CS2040 – Data Structures and Algorithms

Lecture 14 – Connecting People – MST

axgopala@comp.nus.edu.sg

NUS National University of Singapore
School of Computing

Outline

- Minimum Spanning Tree (MST)
  - Motivating Example
  - Some Definitions

- Two Algorithms to solve MST (you have a choice!)
  - Jarnik's/Prim's (greedy algorithm with **PriorityQueue**)

  - Kruskal's (greedy algorithm, uses sorting and **UFDS**)

- https://www.visualgo.net/mst

Review – Definitions

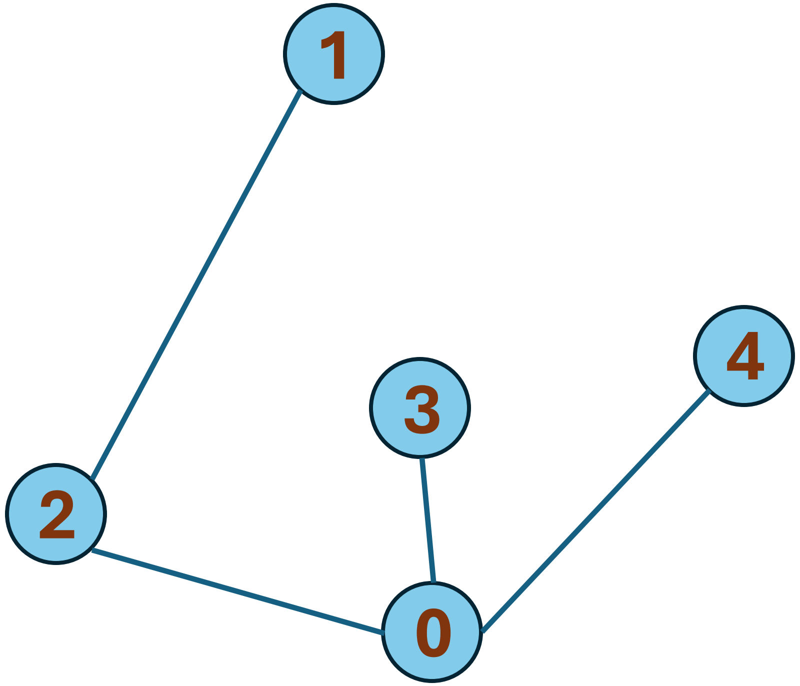
- **Tree T**
  - **T** is a **connected graph** that has **V** vertices and **V – 1** edges
  - Important: One unique path between any two pair of vertices in **T**
- **Spanning** Tree **ST** of connected graph **G**
  - **ST** is a tree that spans (covers) every vertex in **G**
  - Recall the **BFS and DFS Spanning Tree**

Live Quiz

- Is this a tree?

    A) **Yes, absolutely**

    B) You must be joking!

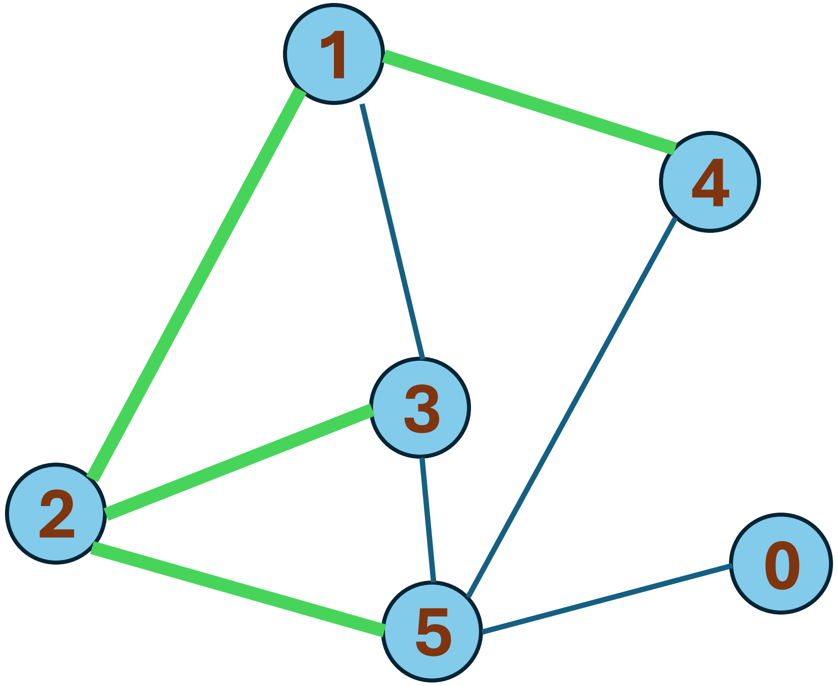


Live Quiz

- Is the tree denoted in green a spanning tree in this graph?

A) Yes ☺

B) Nope!

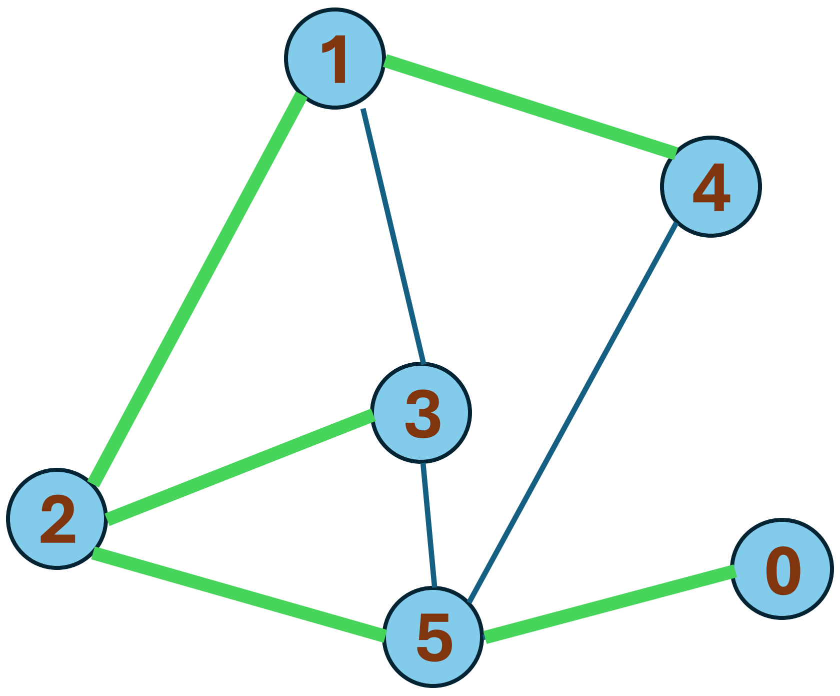


Live Quiz

- Is the tree denoted in green a spanning tree in this graph?

