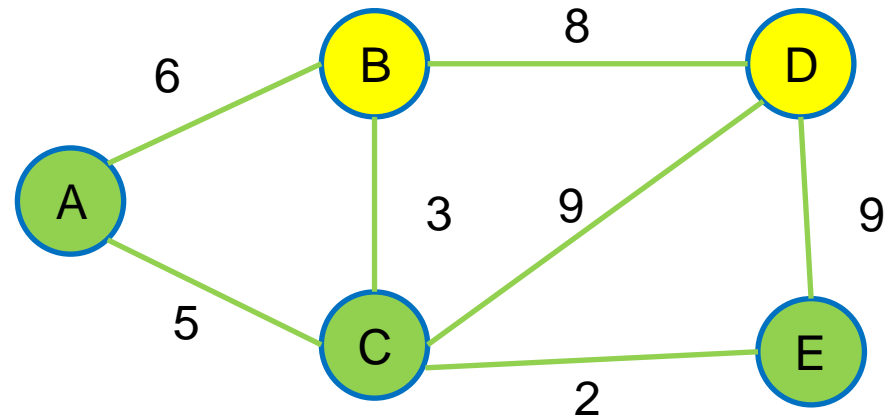


A	B	C	D	E
0	3, C	5, A	9, C	2, C

Prim's Algorithm

Growing of MST

- Very similar to Dijkstra
 - Choosing nearest vertex to tree
 - Let us try it out
 - Start from A
 - Update adjacent B and C
 - Choose closest C
 - Update adjacent B, D, E
 - Choose closest E
 - Update adjacent D

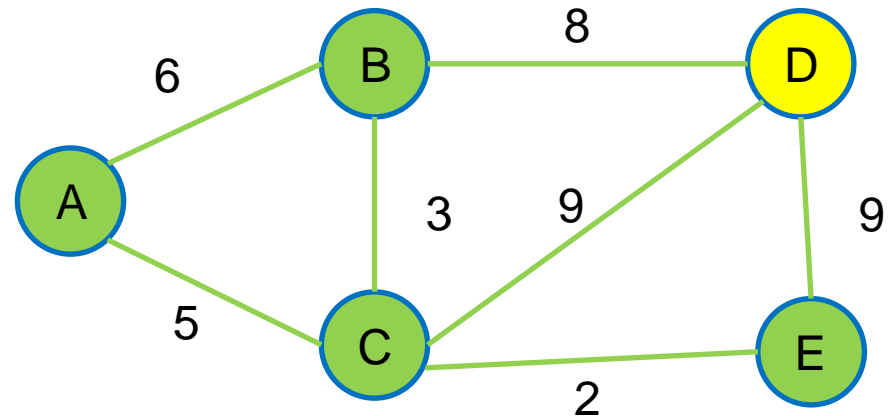


A	B	C	D	E
0	3, C	5, A	9, C	2, C

Prim's Algorithm

Growing of MST

- Very similar to Dijkstra
 - Choosing nearest vertex to tree
 - Let us try it out
 - Start from A
 - Update adjacent B and C
 - Choose closest C
 - Update adjacent B, D, E
 - Choose closest E
 - Update adjacent D
 - Choose closest B

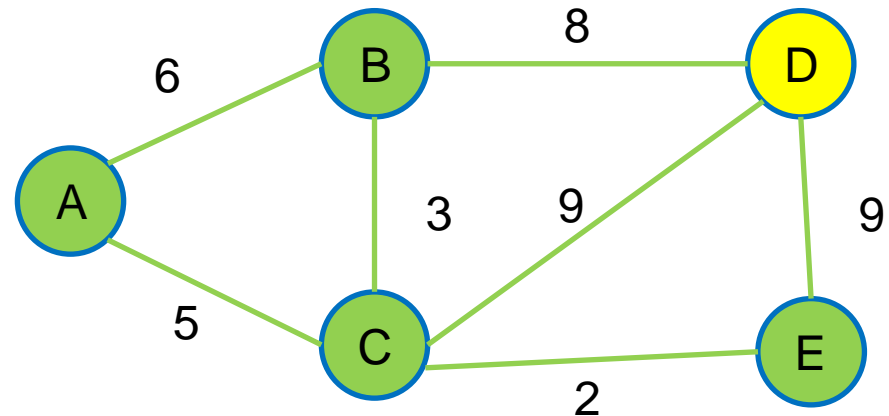


A	B	C	D	E
0	3, C	5, A	9, C	2, C

Prim's Algorithm

Growing of MST

- Very similar to Dijkstra
 - Choosing nearest vertex to tree
 - Let us try it out
 - Start from A
 - Update adjacent B and C
 - Choose closest C
 - Update adjacent B, D, E
 - Choose closest E
 - Update adjacent D
 - Choose closest B
 - Update adjacent D

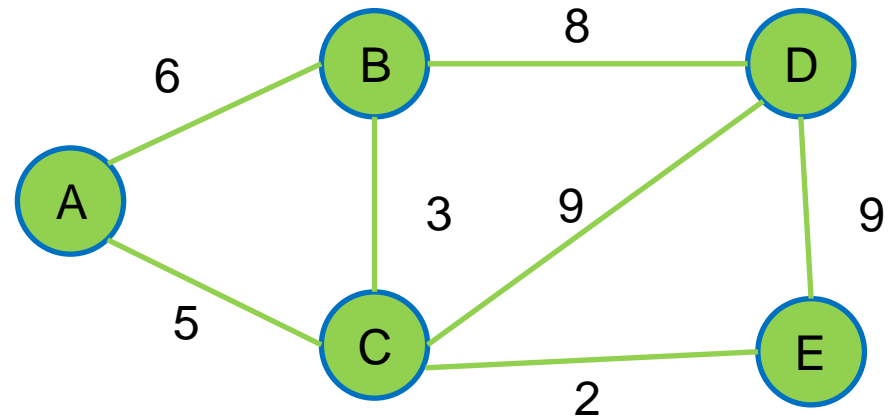


A	B	C	D	E
0	3, C	5, A	8, B	2, C

Prim's Algorithm

Growing of MST

- Very similar to Dijkstra
 - Choosing nearest vertex to tree
 - Let us try it out
 - Start from A
 - Update adjacent B and C
 - Choose closest C
 - Update adjacent B, D, E
 - Choose closest E
 - Update adjacent D
 - Choose closest B
 - Update adjacent D
 - Choose closest D

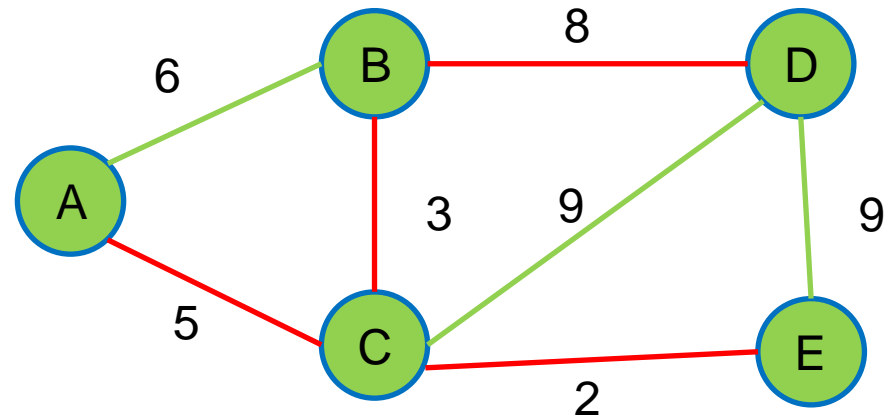


A	B	C	D	E
0	3, C	5, A	8, B	2, C

Prim's Algorithm

Growing of MST

- Very similar to Dijkstra
 - Choosing nearest vertex to tree
 - Let us try it out
 - Start from A
 - Update adjacent B and C
 - Choose closest C
 - Update adjacent B, D, E
 - Choose closest E
 - Update adjacent D
 - Choose closest B
 - Update adjacent D
 - Choose closest D
 - Have all of the edges

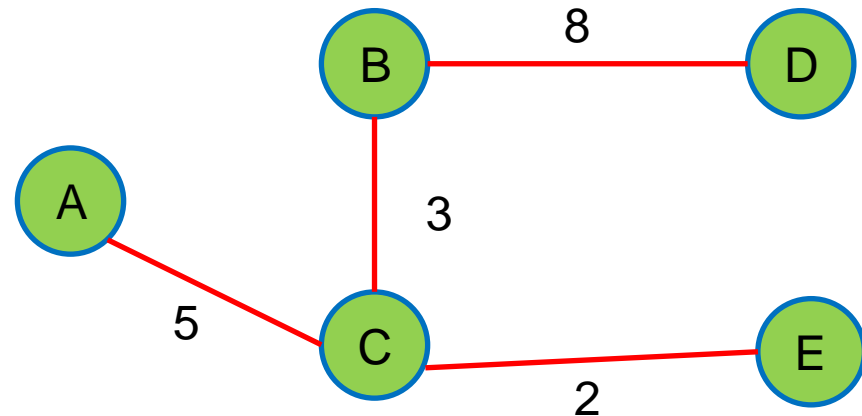


A	B	C	D	E
0	3, C	5, A	8, B	2, C

Prim's Algorithm

Growing of MST

- Very similar to Dijkstra
 - Choosing nearest vertex to tree
 - Let us try it out
 - Start from A
 - Update adjacent B and C
 - Choose closest C
 - Update adjacent B, D, E
 - Choose closest E
 - Update adjacent D
 - Choose closest B
 - Update adjacent D
 - Choose closest D
 - Have all of the edges



A	B	C	D	E
0	3, C	5, A	8, B	2, C

Prim's Algorithm

Growing of MST

- So take Dijkstra
 - Modify the distance update/ calculation for edge $\langle u, v, w \rangle$
 - Instead of $v.\text{distance} = u.\text{distance} + w$
 - Change to $v.\text{distance} = w$

Prim's Algorithm

Growing of MST

- So take Dijkstra
 - Modify the distance update/ calculation for edge $\langle u, v, w \rangle$
 - Instead of $v.\text{distance} = u.\text{distance} + w$
 - Change to $v.\text{distance} = w$
 - Perform relaxation only if distance is smaller

Prim's Algorithm

Growing of MST

- So take Dijkstra
 - Modify the distance update/ calculation for edge $\langle u, v, w \rangle$
 - Instead of $v.\text{distance} = u.\text{distance} + w$
 - Change to $v.\text{distance} = w$
 - Perform relaxation only if distance is smaller
- So what is the complexity?

Prim's Algorithm

Growing of MST

- So take Dijkstra
 - Modify the distance update/ calculation for edge $\langle u, v, w \rangle$
 - Instead of $v.\text{distance} = u.\text{distance} + w$
 - Change to $v.\text{distance} = w$
 - Perform relaxation only if distance is smaller
- So what is the complexity?
 - Same as Dijkstra $O(V \log V + E \log V)$
 - Thus $O(E \log V)$

Questions?

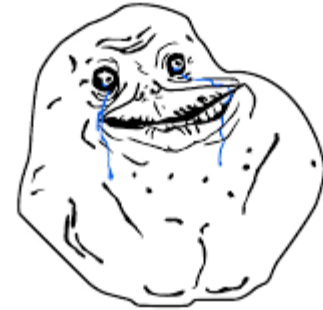
Kruskal's Algorithm

Combining (Union of) Trees

Kruskal's Algorithm

Combining (Union of) Trees

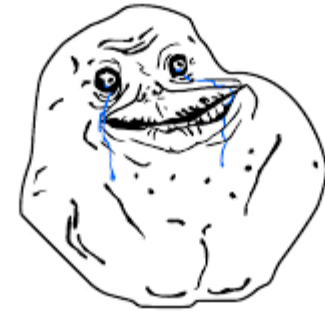
- Imagine every vertex is a tree
 - Only 1 node #ForeverAlone



Kruskal's Algorithm

Combining (Union of) Trees

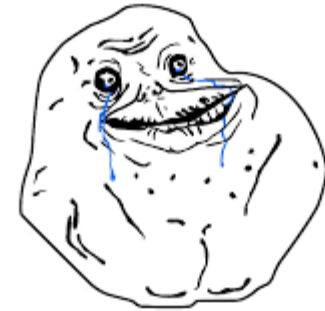
- Imagine every vertex is a tree
 - Only 1 node #ForeverAlone
 - Trees are connected by edges



Kruskal's Algorithm

Combining (Union of) Trees

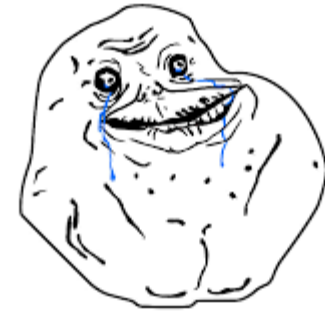
- Imagine every vertex is a tree
 - Only 1 node #ForeverAlone
 - Trees are connected by edges
 - Adding edge $\langle u, v, w \rangle$ combine the trees of vertex u and vertex v



Kruskal's Algorithm

Combining (Union of) Trees

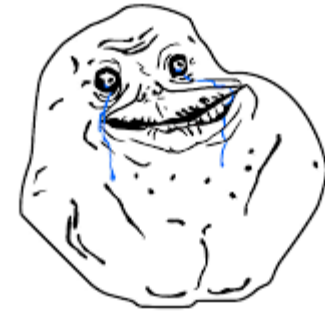
- Imagine every vertex is a tree
 - Only 1 node #ForeverAlone
 - Trees are connected by edges
 - Adding edge $\langle u, v, w \rangle$ combine the trees of vertex u and vertex v
 - Only add if vertex u and vertex v are not in the same tree. Why?



