MONASH University

MONASH
INFORMATION
TECHNOLOGY

FIT2004
Algorithms and Data Structures

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

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

GROUP OF EIGHT AUSTRALIA

Faculty of Information Technology, Monash University

MONASH University

Ready?

Agenda

- Minimum Spanning Tree (MST)

Agenda

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

Monash University

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?

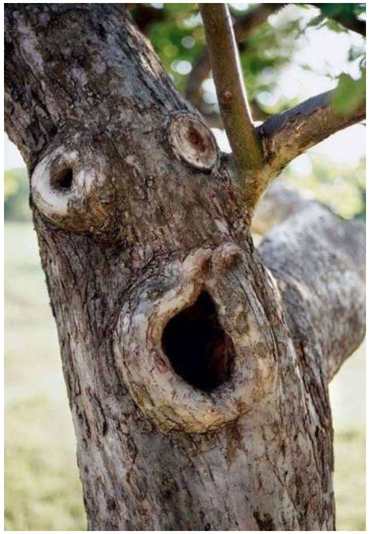
- A tree
- Spanning every vertex
- Minimum total edges

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?

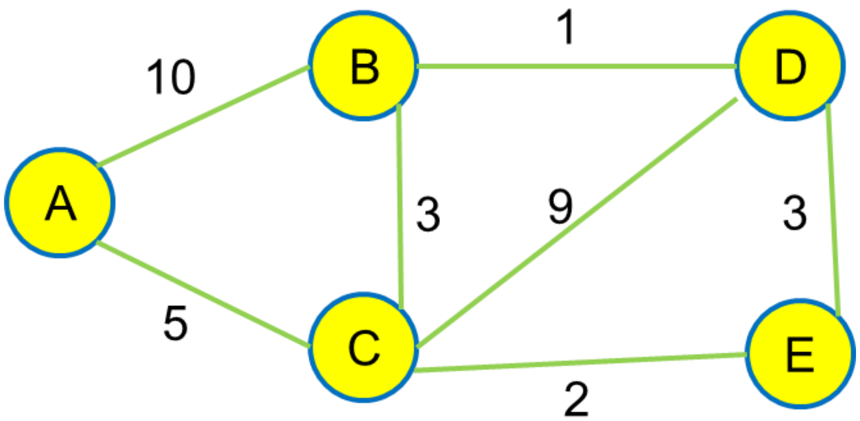
- A tree
  - No cycle
  - Undirected
- Spanning every vertex
  - Minimum number of edges to connect all vertex
  - Maximum number of edges in graph without cycle at most v -1
- Minimum total edges weight

Spanning Tree:

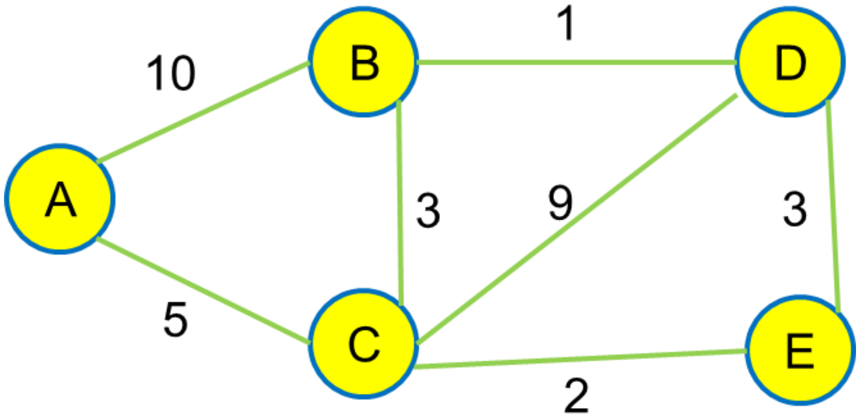
- A spanning tree of a general undirected weighted graph G is a tree that spans G (i.e., a tree that includes every vertex of G) and is a subgraph of G (i.e., every edge in the spanning tree belongs to G).

Minimum Spanning Tree
What is it?

- Let say we have a graph



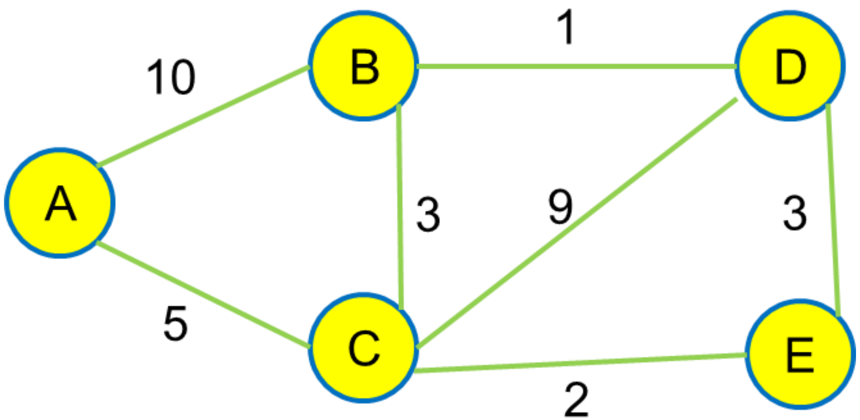
- Let say we have a graph
  - Can you form spanning trees?

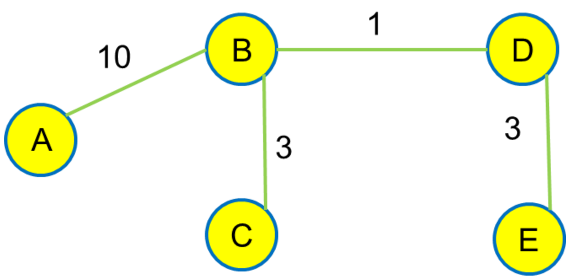


Minimum Spanning Tree

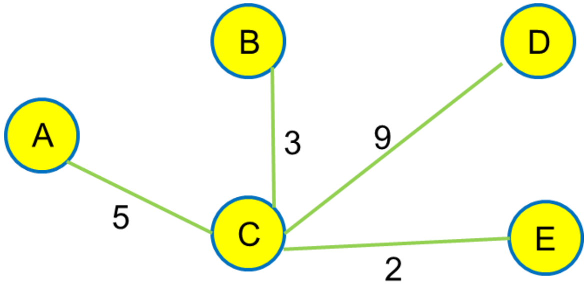
What is it?

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

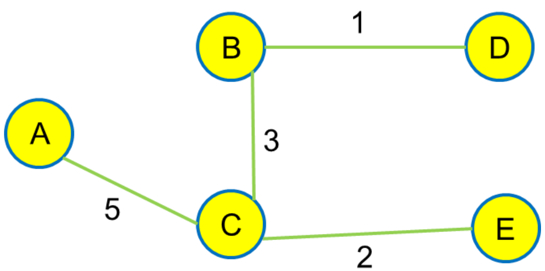




Spanning Tree 1



Spanning Tree 2

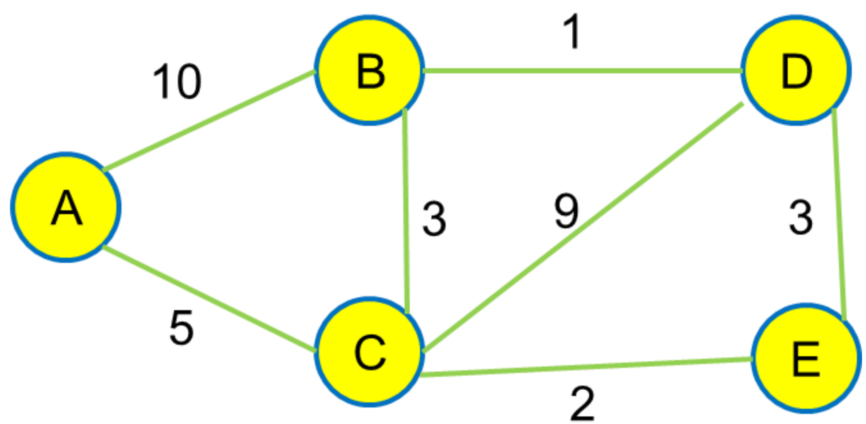


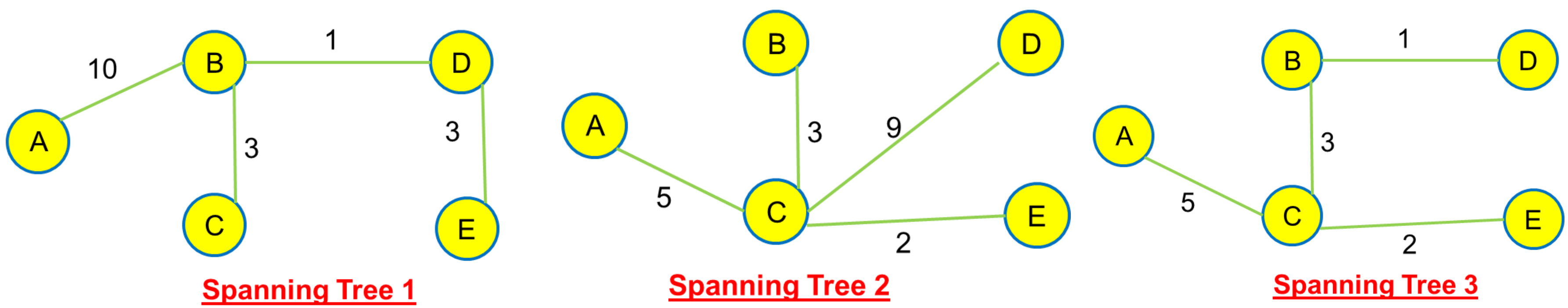
Spanning Tree 3

Minimum Spanning Tree

What is it?

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

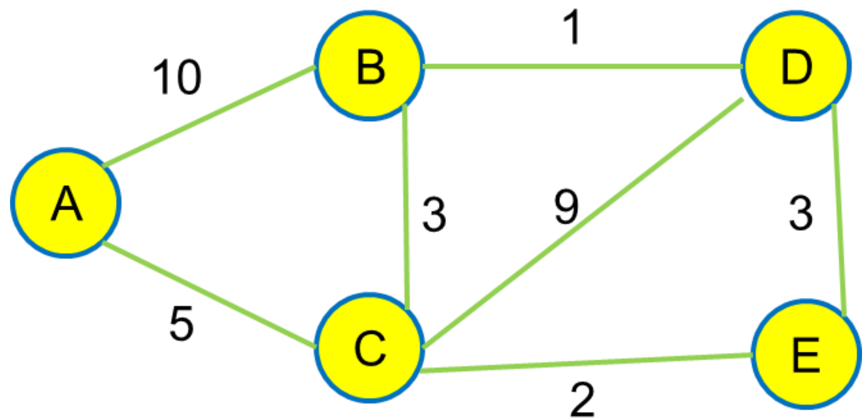




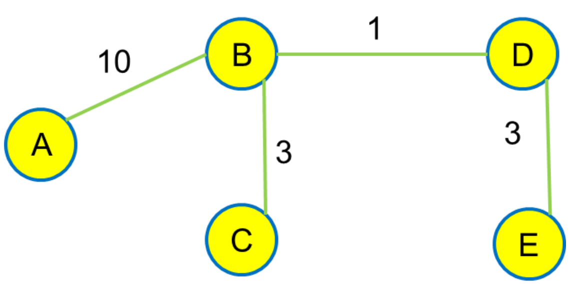
Minimum Spanning Tree

What is it?

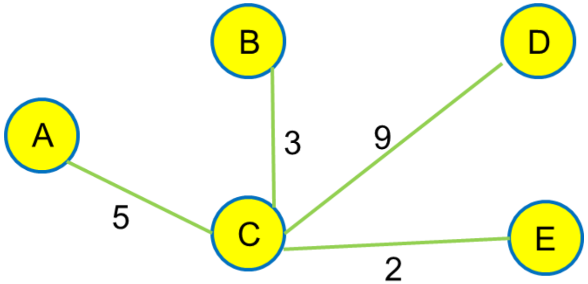
- Let say we have a graph
  - Can you form spanning trees?
  - Which is the minimum?
    - Tree 1 = 10 + 1 + 3 + 3
    - Tree 2 = 5 + 3 + 9 + 2
    - Tree 3 = 5 + 3 + 2 + 1



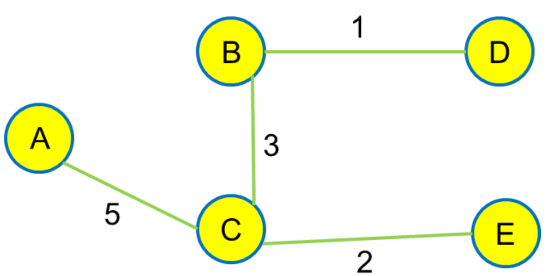
Minimum spanning tree may not be unique



Spanning Tree 1



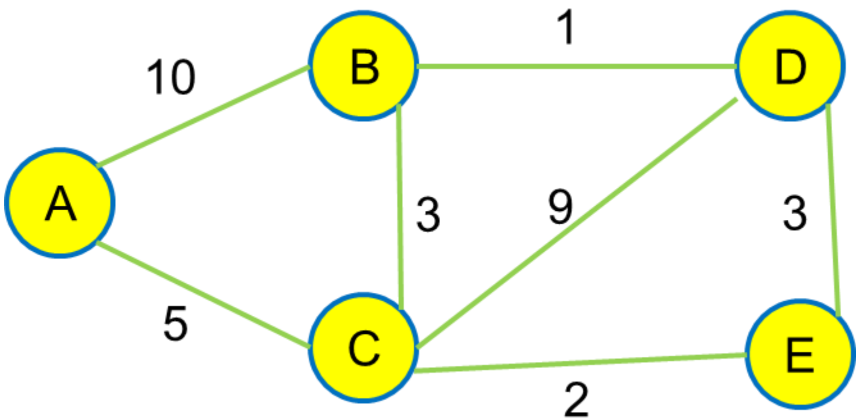
Spanning Tree 2

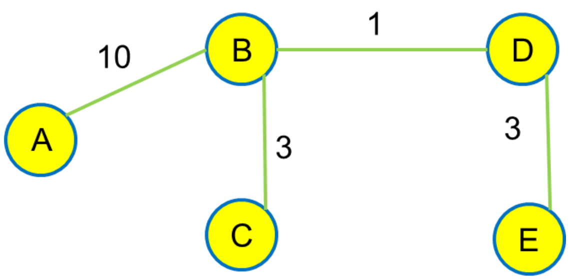


Spanning Tree 3

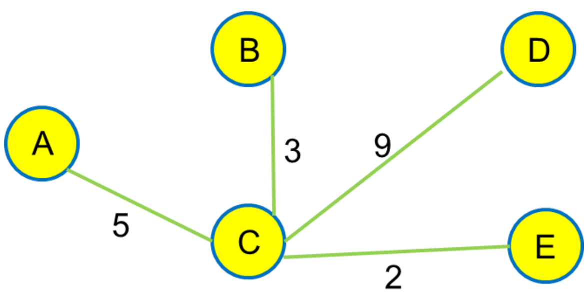
Minimum Spanning Tree
What is it?

- 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

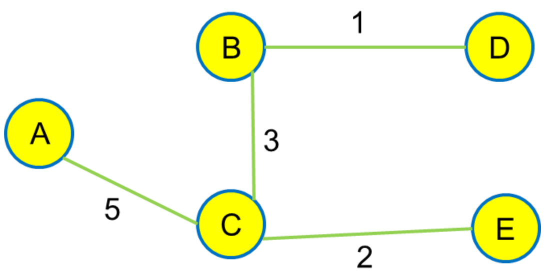




Spanning Tree 1



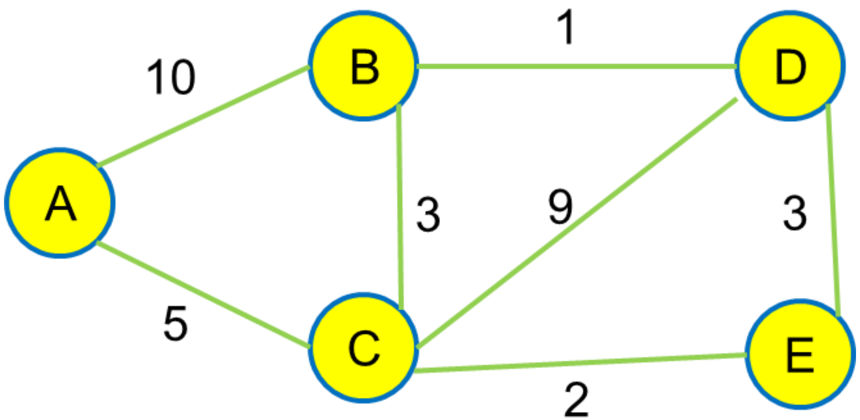
Spanning Tree 2



Spanning Tree 3

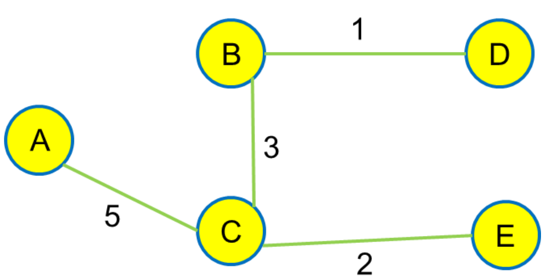
Minimum Spanning Tree

What is it?

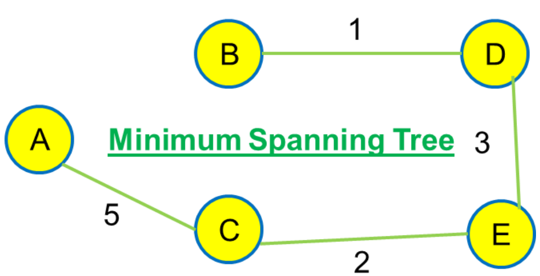


- Let say we have a graph
  - Can you form spanning trees?
  - Which is the minimum?
    - Tree 1 = 10 + 1 + 3 + 3 = 17
    - Tree 2 = 5 + 3 + 9 + 2 = 19
    - Tree 3 = 5 + 3 + 2 + 1 = 11
    - Tree 4 = 5 + 2 + 3 + 1 = 11

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

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

- Both are greedy

- 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

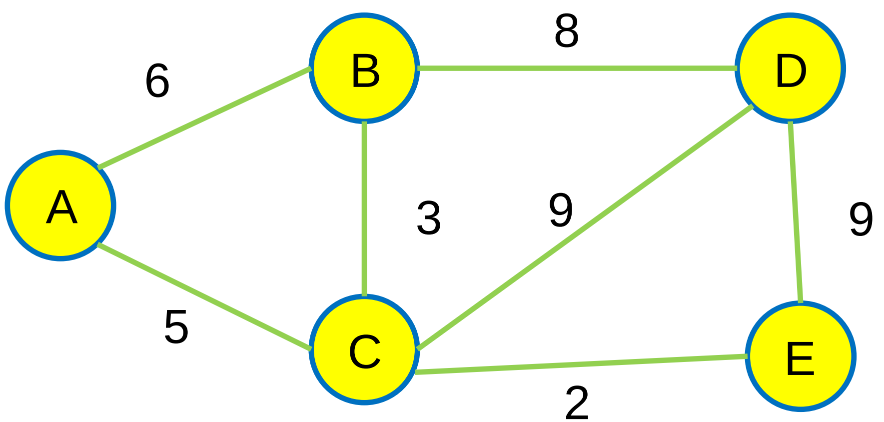
- Both are greedy
  - Choose local optimal
  - Believe to get global optimal
  - We will learn to prove it later

Questions?