  A) **Now, Yes ☺**

  B) Nope!



Motivation – Project

- Want to link rural villages with roads
- The cost to build a road depends on the terrain, etc.
- Budget is limited
- How are you going to build the roads?

Definition time!

Definitions – Preliminary

- Vertex set **V** (e.g., street intersections, houses, etc.)

- Edge set **E** (e.g., streets, roads, avenues, etc.)
    - Generally undirected (e.g. bidirectional road, etc.)
    - Weighted (e.g., distance, time, toll, etc.)

- Weight function **w(a, b): E → R**
    - Sets the weight of edge from **a** to **b**

Definitions – Preliminary

- **Weighted** Graph **G**: **G(V, E), w(a, b): E → R**

- **Connected** undirected graph **G**
  - There is a path from any vertex **a** to any other vertex **b** in **G**

- The graph we are concerned with is **connected**, **undirected**, and **weighted** when dealing with **MST**

Definitions – Main

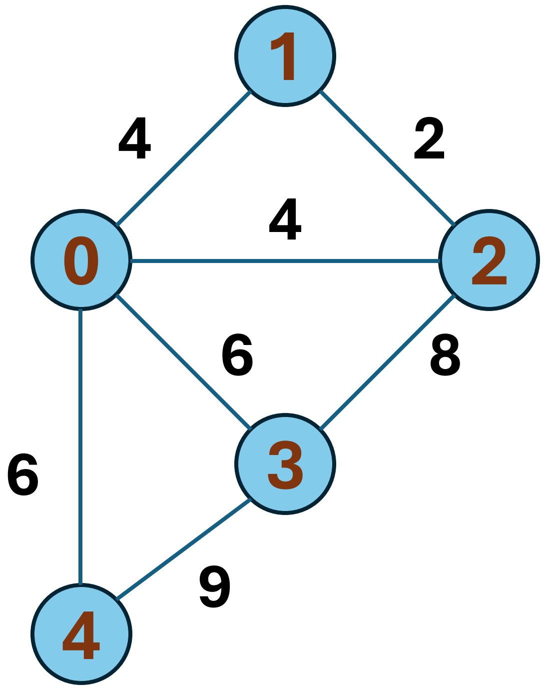
- Spanning Tree **ST** of connected undirected weighted graph **G**
  - Let **w(ST)**, weight of **ST**, denotes the total weight of edges in **ST** ➔ $w(ST) = \sum_{(a,b)\,\in\, ST} w(a,b)$

- **Minimum Spanning Tree (MST)** of connected undirected weighted graph **G**
  - **MST** of **G** is a **ST** of **G** with the minimum possible **w(ST)**

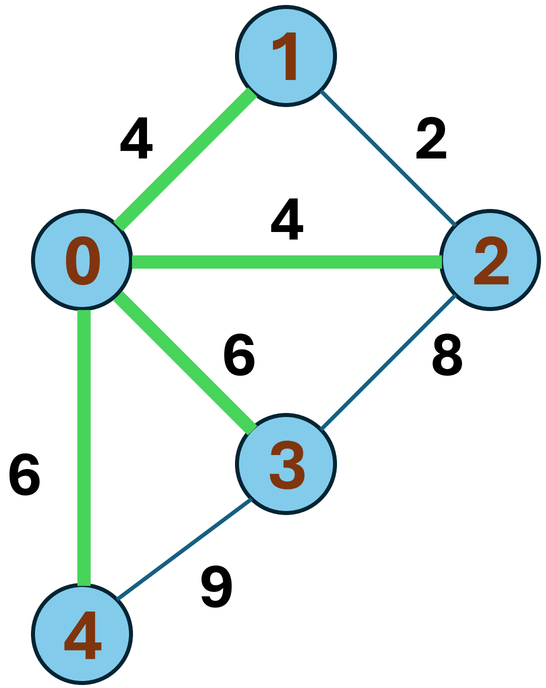
Definition – Standard MST problem

- Input: Connected undirected weighted graph **G(V, E)**
- Output: **Minimum Spanning Tree** (MST) of **G**

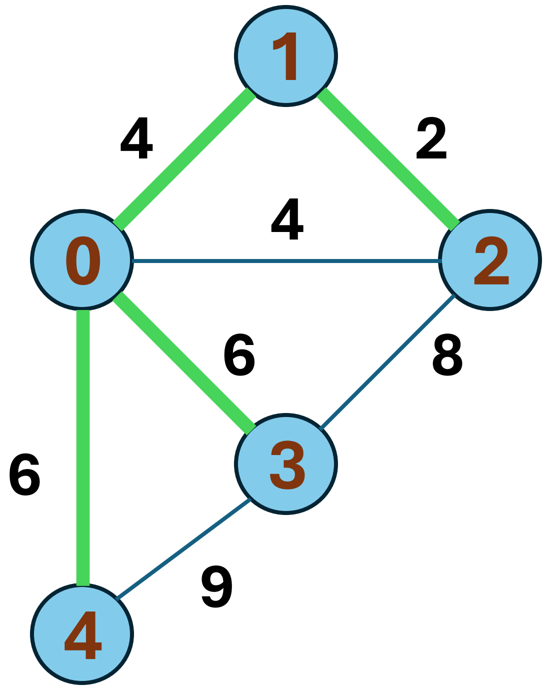
Example



Original Graph



ST with cost = 20
(6 + 6 + 4 + 4)



MST with cost = 18
(6 + 6 + 4 + 2)