Kruskal's Algorithm

Combining (Union of) Trees

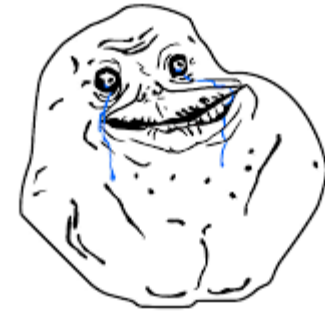
- Imagine every vertex is a tree
 - Only 1 node #ForeverAlone
 - Trees are connected by edges
 - Adding edge $\langle u, v, w \rangle$ combine the trees of vertex u and vertex v
 - Only add if vertex u and vertex v are not in the same tree. Why? **NO CYCLE**



Kruskal's Algorithm

Combining (Union of) Trees

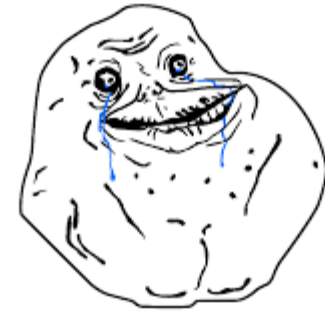
- Imagine every vertex is a tree
 - Only 1 node #ForeverAlone
 - Trees are connected by edges
 - Adding edge $\langle u, v, w \rangle$ combine the trees of vertex u and vertex v
 - Only add if vertex u and vertex v are not in the same tree. Why? **NO CYCLE**
- So how do we do it?



Kruskal's Algorithm

Combining (Union of) Trees

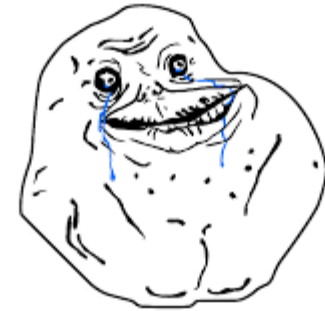
- Imagine every vertex is a tree
 - Only 1 node #ForeverAlone
 - Trees are connected by edges
 - Adding edge $\langle u, v, w \rangle$ combine the trees of vertex u and vertex v
 - Only add if vertex u and vertex v are not in the same tree. Why? **NO CYCLE**
- So how do we do it?
 - Take add edges
 - Sort it
 - Then go through the edges one by one



Kruskal's Algorithm

Combining (Union of) Trees

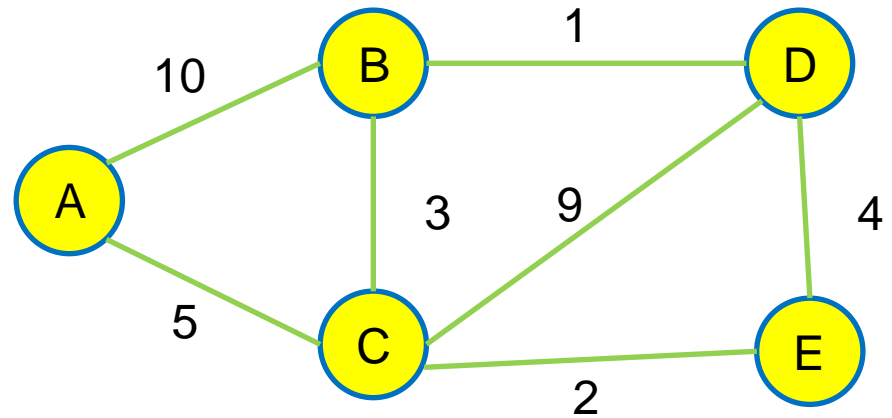
- Imagine every vertex is a tree
 - Only 1 node #ForeverAlone
 - Trees are connected by edges
 - Adding edge $\langle u, v, w \rangle$ combine the trees of vertex u and vertex v
 - Only add if vertex u and vertex v are not in the same tree. Why? **NO CYCLE**
- So how do we do it?
 - Take add edges
 - Sort it
 - Then go through the edges one by one
 - Let us visualize it...



Kruskal's Algorithm

Combining (Union of) Trees

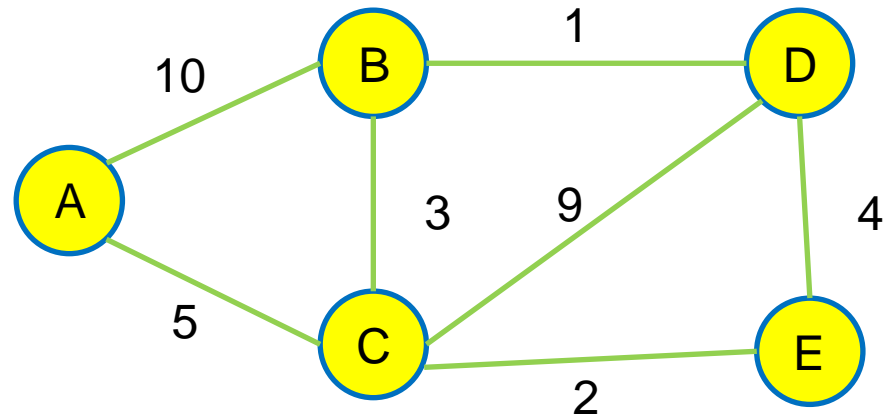
- Look at the graph



Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges

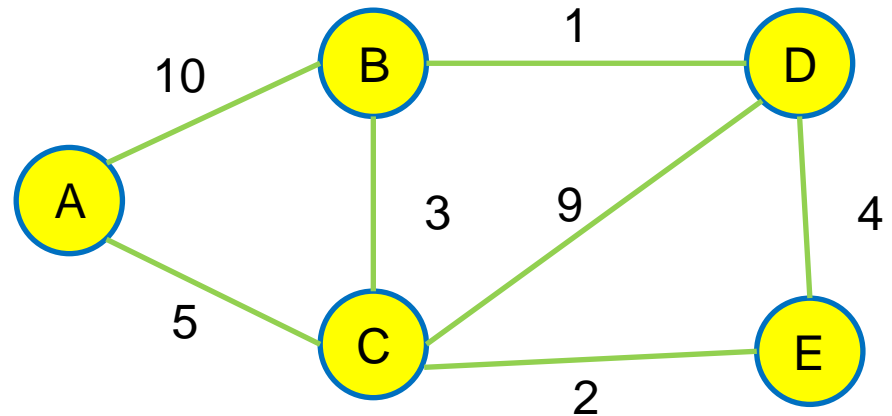


AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges
 - Sort it



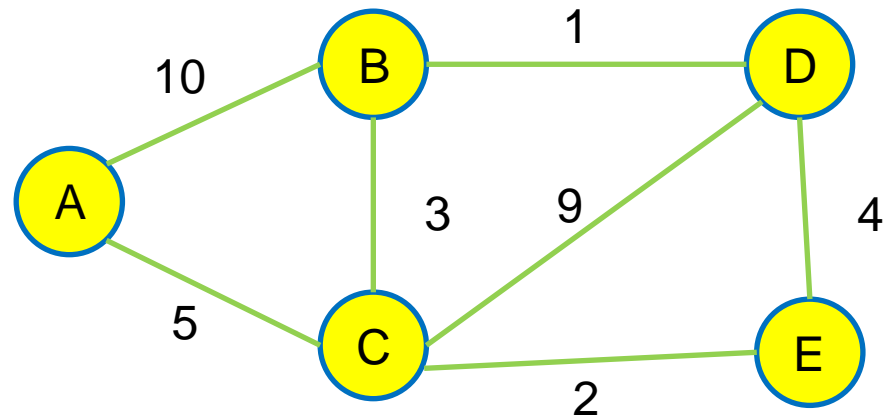
AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10

Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges
 - Sort it
 - Go through the edges one by one



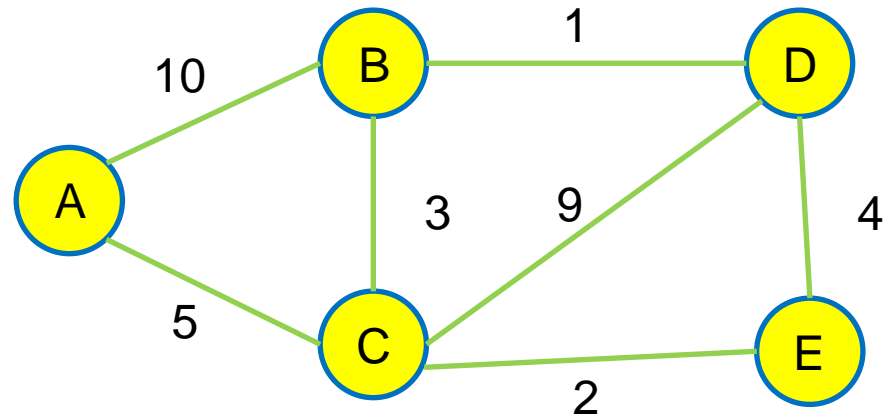
AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10

Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges
 - Sort it
 - Go through the edges one by one



AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

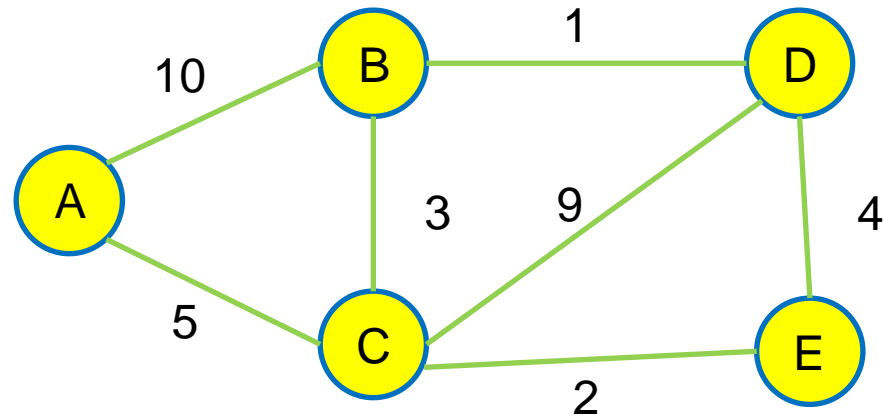
BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10



Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges
 - Sort it
 - Go through the edges one by one
 - Add if u and w not same tree



AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

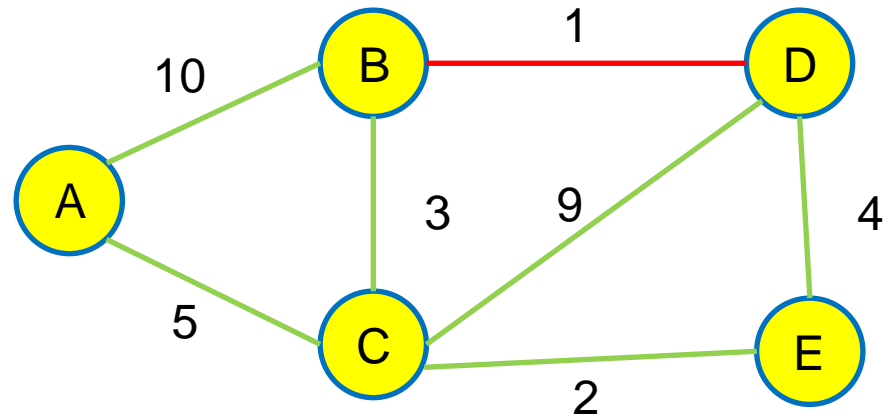
BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10



Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges
 - Sort it
 - Go through the edges one by one
 - Add if u and w not same tree



AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

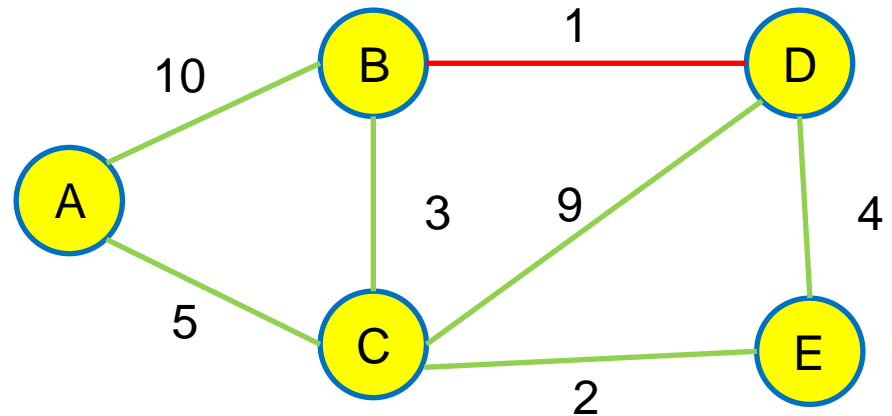
BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10



Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges
 - Sort it
 - Go through the edges one by one
 - Add if u and w not same tree



AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

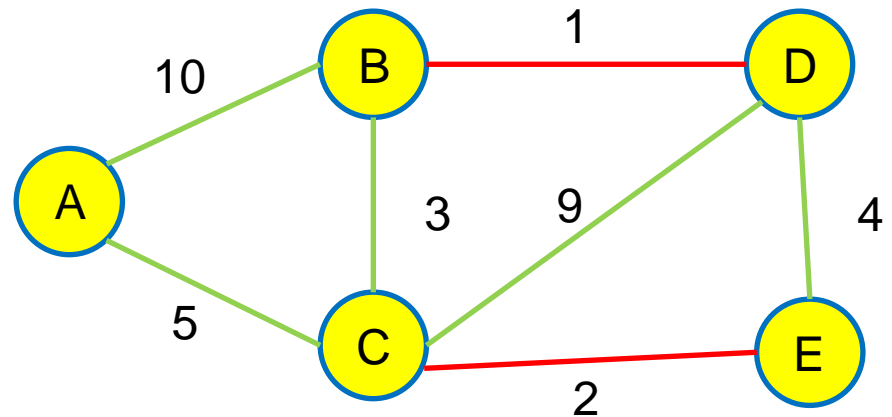
BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10



Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges
 - Sort it
 - Go through the edges one by one
 - Add if u and w not same tree



AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

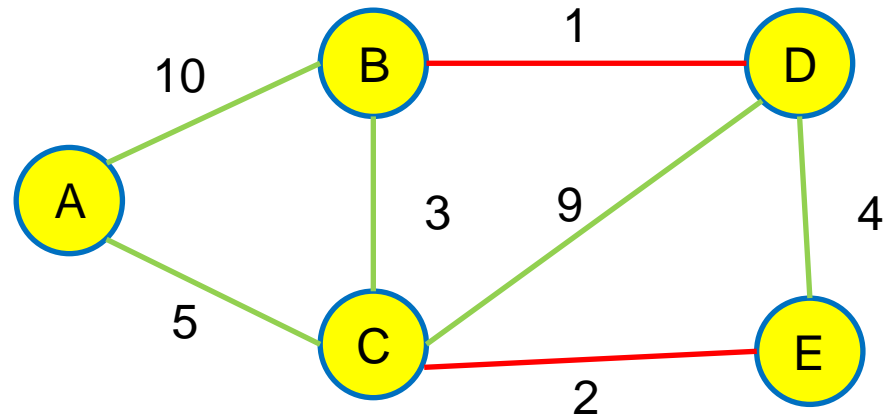
BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10



Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges
 - Sort it
 - Go through the edges one by one
 - Add if u and w not same tree



AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

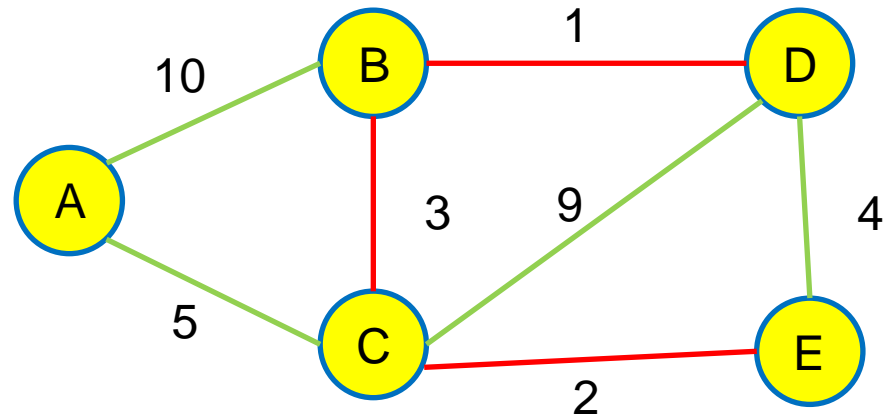
BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10



Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges
 - Sort it
 - Go through the edges one by one
 - Add if u and w not same tree



AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

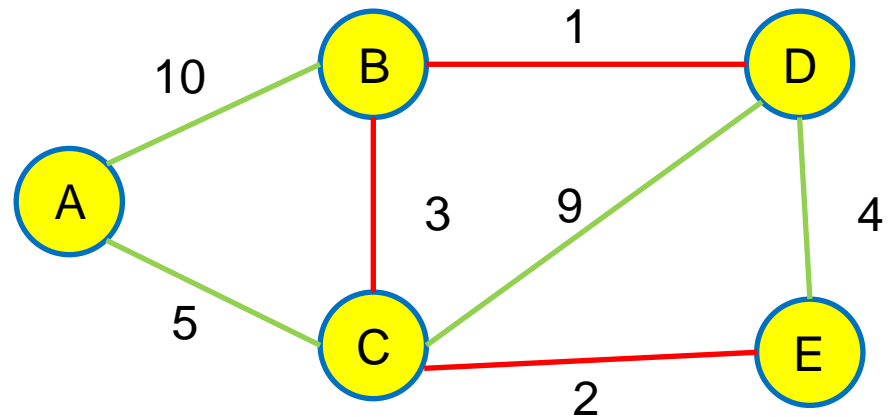
BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10



Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges
 - Sort it
 - Go through the edges one by one
 - Add if u and w not same tree



AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

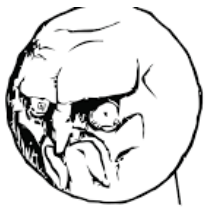
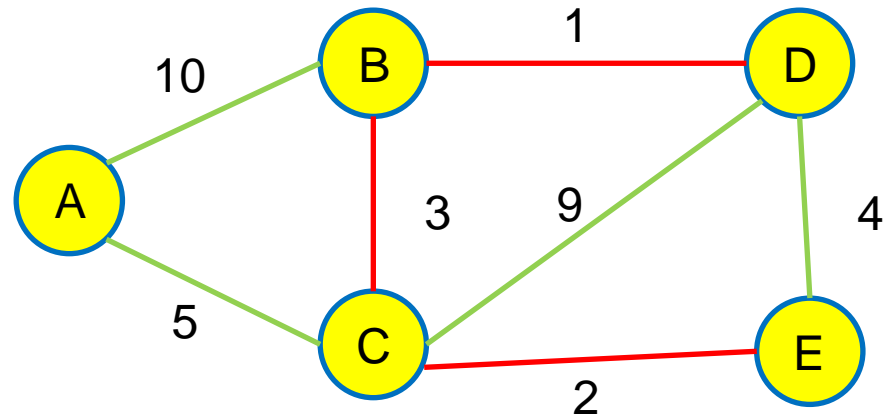
BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10



Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges
 - Sort it
 - Go through the edges one by one
 - Add if u and w not same tree



NO.

AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

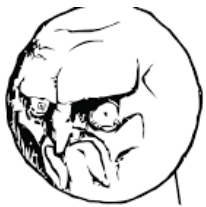
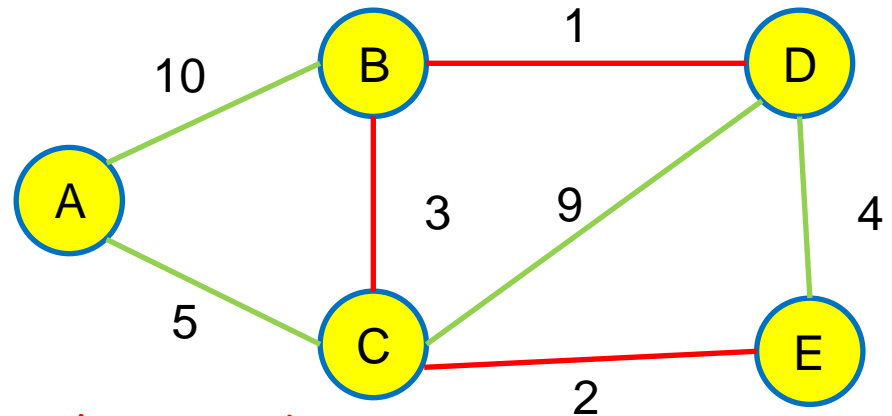
BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10



Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges
 - Sort it
 - Go through the edges one by one
 - Add if u and w not same tree. **Don't want cycle**



NO.

AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

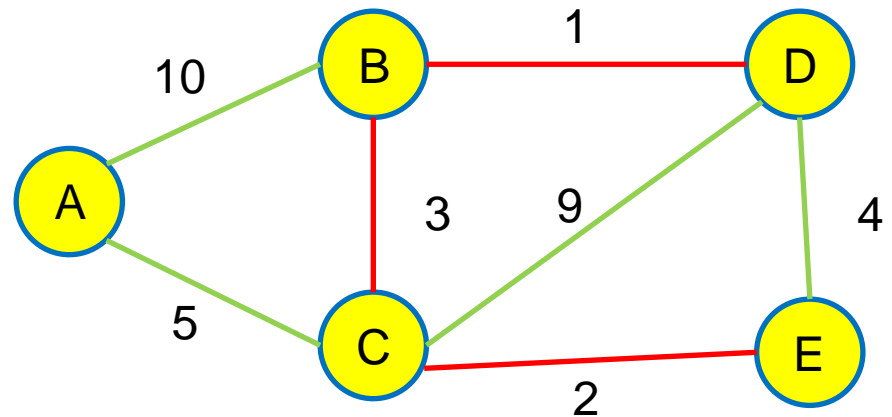
BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10



Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges
 - Sort it
 - Go through the edges one by one
 - Add if u and w not same tree



AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

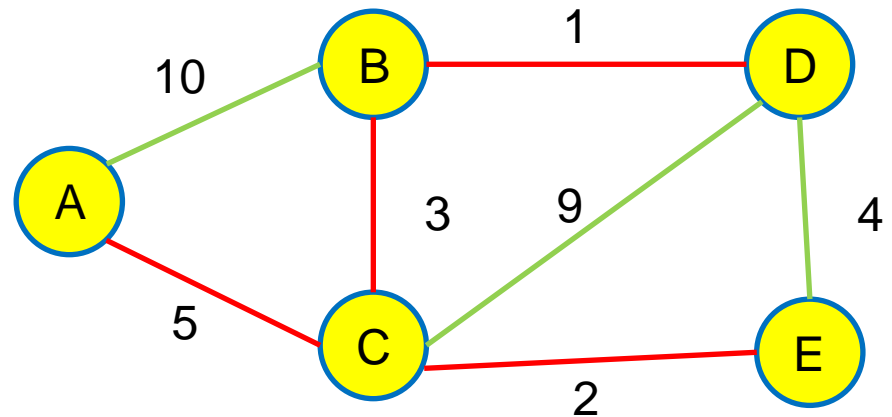
BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10



Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges
 - Sort it
 - Go through the edges one by one
 - Add if u and w not same tree



AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

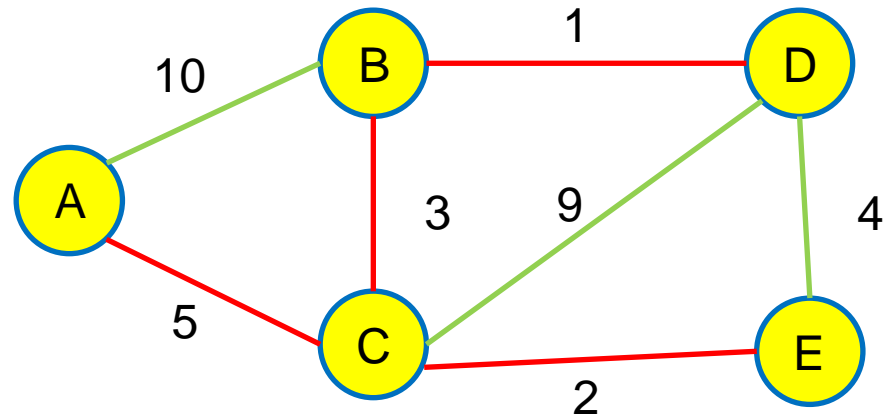
BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10



Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges
 - Sort it
 - Go through the edges one by one
 - Add if u and w not same tree



AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

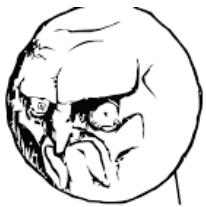
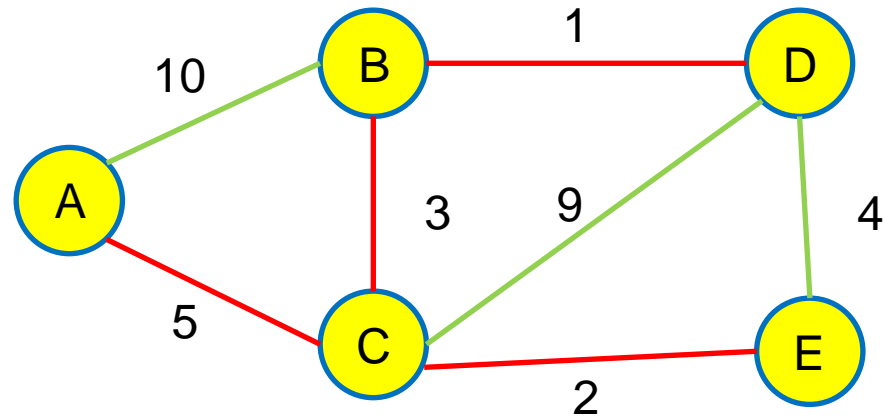
BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10



Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges
 - Sort it
 - Go through the edges one by one
 - Add if u and w not same tree



NO.

AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

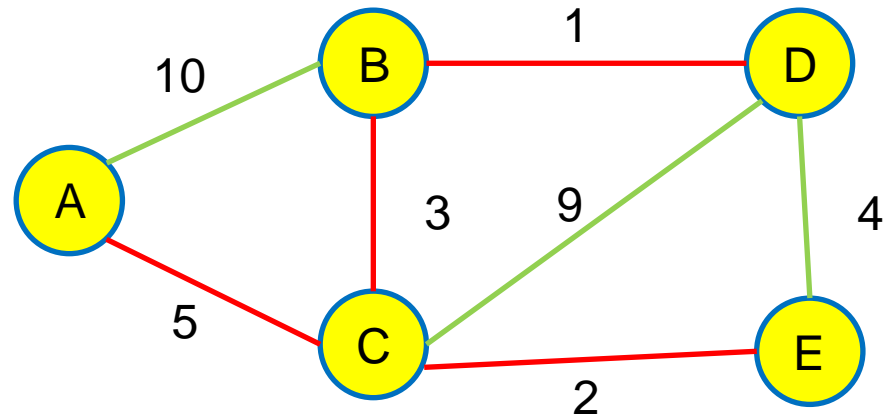
BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10



Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges
 - Sort it
 - Go through the edges one by one
 - Add if u and w not same tree



AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

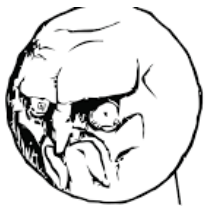
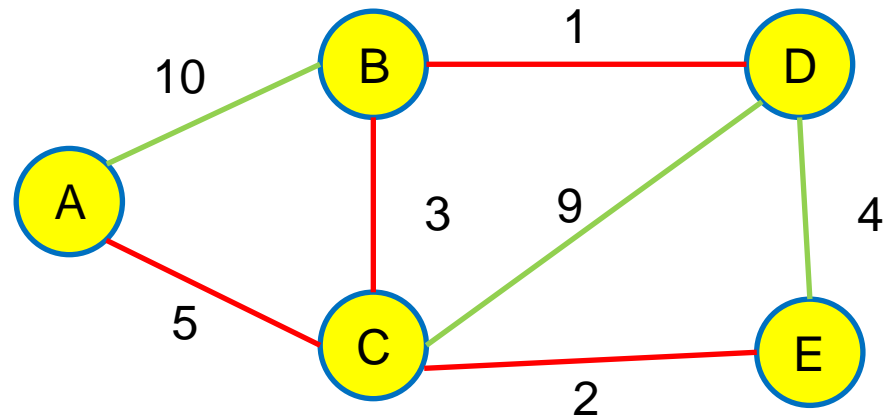
BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10



Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges
 - Sort it
 - Go through the edges one by one
 - Add if u and w not same tree



NO.

AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

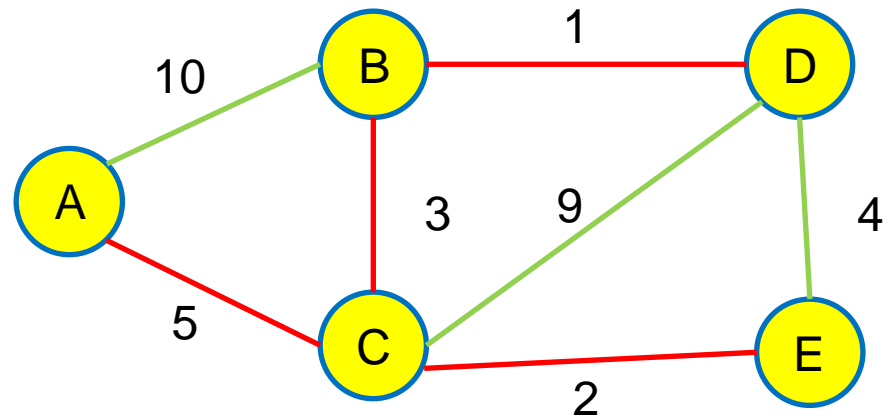
BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10



Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges
 - Sort it
 - Go through the edges one by one
 - Add if u and w not same tree
 - And we are done!



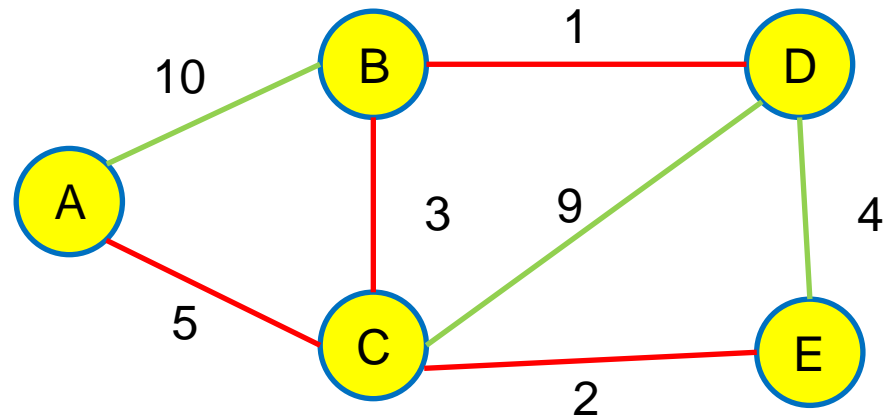
BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10



Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges
 - Sort it
 - Go through the edges one by one
 - Add if u and w not same tree
 - And we are done!
 - But how do we implement it?



BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10



Kruskal's Algorithm

Combining (Union of) Trees

- Look at the graph
 - Take all the edges
 - Sort it
 - Go through the edges one by one
 - Add if u and w not same tree
 - And we are done!
 - But how do we implement it?

Kruskal's Algorithm

Combining (Union of) Trees

- But how do we implement it?

Kruskal's Algorithm

Combining (Union of) Trees

- But how do we implement it?
 - Take the list of edges and sort

Kruskal's Algorithm

Combining (Union of) Trees

- But how do we implement it?
 - Take the list of edges and sort.
 - Easy... just use quicksort

Kruskal's Algorithm

Combining (Union of) Trees

- But how do we implement it?
 - Take the list of edges and sort.
 - Easy... just use quicksort
 - Why not Counting or Radix?

Kruskal's Algorithm

Combining (Union of) Trees

- But how do we implement it?
 - Take the list of edges and sort.
 - Easy... just use quicksort
 - Check if vertex u and vertex v in $\langle u, v, w \rangle$ is in the same tree

Kruskal's Algorithm

Combining (Union of) Trees

- But how do we implement it?
 - Take the list of edges and sort.
 - Easy... just use quicksort
 - Check if vertex u and vertex v in $\langle u, v, w \rangle$ is in the same tree
 - HOW?

Kruskal's Algorithm

Combining (Union of) Trees

- But how do we implement it?
 - Take the list of edges and sort.
 - Easy... just use quicksort
 - Check if vertex u and vertex v in $\langle u, v, w \rangle$ is in the same tree
 - HOW?
 - We use set! Any set data structure

Kruskal's Algorithm

Combining (Union of) Trees

- But how do we implement it?
 - Take the list of edges and sort.
 - Easy... just use quicksort
 - Check if vertex u and vertex v in $\langle u, v, w \rangle$ is in the same tree
 - HOW?
 - We use set! Any set data structure
 - Built-in Python Set (which is based on Dictionary/ Hashtable)

Kruskal's Algorithm

Combining (Union of) Trees

- But how do we implement it?
 - Take the list of edges and sort.
 - Easy... just use quicksort
 - Check if vertex u and vertex v in $\langle u, v, w \rangle$ is in the same tree
 - HOW?
 - We use set! Any set data structure
 - Built-in Python Set (which is based on Dictionary/ Hashtable)
 - Disjoint-Set (which you learn in FIT3155)

Kruskal's Algorithm

Combining (Union of) Trees

- But how do we implement it?
 - Take the list of edges and sort.
 - Easy... just use quicksort
 - Check if vertex u and vertex v in $\langle u, v, w \rangle$ is in the same tree (**Find**)
 - HOW?
 - We use set! Any set data structure
 - Built-in Python Set (which is based on Dictionary/ Hashtable)
 - Disjoint-Set (which you learn in FIT3155)
 - If not the same set, you joint them with the edge (**Union**)

Kruskal's Algorithm

Combining (Union of) Trees

- But how do we implement it?
 - Take the list of edges and sort.
 - Easy... just use quicksort
 - Check if vertex u and vertex v in $\langle u, v, w \rangle$ is in the same tree (**Find**)
 - HOW?
 - We use set! Any set data structure
 - Built-in Python Set (which is based on Dictionary/ Hashtable)
 - Disjoint-Set (which you learn in FIT3155)
 - If not the same set, you joint them with the edge (**Union**)
 - Thus, known as Union-Find

Kruskal's Algorithm

Combining (Union of) Trees

- But how do we implement it?
 - Take the list of edges and sort.
 - Easy... just use quicksort
 - Check if vertex u and vertex v in $\langle u, v, w \rangle$ is in the same tree (**Find**)
 - We use set! Any set data structure
 - Built-in Python Set (which is based on Dictionary/ Hashtable)
 - Disjoint-Set (which you learn in FIT3155)
 - If not the same set, you joint them with the edge (**Union**)
 - Thus, known as Union-Find
 - Complexity?

Kruskal's Algorithm

Combining (Union of) Trees

- But how do we implement it?
 - Take the list of edges and sort.
 - Easy... just use quicksort
 - This is $O(E \log E)$
 - Check if vertex u and vertex v in $\langle u, v, w \rangle$ is in the same tree (**Find**)
 - We use set! Any set data structure
 - Built-in Python Set (which is based on Dictionary/ Hashtable)
 - Disjoint-Set (which you learn in FIT3155)
 - If not the same set, you joint them with the edge (**Union**)
 - Thus, known as Union-Find
 - Complexity?

Kruskal's Algorithm

Combining (Union of) Trees

- But how do we implement it?
 - Take the list of edges and sort.
 - Easy... just use quicksort
 - This is $O(E \log E)$
 - Check if vertex u and vertex v in $\langle u, v, w \rangle$ is in the same tree (**Find**)
 - We use set! Any set data structure
 - Built-in Python Set (which is based on Dictionary/ Hashtable)
 - Disjoint-Set (which you learn in FIT3155)
 - This is $O(1)$
 - If not the same set, you joint them with the edge (**Union**)
 - This is $O(V)$ for now
 - Thus, known as Union-Find
 - Complexity?

Kruskal's Algorithm

Combining (Union of) Trees

- But how do we implement it?
 - Take the list of edges and sort.
 - Easy... just use quicksort
 - This is $O(E \log E)$
 - Check if vertex u and vertex v in $\langle u, v, w \rangle$ is in the same tree (**Find**)
 - We use set! Any set data structure
 - Built-in Python Set (which is based on Dictionary/ Hashtable)
 - Disjoint-Set (which you learn in FIT3155)
 - This is $O(1)$
 - If not the same set, you joint them with the edge (**Union**)
 - This is $O(V)$ for now
 - Thus, known as Union-Find
 - Complexity?

For each edge

Kruskal's Algorithm

Combining (Union of) Trees

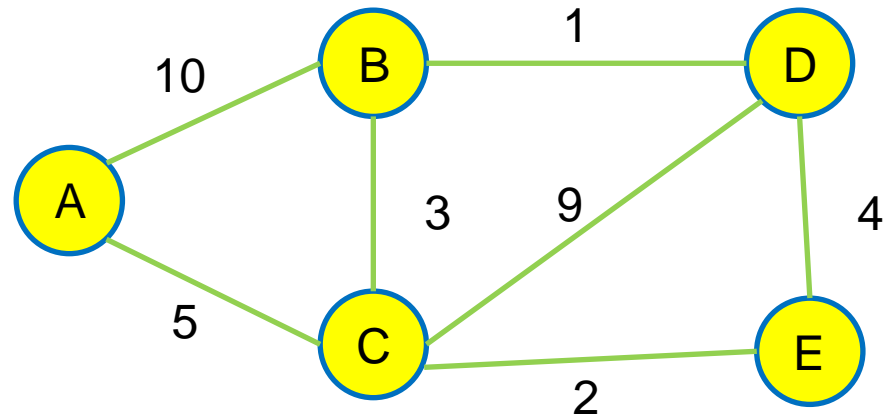
- But how do we implement it?
 - Take the list of edges and sort.
 - Easy... just use quicksort
 - This is $O(E \log E)$
 - Check if vertex u and vertex v in $\langle u, v, w \rangle$ is in the same tree (**Find**)
 - We use set! Any set data structure
 - Built-in Python Set (which is based on Dictionary/ Hashtable)
 - Disjoint-Set (which you learn in FIT3155)
 - This is $O(1)$
 - If not the same set, you joint them with the edge (**Union**)
 - This is $O(V)$ for now
 - Thus, known as Union-Find
 - Complexity? $O(E \log E + E(1+V)) = O(EV)$

