Faculty of Information Technology, Monash University

FIT2004: Algorithms and Data Structures

Week 10: Minimum Spanning Trees

These slides are prepared by M. A. Cheema and are based on the material developed by Arun Konagurthu and Lloyd Allison.

Announcements

- Assignment 4 released
  - Due:31-May-2019, 23:55:00
- Start preparing for the final exam
  - Listen to the lectures (or read slides)
  - Read Lecture Notes
  - Solve tutorial questions
  - Most importantly, do not hesitate to seek help

Recommended reading

- Lecture Notes: Chapters 14 and 15
- Cormen et al. Introduction to Algorithms.
  - Chapter 23, Pages 624-638
- http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Graph/Undirected/
- http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Graph/DAG/

Outline

1. Introduction
2. Prim's Algorithm
3. Kruskal's Algorithm

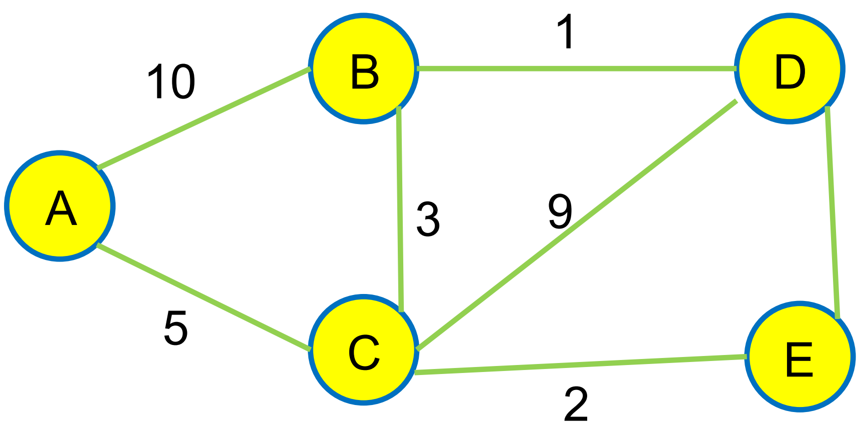
What is a Spanning Tree

Tree:

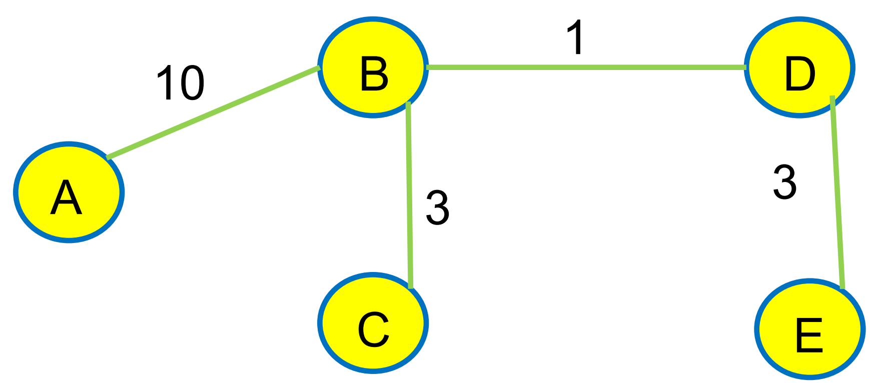
A tree is a connected undirected graph with no cycles in it.

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).

