NUS National University of Singapore

School of Computing

Tutorial 9: Graphs Traversal II & MST

October 25, 2022

Gu Zhenhao

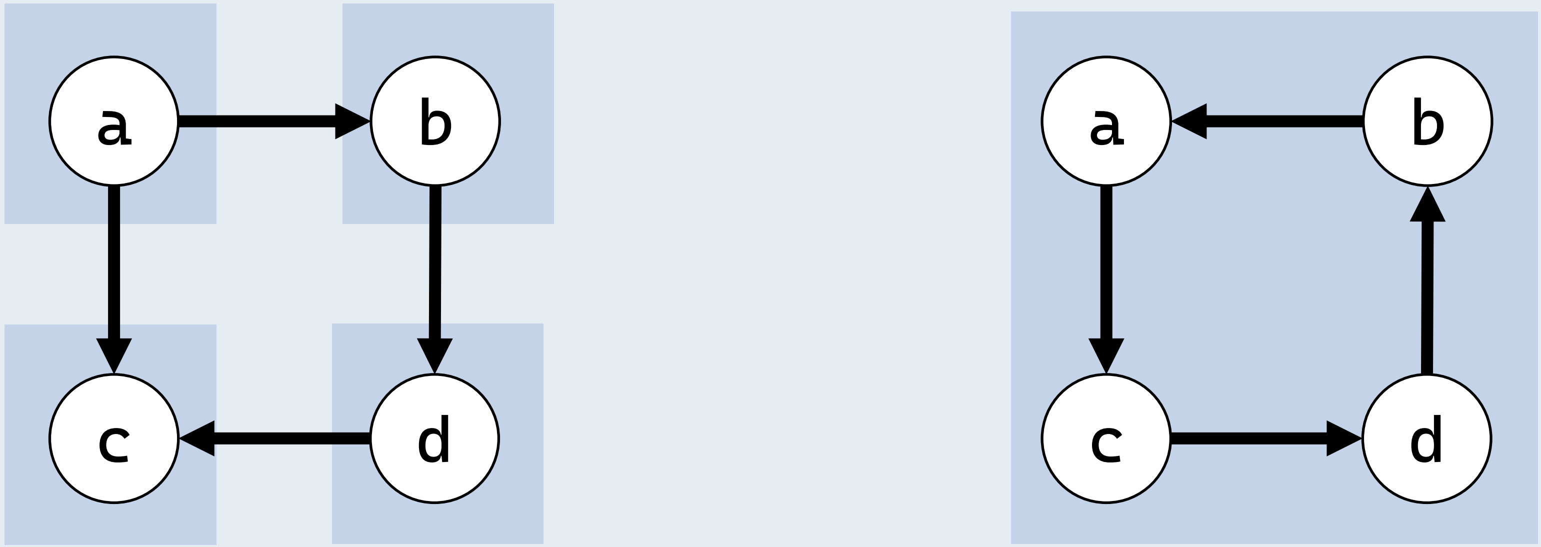
* *Partly adopted from tutorial slides by [Wang Zhi Jian].*

Strongly Connected Component

What are SCCs and what do they imply?

Strongly Connected Components

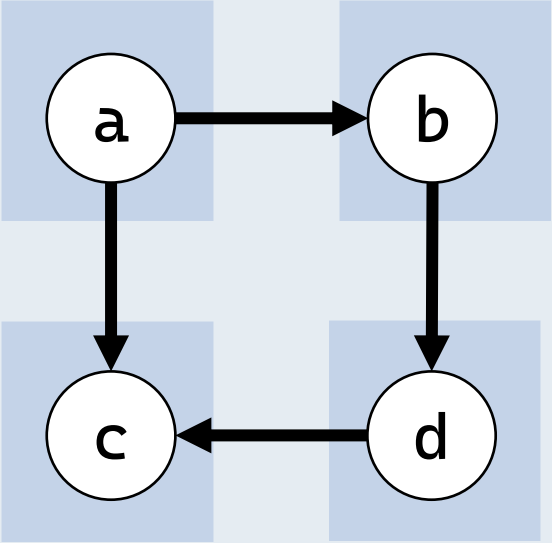
In directed graphs, a ***strongly connected component (SCC)*** is a subgraph where there is a path between **ALL** pairs of vertices.