For each edge

Kruskal's Algorithm

Combining (Union of) Trees

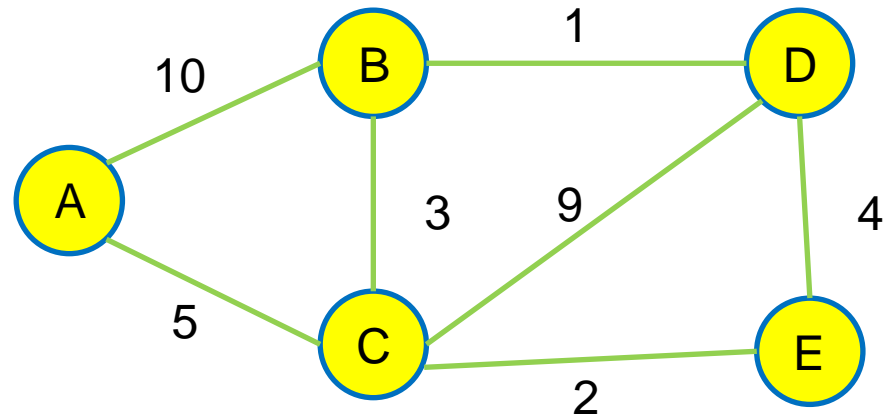
- Union-Find with sets



Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets

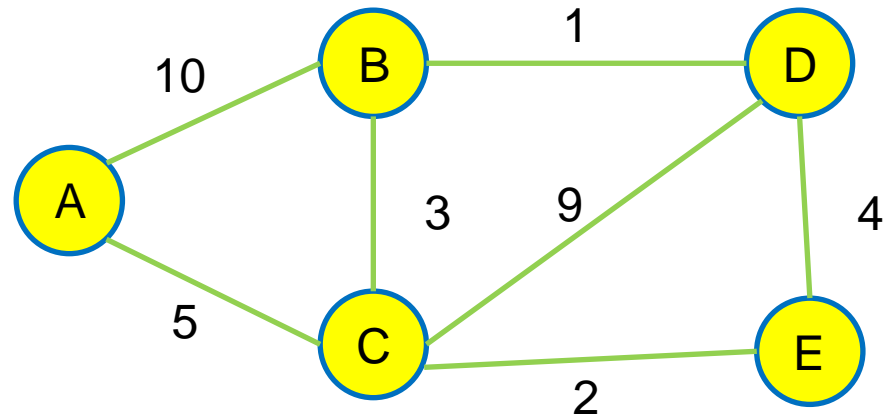


A	B	C	D	E
1	2	3	4	5

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets



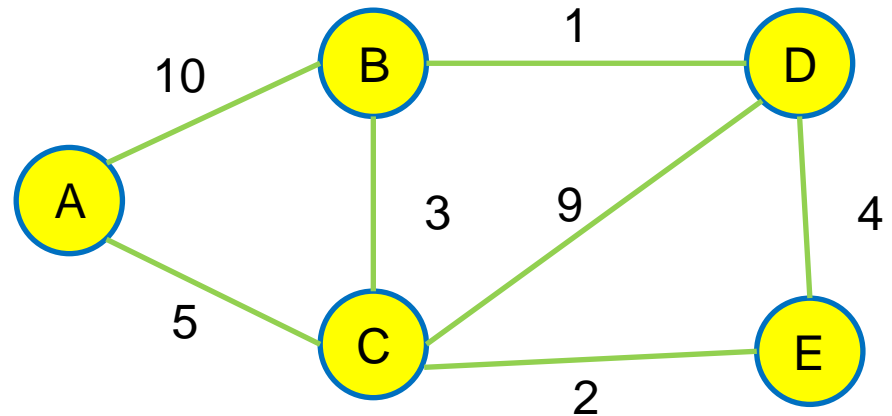
1	2	3	4	5
A	B	C	D	E

A	B	C	D	E
1	2	3	4	5

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets



1	2	3	4	5
A	B	C	D	E

Set Array

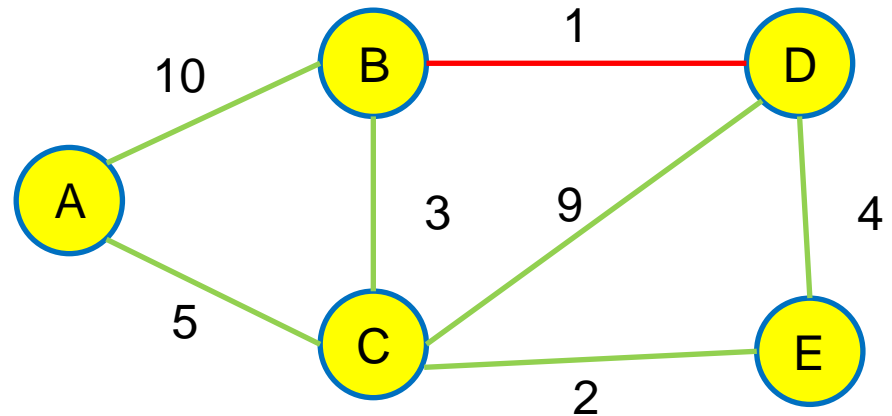
A	B	C	D	E
1	2	3	4	5

Map Array

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets



1	2	3	4	5
A	B	C	D	E

Set Array

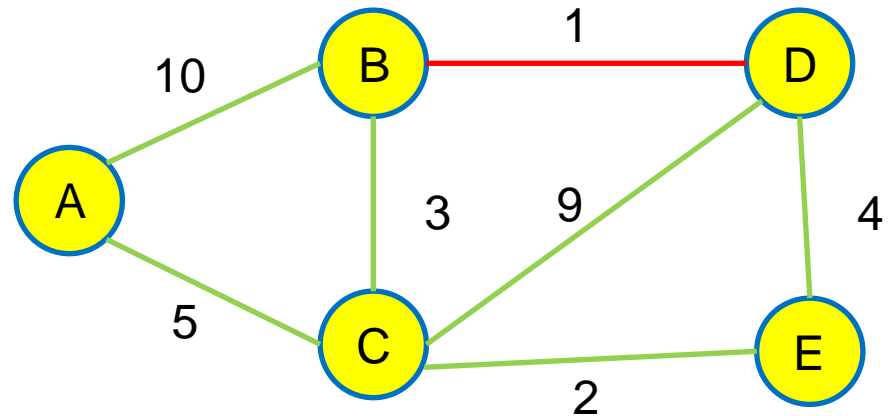
A	B	C	D	E
1	2	3	4	5

Map Array

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets



1	2	3	4	5
A	B	C	D	E

Set Array

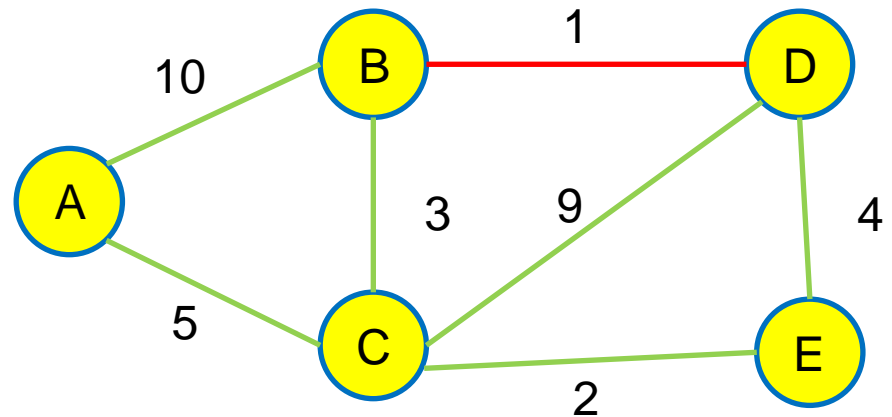
A	B	C	D	E
1	2	3	4	5

Map Array

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets



1	2	3	4	5
A	B,D	C		E

Set Array

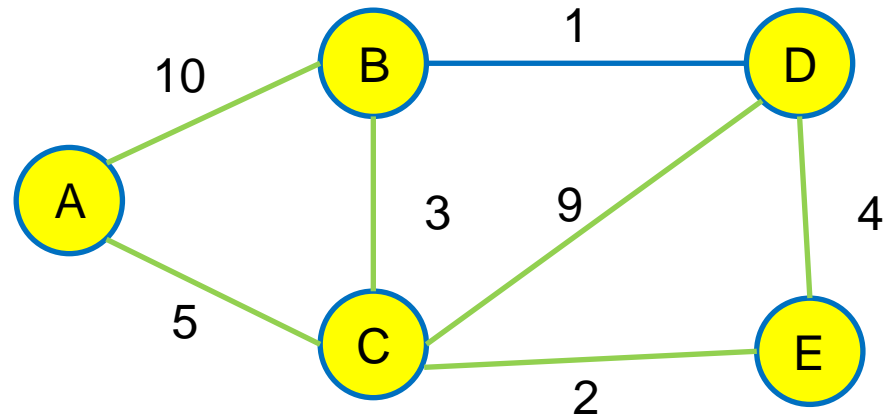
A	B	C	D	E
1	2	3	2	5

Map Array

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets



1	2	3	4	5
A	B,D	C		E

Set Array

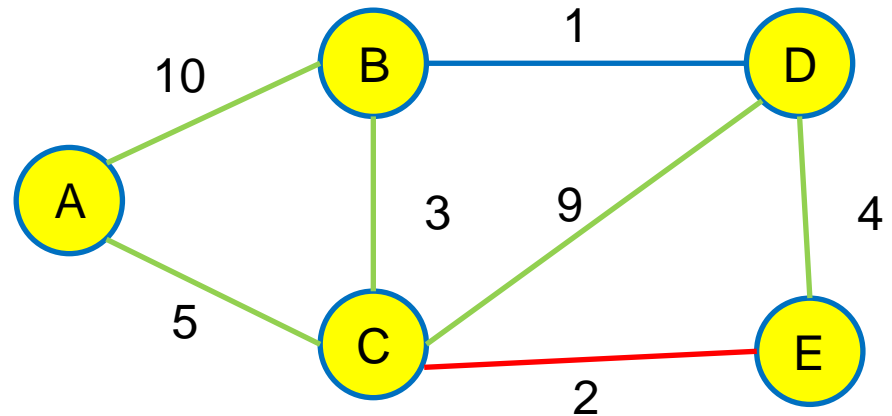
A	B	C	D	E
1	2	3	2	5

Map Array

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets



1	2	3	4	5
A	B,D	C		E

Set Array

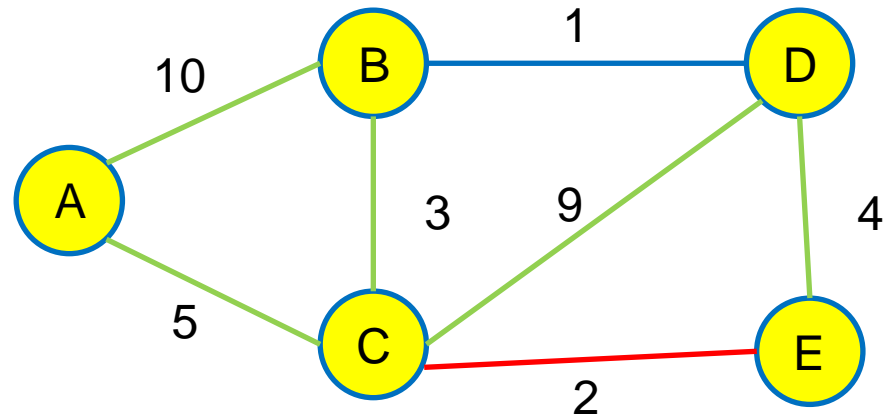
A	B	C	D	E
1	2	3	2	5

Map Array

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets



1	2	3	4	5
A	B,D	C		E

Set Array

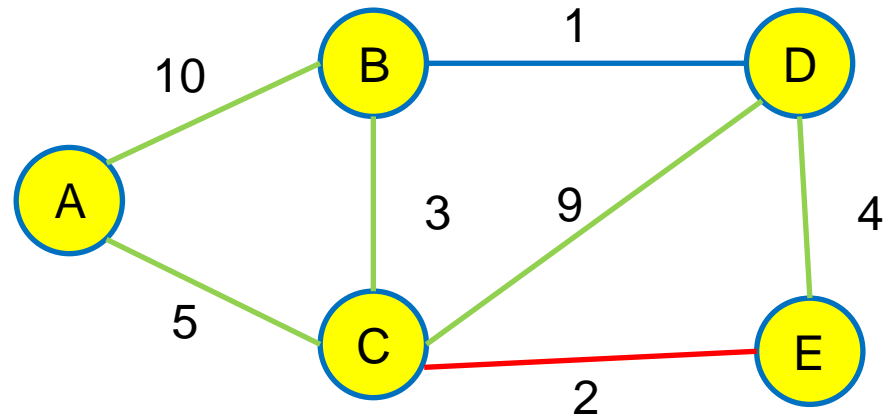
A	B	C	D	E
1	2	3	2	5

Map Array

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets



1	2	3	4	5
A	B,D	C,E		E

Set Array

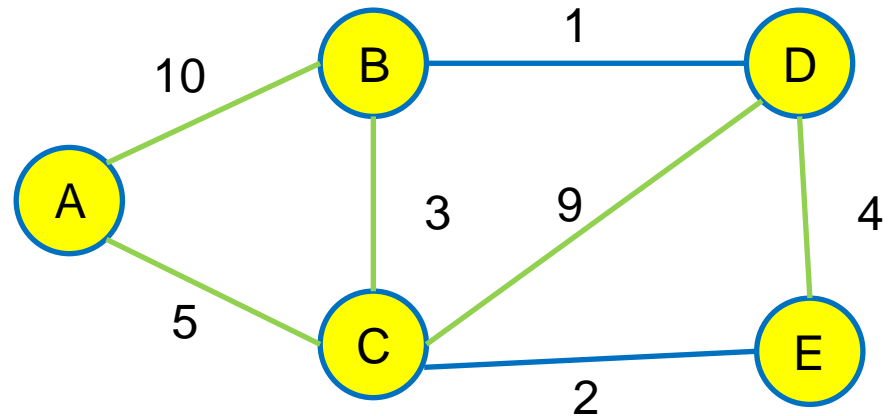
A	B	C	D	E
1	2	3	2	3

Map Array

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets



1	2	3	4	5
A	B,D	C,E		

Set Array

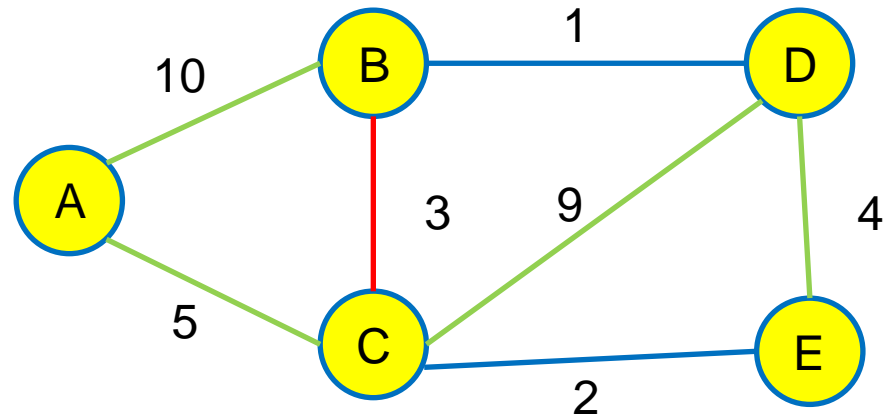
A	B	C	D	E
1	2	3	2	3

Map Array

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets



1	2	3	4	5
A	B,D	C,E		

Set Array

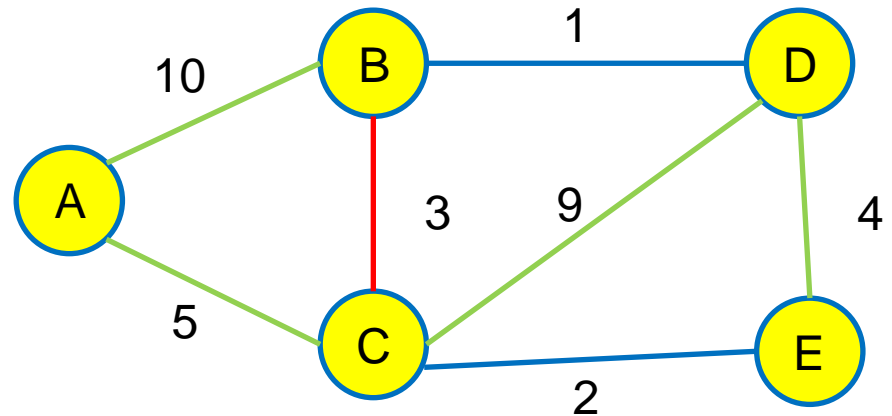
A	B	C	D	E
1	2	3	2	3

Map Array

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets



1	2	3	4	5
A	B,D	C,E		

Set Array

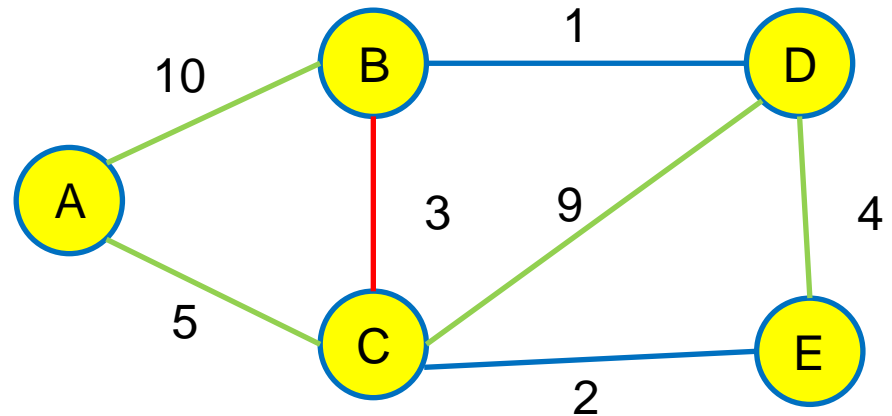
A	B	C	D	E
1	2	3	2	3

Map Array

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets



1	2	3	4	5
A	B,D	C,E		

Set Array

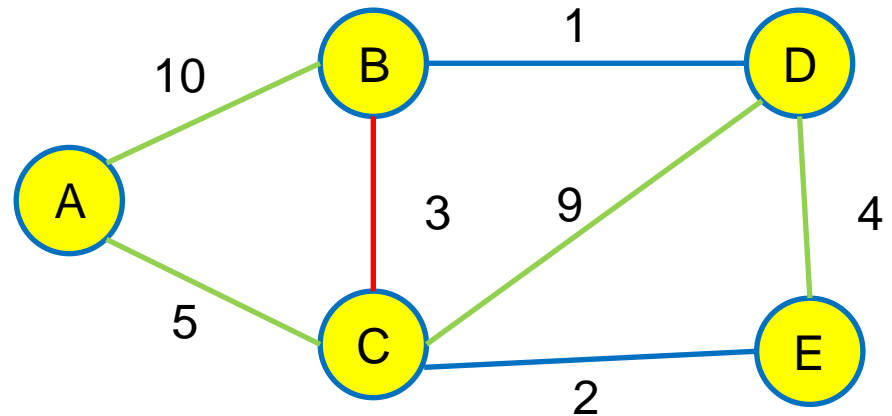
A	B	C	D	E
1	2	3	2	3

Map Array

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets



1	2	3	4	5
A	B,D	<u>C</u> ,E		

Set Array

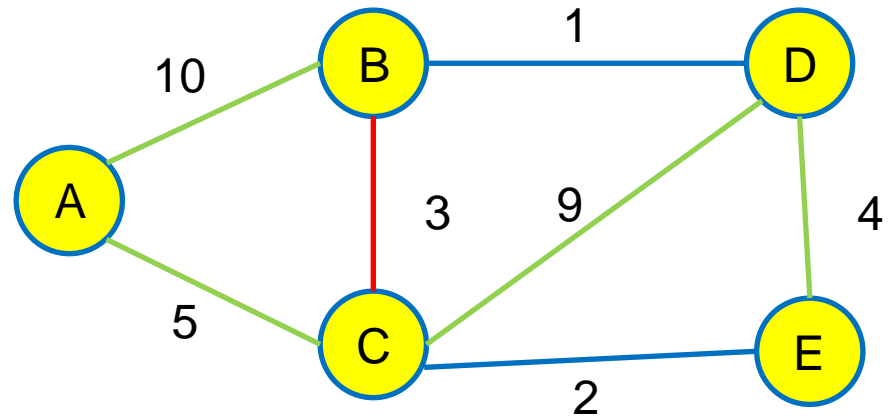
A	B	C	D	E