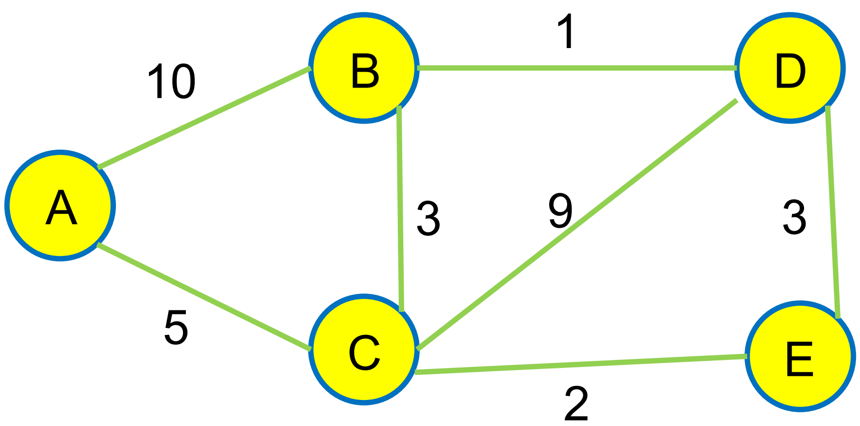


An undirected graph

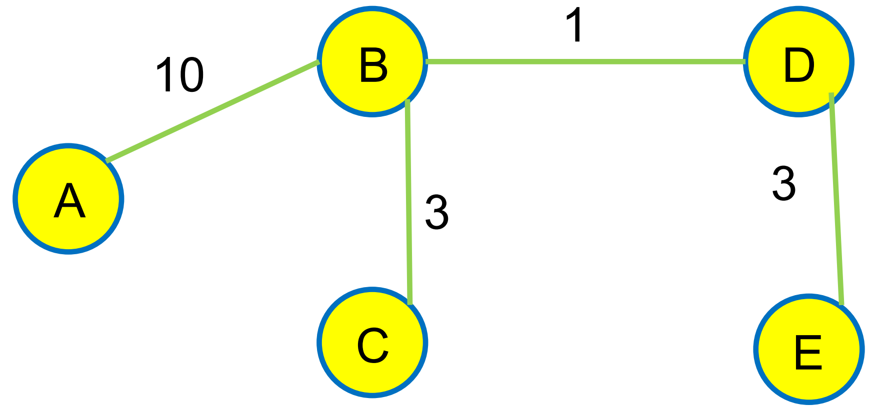


Spanning Tree

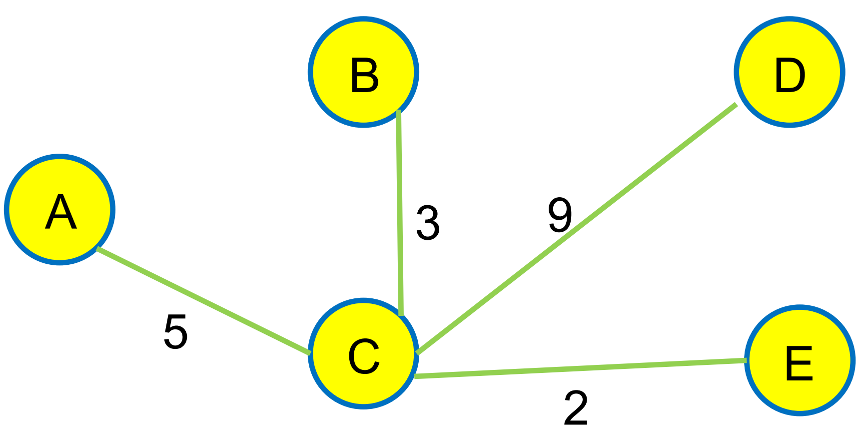
Spanning Tree Examples



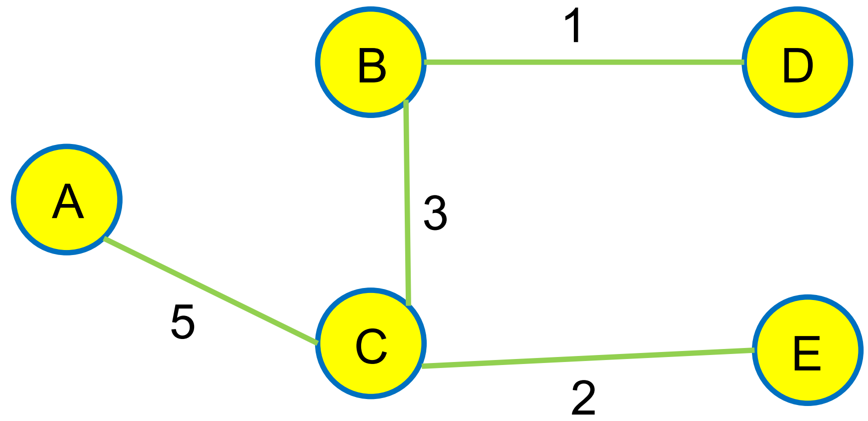
Graph



Spanning Tree 1



Spanning Tree 2



Spanning Tree 3

What is a Spanning Tree

Tree:

A tree is a connected undirected graph with no cycles in it.

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).

Is it true that a spanning tree of a connected graph G is a maximal set of edges of G that contains no cycles?

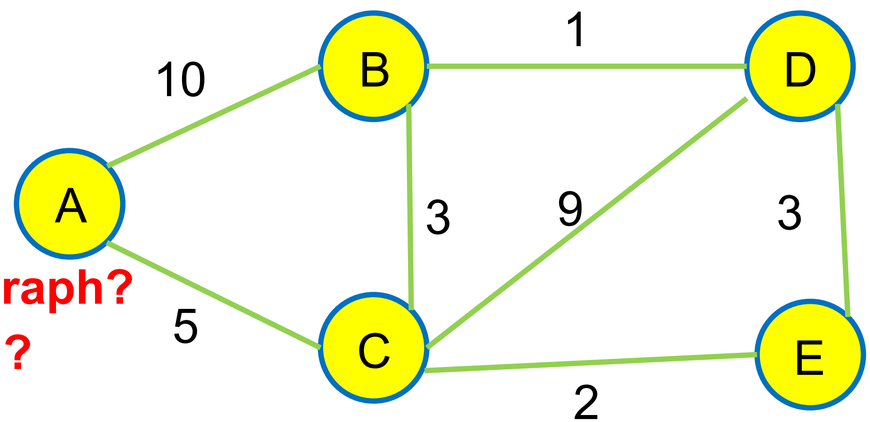
  - Yes

Is it true that a spanning tree of a connected graph G is a minimal set of edges that connect all vertices?

  - Yes

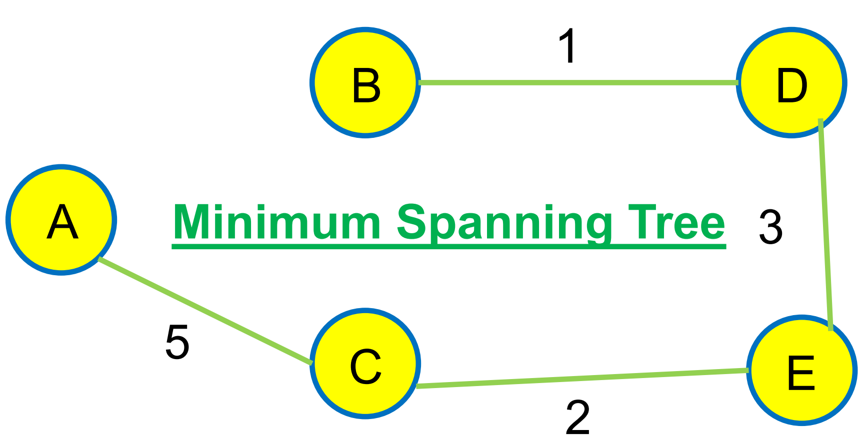
Minimum Spanning Tree (MST)

- Weight of a spanning tree is the sum of the weights of the edges in the tree.
- A Minimum spanning tree of a weighted general graph G is a tree that spans G, whose weight is minimum over all possible spanning trees for this graph.
- There may be more than one minimum spanning trees for a graph G (e.g., two or more spanning trees with the same minimum weight).