Prim’s Algorithm
Growing of MST

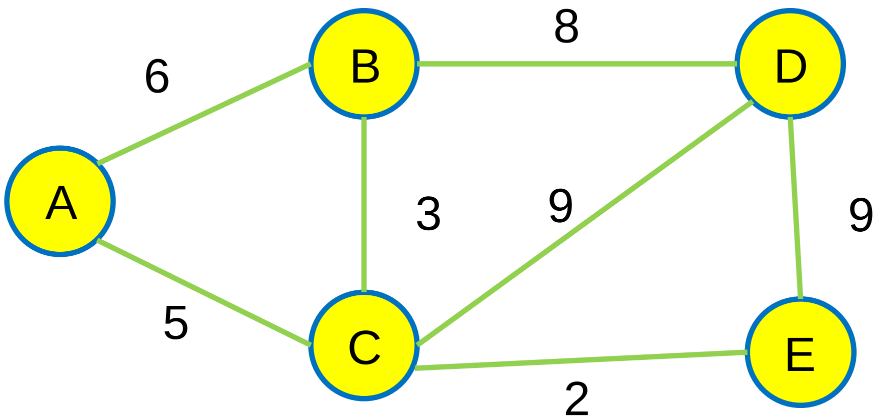
- Very similar to Dijkstra
  - Choosing nearest vertex to tree

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



MONASH University

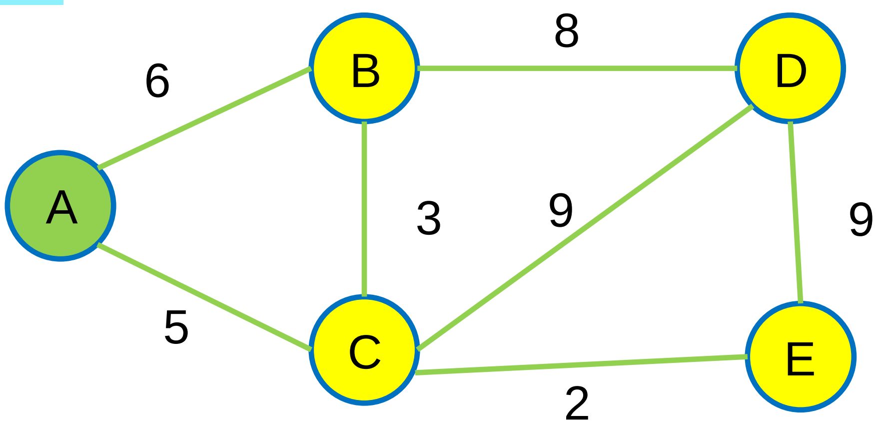
- Very similar to Dijkstra
  - Choosing nearest vertex to tree
  - Let us try it out



A	B	C	D	E
0	inf	inf	inf	inf

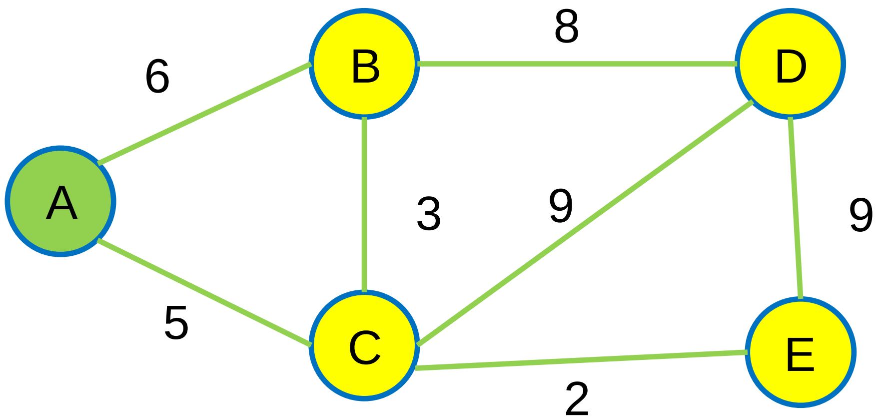
Prim's Algorithm
Growing of MST

- 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

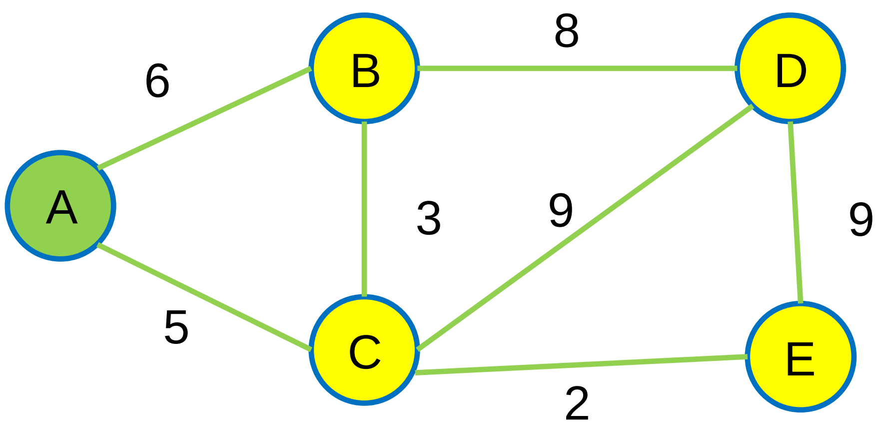
- Very similar to Dijkstra
  - Choosing nearest vertex to tree
  - Let us try it out
  - Start from A
  - Update adjacent B and C



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

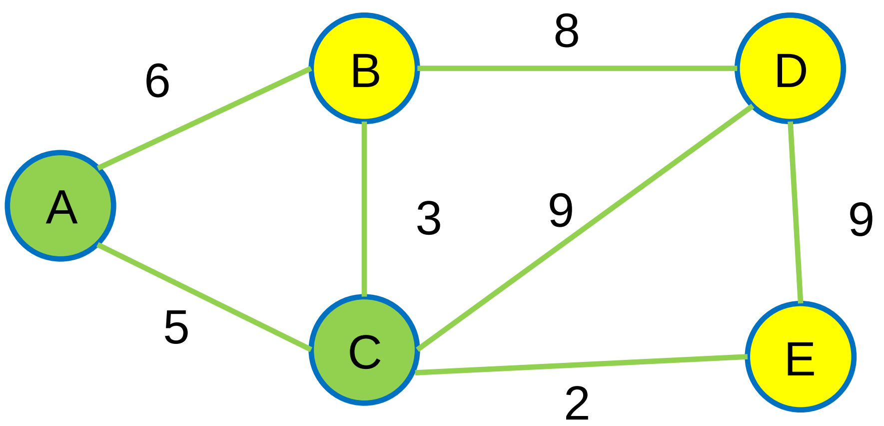
Prim’s Algorithm
Growing of MST

- 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

- 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

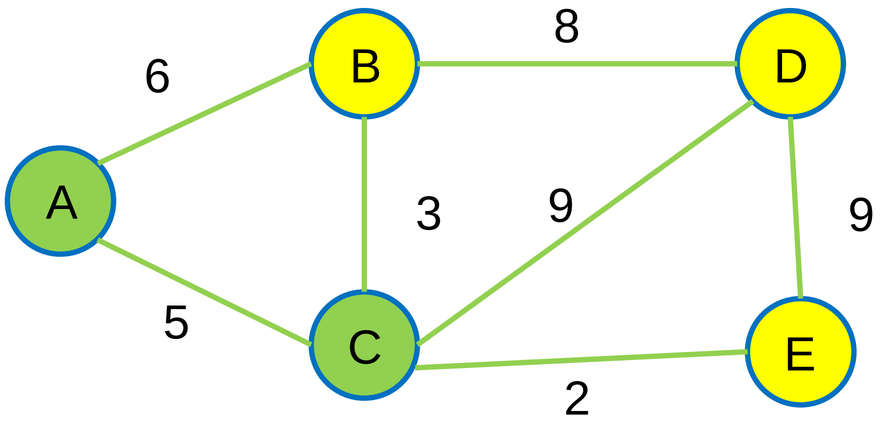


A	B	C	D	E
0	6, A	5, A	inf	inf

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

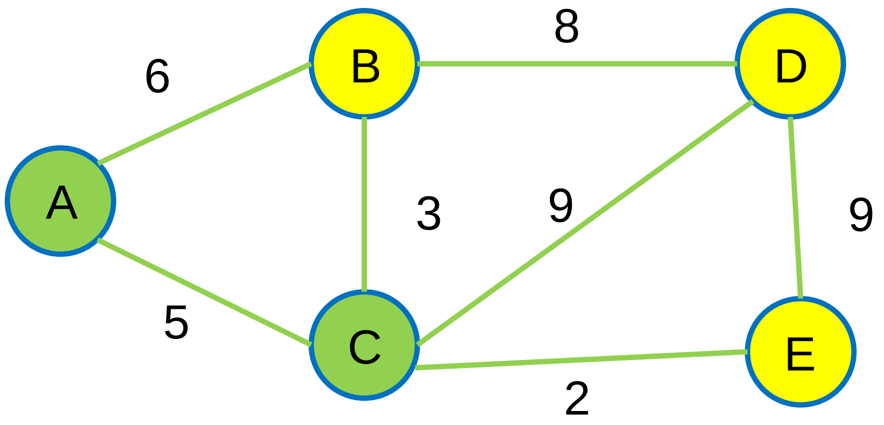


A	B	C	D	E
0	6	5, A	inf	inf

Prim's Algorithm

Growing of MST

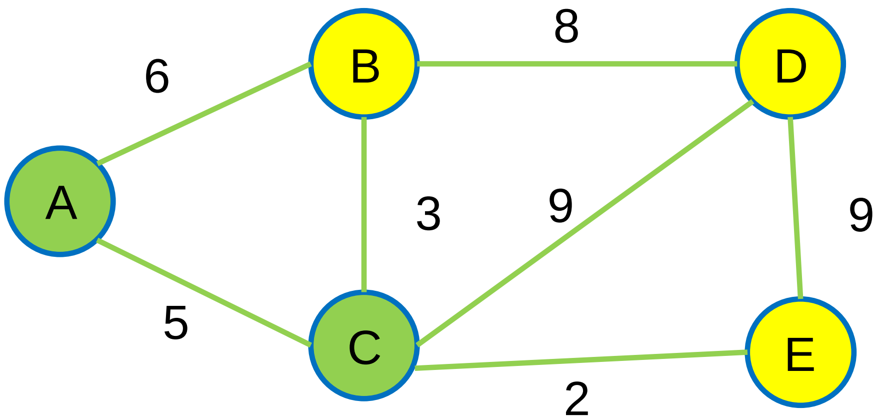
- Very similar to Dijkstra
  - Choosing nearest vertex to tree
  - Let us try it out
  - Start from A
  - Update adjacent B and C
  - Choose closest C
  - Update adjacent B, D, E



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

MONASH University

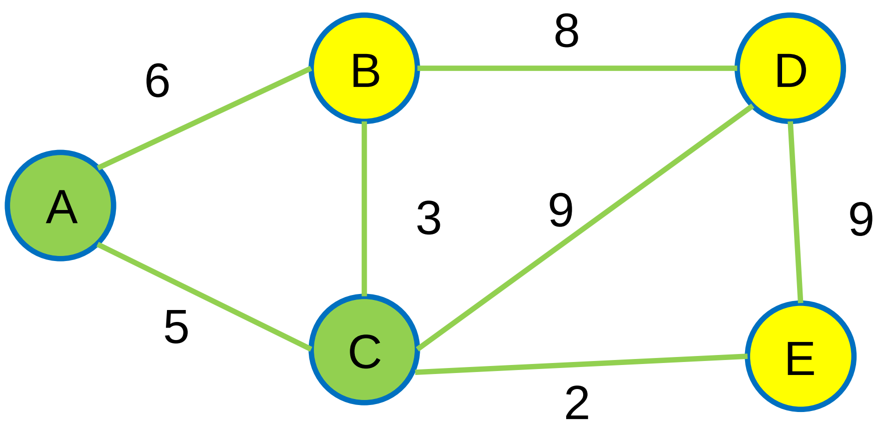
- Very similar to Dijkstra
  - Choosing nearest vertex to tree
  - Let us try it out
  - Start from A
  - Update adjacent B and C
  - Choose closest C
  - Update adjacent B, D, E



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

Prim's Algorithm
Growing of MST

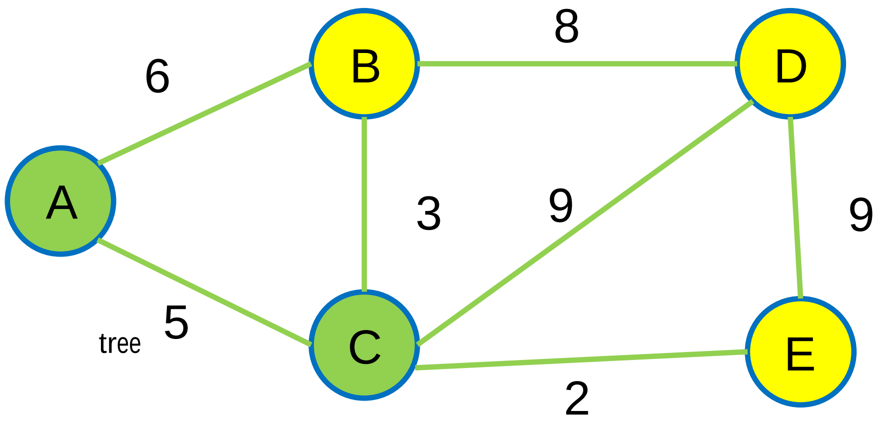
- Very similar to Dijkstra
  - Choosing nearest vertex to tree
  - Let us try it out
  - Start from A
  - Update adjacent B and C
  - Choose closest C
  - Update adjacent B, D, E



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

Prim's Algorithm
Growing of MST

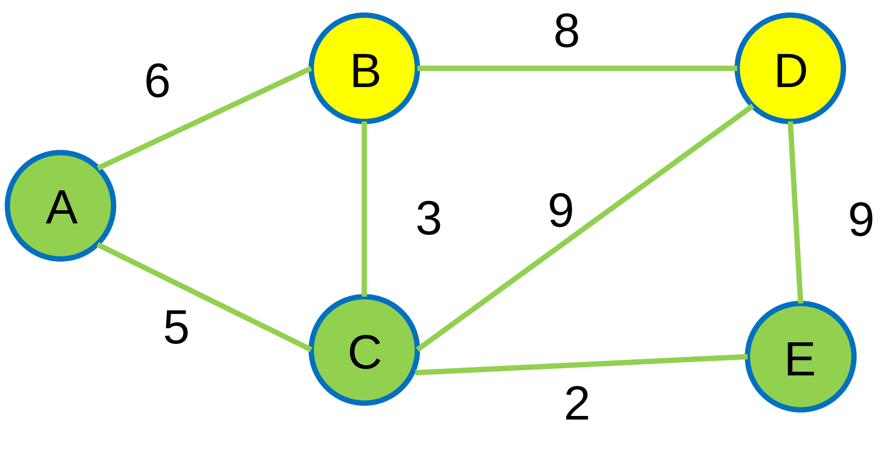
- Very similar to Dijkstra
  - Choosing nearest vertex to tree
  - Let us try it out
  - Start from A
  - Update adjacent B and C
  - Choose closest C
  - Update adjacent B, D, E



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

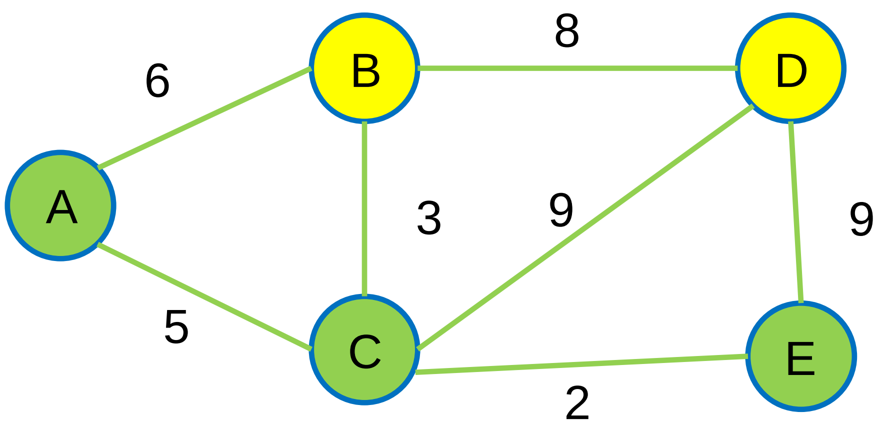
MONASH University

- 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



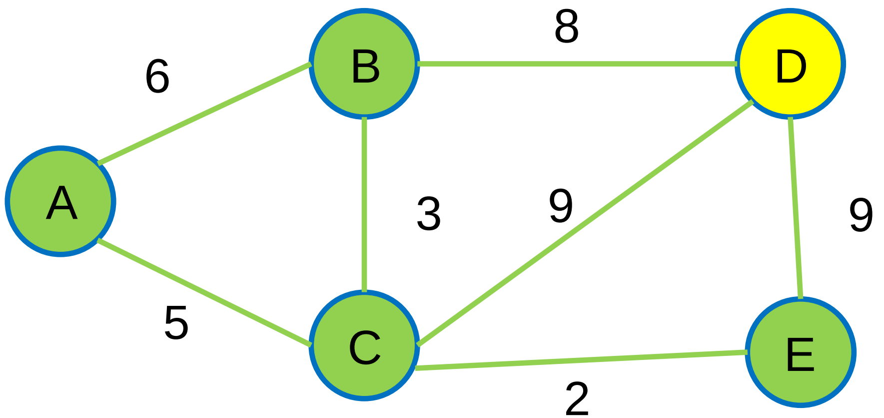
A	B	C	D	E
0	3, C	5, A	9, C	2, C

- 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

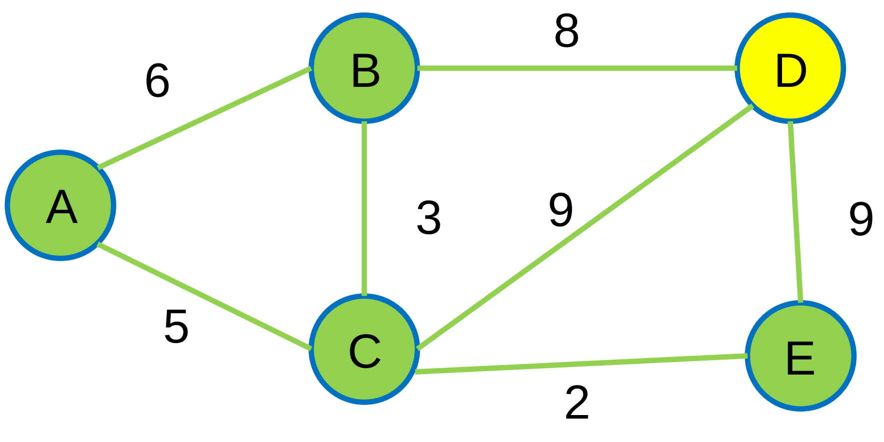
- Very similar to Dijkstra
  - Choosing nearest vertex to tree
  - Let us try it out
  - Start from A
  - Update adjacent B and C
  - Choose closest C
  - Update adjacent B, D, E
  - Choose closest E
  - Update adjacent D
  - Choose closest B