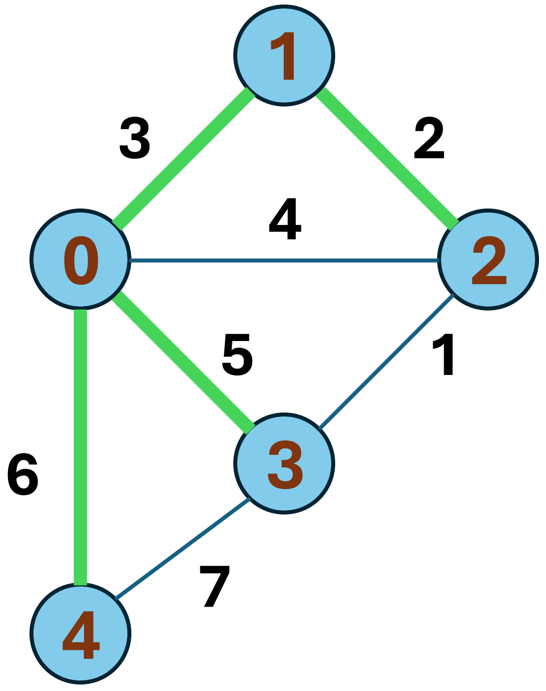
Live Quiz

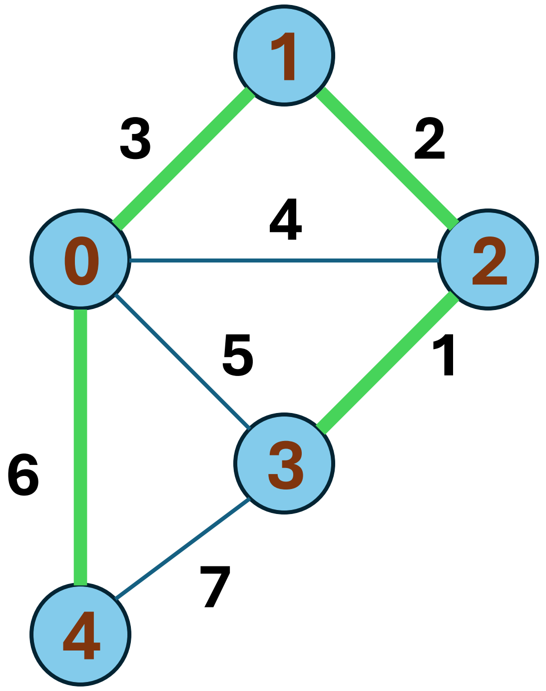
- Do the highlighted edges form an MST in this graph?

  A) No, we must replace edge 0-3 with edge 2-3

  B) No, we must replace edge 1-2 with 0-2

  C) Yes





MST Algorithms

- MST is a well-known Computer Science problem
- Several efficient (polynomial) algorithms
- Jarnik's/Prim's greedy algorithm
  - Uses PriorityQueue Data Structure (covered in Lecture 09)
- Kruskal's greedy algorithm
  - Uses Union-Find Data Structure (covered in Lecture 10)
- Boruvka's greedy algorithm (not discussed here)
- And a few more advanced variants/special cases ...

Greedy Algorithm?

- Class of algorithms that make **locally optimal** choices at each step
  - Key Idea: Select the best possible choice at each step – may not be the most optimal but is often good enough
  - Hope is to find a **global optimum** solution

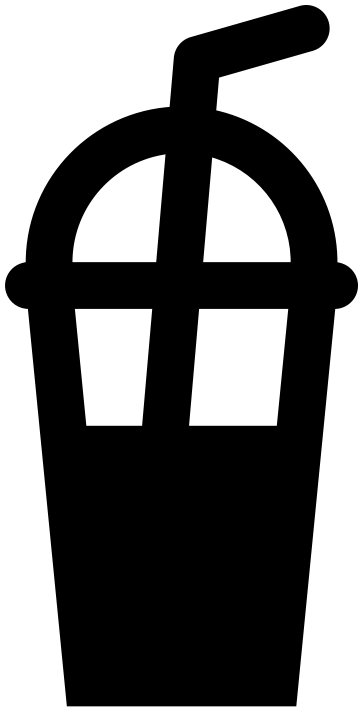
Brute force/complete search application

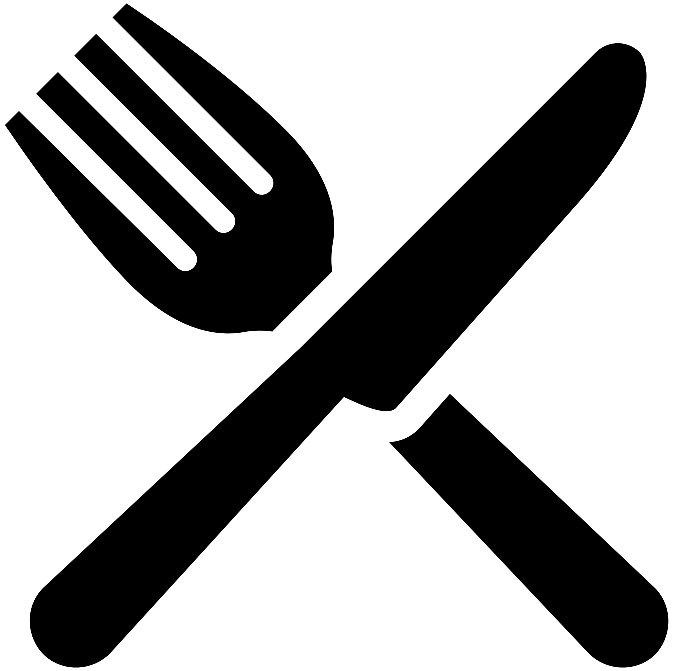
- Consider all cycles in the graph and break them!
  - For each cycle, remove the largest edge
  - If 1 or more edges in a cycle has already been removed previously move on to the next cycle

- Cycle property: For any cycle C in graph G(V,E), if weight of an edge e is larger than every other edge in C, e cannot be included in the MST of G(V,E)

- How to get all cycles in the graph?
  - Not so easy except for some special graphs ...
  - Can have up to O(V!) different cycles!
  - Listing down one by one is slow

Take a Break





Jarnik's/Prim's Algorithm

- Very simple pseudo code

```
1. T ← {s}, a starting vertex s (usually vertex 0)

2. enqueue edges connected to s (only the other ending
   vertex and edge weight) into a priority queue PQ
   that orders elements based on increasing weight

3. while there are edges left in PQ
     take out the front most edge e
     if vertex v linked with this edge e is not yet in T
       T ← T ∪ v (including this edge e)
       enqueue each edge adjacent to v into the PQ if the
       other ending vertex of that edge is not already in T

4. T is an MST
```

Easy Java Implementation

- Use two known Data Structures to implement Jarnik's/Prim's algorithm
  - A priority queue **PQ** (we can use Java PriorityQueue), and
  - A boolean array **taken** (to decide if a vertex has been taken or not)