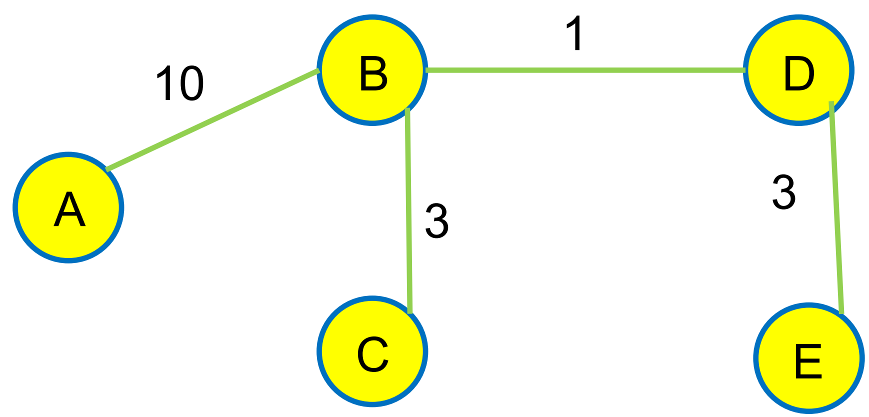


What is the weight of the MST in this graph?
How many MSTs we have in this graph?

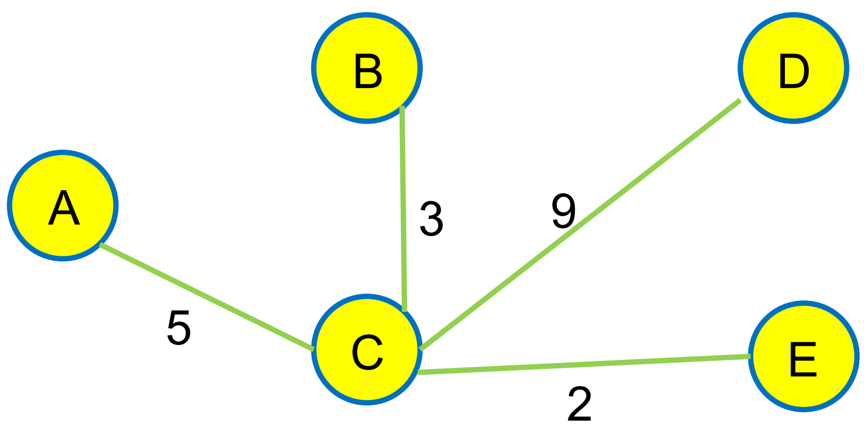
Spanning Trees and MSTs



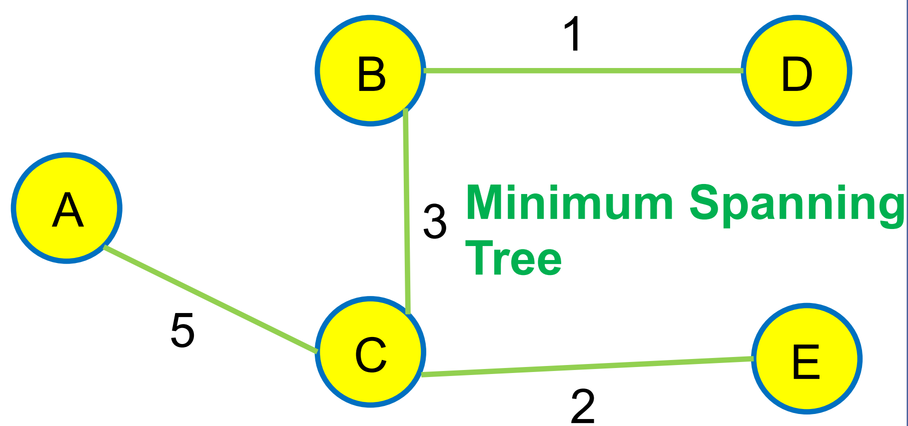
Spanning Tree 1: Weight 11



Spanning Tree 2: Weight 17



Spanning Tree 3: Weight 19



Spanning Tree 4: Weight 11

MST Algorithms

Let M denote the MST we are constructing, initialized to be empty

An edge e is said to be safe if {M U e} is a subset of a MST

General Strategy:

- M = null
- **while** M can be grown safely:
- find an edge e=<x,y> along which M **is** safe to grow
- M = {M} union {<x,y>}
- **return** M

We will study two **greedy** algorithms that follow this strategy

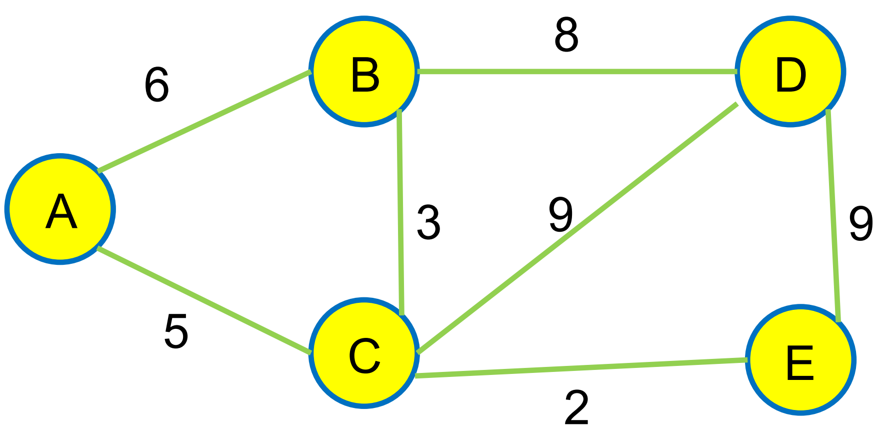
- Prim's Algorithm (very similar to Dijkstra's Algorithm)
  - In fact, Dijkstra published his algorithm for both MST and shortest path in the same paper (1959)
  - The algorithm is also often called Prim-Dijkstra Algorithm
  - M is always a tree and, in each iteration, we choose the shortest edge connected to M avoiding cycles
  - Time complexity: O(E log V)
- Kruskal's Algorithm
  - We process edges in ascending order of edge weights and M is a forest (i.e., a set of trees)
  - Time complexity: O(E log V)