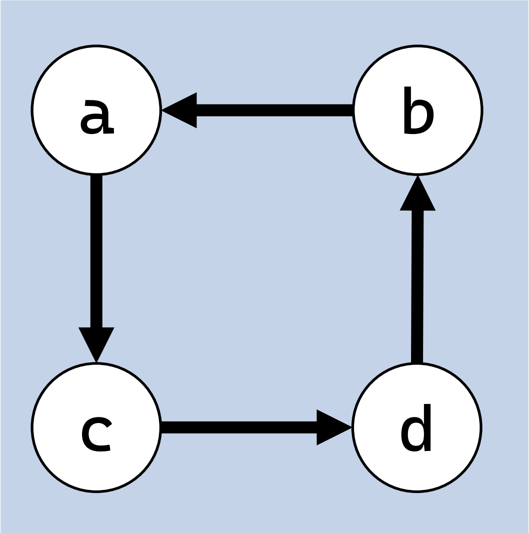


Strongly Connected Components

Claim: A graph contains SCC of size > 1 vertex $\Leftrightarrow$ graph contains a cycle.

Idea: To detect a cycle, just use **Kosaraju's algorithm** to find the SCC! Time is still $O(|V| + |E|)$.





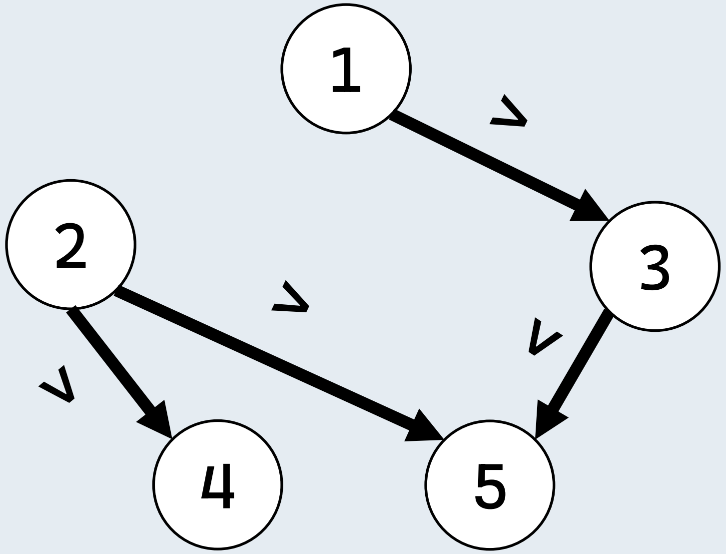
Graph Traversal (Continued)

How can we utilize traversal algorithms?

Problem 1.a

- n skyscrapers, numbered 1 to n,
- m pairwise comparisons of the height of skyscrapers,
- **Goal**: give one possible ordering of the height.

Info
1 taller than 3
2 taller than 5
3 taller than 5
2 taller than 4



Possible orderings:

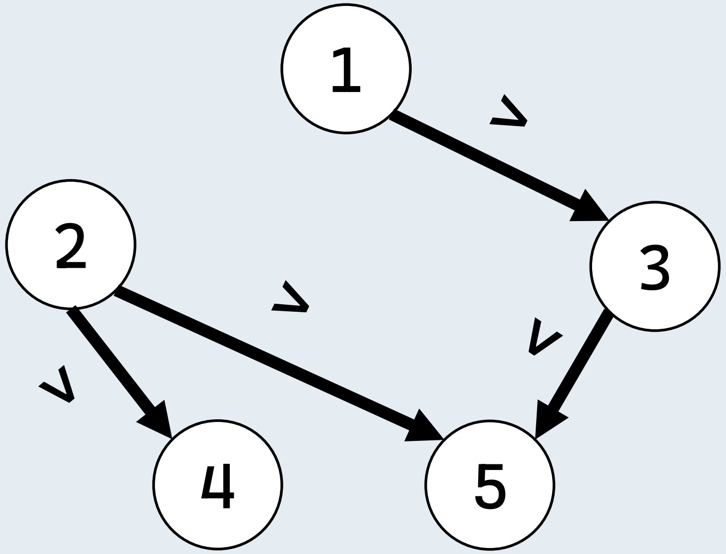
2 > 1 > 3 > 4 > 5

1 > 3 > 2 > 5 > 4

...