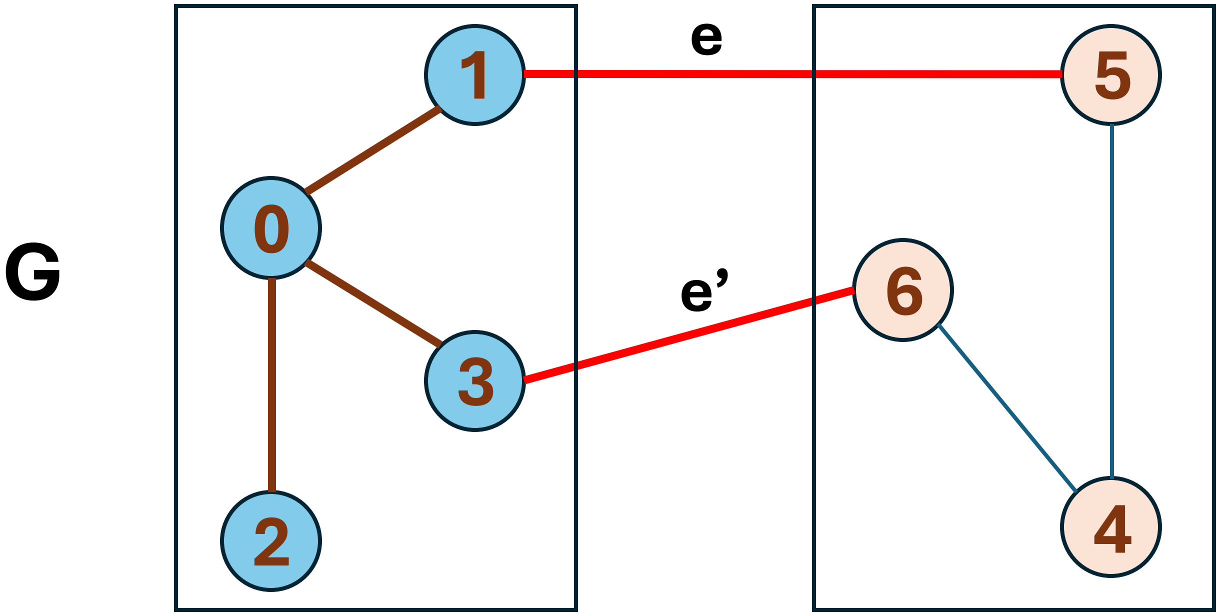
- With these DSes, we can run Prim's in O(**E** log **V**) using Adjacency list
  - We process each edge only once (enqueue and dequeue it), O(**E**)
  - Can do each enqueue/dequeue from a PQ in O(log **E**)
  - As $\mathbf{E} = O(\mathbf{V}^2)$, we have $O(\log \mathbf{E}) = O(\log \mathbf{V}^2) = O(2 \log \mathbf{V}) = O(\log \mathbf{V})$
  - Total time $O(\mathbf{E}) * O(\log \mathbf{V}) = O(\mathbf{E} \log \mathbf{V})$

Why does Jarnik's/Prim's Algorithm work?

- **Jarnik's/Prim's algorithm** is a **greedy algorithm**
- **At each step**, it always try to select the next valid edge **e** with **minimal weight** (greedy!)
- Greedy algorithm is usually simple to implement
    - However, it usually requires "proof of correctness"
    - Here, we will just see a quick proof

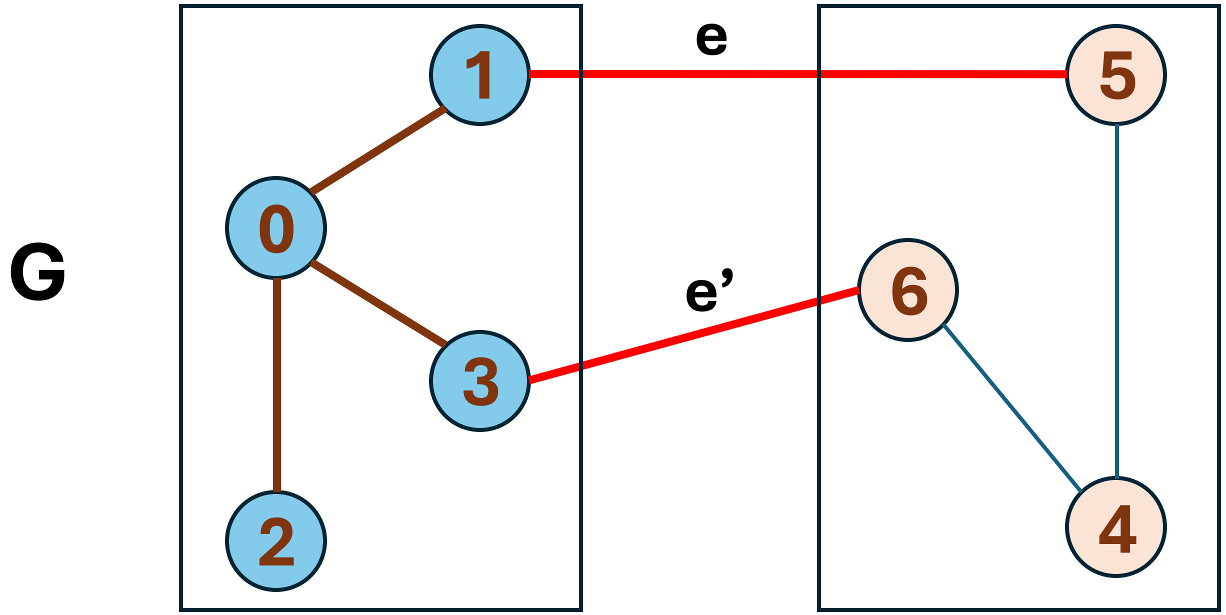
Cut Property of a Connected Graph G

- **Cut of a connected graph**: any partition of vertices of G into 2 disjoint subsets (vertices in one set are not in the other)
- **Cut Set**: The set of edges that cross a cut (e and e' in the example)



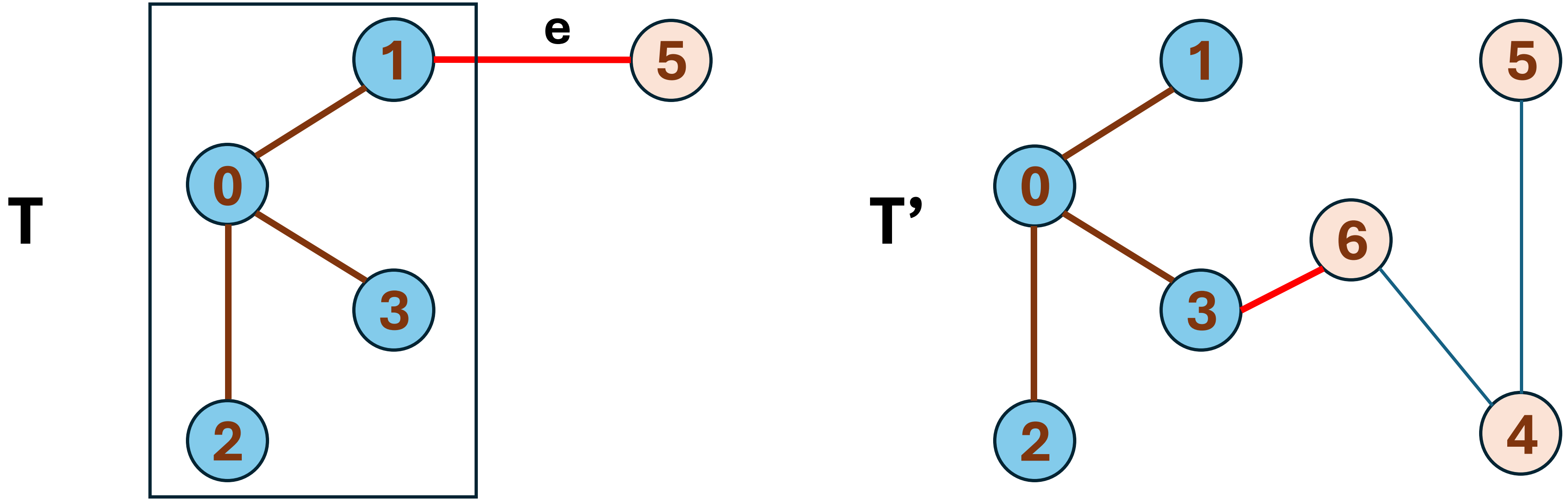
Cut Property of a Connected Graph G

- **Cut Property of a connected graph**: For any cut of the graph, if the weight of an edge **e** in the cut-set is strictly smaller than the weights of other edges of the Cut Set, then **e** belongs to all MSTs of G