Outline

1. Introduction
2. Prim’s Algorithm
3. Kruskal’s Algorithm

Prim’s Algorithm: Overview

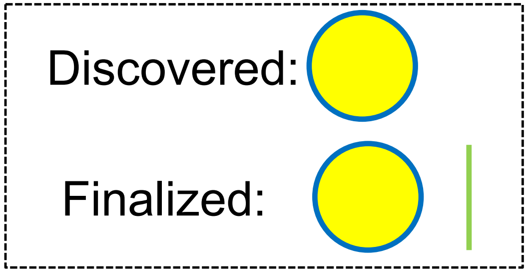
- Start by picking any vertex v to be the root of the tree M.
- While the tree M does not contain **all** vertices in the graph
  - Find shortest edge e connected to the growing subtree M that does not create a cycle
  - add e to the tree M

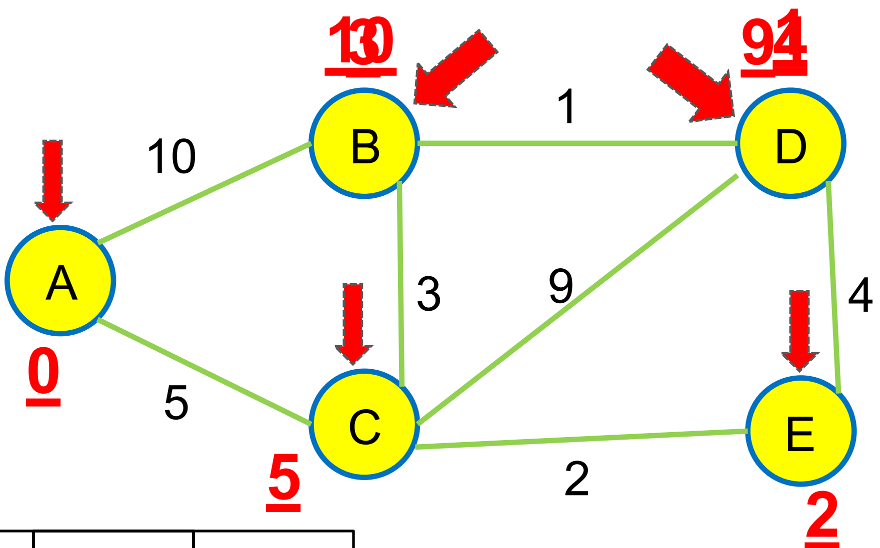


Prim's Algorithm

Differences with Dijkstra's are shown in red

- Initialize a list called Discovered and insert a random node A in it with distance 0
- While Discovered is not empty
  - Get the vertex v from the Discovered List with smallest weight
  - For each outgoing edge (v, u, w) of v
    - If u is not in Discovered/Finalized
      - Insert u in Discovered with weight w and edge v →u
    - Else If u.weight > w
      - If u is not finalized, update the weight of u in Discovered to w and edge to v→u
  - Move v from Discovered to Finalized along with its corresponding edge





Discovered:

A, 0	C→B,3 / A→B,10	A→C,5	B→D,1 / C→D,9	C→E,2

Finalized (in MST):

A	A→C	C→E	B→C	B→D

Prim's Algorithm

```
# Initializations
Discovered = random(V) # Start by choosing any vertex randomly
Finalized = null; # Initially the MST is null
while Discovered not_empty: # loops |V| times
#INV: Finalized is a (growing) subset of a minimum spanning tree
    v  = EXTRACT_MIN(Discovered) # get vertex v from Discovered with minimum weight
    Finalized = Finalized + <x,v> # <x,v> is the edge corresponding to the weight of v
    for each adjacent edge of (v,u,w) adjacent to v:
        if u is not Discovered/Finalized:
            insert u in Discovered with weight w and edge <v,u>
        else:
            if u is not Finalized:
                if u.weight > w:
                    update weight of u to w and edge to <v,u>
Return Finalized
```

Time Complexity?

Prim's Algorithm: Complexity

It is very similar to Dijkstra's Algorithm and its complexity is the same as Dijkstra's Algorithm

- $O(V \log V + E \log V)$ if min-heap is used

Since the input graph G is connected, $E \geq V-1$. Hence, the complexity can be simplified to $O(E \log V)$.