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

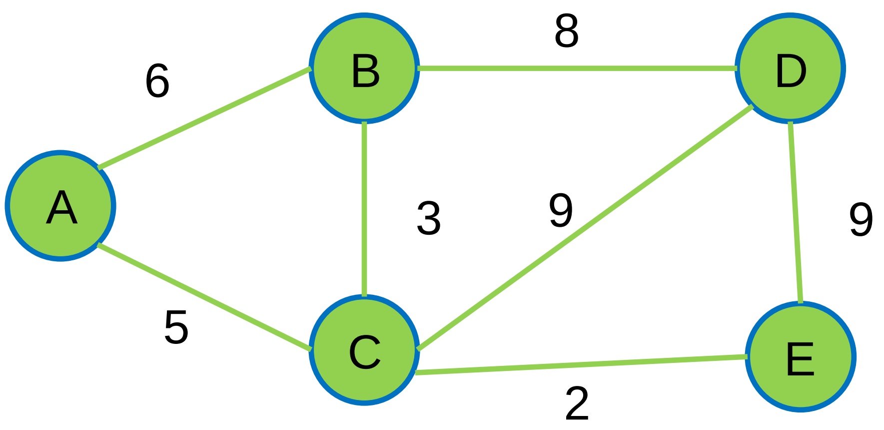


A	B	C	D	E
0	3, C	5, A	8, B	2, C

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

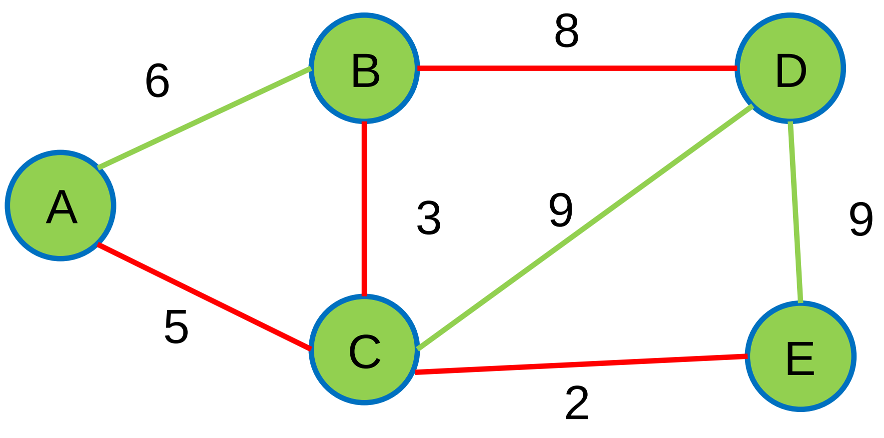


A	B	C	D	E
0	3, C	5, A	8, B	2, C

Prim's Algorithm

Growing of MST

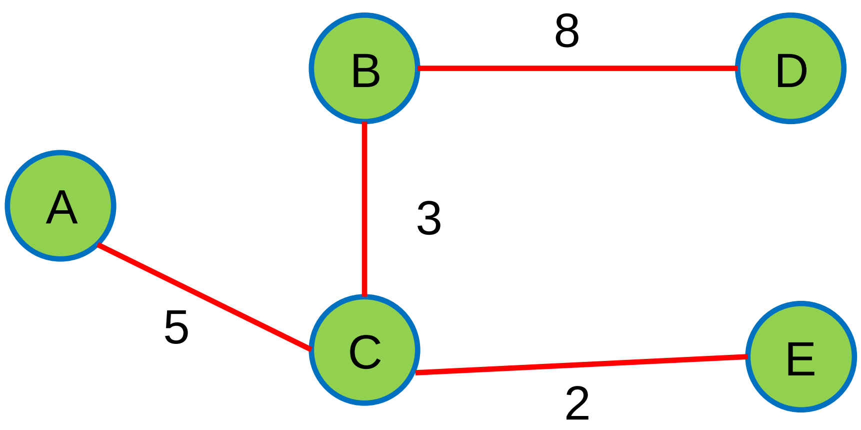
- Very similar to Dijkstra
  - Choosing nearest vertex to tree
  - Let us try it out
  - Start from A
  - Update adjacent B and C
  - Choose closest C
  - Update adjacent B, D, E
  - Choose closest E
  - Update adjacent D
  - Choose closest B
  - Update adjacent D
  - Choose closest D
  - Have all of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



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

MONASH University

- 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

MONASH University

- So take Dijkstra
  - Modify the distance update/ calculation for edge <u,v,w>
    - Instead of v.distance = u.distance + w
    - Change to v.distance = w

u.distance = u.distance

Prim's Algorithm
Growing of MST

- So take Dijkstra
  - Modify the distance update/ calculation for edge <u,v,w>
    - Instead of v.distance = u.distance + w
    - Change to v.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 <u,v,w>
    - Instead of v.distance = u.distance + w
    - Change to v.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 <u,v,w>
    - Instead of v.distance = u.distance + w
    - Change to v.distance = w
    - Perform relaxation only if distance is smaller

- So what is the complexity?
  for every edge get
  - Same as Dijkstra O(V log V + E log V) MinHeap
    for every u find a next vertex to go through corresponding edge
  - Thus O(E log V)

Questions?

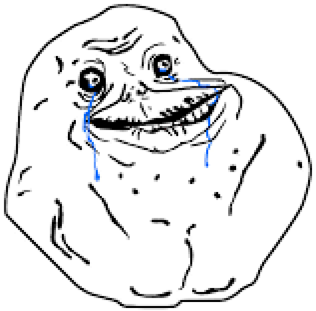
Kruskal's Algorithm

Combining (Union of) Trees

Kruskal's Algorithm
Combining (Union of) Trees

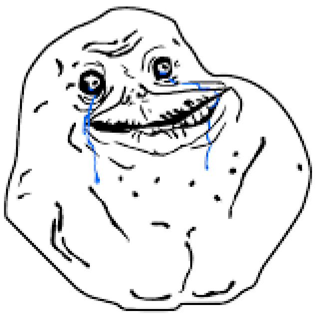
Merge

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



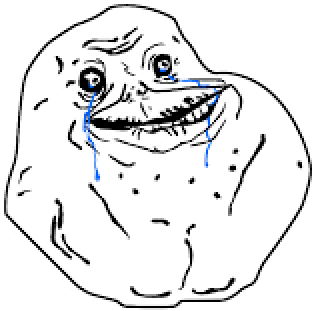
Kruskal's Algorithm
Combining (Union of) Trees

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



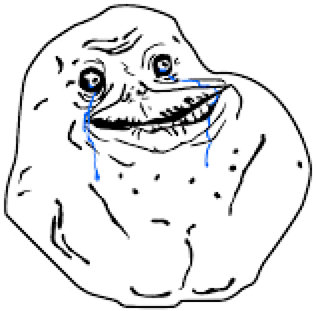
Kruskal’s Algorithm
Combining (Union of) Trees

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



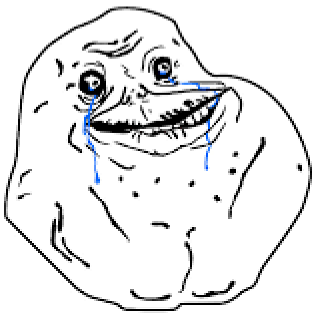
Kruskal's Algorithm
Combining (Union of) Trees

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



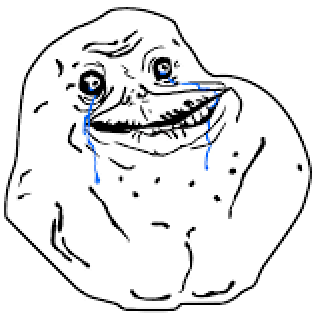
Kruskal's Algorithm
Combining (Union of) Trees

- Imagine every vertex is a tree
  - Only 1 node #ForeverAlone
  - Trees are connected by edges
    - Adding edge <u,v,w> 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

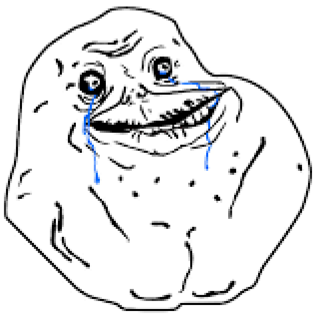
- Imagine every vertex is a tree
  - Only 1 node #ForeverAlone
  - Trees are connected by edges
    - Adding edge <u,v,w> combine the trees of vertex u and vertex v
    - Only add if vertex u and vertex v are not in the same tree. Why? NO CYCLE



- So how do we do it?

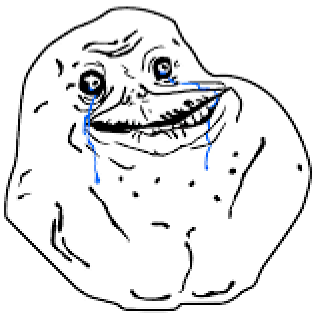
Kruskal's Algorithm
Combining (Union of) Trees

- Imagine every vertex is a tree
  - Only 1 node #ForeverAlone
  - Trees are connected by edges
    - Adding edge <u,v,w> 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

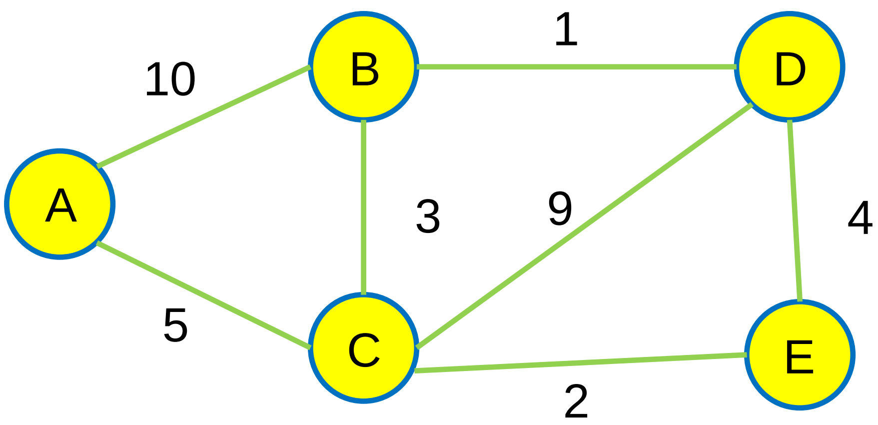


- Imagine every vertex is a tree
  – Only 1 node #ForeverAlone
  – Trees are connected by edges
    - Adding edge <u,v,w> combine the trees of vertex u and vertex v
    - Only add if vertex u and vertex v are not in the same tree. Why? NO CYCLE

- So how do we do it?
  – Take add edges
  – Sort it
  – Then go through the edges one by one
  – Let us visualize it…

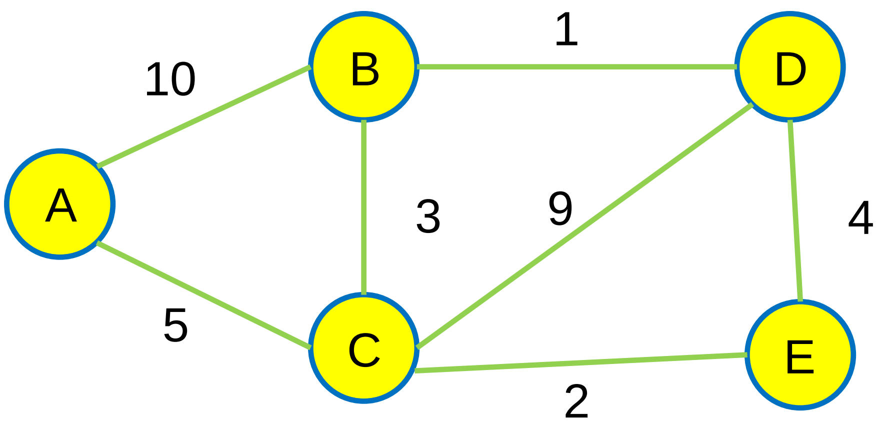
Kruskal's Algorithm
Combining (Union of) Trees

- Look at the graph



Kruskal's Algorithm
Combining (Union of) Trees

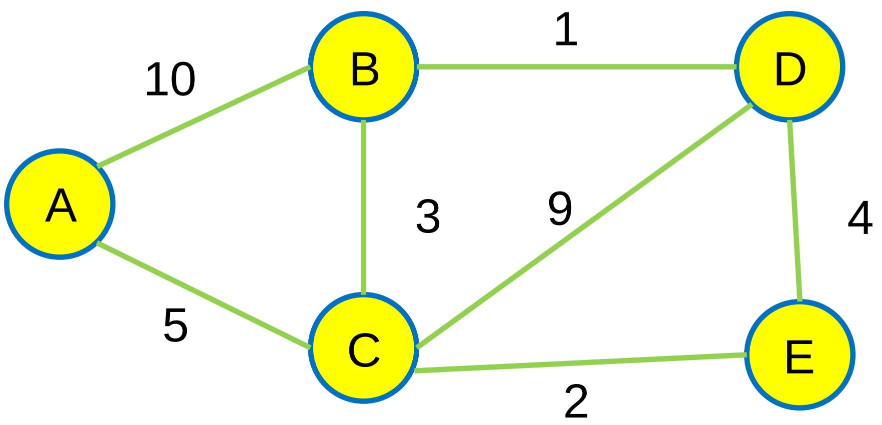
- Look at the graph
  - Take all the edges



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

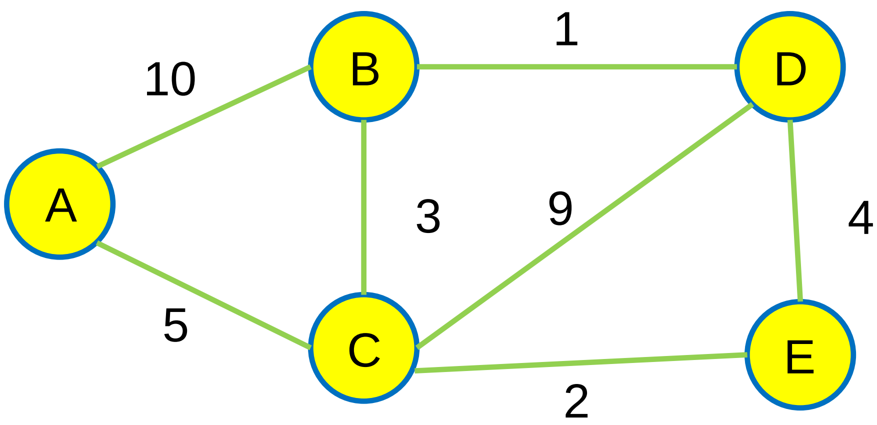


AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

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



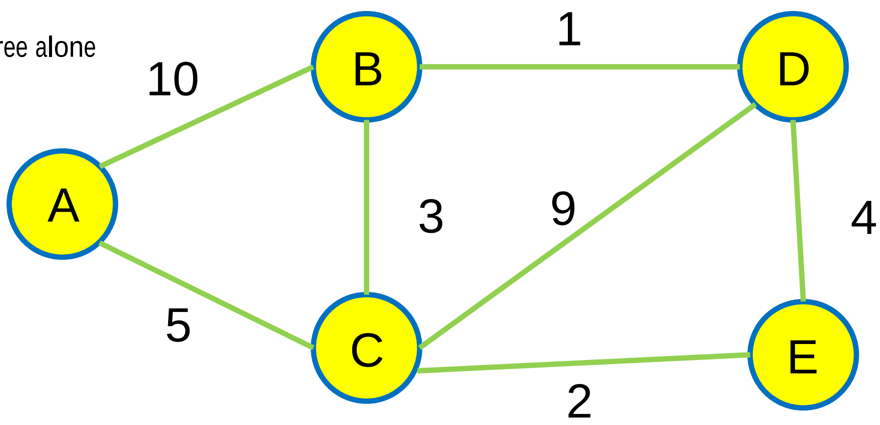
AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

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

everyone is the tree alone



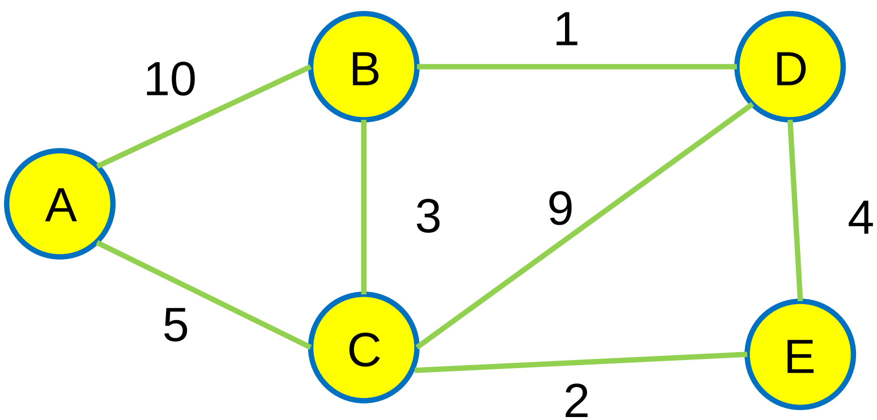
AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

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

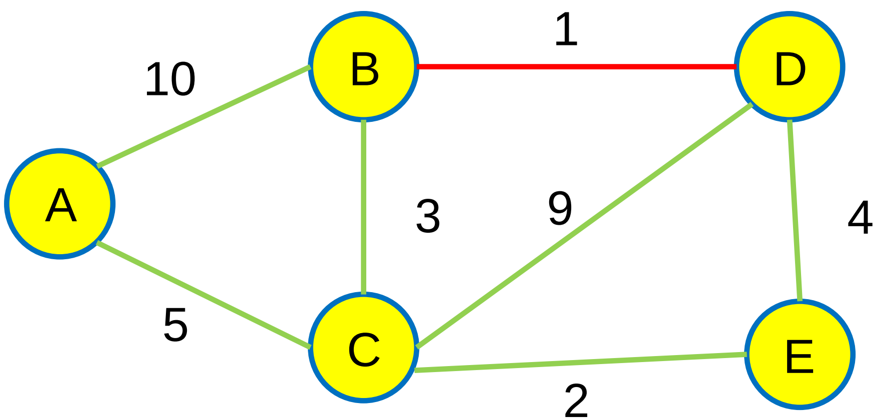


AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

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



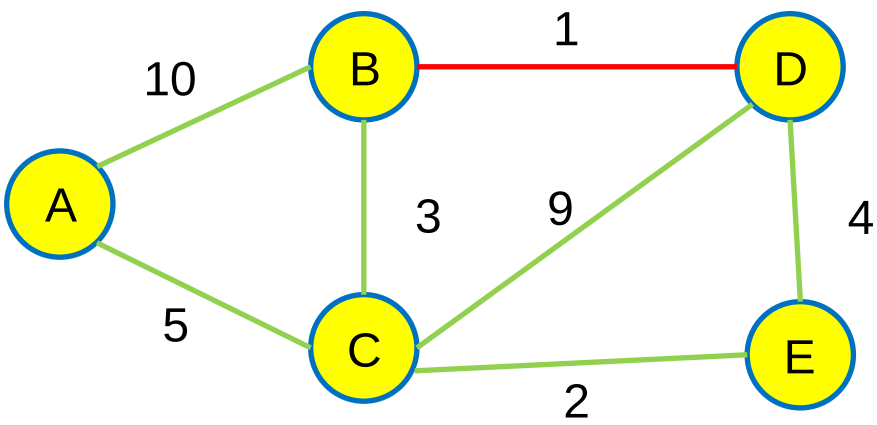
AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