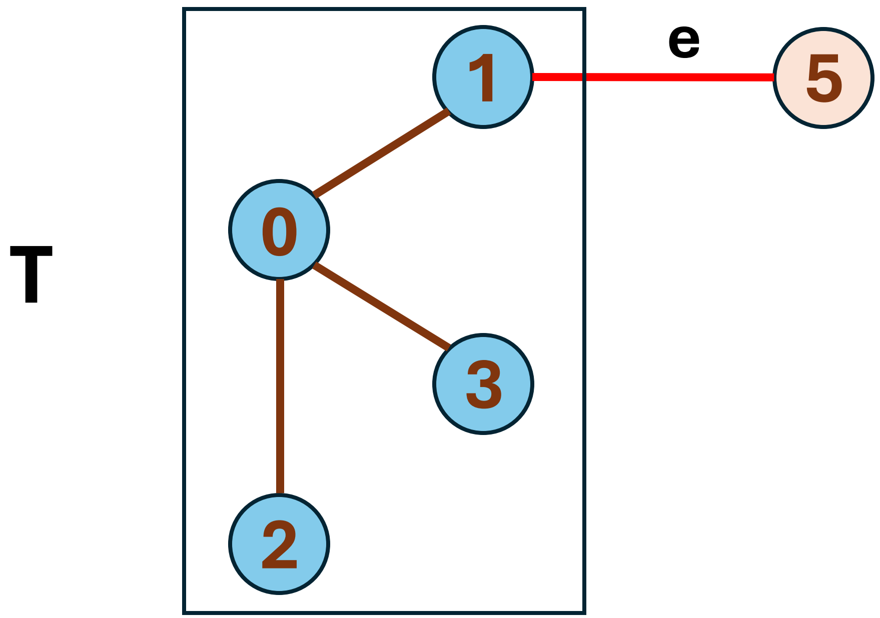
Proof (By Contradiction)

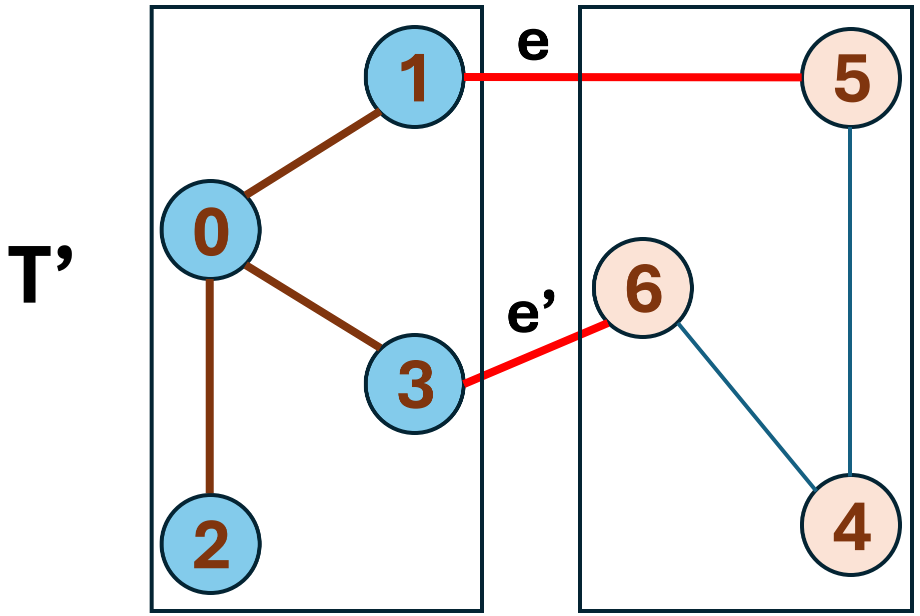
- Assume that edge **e** is the first edge at iteration k chosen by the algorithm which is not in any valid MST
- Let **T** be the tree generated before adding **e**
- Now **T** must be a subtree of some valid MST **T'**



Why does Jarnik's/Prim's Algorithm Work?

- Adding edge **e** to **T'** will now create a cycle
- Since e has 1 endpoint in **T** (the valid endpoint) and one endpoint outside **T**, trace around this cycle in **T'** until we get to some edge **e'** that goes back to **T**
- Removing **e** and **e'** will disconnect T' into 2 components ({0,1,2,3} which is T and {4,5,6} in the example). This is a cut of **T'**, where {**e**, **e'**} is the cut set as illustrated