1	2	3	2	3

Map Array

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets



1	2	3	4	5
A	B,D,C,E			

Set Array

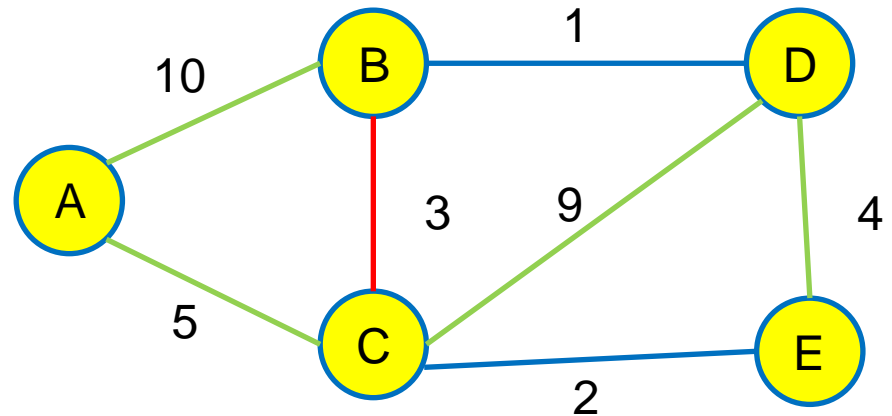
A	B	C	D	E
1	2	3	2	3

Map Array

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets



1	2	3	4	5
A	B,D,C,E			

Set Array

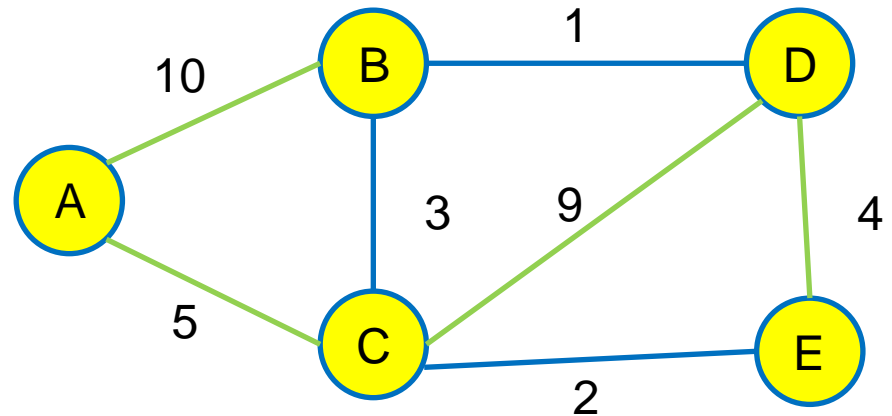
A	B	C	D	E
1	2	2	2	2

Map Array

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets



1	2	3	4	5
A	B,D,C,E			

Set Array

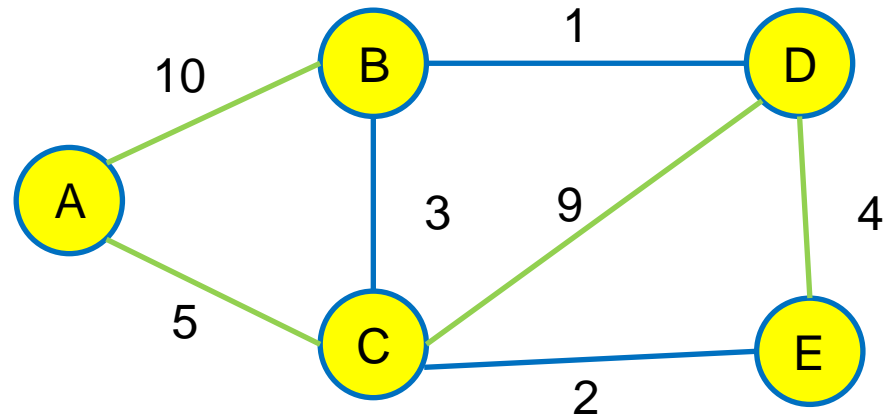
A	B	C	D	E
1	2	2	2	2

Map Array

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets
... and so on
you get the idea...



1	2	3	4	5
A	B,D,C,E			

Set Array

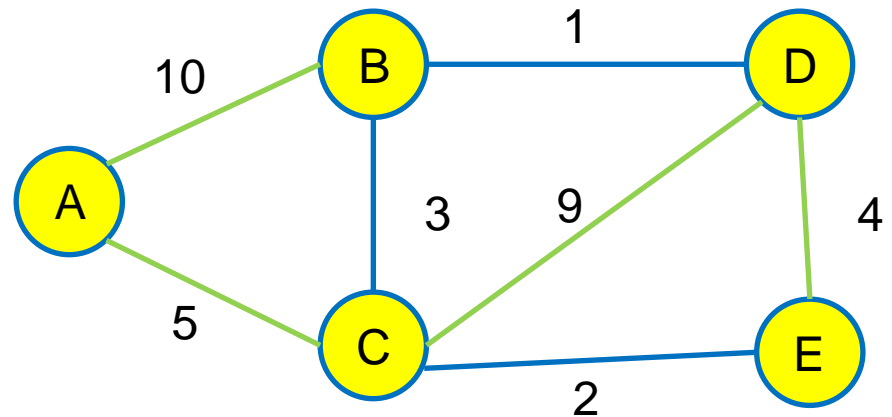
A	B	C	D	E
1	2	2	2	2

Map Array

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets
 - Check the set for vertex
 - Merge vertex set
 - Smaller set -> bigger set
 - Update the map array...



1	2	3	4	5
A	B,D,C,E			

Set Array

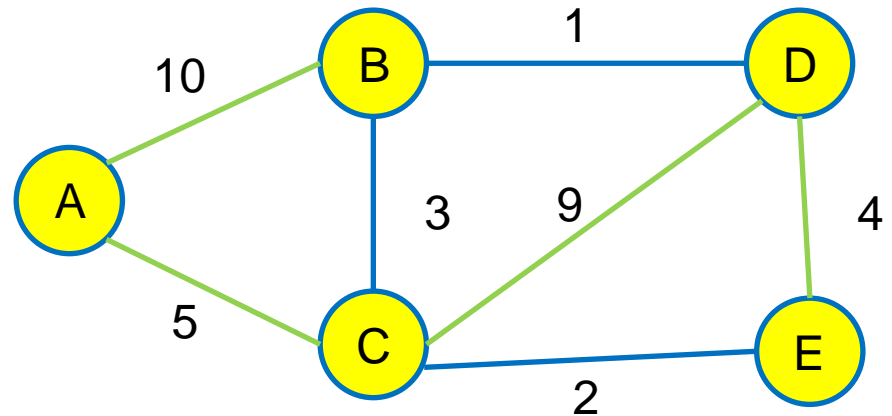
A	B	C	D	E
1	2	2	2	2

Map Array

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets
 - Check the set for vertex
 - Merge vertex set
 - Smaller set -> bigger set
 - Update the map array...
 - Repeat...



1	2	3	4	5
A	B,D,C,E			

Set Array

A	B	C	D	E
1	2	2	2	2

Map Array

Kruskal's Algorithm

Combining (Union of) Trees

- But how do we implement it?
 - Take the list of edges and sort.
 - Easy... just use quicksort
 - This is $O(E \log E)$
 - Check if vertex u and vertex v in $\langle u, v, w \rangle$ is in the same tree (**Find**)
 - We use set! Any set data structure
 - Built-in Python Set (which is based on Dictionary/ Hashtable)
 - Disjoint-Set (which you learn in FIT3155)
 - This is $O(1)$
 - If not the same set, you joint them with the edge (**Union**)
 - This is $O(V)$ for now
 - Thus, known as Union-Find
 - Complexity? $O(EV)$ but this is $O(E \log V)$ amortized

For each edge

Kruskal's Algorithm

Combining (Union of) Trees

- But how do we implement it?
 - Take the list of edges and sort.
 - Easy... just use quicksort
 - This is $O(E \log E)$
 - Check if vertex u and vertex v in $\langle u, v, w \rangle$ is in the same tree (**Find**)
 - We use set! Any set data structure
 - Built-in Python Set (which is based on Dictionary/ Hashtable)
 - Disjoint-Set (which you learn in FIT3155)
 - This is $O(1)$
 - If not the same set, you joint them with the edge (**Union**)
 - This is $O(V)$ for now
 - Thus, known as Union-Find
 - Complexity? $O(EV)$ but this is $O(E \log V)$ amortized (worst case merge 2 same size)

For each edge

Questions?