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

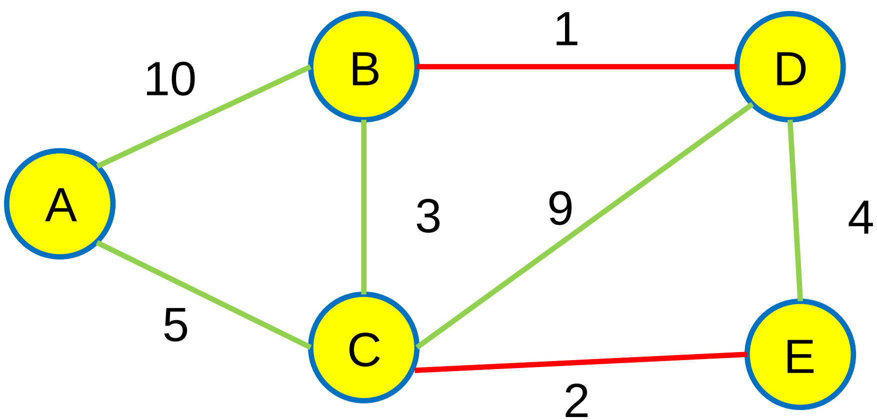


AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

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

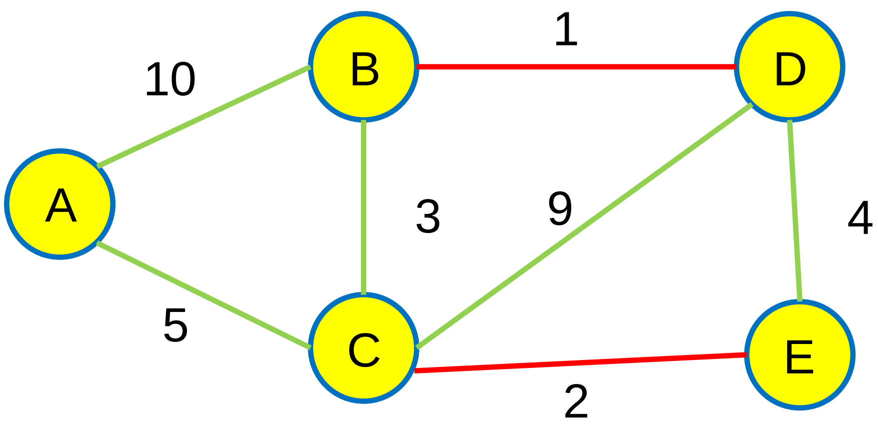


AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

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

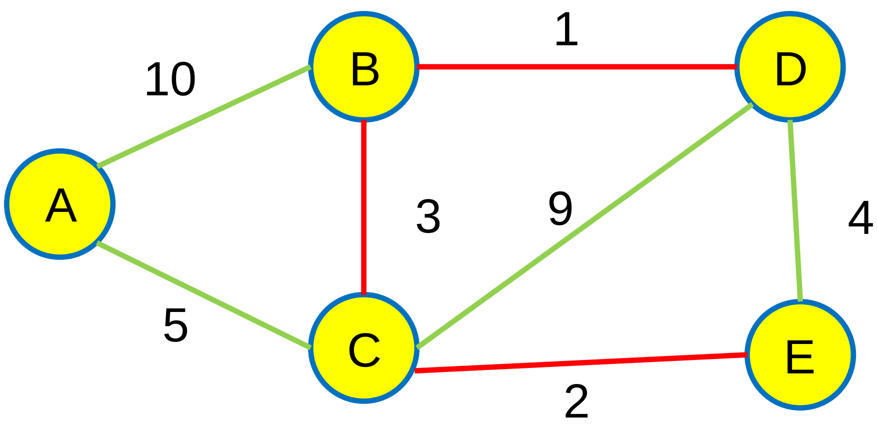


AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

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

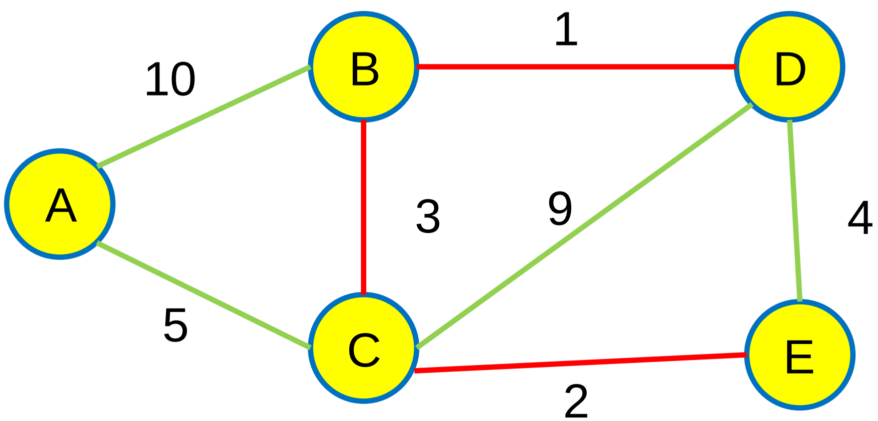


AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

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

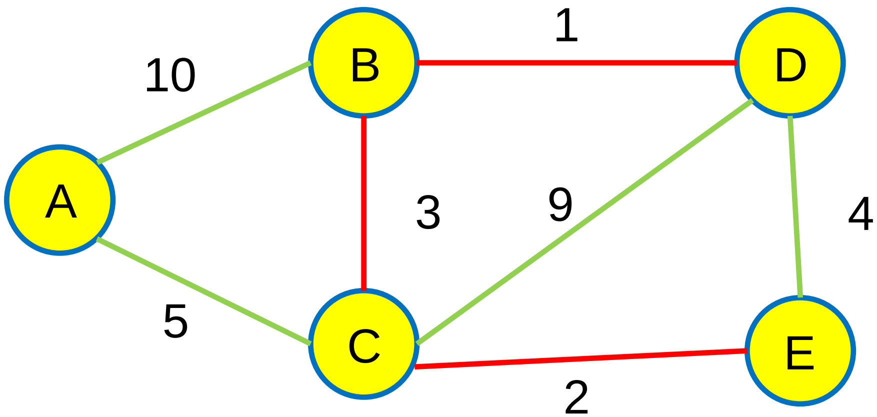


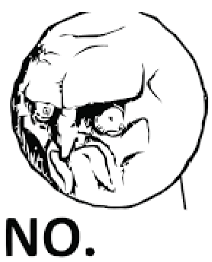
AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

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



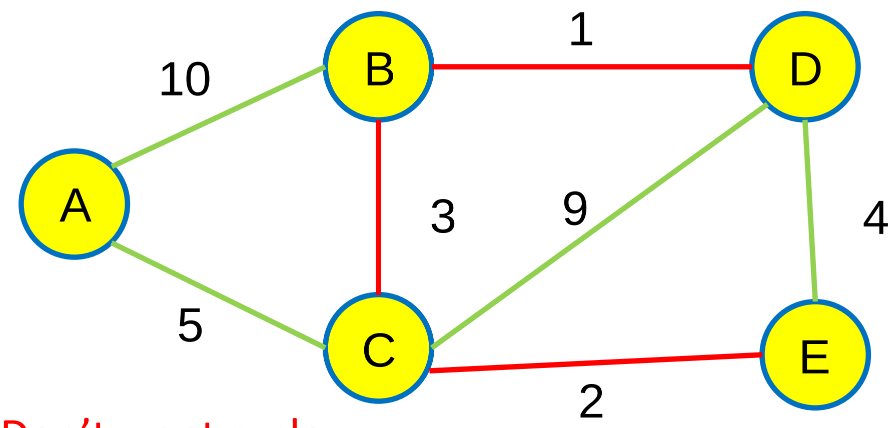


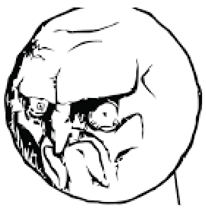
AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

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





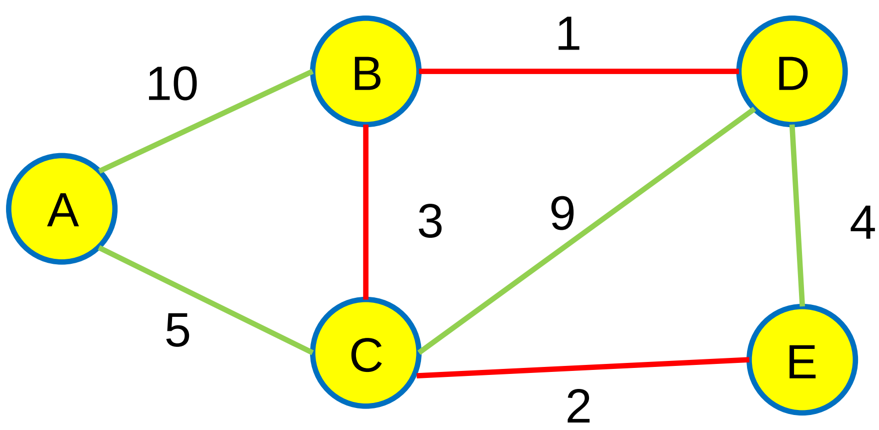
NO.

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

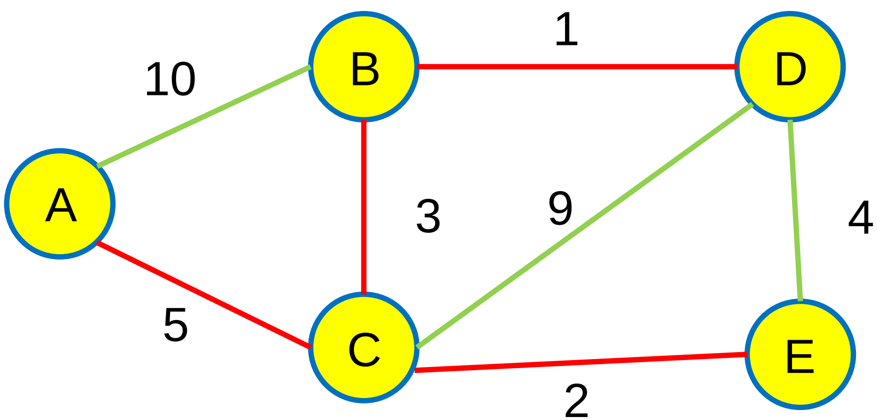


AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

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

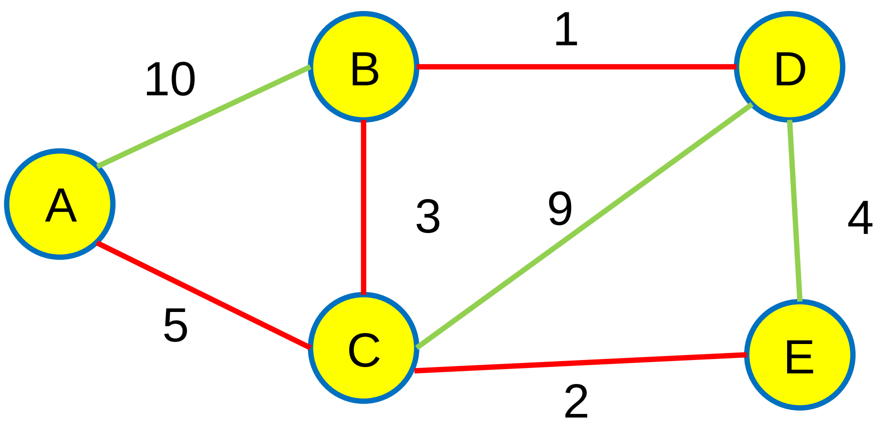


AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

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



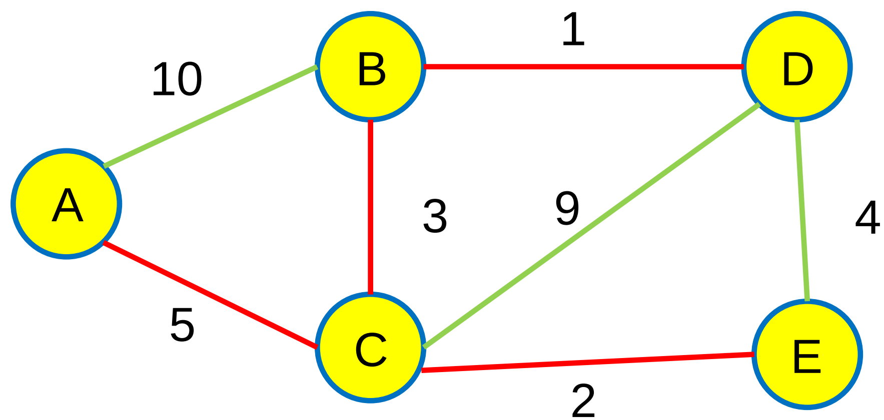
AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

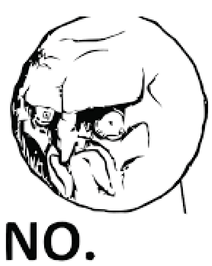
BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10

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



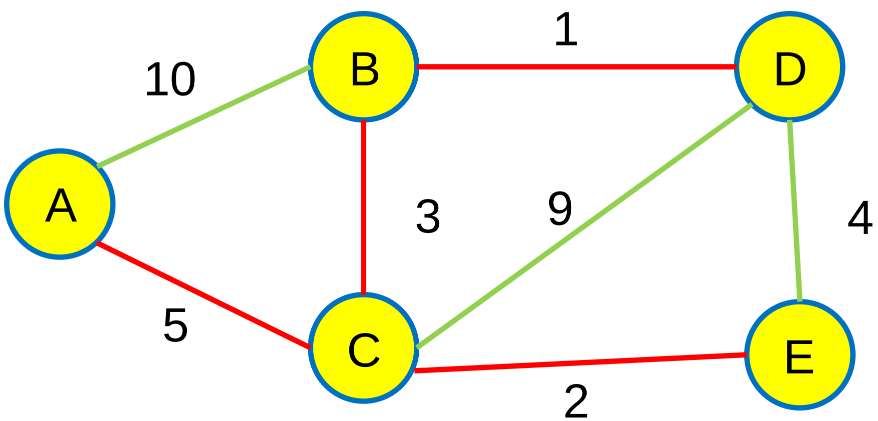


AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

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

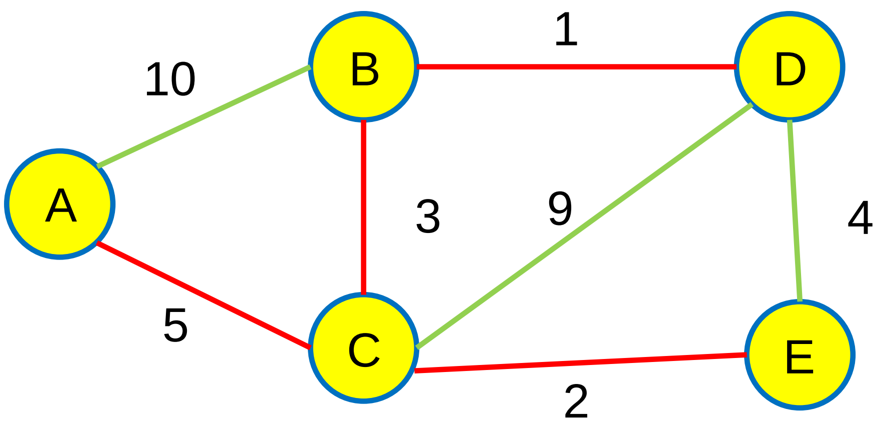


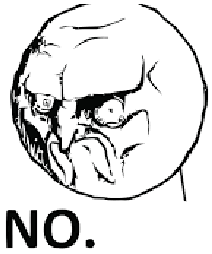
AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

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



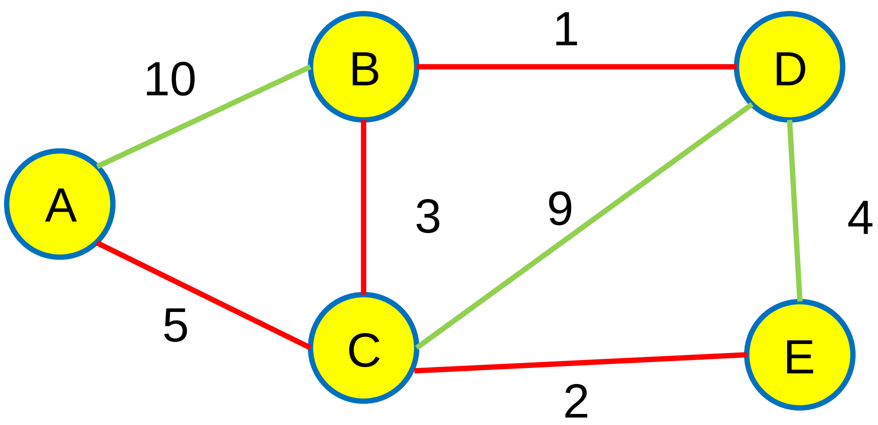


AB	AC	BC	BD	CD	CE	DE
10	5	3	1	9	2	4

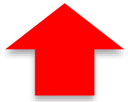
BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10

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
  - And we are done!

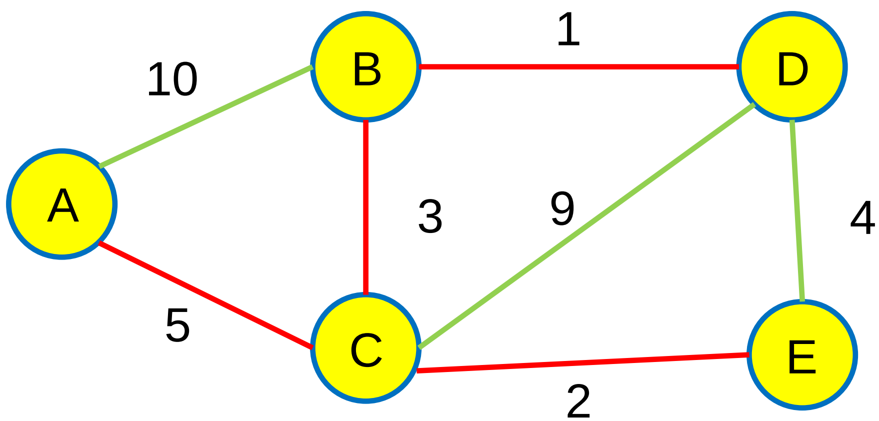


BD	CE	BC	DE	AC	CD	AB
1	2	3	4	5	9	10



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! When we have V - 1 edges or merges
  - 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?