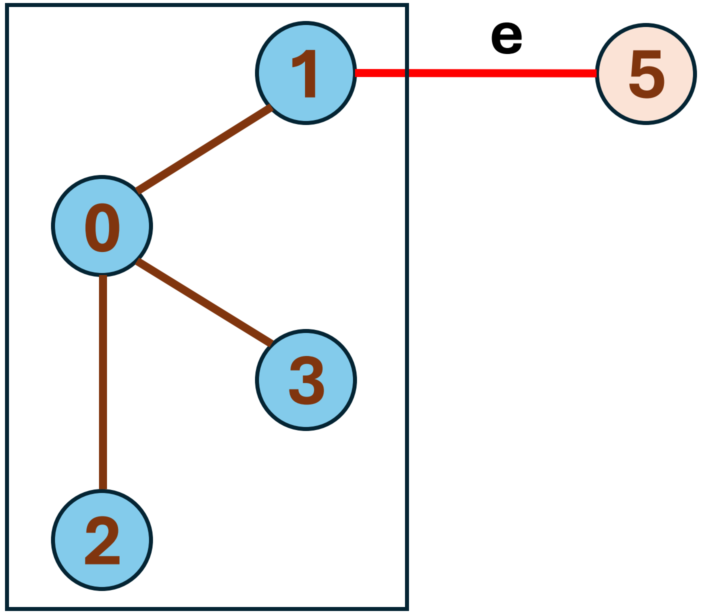




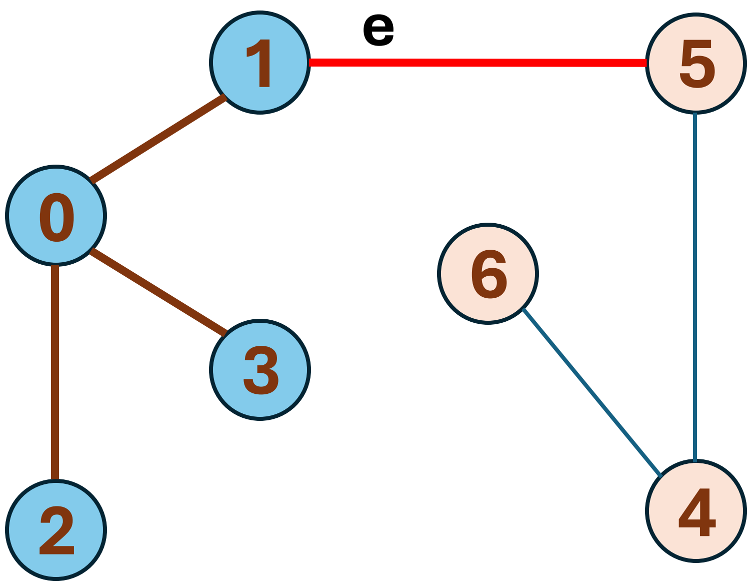
Why does Jarnik’s/Prim’s Algorithm Work?

- By algorithm, **e** and **e’** must be candidate edges at iteration k, but **e** was chosen meaning w(**e**) ≤ w(**e’**) by the **cut property**
- Now replacing **e’** with **e** in **T’** must give us tree **T’’** covering all vertices of the graph such that w(**T’’**) ≤ w(**T’**)
- Contradiction that **e** is first edge chosen wrongly

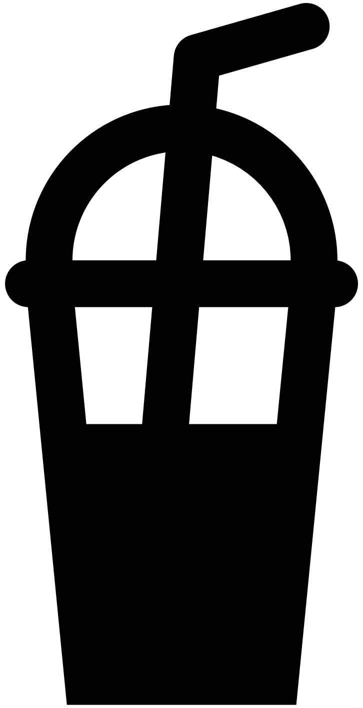
T

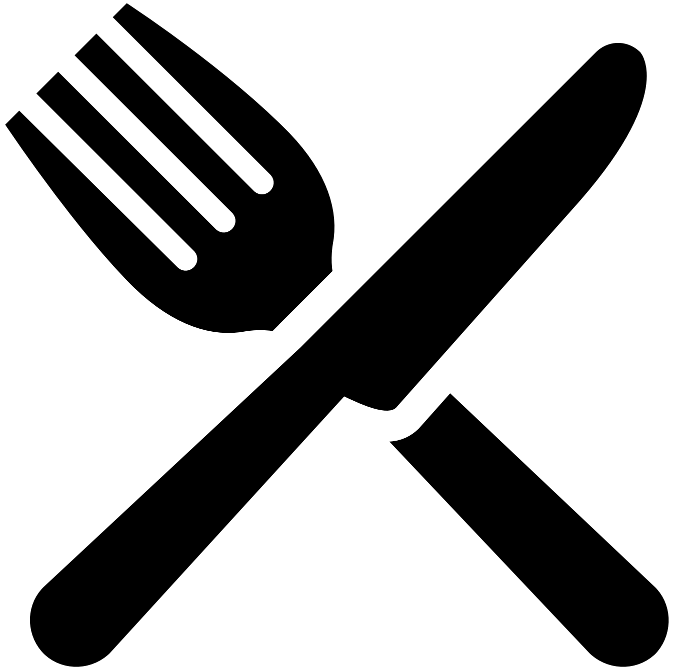


T’’



Take a Break





Kruskal's Algorithm

- Very simple pseudo code

```
1. sort the set of E edges by increasing weight

2. T ← {}

3. while there are unprocessed edges left
     pick an unprocessed edge e with min cost
     if adding e to T does not form a cycle
       add e to T

4. T is an MST
```

Kruskal's Algorithm

- Very simple pseudo code

```
1. sort the set of E edges by increasing weight // O(?)

2. T ← {}

3. while there are unprocessed edges left
     pick an unprocessed edge e with min cost // O(?)
     if adding e to T does not form a cycle // O(?)
       add e to T // O(1)

4. T is an MST
```

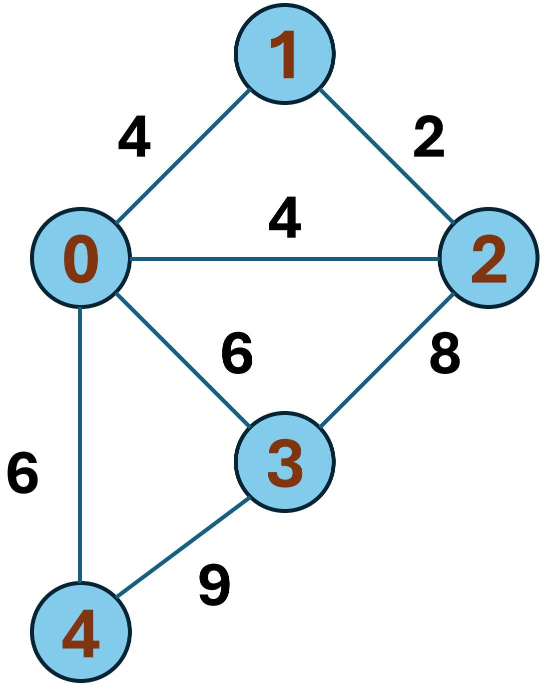
Kruskal’s Algorithm – Data Structures

- Sorting edges
  - Use an **Edge List** to store them
  - Sort using ‘any’ sorting algorithm we know
- Testing for cycles
  - Use UFDS

Time Complexity?