Proof of correctness

```
# Initializations
Discovered = random(V) # Start by choosing any vertex randomly
Finalized = null; # Initially the MST is null
while Discovered not_empty: # loops |V| times
#INV: Finalized is a (growing) subset of a minimum spanning tree
    v  = EXTRACT_MIN(Discovered) # get vertex v from Discovered with minimum weight
    Finalized = Finalized + <x,v> # <x,v> is the edge corresponding to the weight of v
    for each adjacent edge of (v,u,w) adjacent to v:
        if u is not Discovered/Finalized:
            insert u in Discovered with weight w and edge <v,u>
        else:
            if u is not Finalized:
                if u.weight > w:
                    update weight of u to w and edge to <v,u>
Return Finalized
```

Prim's Algorithm: Correctness

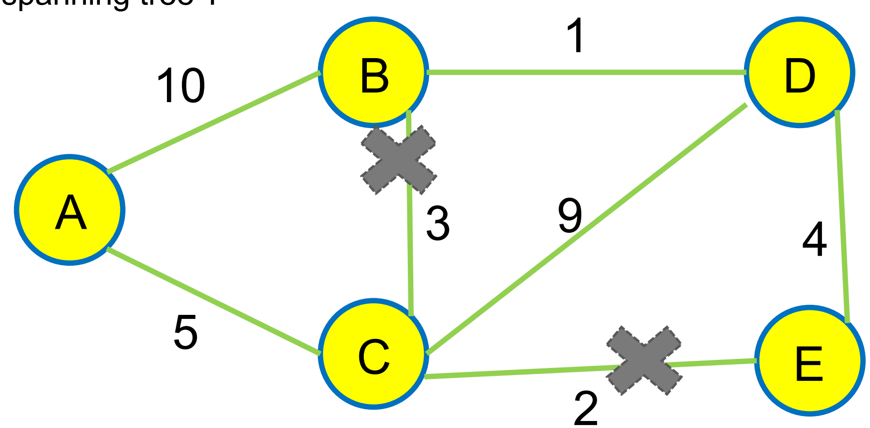
#INV: Finalized is a (growing) subset of <u>a</u> minimum spanning tree

Base Case:

- The invariance is true initially when Finalized is empty

Inductive step:

- Assume Finalized is currently a subset of a MST. We show that after Prim's algorithm adds an edge, invariance still holds
- Suppose the algorithm chooses a vertex E and an edge <C,E> having minimum weight w
- Assume {Finalized Union <C,E>} is **<u>not</u>** a subset of **<u>any</u>** minimum spanning tree. We show that this assumption is wrong.
- Let M be a minimum spanning tree that contains Finalized **<u>but</u>** excludes <C,E>.
  - E must be connected to Finalized in M (because M is a spanning tree). Since M does not contain <C,E>, there must be a path that connects Finalized (e.g., red vertices) with E (e.g., see blue edges).
  - Let <C,B> be the first edge on the path that connects Finalized to E.
- If we remove <C,B> from M and add <C,E> we will still get a spanning tree. Let this spanning tree be called T.
- Since the weight of <C,E> is smaller or equal to the weight of <C,B>, the weight of T is smaller than or equal to M. Hence, either M is not a minimum spanning tree or T is also a minimum spanning tree.
- Hence, Finalized after adding <C,E> is a subset of a minimum spanning tree T
  - i.e., the invariance holds after adding the edge <C,E>



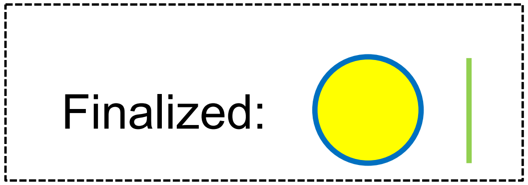
Outline

1. Introduction
2. Prim's Algorithm
3. Kruskal's Algorithm

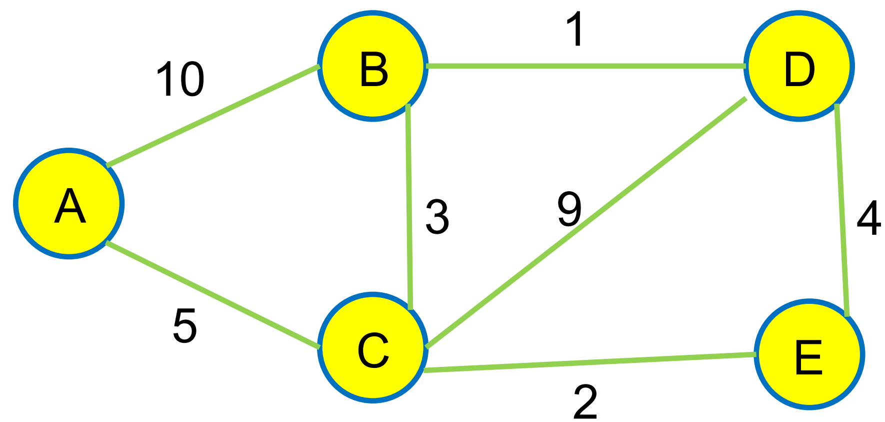
Kruskal's Algorithm

It is also a greedy algorithm like Prim's

- Sort the edges in ascending order of weights
- For each edge (v, u) in ascending order
  - If adding (v,u) does not create a cycle in Finalized
    - Add (v,u) in Finalized
- Return Finalized



How to determine if the edge will create a cycle???



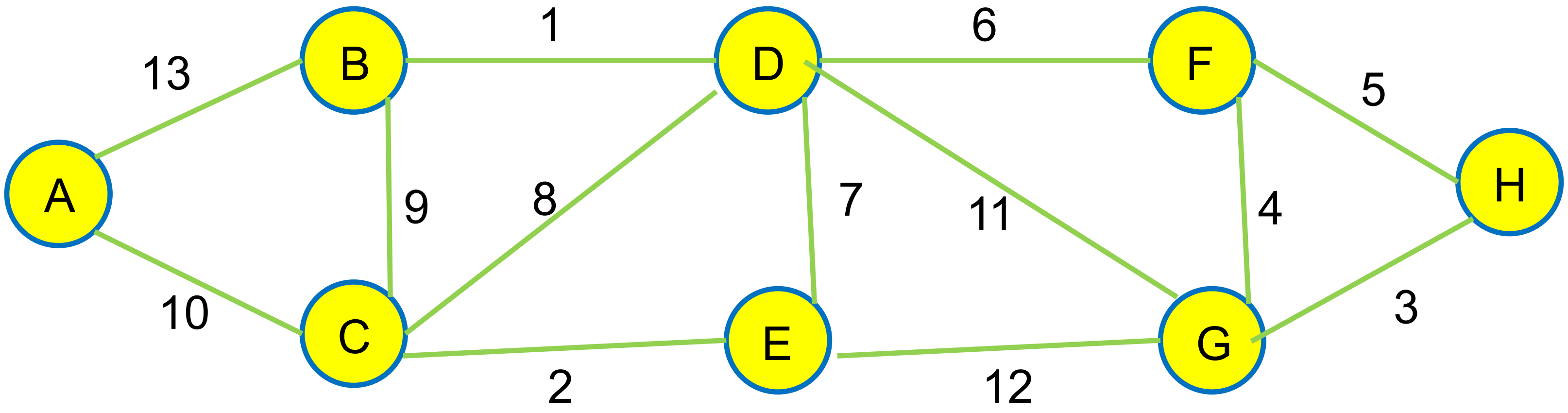
Sorted Edges:

B→D,1	C→E,2	C→B,3	E→D,4	A→C,5	C→D,9	A→B,10

Finalized (in MST):

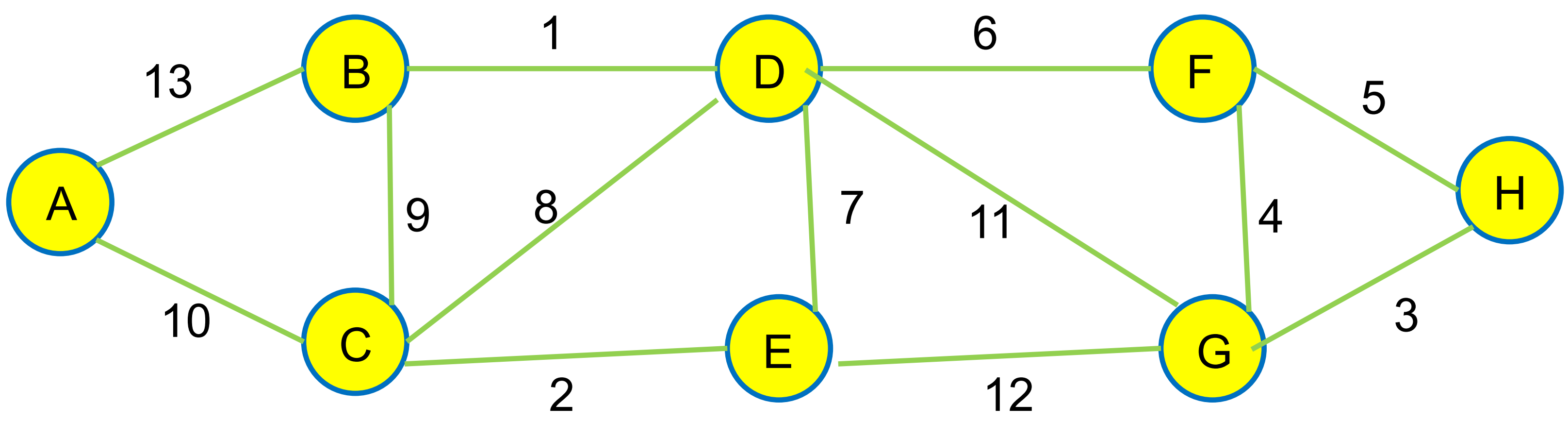
B→D	C→E	C→B	A→C

Kruskal's Algorithm



Each connected component is considered a set.
Is it true that an edge (u,v) creates a cycle if and only if both u and v belong to the same connected component (i.e., set)?

Kruskal's Algorithm




```
# Initializations
for each vertex x in V:
        create a new set containing only x
sort edges in increasing order by weight
Finalized  = null # stores edges of MST
for each edge <u,v,w> in sorted order:
# INV: Finalized is a subset of a minimal spanning tree
        if SET_ID(u) != SET_ID(v): # the edge will not create a cycle
                Finalized = Finalized + {<u,v,w>} # add the edge to MST
                UNION_SETS(SET_ID(u), SET_ID(v))
return Finalized
```

How to implement SET_ID and UNION_SETS functions efficiently?

Union-Find Data Structure

- For each set S_i, maintain the vertices in it as a linked list.
- Create an array (called map_array) that will record, for each vertex, the set that it belongs to. E.g.,
  - If vertex 3 belongs to set S4, then map_array[3] = S4

SET_ID(u)

- Return map_array[u] # Cost O(1)

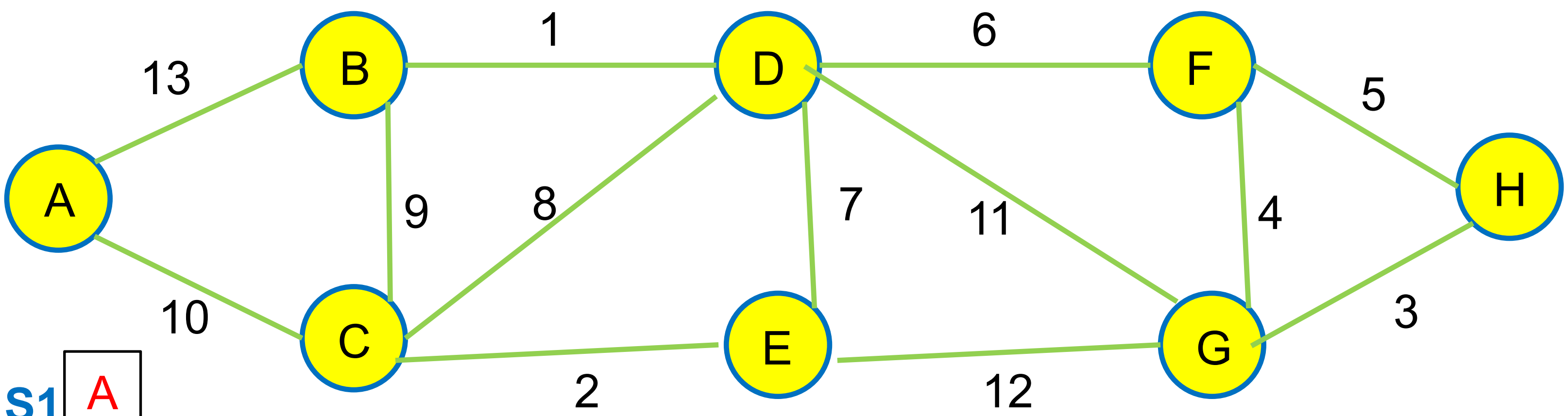
UNION_SETS(S_i, S_j) # Let S_i be the smaller set. We will merge S_i into S_j