Combining (Union of) Trees

- But how do we implement it?

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

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

- But how do we implement it?
  - Take the list of edges and sort.
    - Easy… just use quicksort
    - Why not Counting or Radix? Counting (M + N) N number of items
      M - biggest number edge may have weight way to large

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 <u,v,w> 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 <u,v,w> 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 <u,v,w> is in the same tree
    - HOW?
    - We use set! Any set data structure

- 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 <u,v,w> 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 <u,v,w> 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 <u,v,w> 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 <u,v,w> 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

- 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 <u,v,w> 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?

MONASH University

- 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 <u,v,w> 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) sort edges
  - Check if vertex u and vertex v in <u,v,w> 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 <u,v,w> 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) Union find, O(logV)
  - If not the same set, you joint them with the edge (Union)
    - This is O(V) for now O(1), join root to another root
  - Thus, known as Union-Find

  - Complexity?

For each edge

- But how do we implement it?
  - Take the list of edges and sort.
    - Easy… just use quicksort
    - This is O(E log E) quicksort on every edge
  - Check if vertex u and vertex v in <u,v,w> 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 u and v in the same set would cause cycle
  - If not the same set, you joint them with the edge (Union)
    - This is O(V) for now O(V) for each member in another group to join target Group
  - Thus, known as Union-Find

For each edge

E log V^2
= E * 2 log V
= E log V

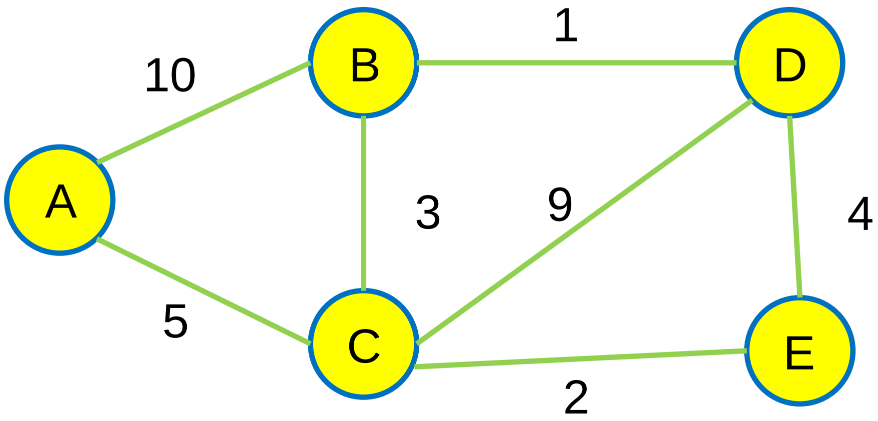
E = O(V^2) E logE = E * log(V^2) = E * 2log(V)

  - Complexity? O(E log E + E(1+V)) = O(EV)

E at most equal (V-1)^2 for undirected graph
log E < V

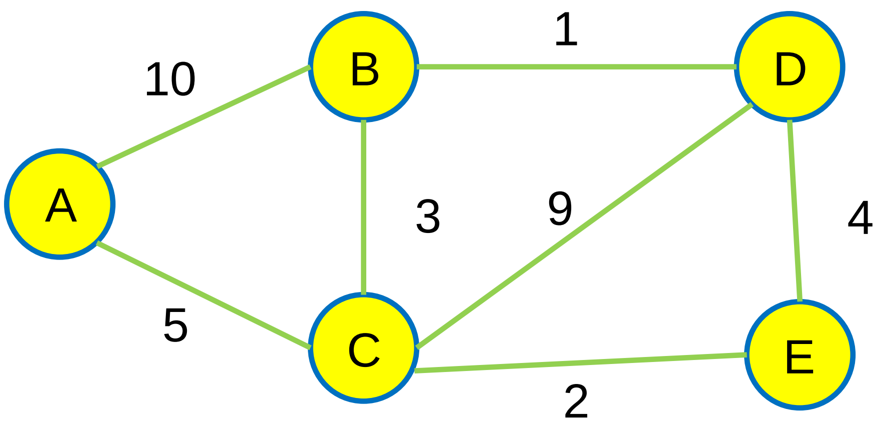
Kruskal's Algorithm
Combining (Union of) Trees

- Union-Find with sets



Kruskal's Algorithm
Combining (Union of) Trees

- Union-Find with sets

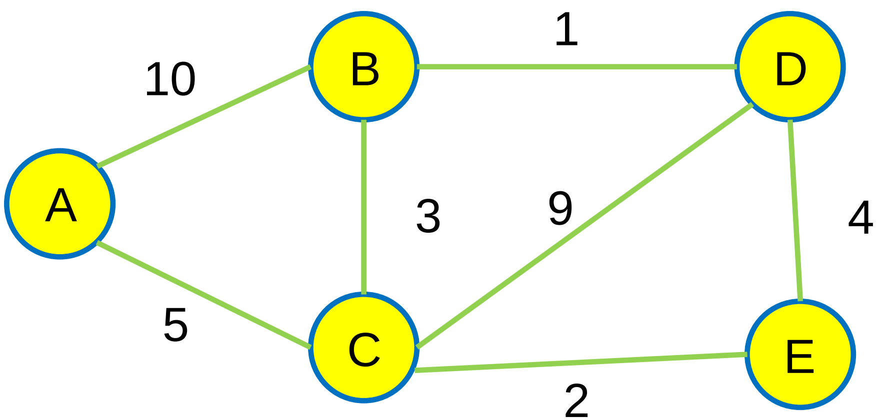


A	B	C	D	E
1	2	3	4	5

original each one in their own group

MONASH University

- Union-Find with sets

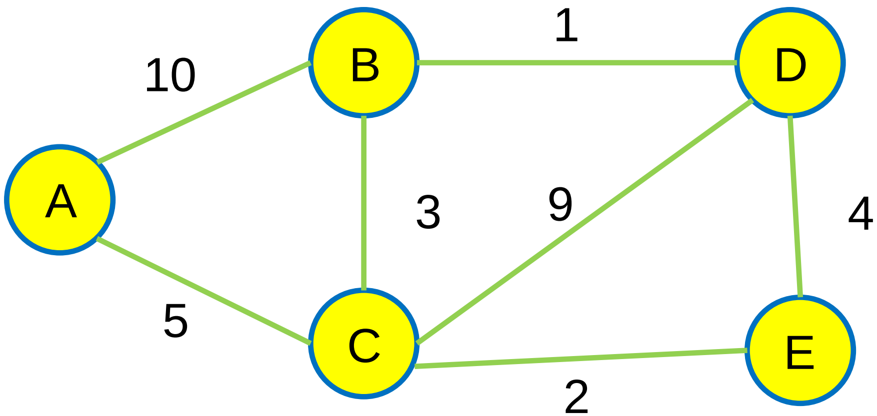


1	2	3	4	5
A	B	C	D	E

A	B	C	D	E
1	2	3	4	5

Combining (Union of) Trees

- Union-Find with sets



1	2	3	4	5
A	B	C	D	E

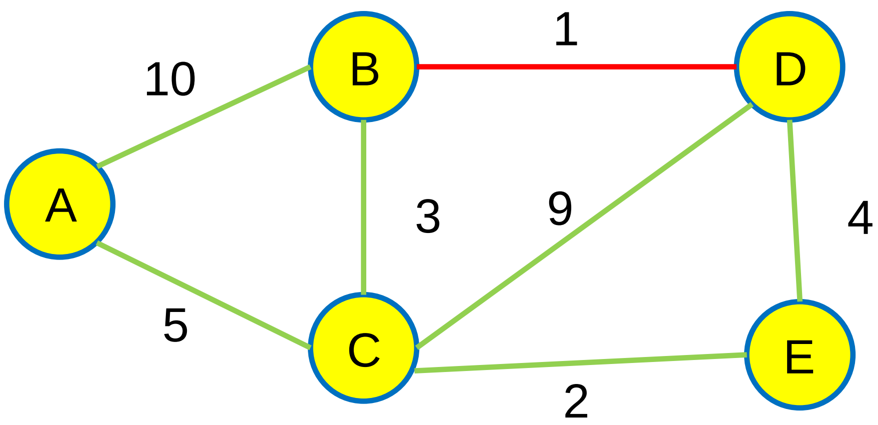
Set Array

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

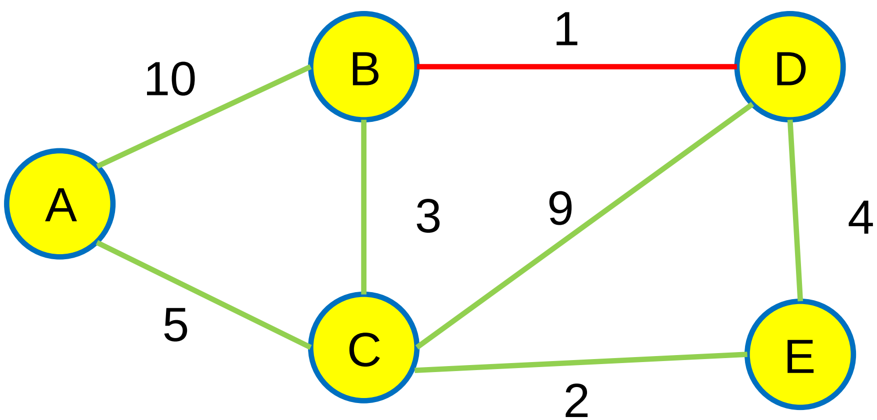
Set Array

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

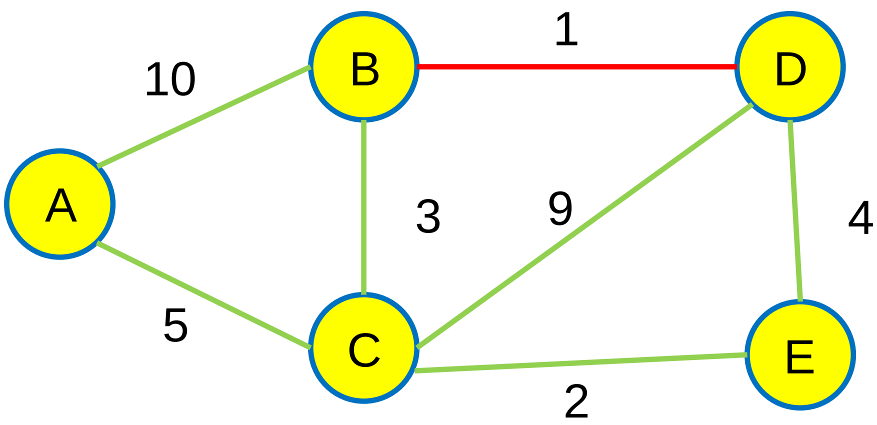
Set Array

A	B	C	D	E
1	2	3	4	5

Map Array

Kruskal's Algorithm
Combining (Union of) Trees

- Union-Find with sets



If

1	2	3	4	5
A	B,D	C		E

Set Array

not in the same set
perform merge between B and D

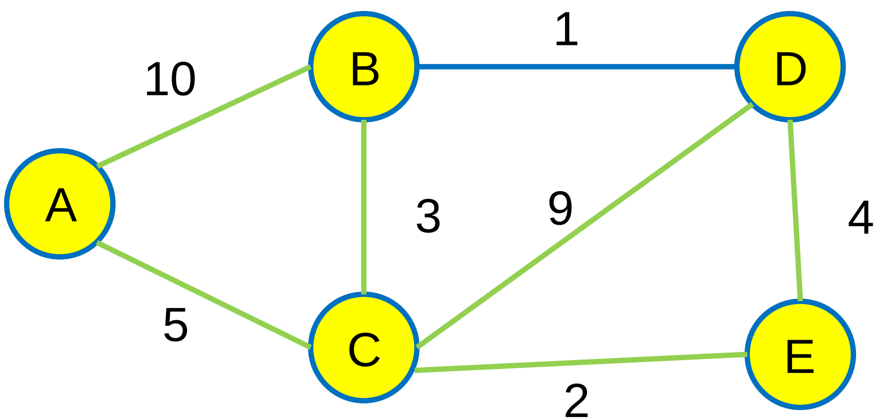
A	B	C	D	E
1	2	3	2	5

Map Array

union (join two vertices by an edge)
by changing corresponding index in Map Array
into the group that join
in D, 4 ->2 to union with B join Group 2