Problem 1.a

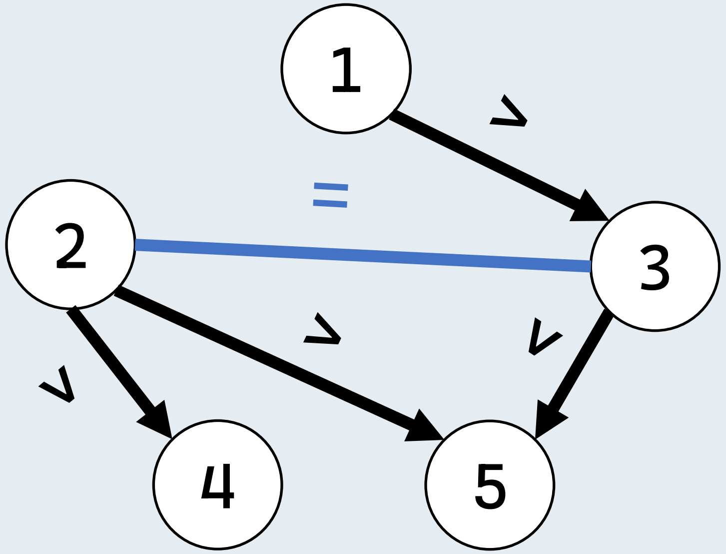
- **Observation**: this is a ***directed acyclic graph (DAG)***.
- **Idea**: We can use topological ordering to ensure that if **u > v**, **u** is put in front of **v** in the ordering.



Problem 1.b

- n skyscrapers, numbered 1 to n,
- m pairwise comparisons/equality of the height of skyscrapers,
- **Goal**: give one possible ordering of the height.

Info
1 taller than 3
2 taller than 5
3 taller than 5
2 taller than 4
2 as tall as 3



Possible orderings:

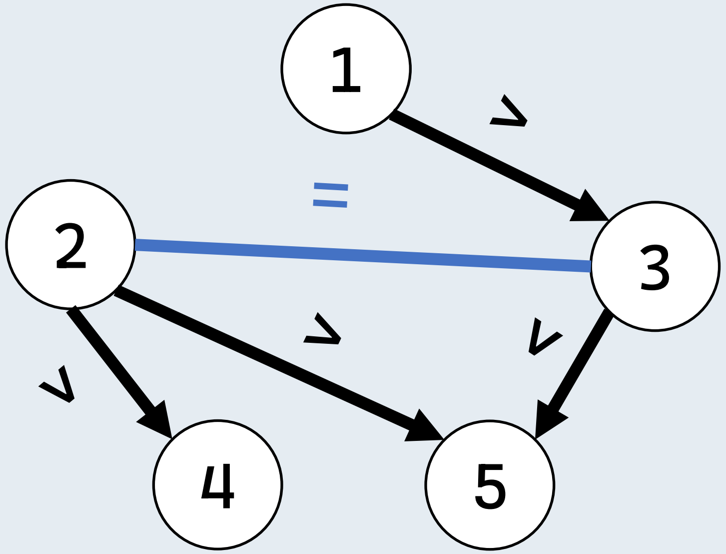
1 > 2 = 3 > 4 > 5

1 > 3 = 2 > 5 > 4

...