- For each vertex v in S_i
  - map_array[v] = S_j # Update the set of v in map_array
- Append S_i to the linked list of S_j

Time complexity of UNION_SETS?

- O(x) where x is the number of elements in the smaller set.

Illustration of Kruskal's Algorithm



Sets (i.e., connected components)

S1	A							
S2	B							
S3	C							
S4	D	B						
S5	E	C						
S6	F							
S7	G	H	F	D	B	E	C	A
S8	H							

Finalized:

B→D	C→E	G→H	F→G	D→F	D→E	A→C

map_array:

A	B	C	D	E	F	G	H
S1 / S7	S2 / S4 / S7	S3 / S5 / S7	S4 / S7	S5 / S7	S6 / S7	S7	S8 / S7

Kruskal's Algorithm: Complexity

```
# Initializations
for each vertex x in V:
        create a new set containing only x
sort edges in increasing order by weight
Finalized  = null # stores edges of MST
for each edge <u,v,w> in sorted order:
# INV: Finalized is a subset of a minimal spanning tree
        if SET_ID(u) != SET_ID(v): # the edge will not create a cycle
                Finalized = Finalized + {<u,v,w>} # add the edge to MST
                UNION_SETS(SET_ID(u), SET_ID(v))
return Finalized
```

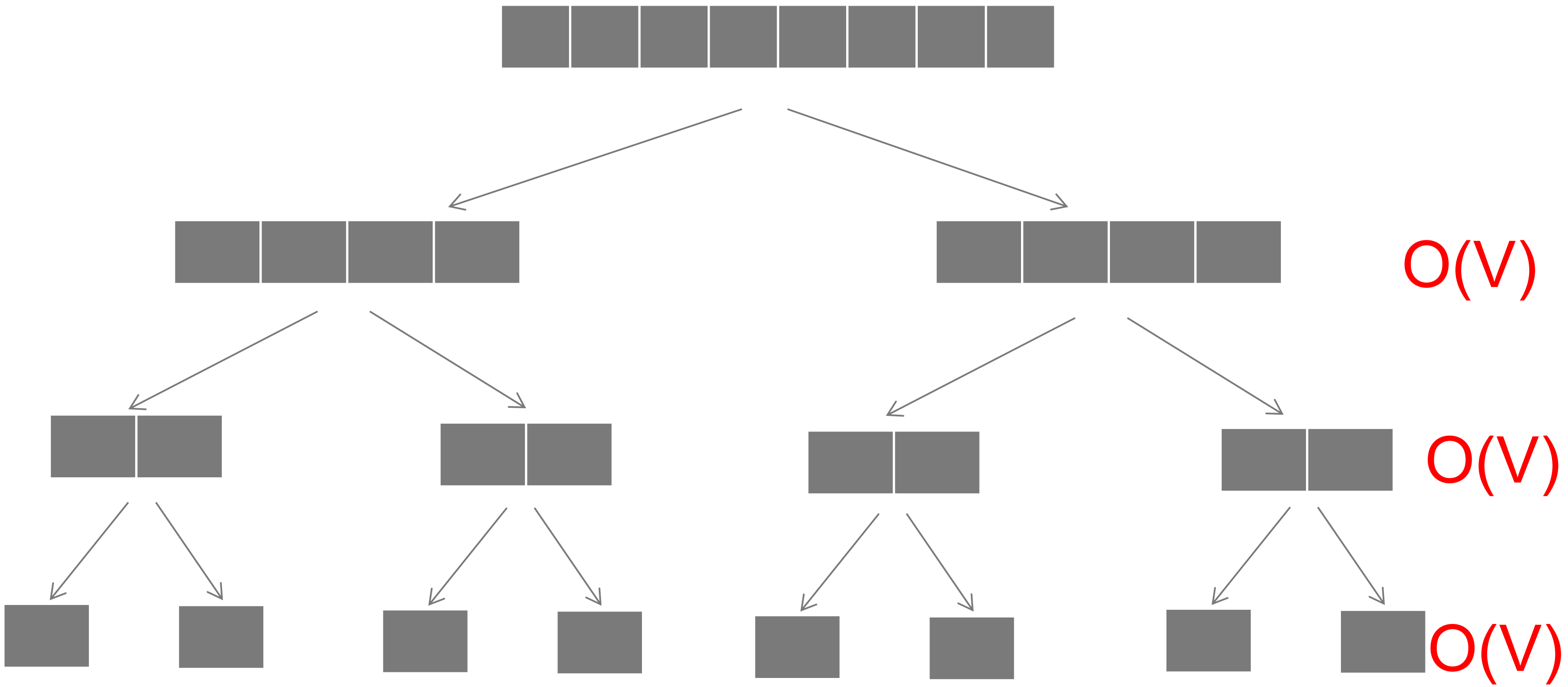
But this is not tight as we assumed the cost of UNION_SETS to be O(V) for each call leading to overall cost of O(EV). A closer look reveals that the total cost of UNION_SETS is O(V log V)

Time Complexity:

- Initialization: O(V)
- Sorting edges: O(E log E)
  - E log E <= E log V^2 = 2 E log V → O(E log V)
- For loop executes O(E) times
  - SET_ID() takes O(1)
  - UNION_SET() takes O(x) where x is the smaller set (in worst case O(V))
- Total cost: O(EV)

Complexity of UNION_SETS

- The cost of UNION_SETS(S1,S2) is O(x) where x is the size of the smaller set.
- What is the worst case for UNION_SETS(S1,S2)?
  - The worst case for UNION_SETS(S1,S2) is when both sets are of equal size.