Kruskal's Algorithm
Combining (Union of) Trees

- Union-Find with sets



1	2	3	4	5
A	B,D	C		E

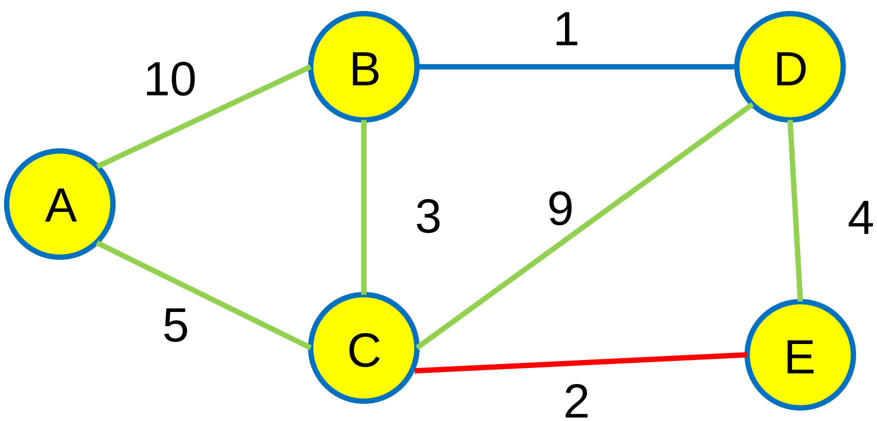
Set Array

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

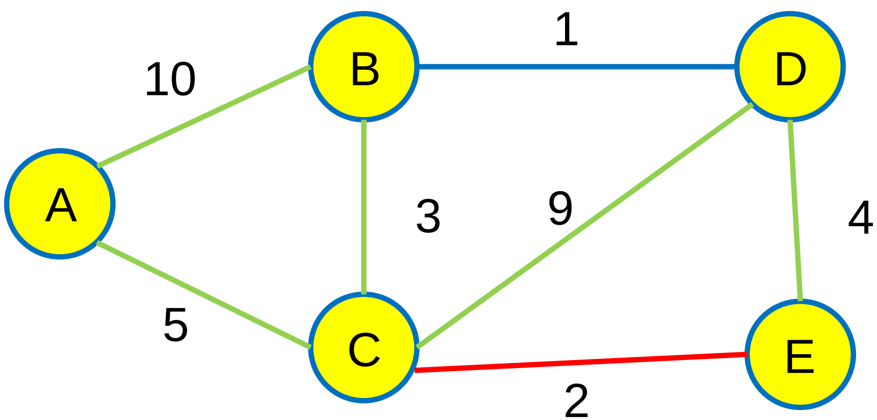
Set Array

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

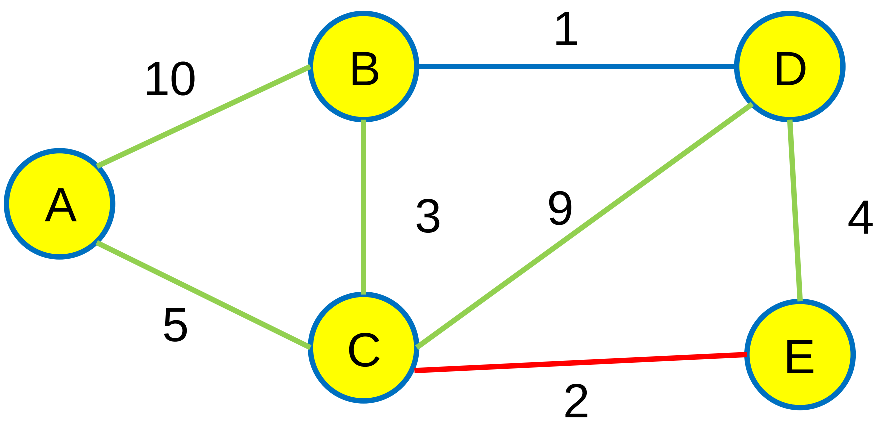
A	B	C	D	E
1	2	3	2	5

Map Array

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets



BD connected by edge, CE connected by edge

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

Set Array

Merge by vertices

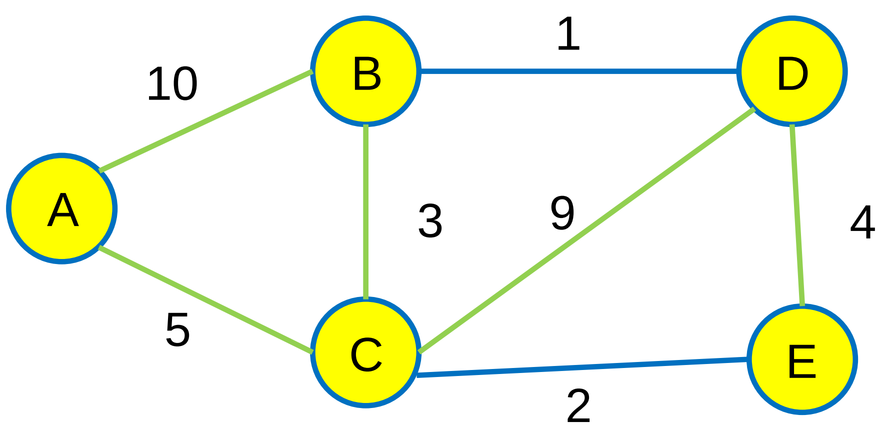
A	B	C	D	E
1	2	3	2	3

Map Array

change to Group 3 aswell

Kruskal's Algorithm
Combining (Union of) Trees

- Union-Find with sets



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

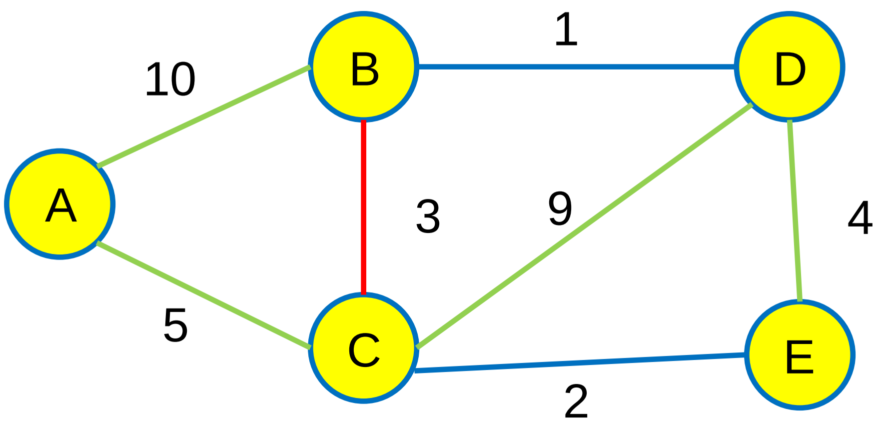
Set Array

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		

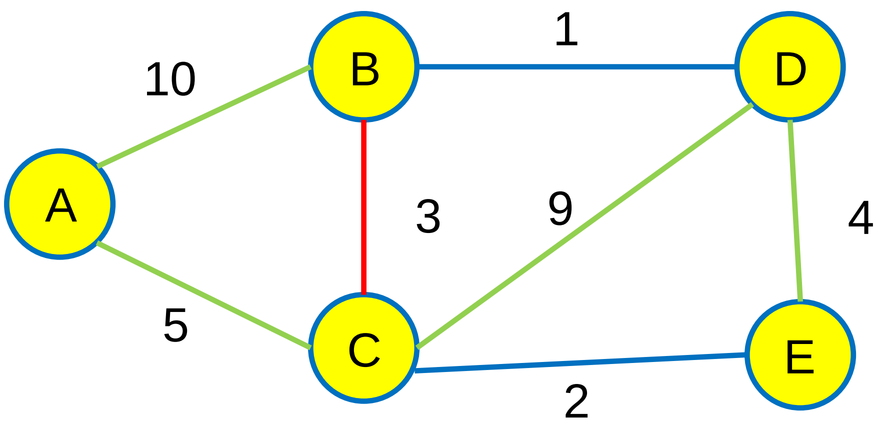
Set Array

A	B	C	D	E
1	2	3	2	3

Map Array

Combining (Union of) Trees

- Union-Find with sets



set element that lower number of vertices goes to the one that has more

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

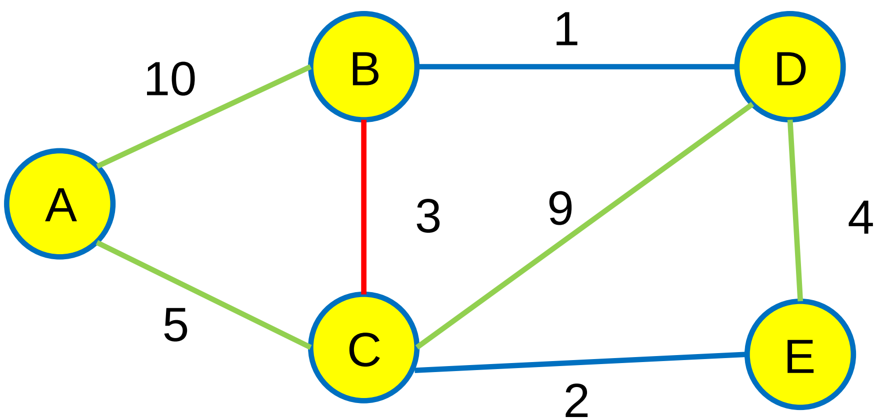
Set Array

A	B	C	D	E
1	2	3	2	3

Map Array

Kruskal's Algorithm
Combining (Union of) Trees

- Union-Find with sets



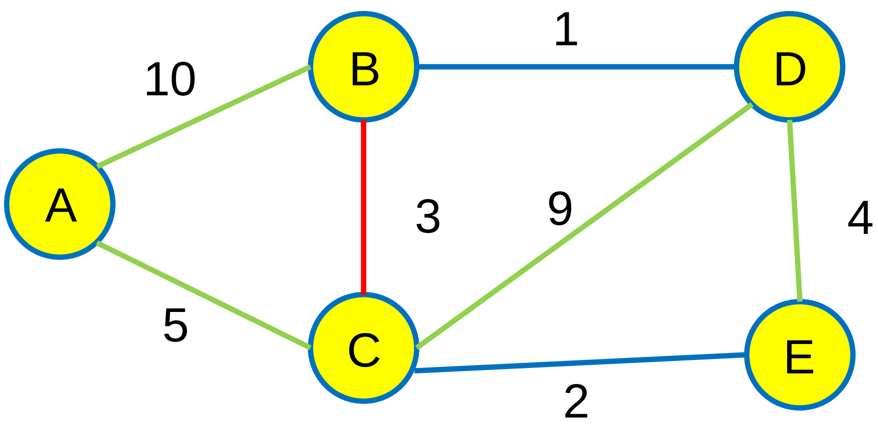
1	2	3	4	5
A	B,D	C,E		

Set Array

A	B	C	D	E
1	2	3	2	3

Map Array

- Union-Find with sets



O(V/2) O(V/2) = O(V)

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

Set Array

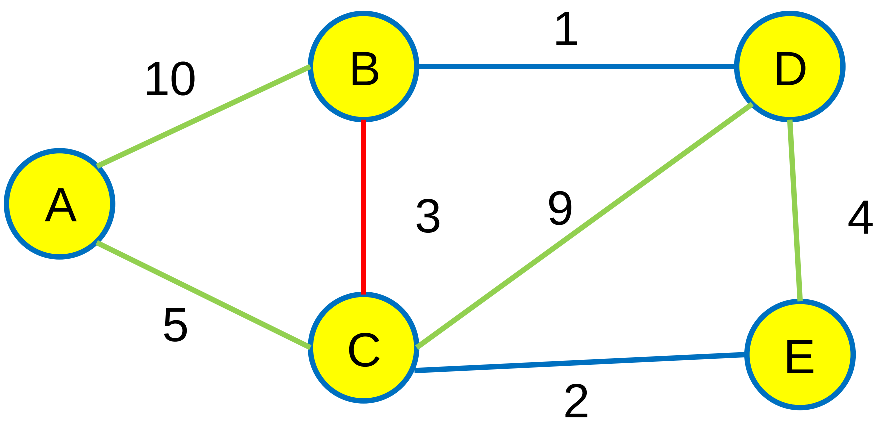
A	B	C	D	E
1	2	3	2	3

Map Array

Kruskal's Algorithm

Combining (Union of) Trees

- Union-Find with sets



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

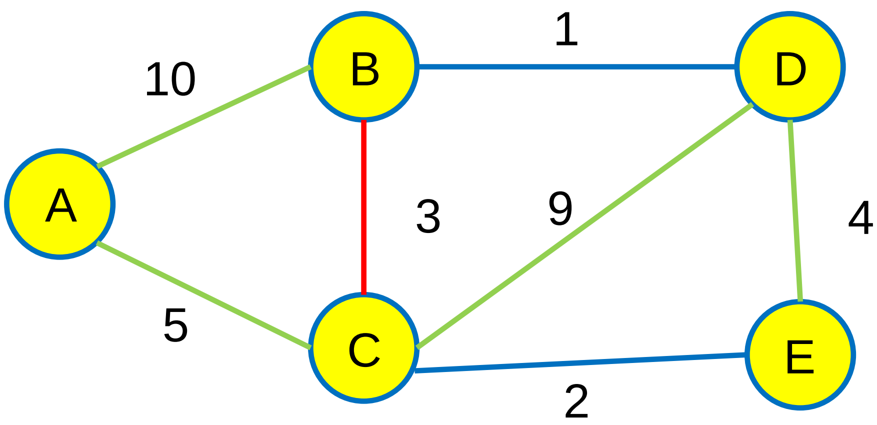
Set Array

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

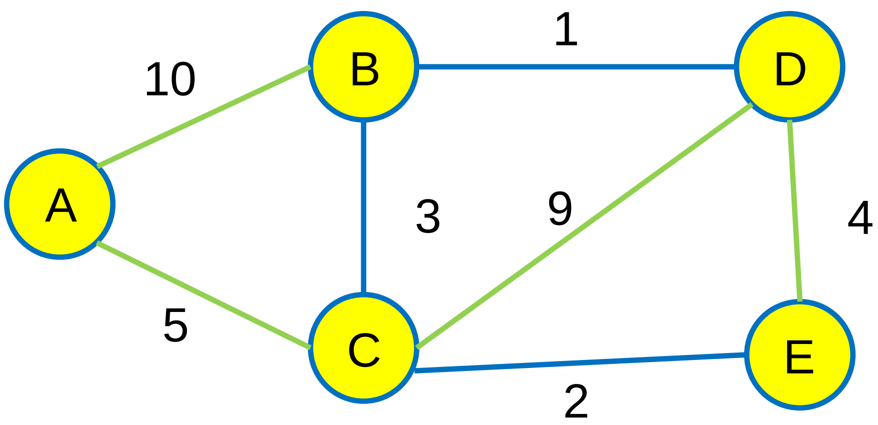
2 and 3 both have 2 members each
so choose either one to join another
join 3 to 2
need to go in Group 3 which has 2 in this case (bouneded by O(n))
for each to assign to Group 2 (bouneded by O(n))

A	B	C	D	E
1	2	2	2	2

Map Array

In the same group don't add

- Union-Find with sets



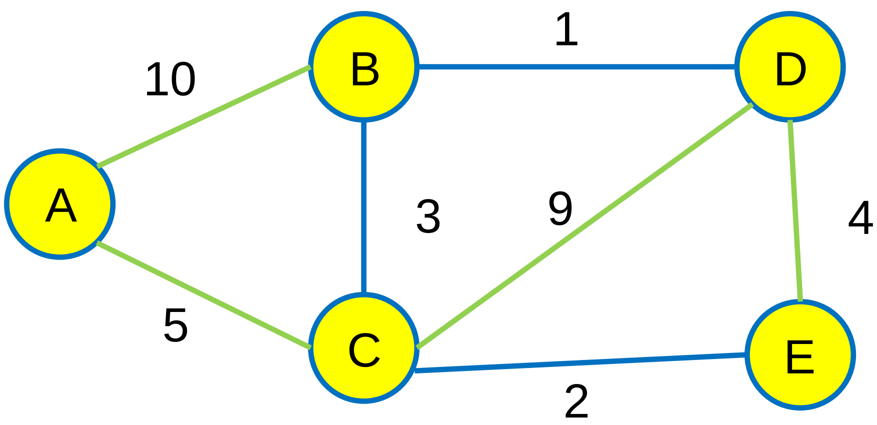
1	2	3	4	5
A	B,D,C,E			

Set Array

A	B	C	D	E
1	2	2	2	2

Map Array

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



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

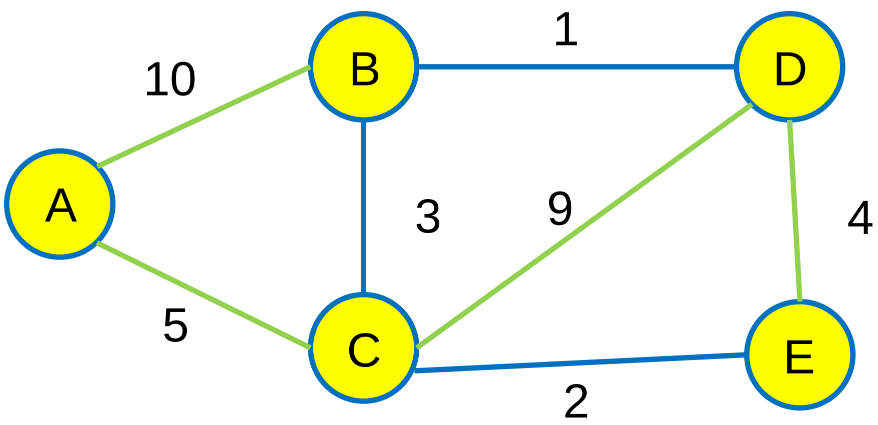
Set Array

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

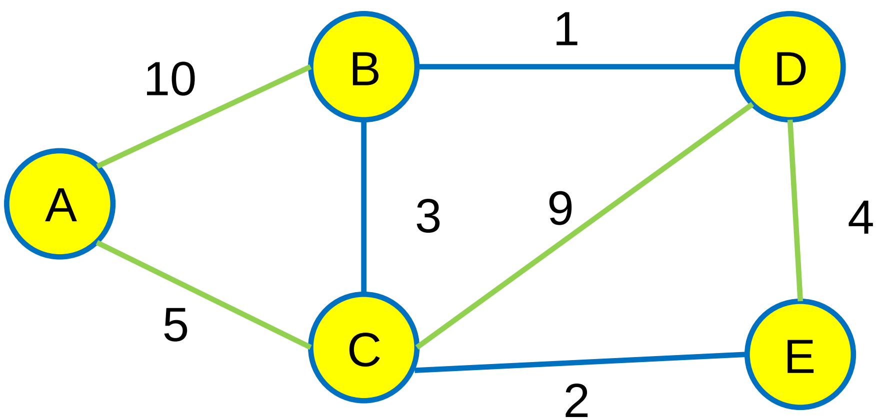
A	B	C	D	E
1	2	2	2	2

Map Array

O(X)
# of vertices

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…



groups

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 <u,v,w> 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

perform the same function over and over again
compose as a series of operations, complexity of this

best … worst
1 … O(V/2)

find is fast, union is slow
amortised complexity of the series operations of find() and union()

amortised is the average complexity for the series of
find(0 and union()

- 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 <u,v,w> 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

For each edge

  - Thus, known as Union-Find

at the begining, every one is forever alone, average all merging complexity (amortised)

  - Complexity? O(EV) but this is O(E log V) amortized : O(E log V)
    (worst case merge 2 same size) best case, another only 1 member

best; 1 ….. Worst; V/2

Questions?

Union-Find

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

Kruskal's

Union-Find

- 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

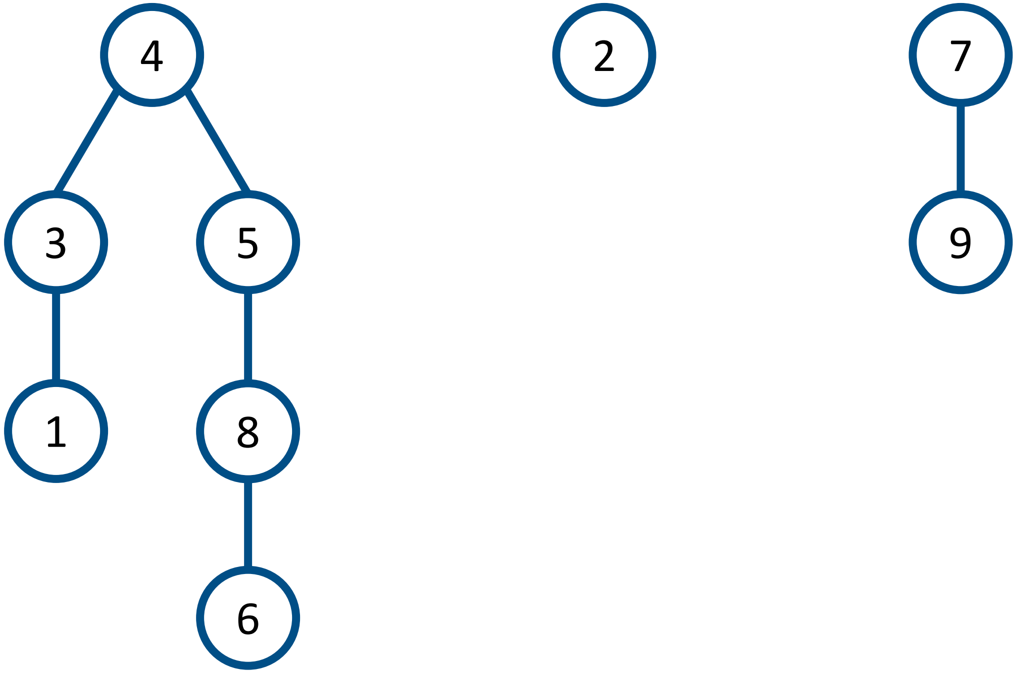
- Union by size
- Union by height/ rank

- 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

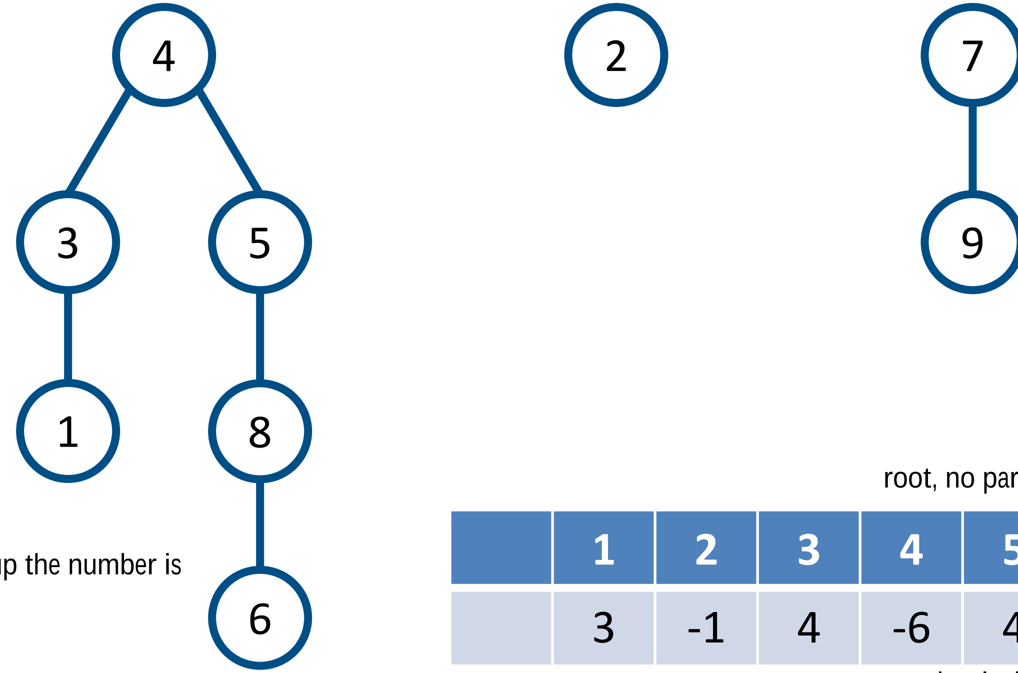
- Union by size
- Union by height/ rank

- 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


4
2
7
3
5
9
1
8
6

Kruskal's

Union-Find



when asking which group the number is
ask their parent

root, no parent,   -2 + (-6) = 8

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

parent list
parent list of 3: 4

just know parent

number inside the tree the number is in

no parent: negative value

6 is the size of the tree with root node 4
the tree has 6 nodes
4 is Group 4 (name after root node)

merge = O(1)
when merging (Union), just merge root and the rest is followed
parent 4: -6 -> 7, parent of 7: -2 -> -8

- Why such an implementation?