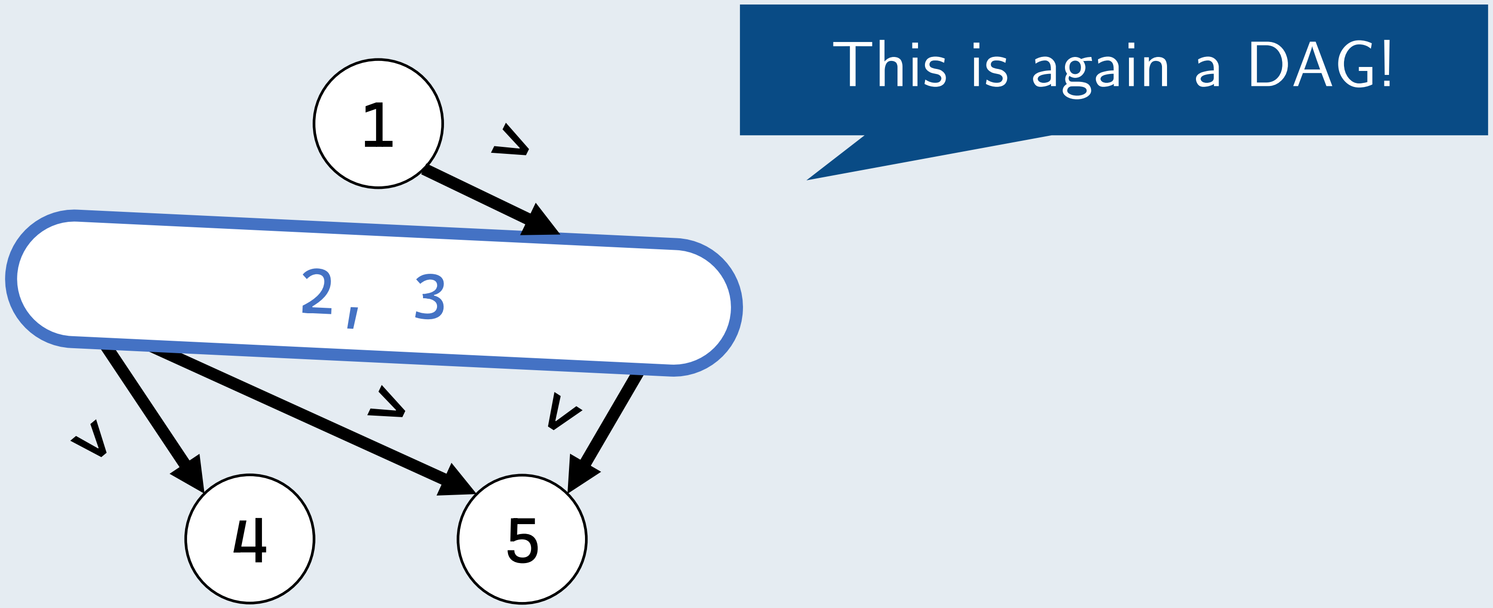
Problem 1.b

- **Observation**: this is no longer a DAG... so we cannot use topological sort!
- **Question**: Is it possible to turn this into a DAG?