- This is a week of content itself for FIT3155
 - Though, probably the shortest and easiest one to learn

- This is a week of content itself for FIT3155
 - Though, probably the shortest and easiest one to learn
- Union by size
- Union by height/ rank

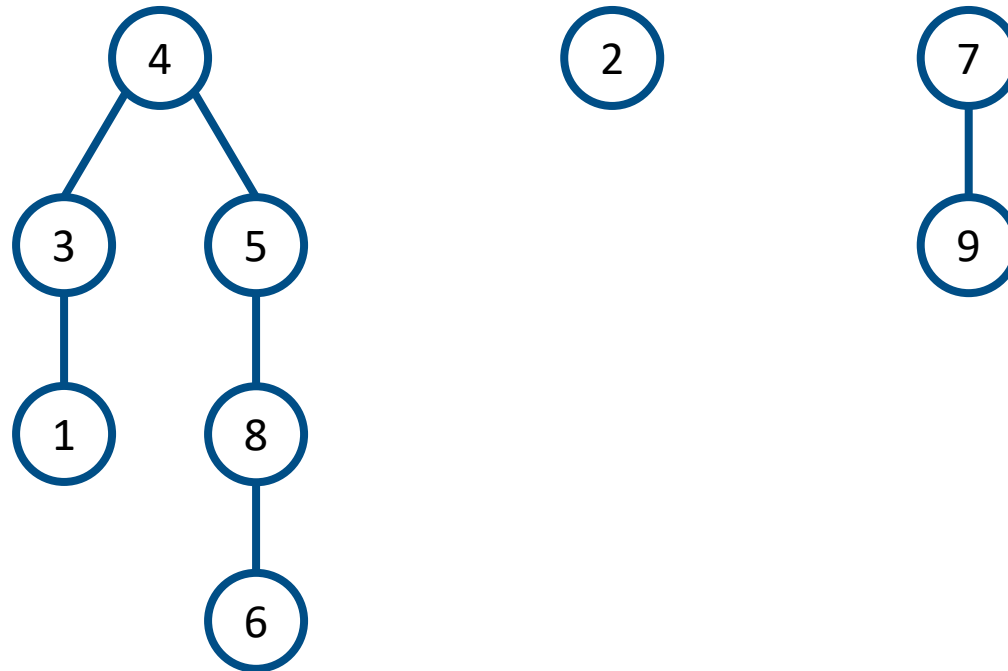
- This is a week of content itself for FIT3155
 - Though, probably the shortest and easiest one to learn
- Union by size
- Union by height/ rank
- Done by using an array, called the parent array

- This is a week of content itself for FIT3155
 - Though, probably the shortest and easiest one to learn
- Union by size
- Union by height/ rank
- Done by using an array, called the parent array
 - Index of the parent, as positive value

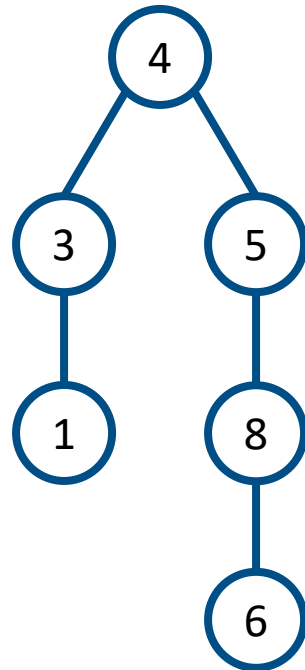
- This is a week of content itself for FIT3155
 - Though, probably the shortest and easiest one to learn
- Union by size
- Union by height/ rank
- Done by using an array, called the parent array
 - Index of the parent, as positive value
 - Size or height, as negative value

- This is a week of content itself for FIT3155
 - Though, probably the shortest and easiest one to learn
- Union by size
- Union by height/ rank
- Done by using an array, called the parent array
 - Index of the parent, as positive value
 - Size or height, as negative value but only at the **root**

Kruskal's Union-Find



Kruskal's Union-Find



	1	2	3	4	5	6	7	8	9
	3	-1	4	-6	4	8	-2	5	7

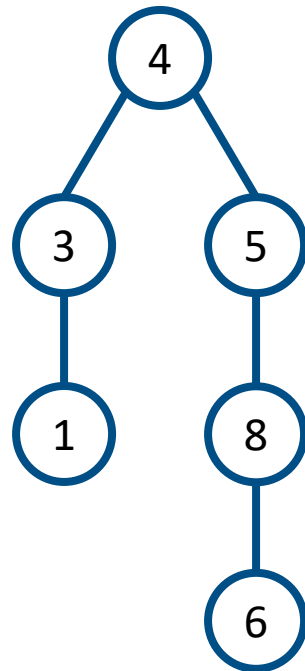
- Why such an implementation?

- Why such an implementation?
 - On find(u)
 - Loop till we reach the root of u (having negative number)

- Why such an implementation?
 - On find(u)
 - Loop till we reach the root of u (having negative number)
 - On find(v)
 - Loop till we reach the root of v (having negative number)

- Why such an implementation?
 - On find(u)
 - Loop till we reach the root of u (having negative number)
 - On find(v)
 - Loop till we reach the root of v (having negative number)
 - Remember: roots always store the size (as a negative number)

Kruskal's Union-Find



	1	2	3	4	5	6	7	8	9
	3	-1	4	-6	4	8	-2	5	7

- Remember: roots always store the size (as a negative number)

- Why such an implementation?
 - On find(u)
 - Loop till we reach the root of u (having negative number)
 - On find(v)
 - Loop till we reach the root of v (having negative number)
 - If both u and v have the same root...

- Why such an implementation?
 - On find(u)
 - Loop till we reach the root of u (having negative number)
 - On find(v)
 - Loop till we reach the root of v (having negative number)
 - If both u and v have the same root...
 - They are in the same team/ set/ tree

Kruskal's Union-Find

- Why such an implementation?
 - On find(u)
 - Loop till we reach the root of u (having negative number)
 - On find(v)
 - Loop till we reach the root of v (having negative number)
 - If both u and v have the same root...
 - They are in the same team/ set/ tree
 - We can't perform union(u, v)



- Why such an implementation?
 - On find(u)
 - Loop till we reach the root of u (having negative number)
 - On find(v)
 - Loop till we reach the root of v (having negative number)
 - If both u and v have the same root...
 - They are in the same team/ set/ tree
 - We can't perform union(u, v)
 - If both u and v have different root...

- Why such an implementation?
 - On $\text{find}(u)$
 - Loop till we reach the root of u (having negative number)
 - On $\text{find}(v)$
 - Loop till we reach the root of v (having negative number)
 - If both u and v have the same root...
 - They are in the same team/ set/ tree
 - We can't perform $\text{union}(u, v)$
 - If both u and v have different root...
 - They are in different team/ set/ tree

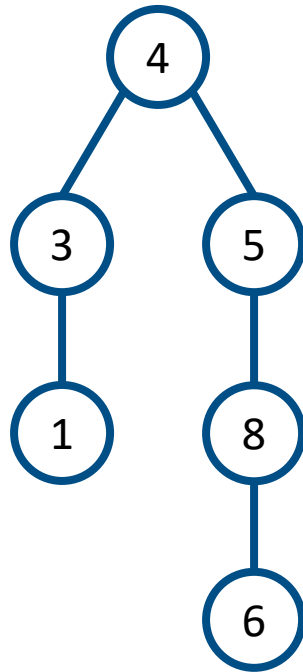


- Why such an implementation?
 - On find(u)
 - Loop till we reach the root of u (having negative number)
 - On find(v)
 - Loop till we reach the root of v (having negative number)
 - If both u and v have the same root...
 - They are in the same team/ set/ tree
 - We can't perform union(u, v)
 - If both u and v have different root...
 - They are in different team/ set/ tree
 - Then we can perform union(u, v)

- Why such an implementation?
 - On find(u)
 - Loop till we reach the root of u (having negative number)
 - On find(v)
 - Loop till we reach the root of v (having negative number)
 - If both u and v have the same root...
 - They are in the same team/ set/ tree
 - We can't perform union(u, v)
 - If both u and v have different root...
 - They are in different team/ set/ tree
 - Then we can perform union(u, v)
 - If tree with u has more items than tree with v ,
root of u becomes parent of root of v
 - ... vice versa

Questions?

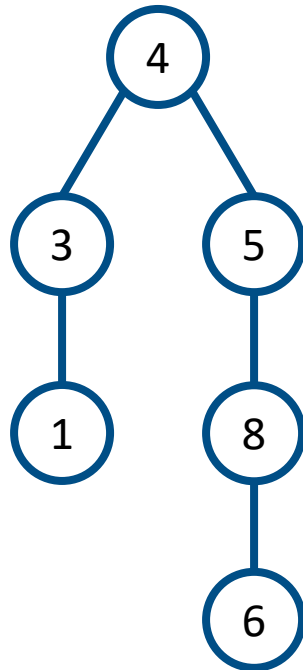
Kruskal's Union-Find



	1	2	3	4	5	6	7	8	9
	3	-1	4	-6	4	8	-2	5	7

– Union(3,8)

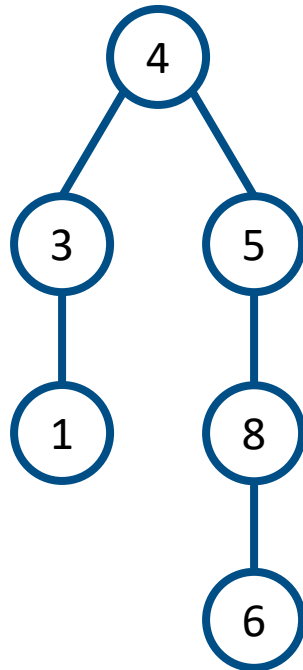
Kruskal's Union-Find



	1	2	3	4	5	6	7	8	9
	3	-1	4	-6	4	8	-2	5	7

- Union(3,8)
 - Find(3)
 - Find(8)

Kruskal's Union-Find

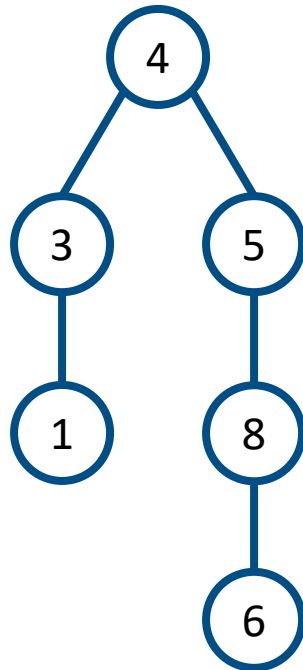


	1	2	3	4	5	6	7	8	9
	3	-1	4	-6	4	8	-2	5	7

– Union(3,8)

▪ Find(3) -> 4

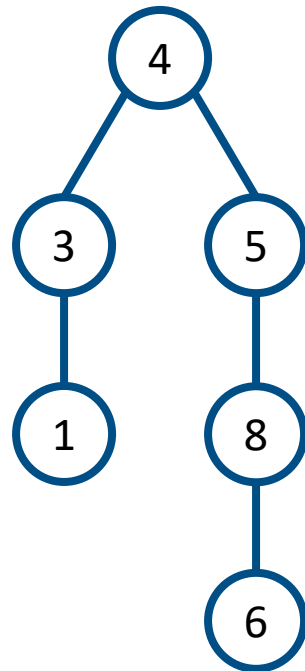
Kruskal's Union-Find



	1	2	3	4	5	6	7	8	9
	3	-1	4	-6	4	8	-2	5	7

- Union(3,8)
 - Find(3) -> 4
 - Find(8) -> 4

Kruskal's Union-Find

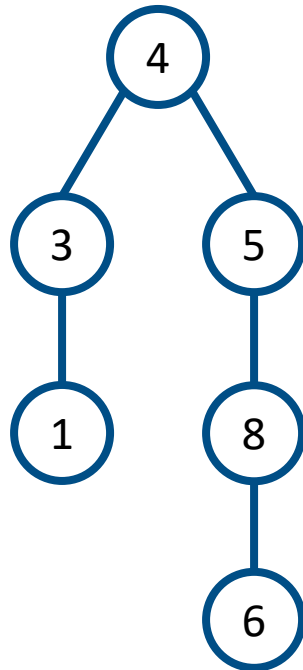


	1	2	3	4	5	6	7	8	9
	3	-1	4	-6	4	8	-2	5	7

- Union(3,8), **can't perform the union**
 - Find(3) -> 4
 - Find(8) -> 4

Questions?