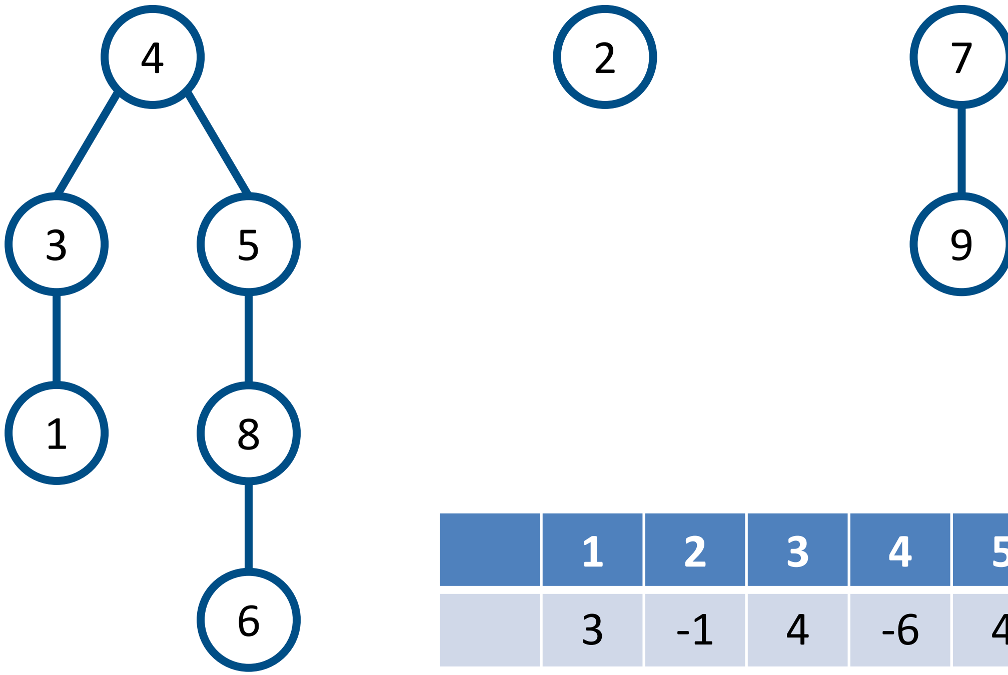
- Why such an implementation?
  - On find(u)
    - Loop till we reach the root of u (having negative number)

- 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…

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

MONASH University

- 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 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

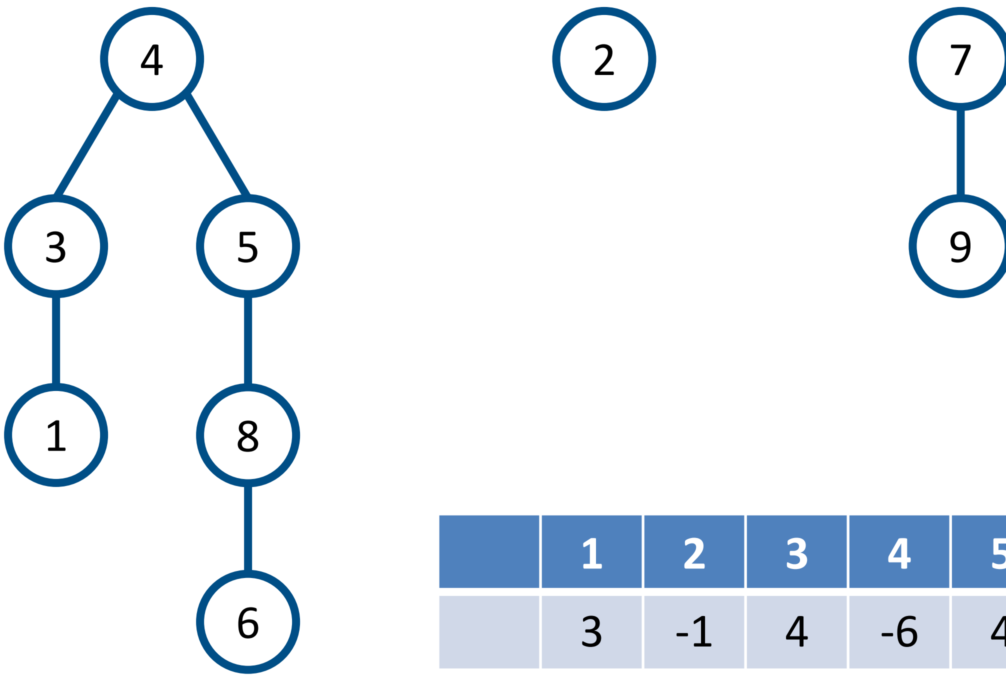
- Why such an implementation?
  - On find(u)
    - Loop till we reach the root of u (having negative number)
  - On find(v)
    - Loop till we reach the root of v (having negative number)
  - If both u and v have the same root…
    - They are in the same team/ set/ tree
    - We can't perform union(u,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

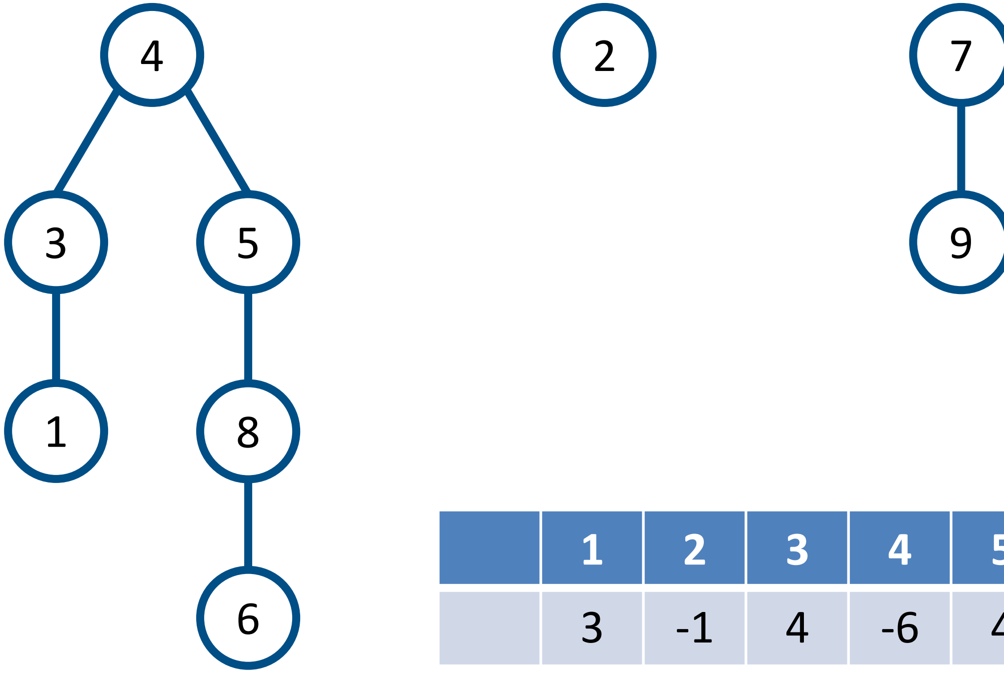
Union-Find



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

– Union(3,8)

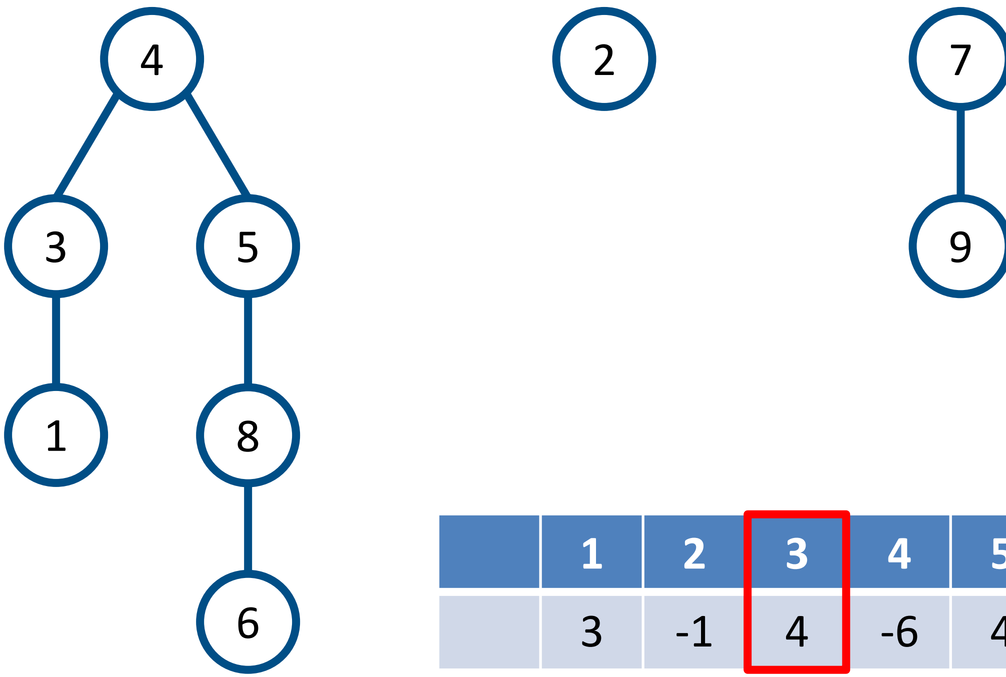
Kruskal's
Union-Find



	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)

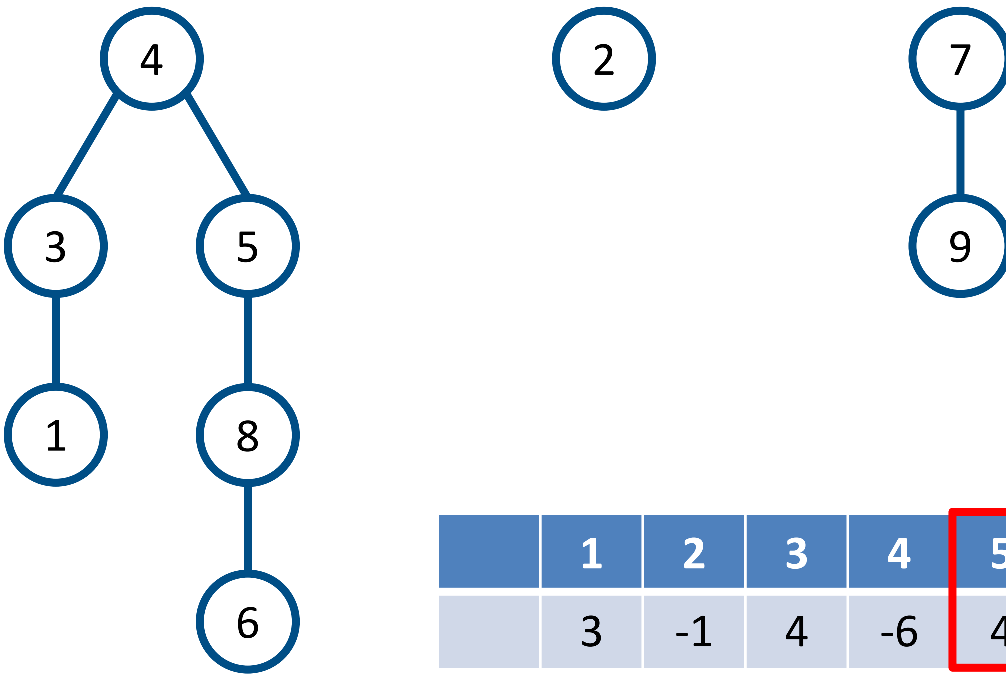
Kruskal's
Union-Find



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

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

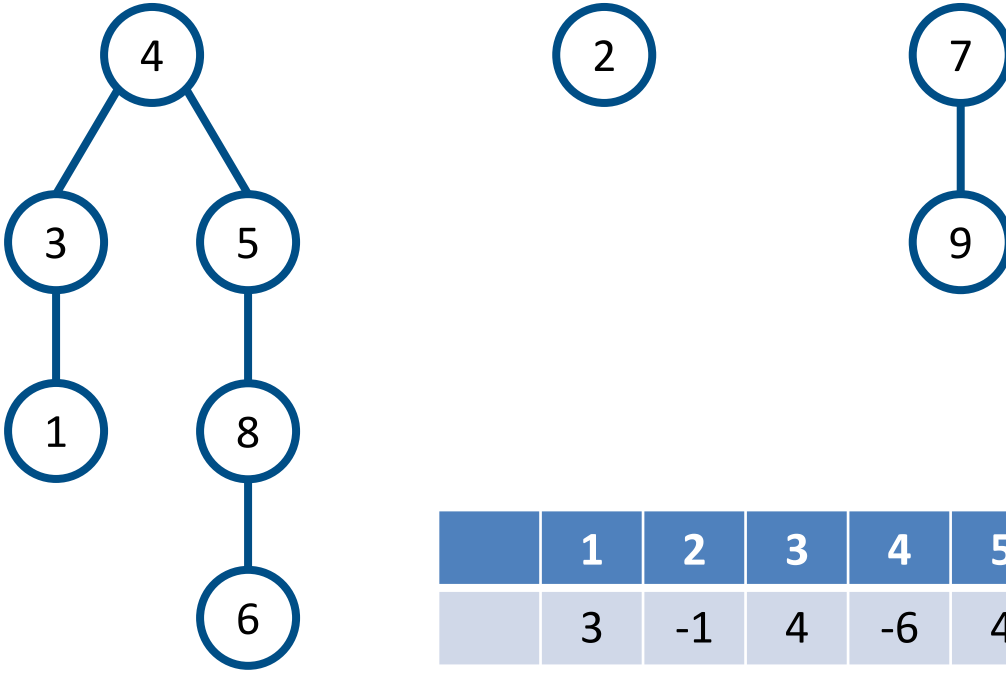
Kruskal's
Union-Find



	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



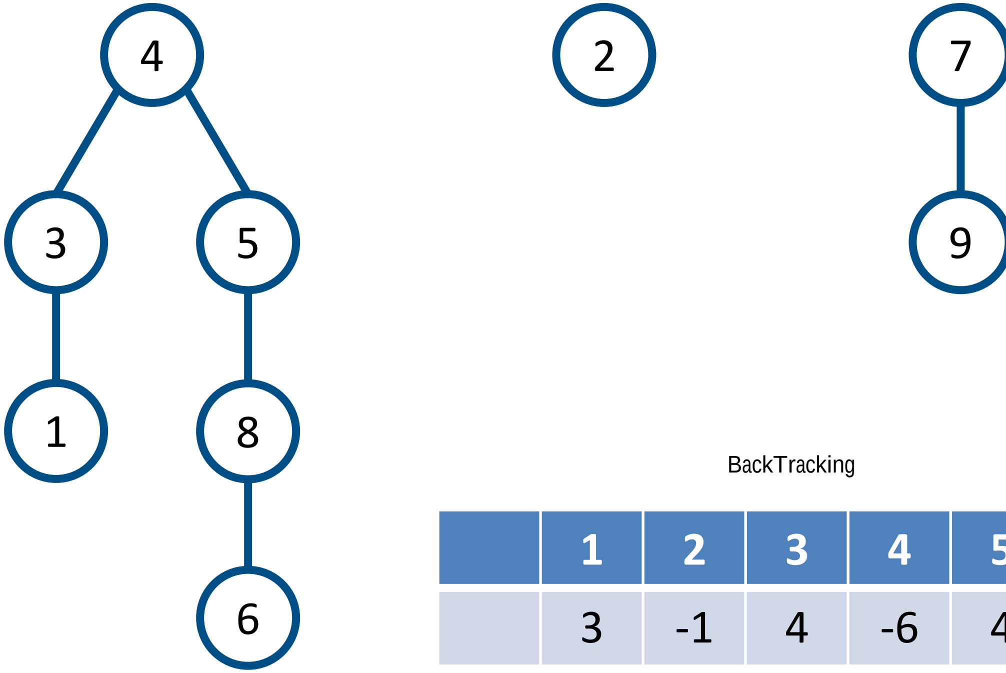
	1	2	3	4	5	6	7	8	9
	3	-1	4	-6	4	8	-2	5	7

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

Questions?

Kruskal's
Union-Find

when merging, the ax at the top of heap and eventually get a minimum spanning tree
then should not use cz since already minimum spanning tree



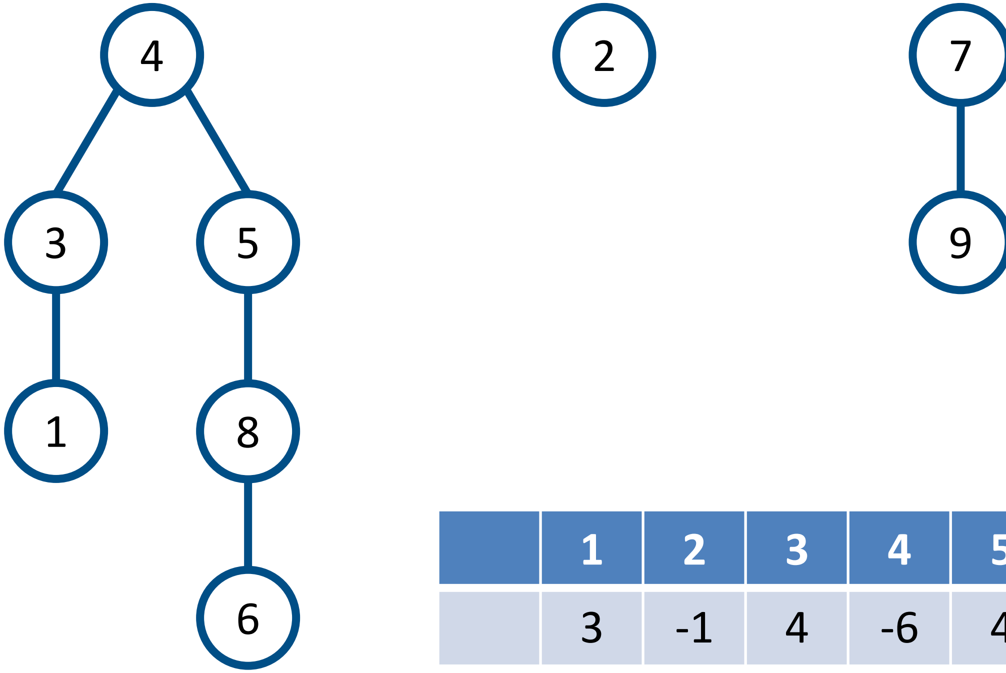
BackTracking

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

– Union(9,8)