Height: O(log V)
Overall Complexity: O(V log V)

Kruskal's Algorithm: Complexity

```
# Initializations
for each vertex x in V:
        create a new set containing only x
sort edges in increasing order by weight
Finalized  = null # stores edges of MST
for each edge <u,v,w> in sorted order:
# INV: Finalized is a subset of a minimal spanning tree
        if SET_ID(u) != SET_ID(v): # the edge will not create a cycle
                Finalized = Finalized + {<u,v,w>} # add the edge to MST
                UNION_SETS(SET_ID(u), SET_ID(v))
return Finalized
```

Time Complexity:

- Initialization: O(V)
- Sorting edges: O(E log E)
  - E log E = E log V^2 = 2 E log V → O(E log V)
- For loop executes O(E) times
  - SET_ID() takes O(1)
- UNION_SET() takes O(V log V) in total
- Total cost: O(E log V + V log V) → O(E log V)

Kruskal's Algorithm: Correctness

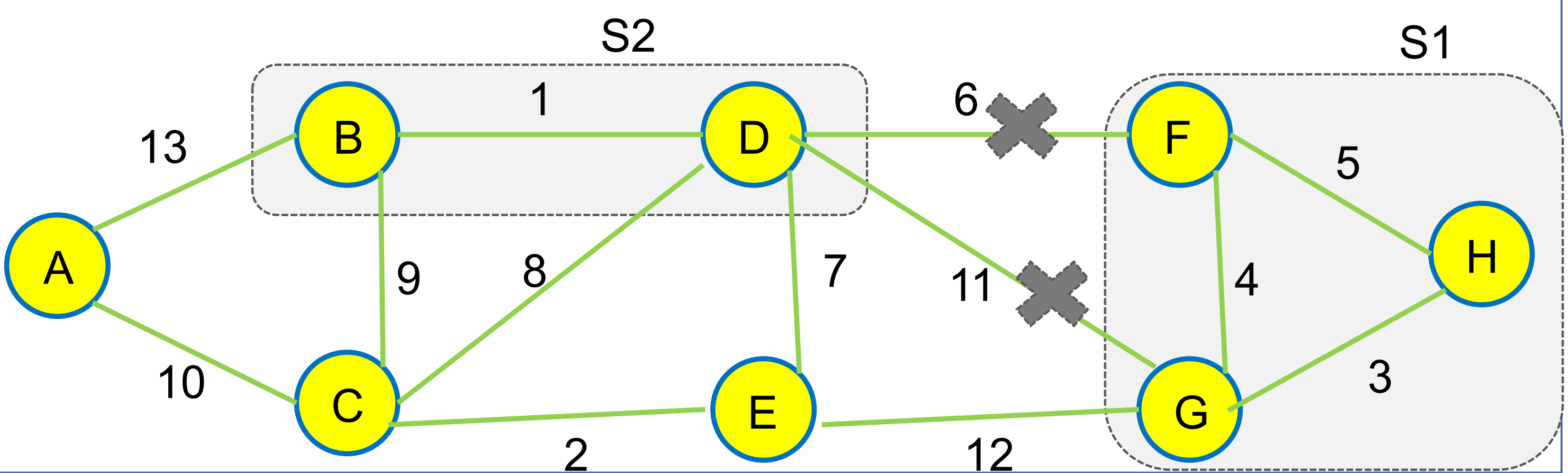
`# INV: Finalized is a subset of a minimal spanning tree`

Base Case:

- The invariance is true initially when Finalized is empty

Inductive step

- Assume that Finalized is currently a subset of a MST. We show that after Kruskal's adds an edge, it is still a subset of a MST.
- At an arbitrary step, the algorithm combines two sets (UNION_SETS) say S1 and S2 using an edge <D,F> with weight w.
- Assume that {Finalized Union <D,F>} is **not** a subset of **any** MST. We show that this cannot be true.
- Let M be a minimum spanning tree that contains Finalized but excludes <D,F> (see the tree formed by red and blue edges).
- The sets S1 and S2 must be connected by at least one edge in every spanning tree (e.g., M).
- Let <D,G> be the first edge on the path that connects S1 and S2 in the minimum spanning tree M.
- We will get a spanning tree if we add <D,F> in M and remove <D,G>. Let's call this spanning tree T.
- The weight of T is smaller or equal to M because the weight of <D,F> is smaller or equal to <D,G>.
- Hence, T is also a minimum spanning tree if M is a minimum spanning tree, i.e., the invariance holds after adding <D,F> in Finalized



Summary

Take home message

- Prim's Algorithm and Kruskal's algorithm both are greedy algorithm that correctly determine minimum spanning trees.

Things to do (this list is not exhaustive)

- Make sure you understand
  - the two algorithms especially how to implement Union-Find data structure for Kruskal's algorithm
  - the proofs of correctness for each of the two algorithms
- Start preparing for the final exam

Coming Up Next

- Network Flow