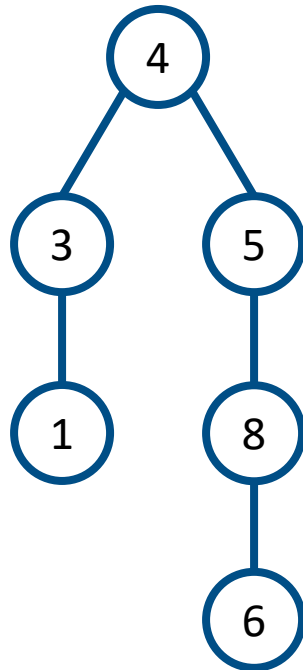
Kruskal's Union-Find



	1	2	3	4	5	6	7	8	9
	3	-1	4	-6	4	8	-2	5	7

– Union(9,8)

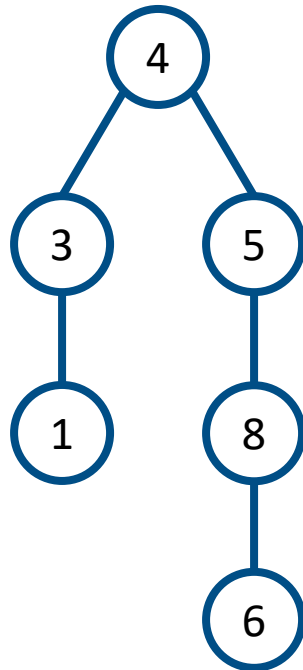
Kruskal's Union-Find



	1	2	3	4	5	6	7	8	9
	3	-1	4	-6	4	8	-2	5	7

- Union(9,8)
 - Find(9)

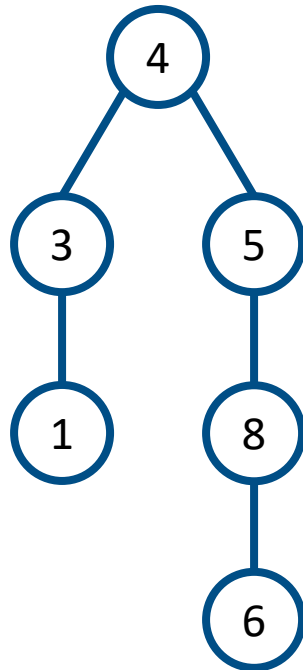
Kruskal's Union-Find



	1	2	3	4	5	6	7	8	9
	3	-1	4	-6	4	8	-2	5	7

- Union(9,8)
 - Find(9) -> 7

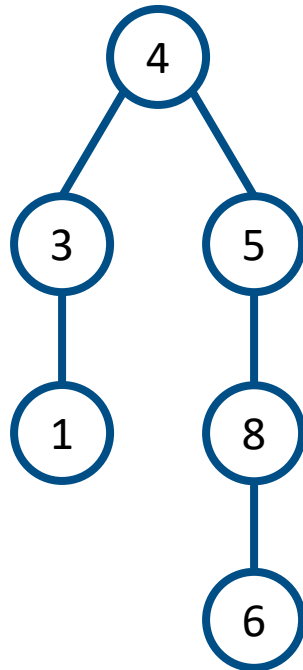
Kruskal's Union-Find



	1	2	3	4	5	6	7	8	9
	3	-1	4	-6	4	8	-2	5	7

- Union(9,8)
 - Find(9) -> 7
 - Find(8)

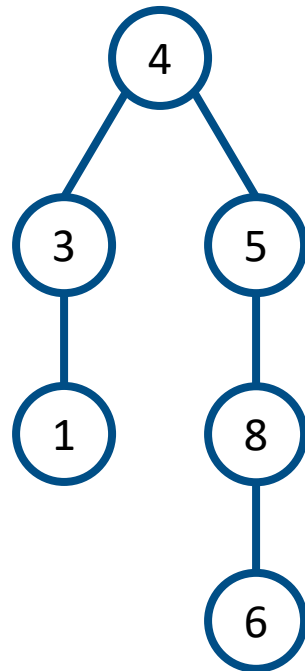
Kruskal's Union-Find



	1	2	3	4	5	6	7	8	9
	3	-1	4	-6	4	8	-2	5	7

- Union(9,8)
 - Find(9) -> 7
 - Find(8) -> 4

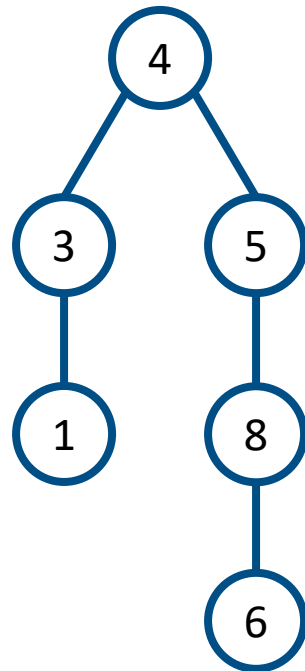
Kruskal's Union-Find



	1	2	3	4	5	6	7	8	9
	3	-1	4	-6	4	8	-2	5	7

- Union(9,8), different tree so we can perform union
 - Find(9) -> 7
 - Find(8) -> 4

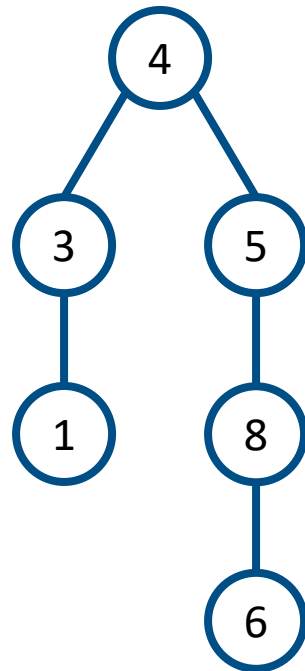
Kruskal's Union-Find



	1	2	3	4	5	6	7	8	9
	3	-1	4	-6	4	8	-2	5	7

- Union(9,8), different tree so we can perform union
 - Find(9) -> 7, size of 2
 - Find(8) -> 4,

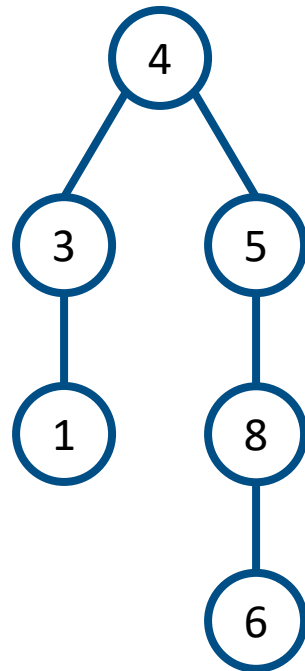
Kruskal's Union-Find



	1	2	3	4	5	6	7	8	9
	3	-1	4	-6	4	8	-2	5	7

- Union(9,8), different tree so we can perform union
 - Find(9) -> 7, size of 2
 - Find(8) -> 4, size of 6

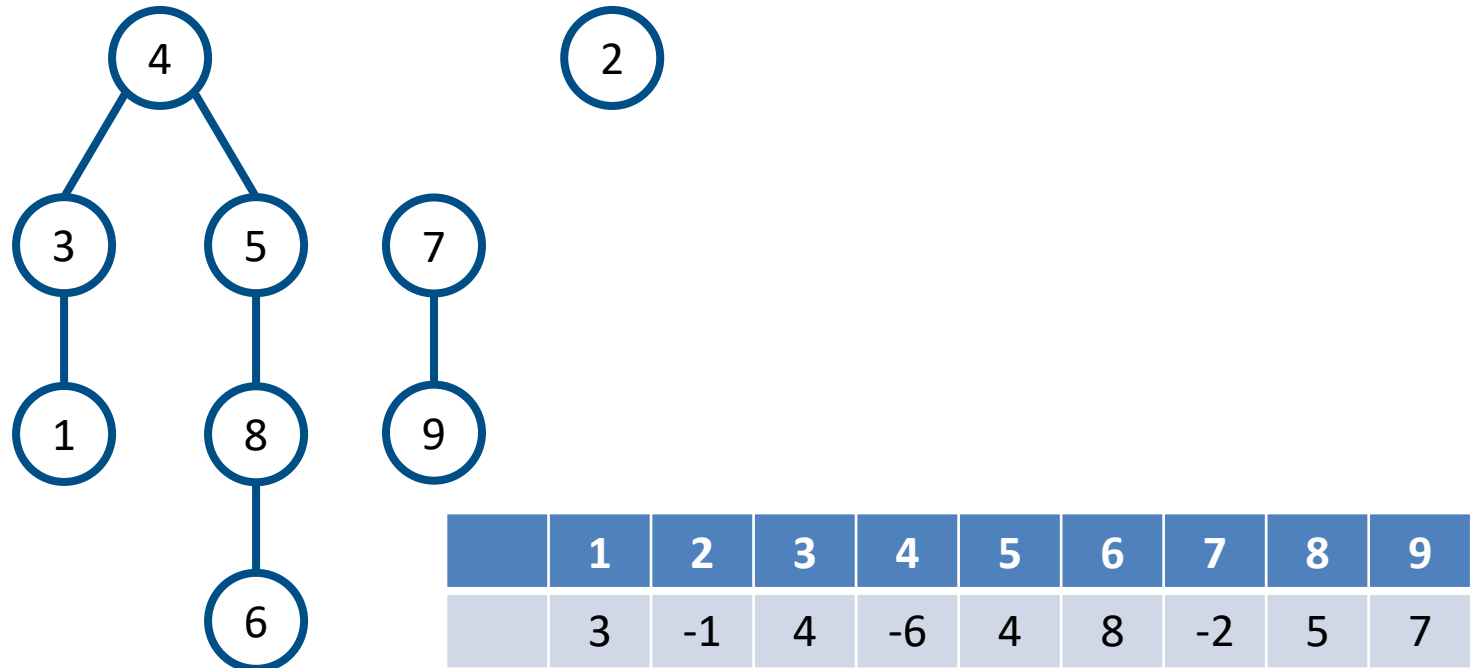
Kruskal's Union-Find



	1	2	3	4	5	6	7	8	9
	3	-1	4	-6	4	8	-2	5	7

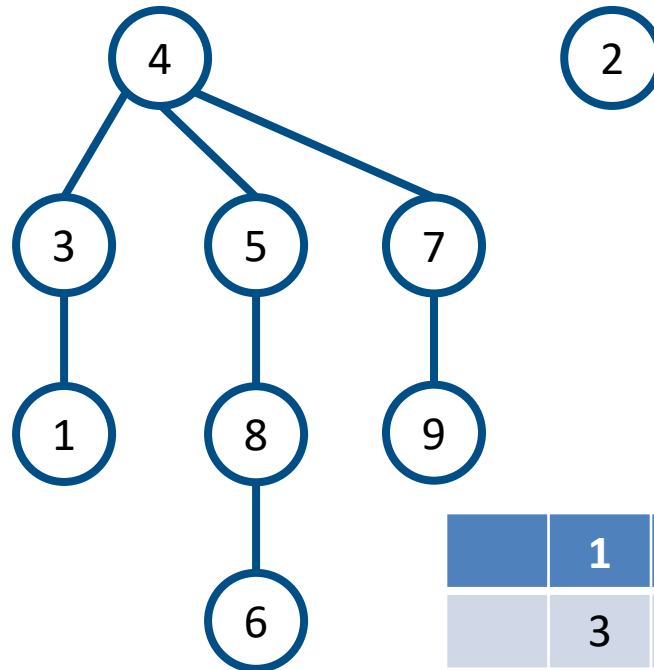
- Union(9,8), different tree so we can perform union
 - Find(9) -> 7, size of 2, smaller tree so merge to bigger tree
 - Find(8) -> 4, size of 6

Kruskal's Union-Find



- Union(9,8), **different tree so we can perform union**
 - Find(9) -> 7, size of 2, **smaller tree so merge to bigger tree**
 - Find(8) -> 4, size of 6

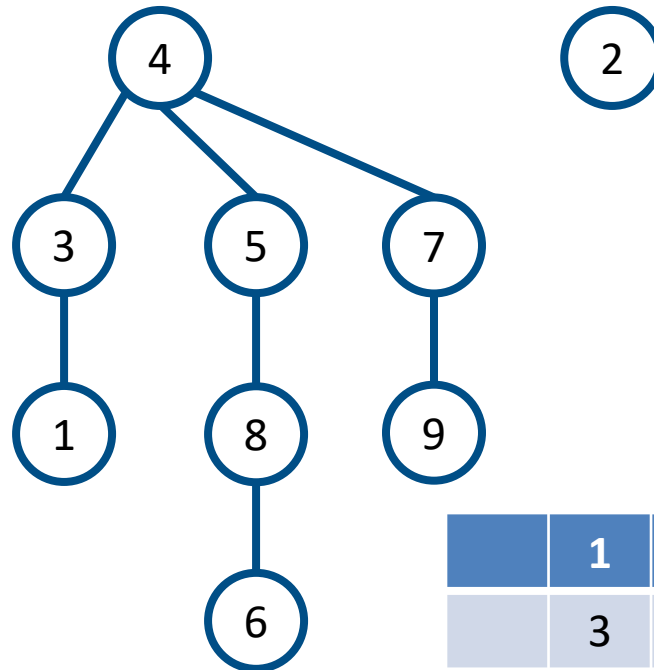
Kruskal's Union-Find



	1	2	3	4	5	6	7	8	9
	3	-1	4	-8	4	8	4	5	7

- Union(9,8), different tree so we can perform union
 - Find(9) -> 7, size of 2, smaller tree so merge to bigger tree
 - Find(8) -> 4, size of 6, size updated to 8

Kruskal's Union-Find



	1	2	3	4	5	6	7	8	9
	3	-1	4	-8	4	8	4	5	7

- Union(9,8), different tree so we can perform union
 - Find(9) -> 7, size of 2, smaller tree so merge to bigger tree
 - Find(8) -> 4, size of 6, size updated to 8

Questions?

- For a graph, can we always find the MST?

- For a graph, can we always find the MST?
- Time to prove it on the whiteboard...
 - Known as proof by contradiction...

Questions?

Prim's and Kruskal's

Does it work?

Prim's and Kruskal's

Does it work?

- For negative edges?

Prim's and Kruskal's

Does it work?

- For negative edges?
 - Prim's work fine cause it will choose the negative one from the tree
 - Kruskal's work fine cause the negative edges is sorted forward

- For negative edges?
 - Prim's work fine cause it will choose the negative one from the tree
 - Kruskal's work fine cause the negative edges is sorted forward

- For negative cycles?

Prim's and Kruskal's

Does it work?

- For negative edges?
 - Prim's work fine cause it will choose the negative one from the tree
 - Kruskal's work fine cause the negative edges is sorted forward
- For negative cycles?
 - Yes we chose the smallest edges without form cycles!

- For negative edges?
 - Prim's work fine cause it will choose the negative one from the tree
 - Kruskal's work fine cause the negative edges is sorted forward
- For negative cycles?
 - Yes we chose the smallest edges without form cycles!
- Can you prove that the greediness is correct?

- For negative edges?
 - Prim's work fine cause it will choose the negative one from the tree
 - Kruskal's work fine cause the negative edges is sorted forward

- For negative cycles?
 - Yes we chose the smallest edges without form cycles!

- Can you prove that the greediness is correct?
 - Yes...
 - I will now work both out on the whiteboard

Prim's and Kruskal's

Does it work?

- For negative edges?
 - Prim's work fine cause it will choose the negative one from the tree
 - Kruskal's work fine cause the negative edges is sorted forward
- For negative cycles?
 - Yes we chose the smallest edges without form cycles!
- Can you prove that the greediness is correct?
 - Yes...
 - I will now work both out on the whiteboard
 - **Invariant: The selected edges will be part of the final MST**

Questions?

Thank You