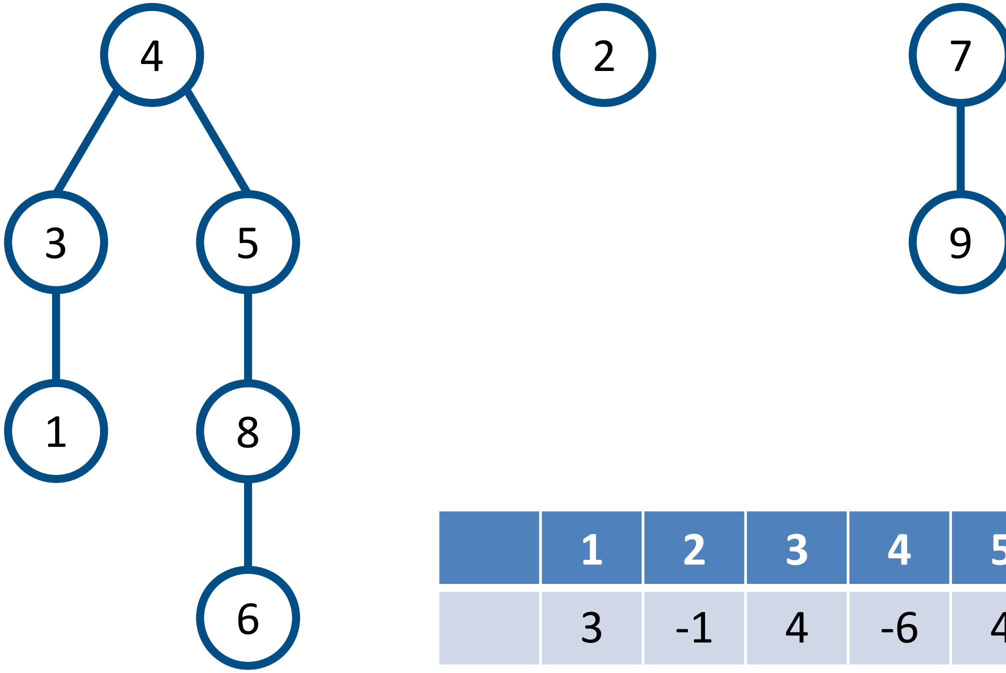
Kruskal's

Union-Find



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

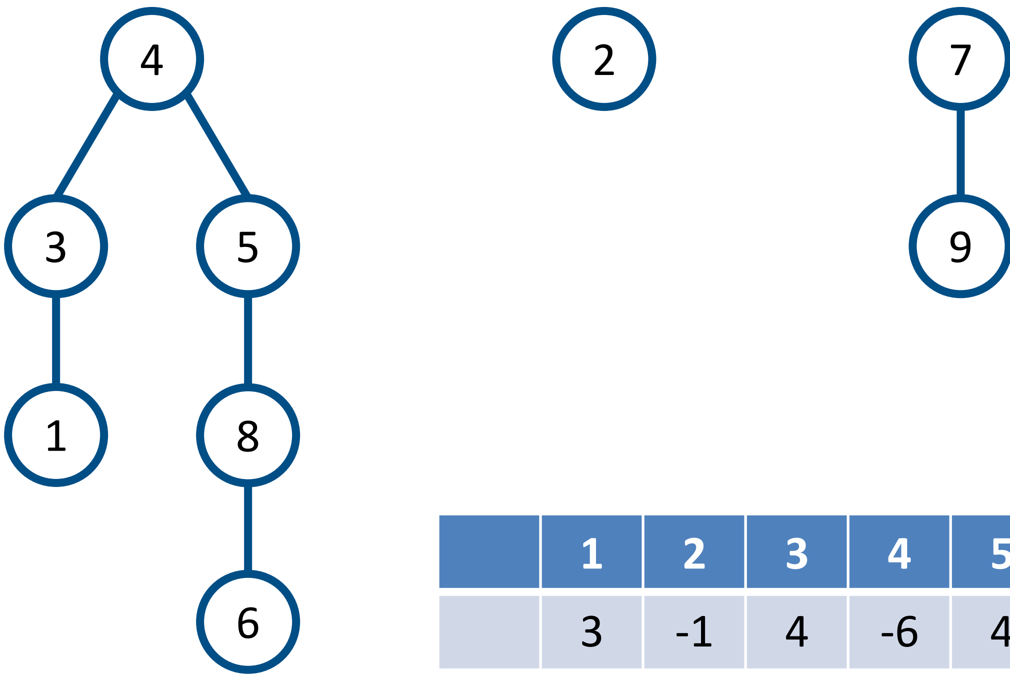
- Union(9,8)
  - Find(9)



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

– Union(9,8)
  - Find(9) -> 7

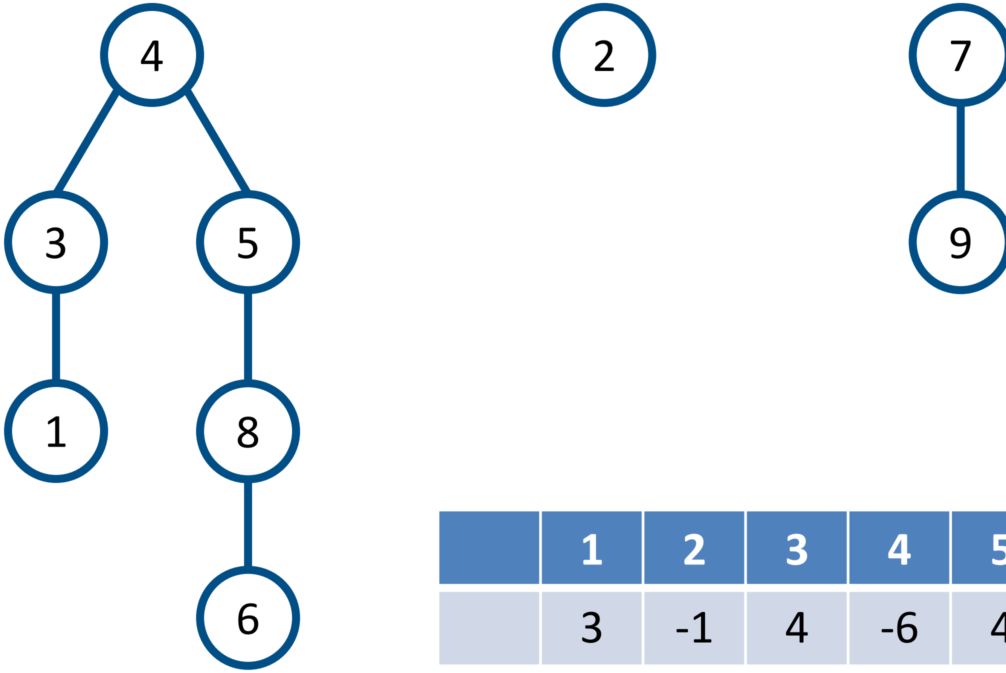
Kruskal's
Union-Find



	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)

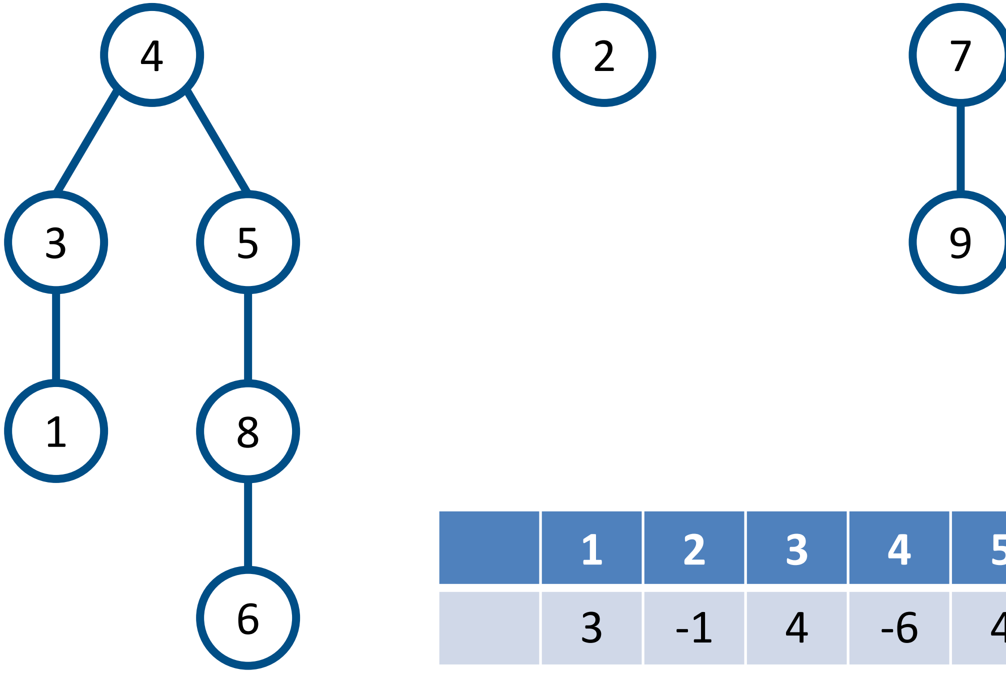
Kruskal's
Union-Find



	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

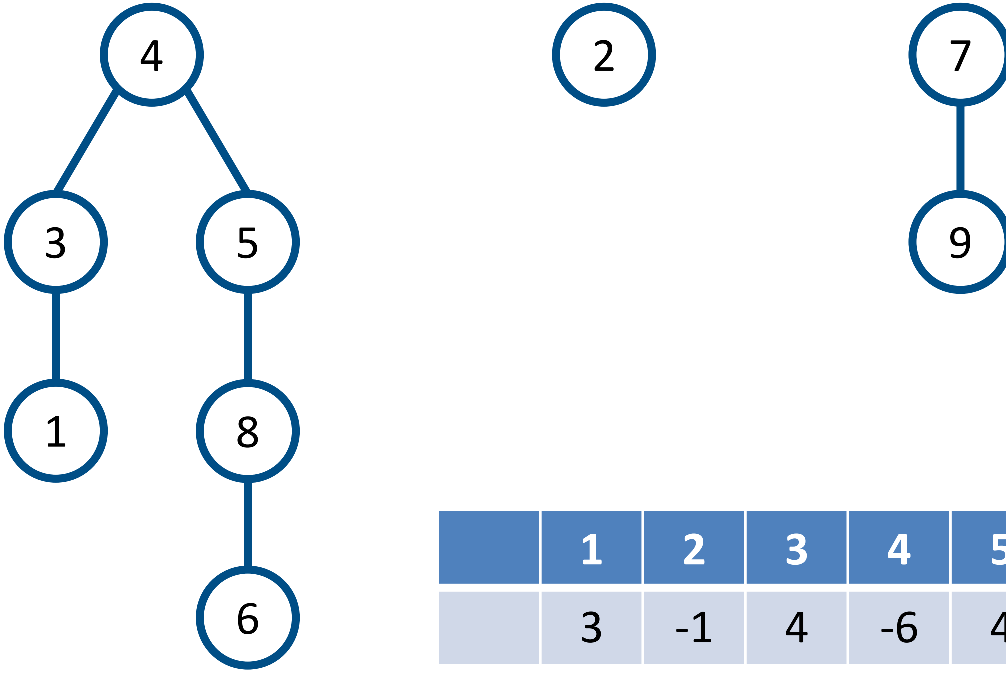
Kruskal's
Union-Find



	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

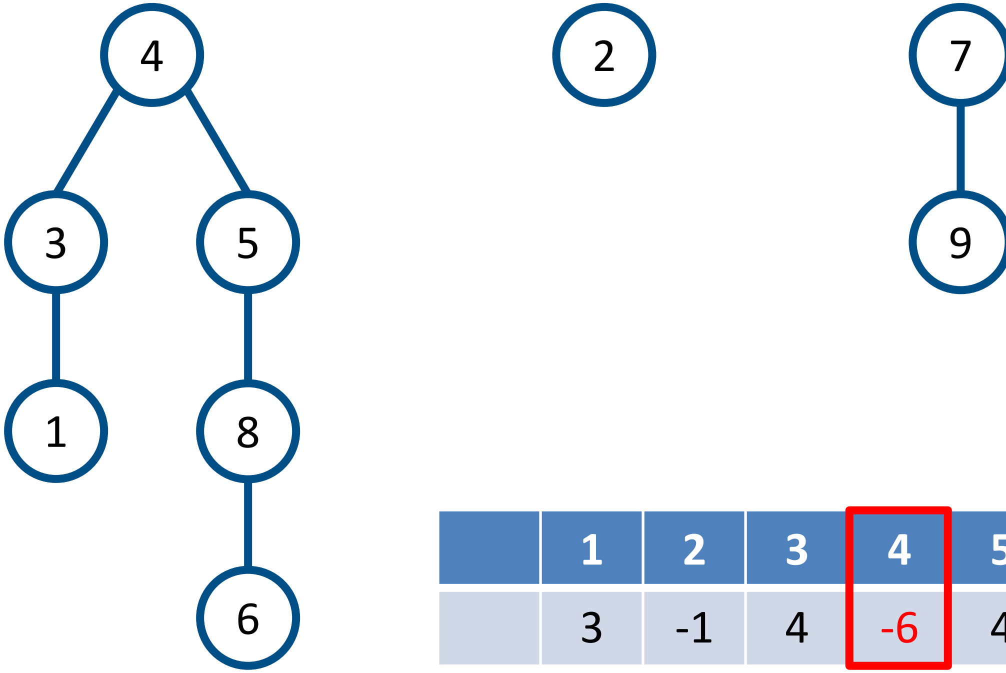
Kruskal's
Union-Find



	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,

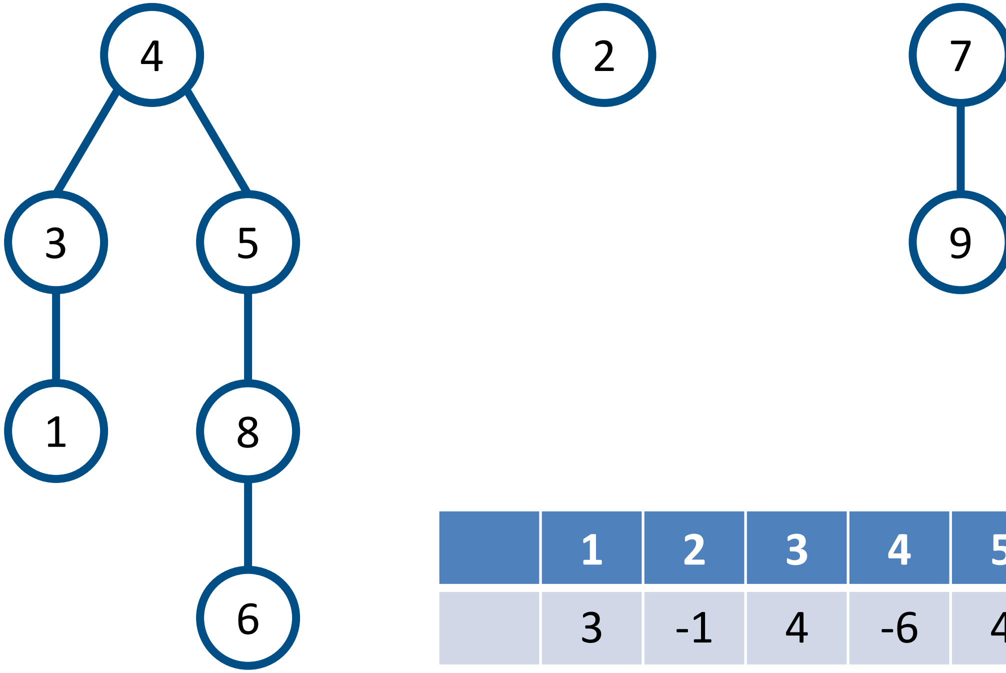
Kruskal's
Union-Find



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

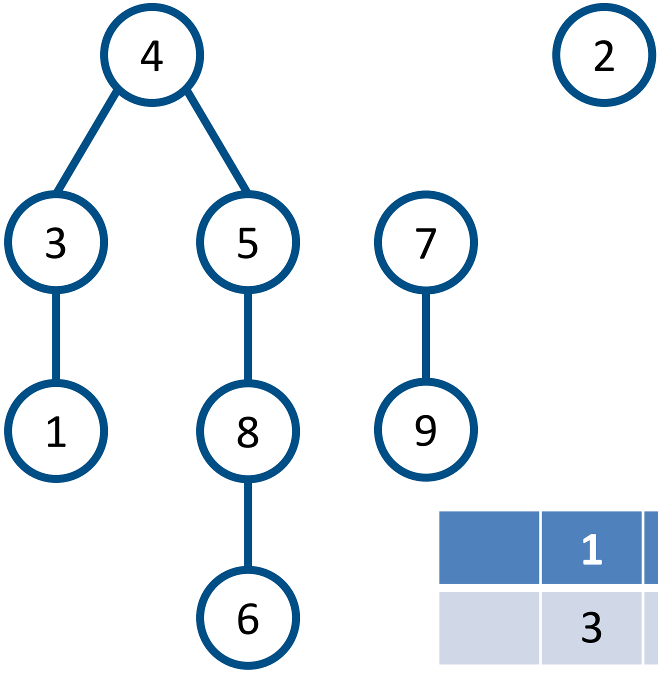
- Union(9,8), different tree so we can perform union
  - Find(9) -> 7, size of 2
  - Find(8) -> 4, size of 6

Kruskal's
Union-Find



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

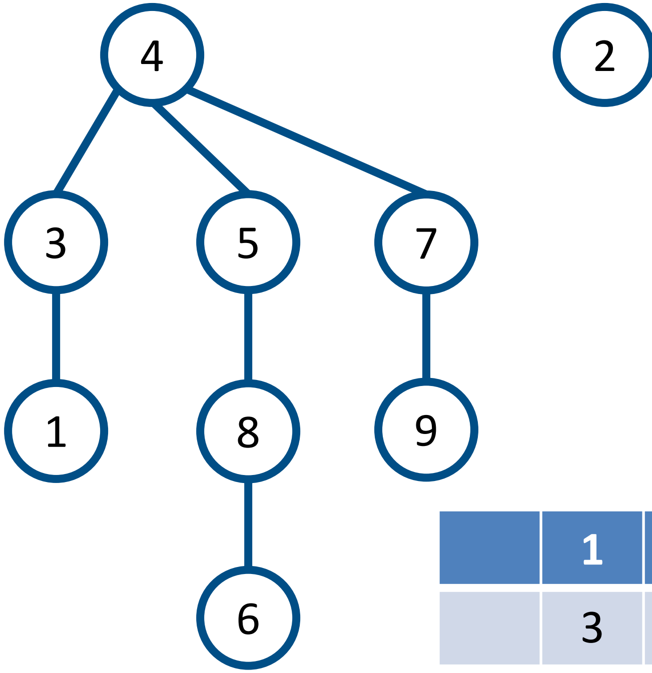
– Union(9,8), different tree so we can perform union
  - Find(9) -> 7, size of 2, smaller tree so merge to bigger tree
  - Find(8) -> 4, size of 6



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

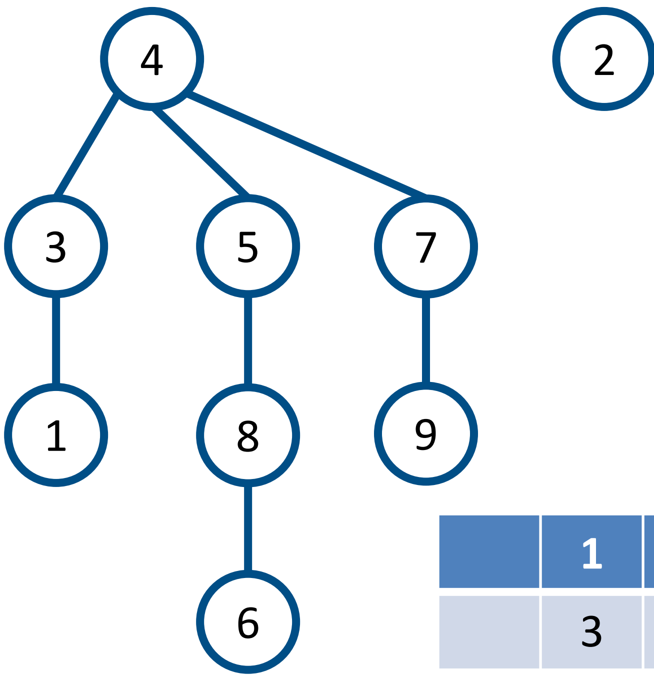
- Union(9,8), different tree so we can perform union
  - Find(9) -> 7, size of 2, smaller tree so merge to bigger tree
  - Find(8) -> 4, size of 6

Kruskal's
Union-Find



	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



	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?

Prim's and Kruskal's

Does it work?

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

if both using prim's algorithm which give ax at this time, give cz
they all can be alright as long as same method and the ax and cz have the same weights (doing the same thing, both can be right or wrong)

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

Prim's and Kruskal's

they don't form cycles

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?

Dija form cycle so does not work

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!

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?

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

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

minimum spanning tree

Questions?

Thank You