Kruskal's Algorithm – Time Complexity

- Sorting edges



i	w	u	v
0	4	0	1
1	4	0	2
2	6	0	3
3	6	0	4
4	2	1	2
5	8	2	3
6	9	3	4

i	w	u	v
0	2	1	2
1	4	0	1
2	4	0	2
3	6	0	3
4	6	0	4
5	8	2	3
6	9	3	4

$O(E \log E)$

Kruskal's Algorithm – Time Complexity

- Testing for cycles
    - Recall: UFDS time complexity is ..... $O(\alpha(V)) \rightarrow O(1)$
- Overall: $O(E \log E + E * \alpha(V)) \rightarrow O(E \log E)$
- $E = O(V^2) \rightarrow O(E \log V^2) \rightarrow$ **$O(E \log V)$**

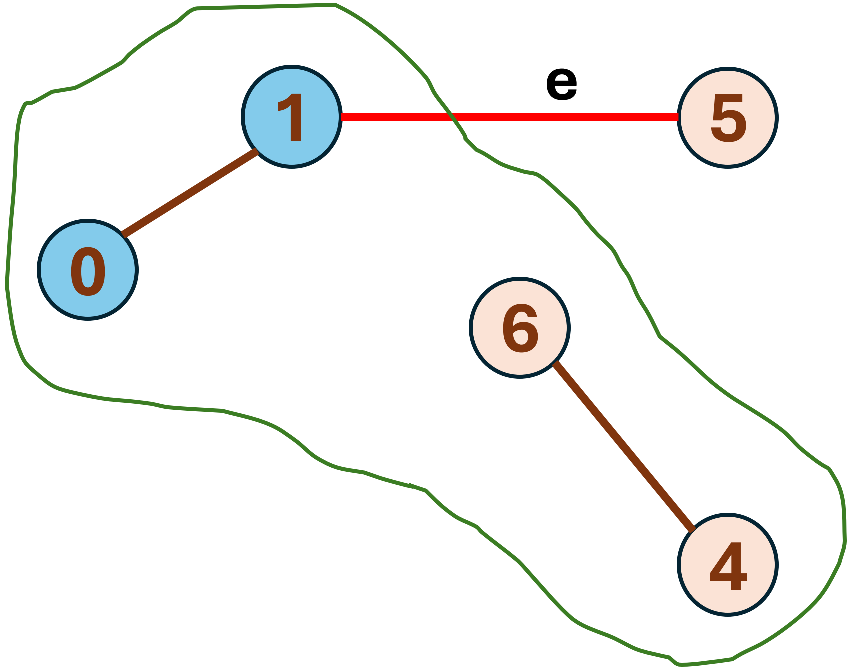
Why does Kruskal’s Algorithm work?

- **Kruskal’s algorithm** is a **greedy algorithm**
- **At each step**, it always try to select the next valid edge e with **minimal weight** (greedy!)
- Proof: almost same as Prim’s algorithm

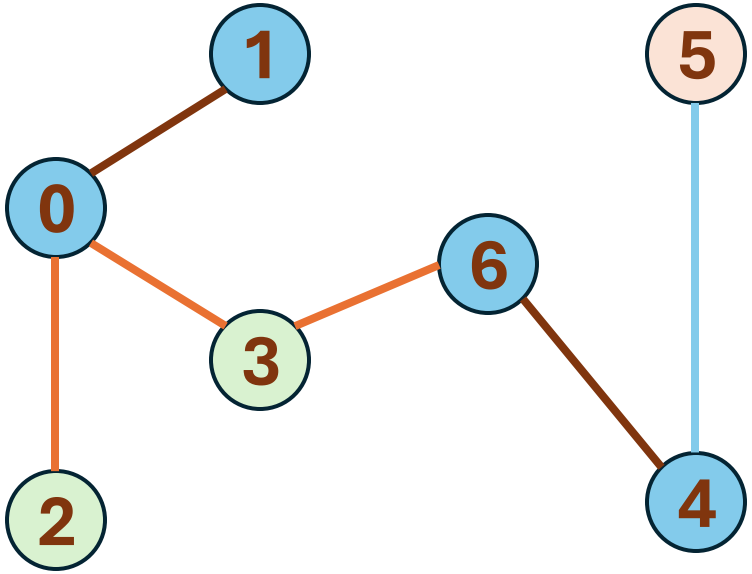
Proof by Contradiction

- Assume that edge **e** is the first edge at iteration k chosen by Kruskal's which is not in any valid MST
- Let **F** be the forest generated by Kruskal's before adding **e**
- Now **F** must be a part of some valid MST **T'**