Problem 1.b

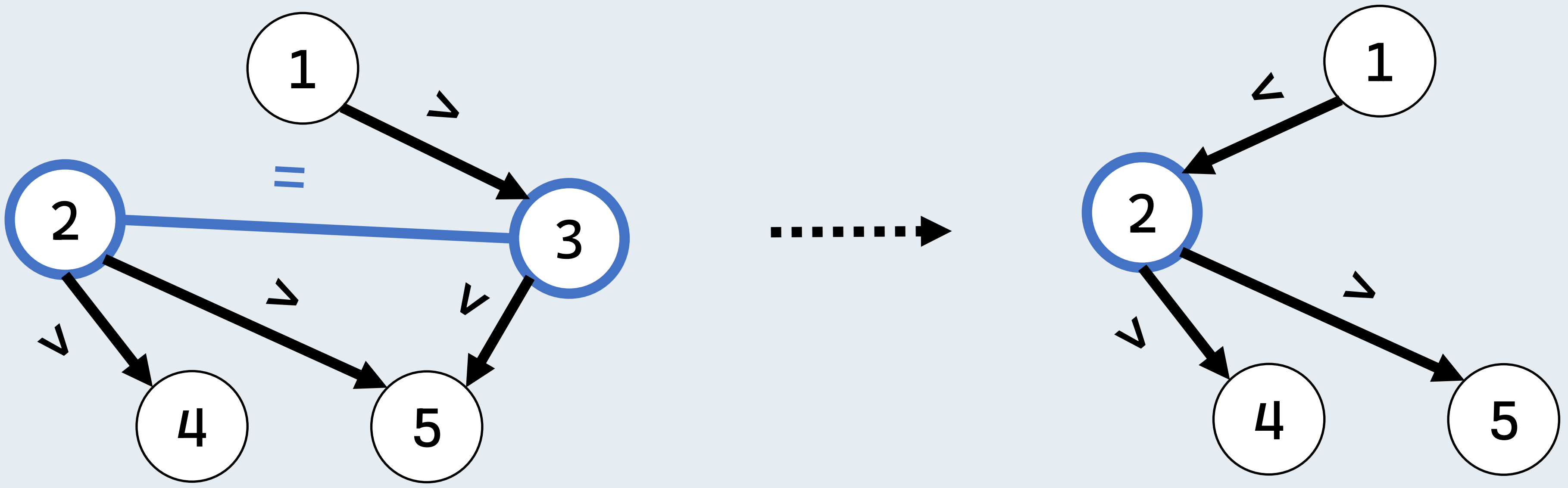
- **Observation**: this is no longer a DAG... so we cannot use topological sort!
- **Question**: Is it possible to turn this into a DAG?
- **Idea**: We can view the vertices of equal height as 1 vertex!



Problem 1.b

Note: to implement this ***edge contraction*** in practice, we can

- Pick a representative vertex, e.g. **2**.
- Connect the edges containing vertex **3** to vertex **2** instead.
- Keep a UFDS storing all the merged vertices.

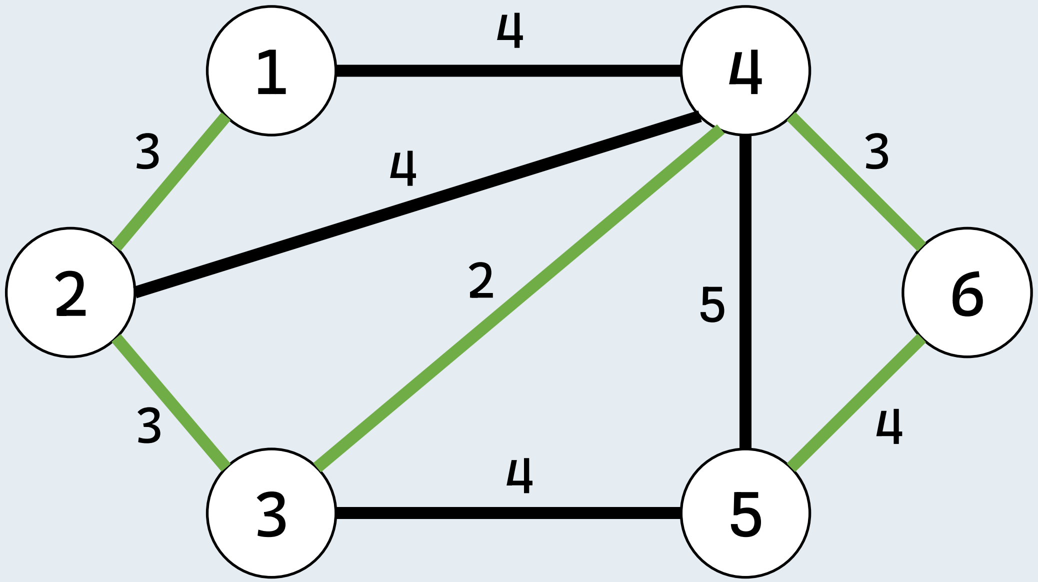


Minimum Spanning Tree

What is MST and how to find the MST?

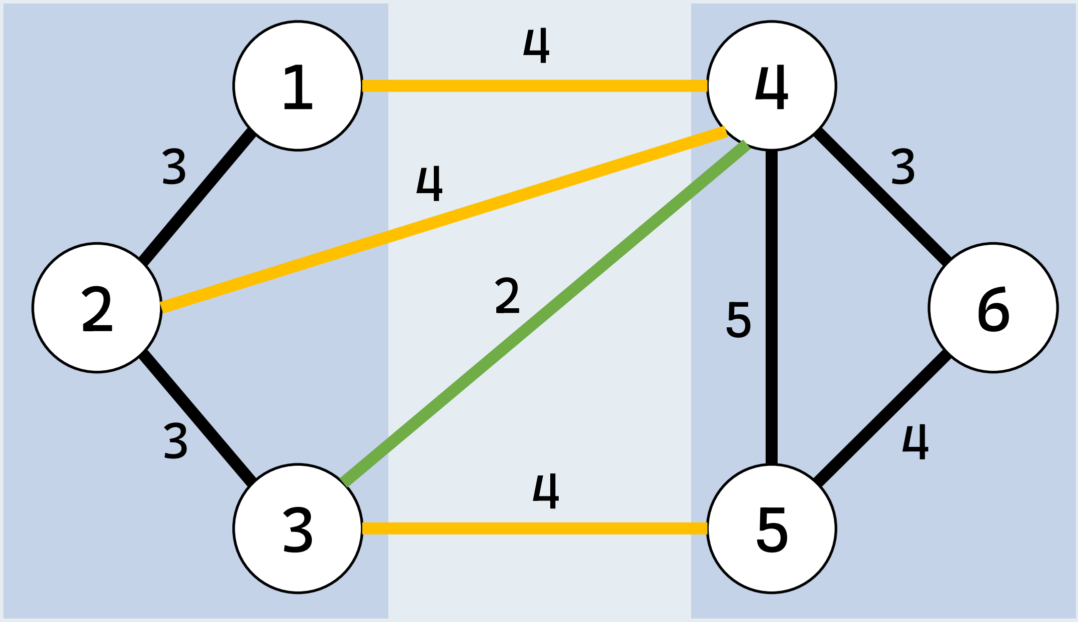
Minimum Spanning Tree

The ***minimum spanning tree*** of a connected, weighted and undirected graph is a tree of minimum total edge weight that connects all vertices.



Minimum Spanning Tree

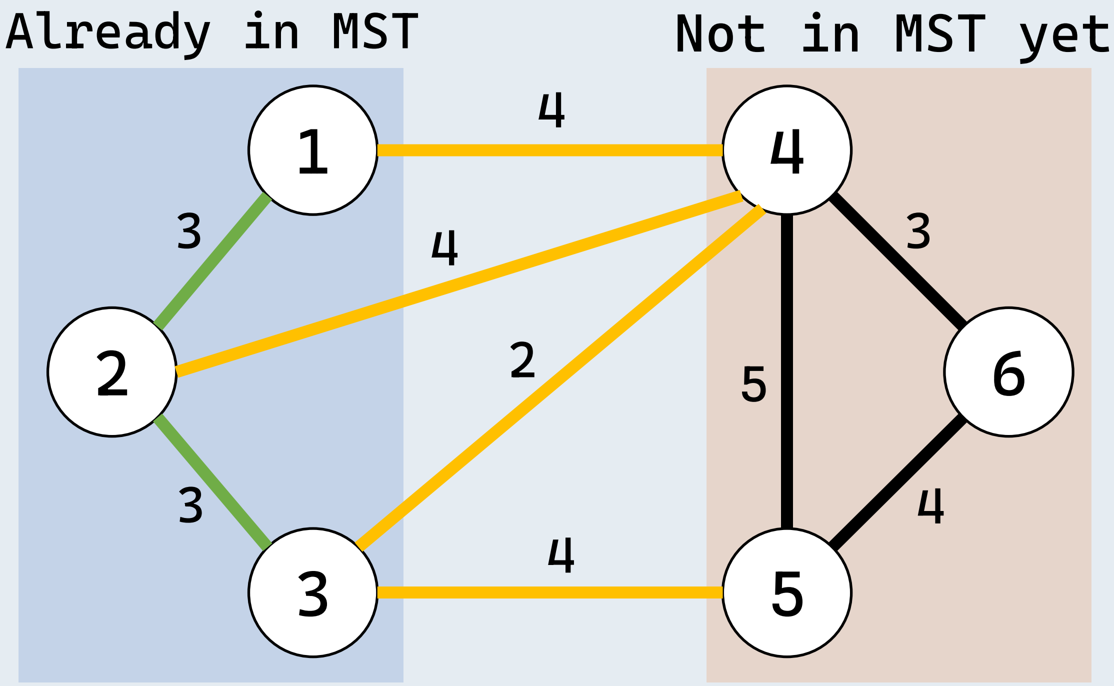
Cut property: For every cut (partitioning of nodes into two sets), the edge with the smallest weight across the cut is in the MST. (Why?)