F

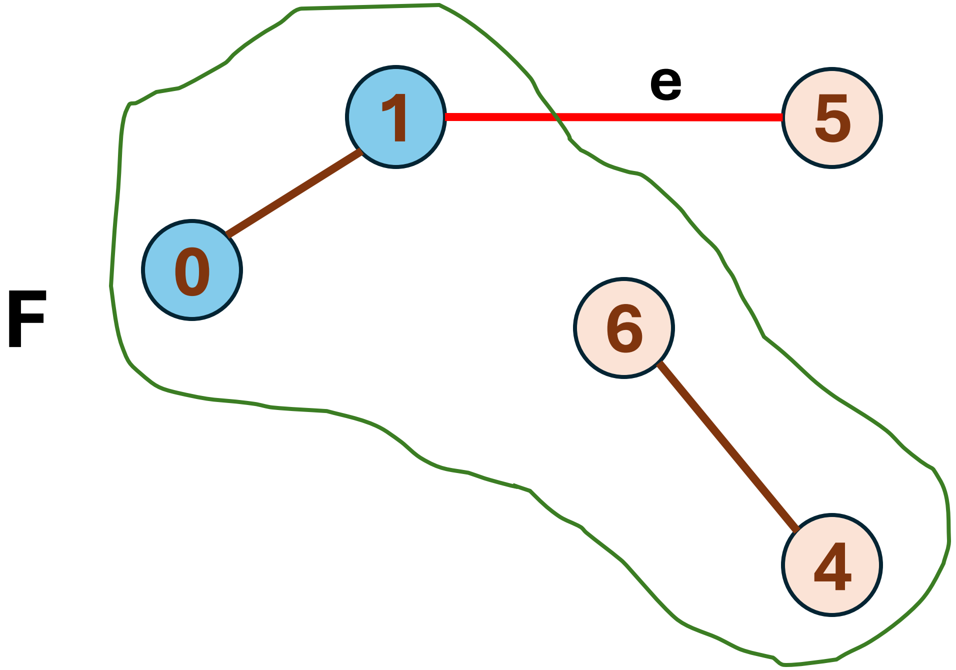


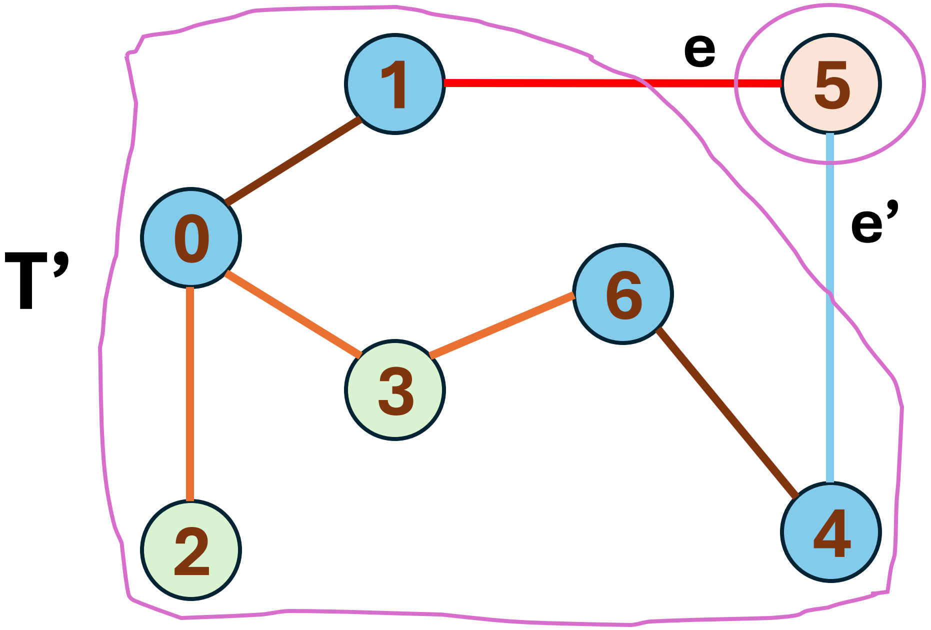
T'



Why does Kruskal’s Algorithm work?

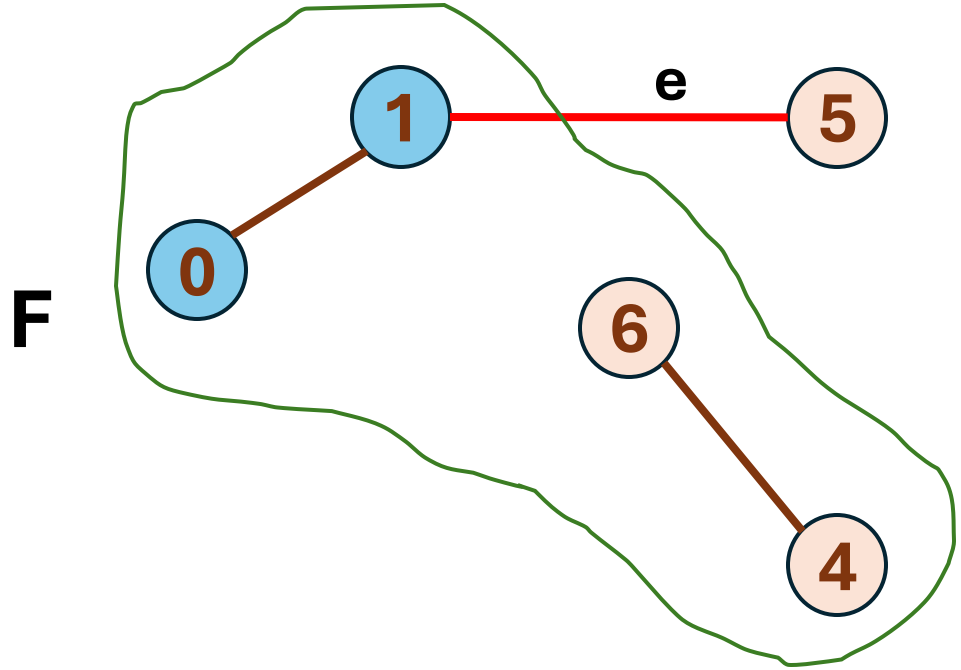
- Putting **e** into **T’** will create a cycle
- Tracing the cycle using **e** to exit **F**, at some point we must come across an edge **e’** leading back to **F**
- Removing **e** and **e’** will create 2 components of T’ ({0,1,2,3,4,6} and {5} in the example). This forms a cut where {**e**, **e’**} is the cut set

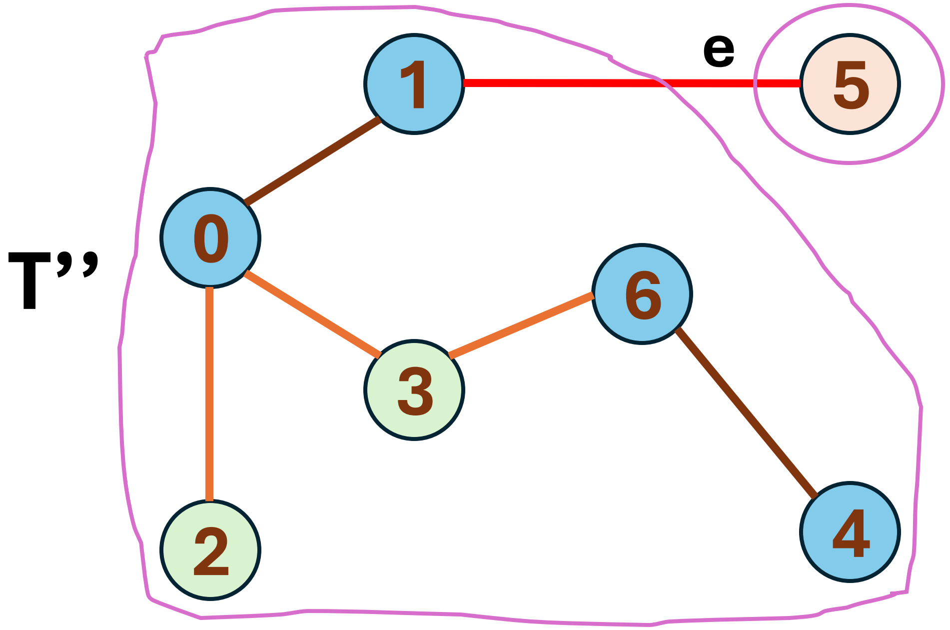




Why does Kruskal’s Algorithm work?

- At iteration k, both **e** and **e’** are candidates and do not form a cycle if chosen
- Since **e** was chosen, w(**e**) ≤ w(**e’**) by the **cut property**
- Now replacing **e’** with **e** in **T’** gives us tree **T’’** covering all vertices of the graph s.t w(**T’’**) ≤ w(**T’**)
- Contradiction that **e** is first edge chosen wrongly





Summary

- Introduced the MST problem

- Discussed 2 algorithms
  - Jarnik's/Prim's algorithm (uses PriorityQueue ADT)
  - Very briefly mentioned a variant for dense graphs where $E = O(V^2)$ (uses an array instead of PQ)

  - Kruskal's algorithm (uses Edge List and UFDS)

  - They use the Cut Property as opposed to the Cycle Property to construct the MST of any given connected weighted graph

Next Week

- Single Source Shortest Paths

Continuous Feedback

https://forms.office.com/r/KsNwmTUD0q