Prim's Algorithm

Idea:

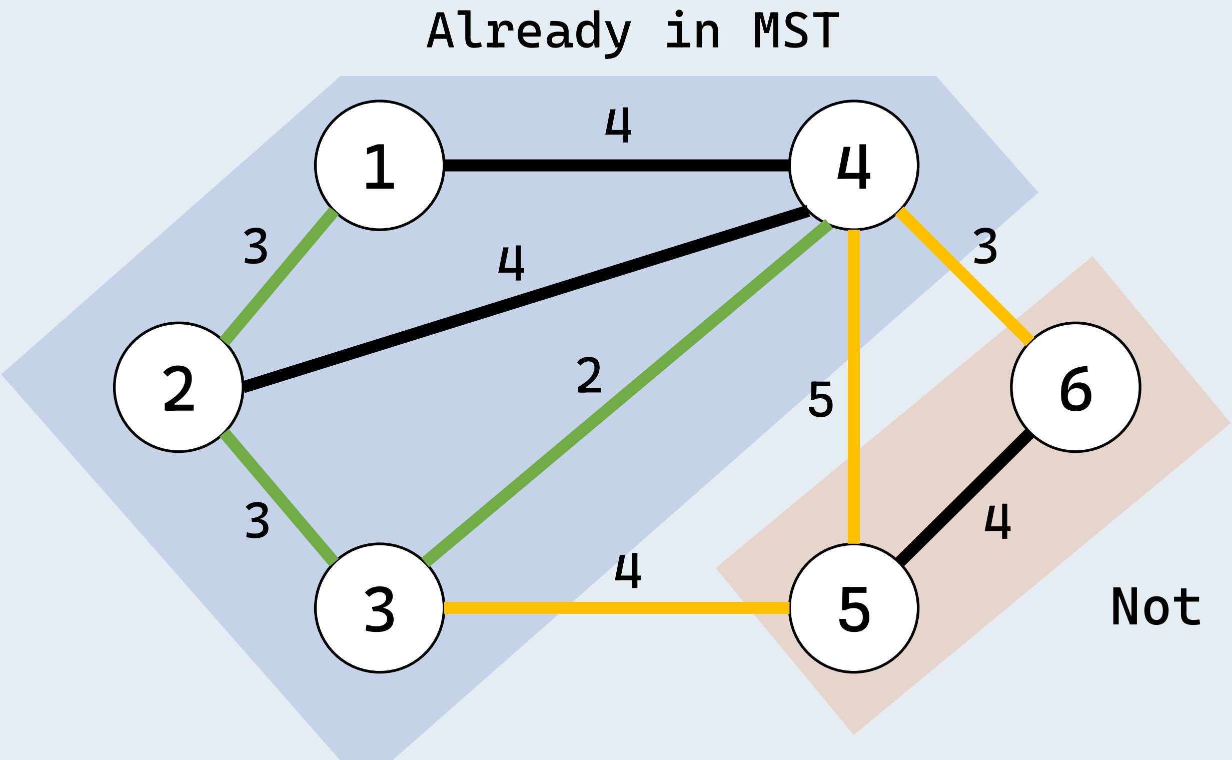
- Partition vertices into 2 sets: those already in our MST and those that are not.
- Find the minimum weighted edge across the cut and put it in MST.



Prim's Algorithm

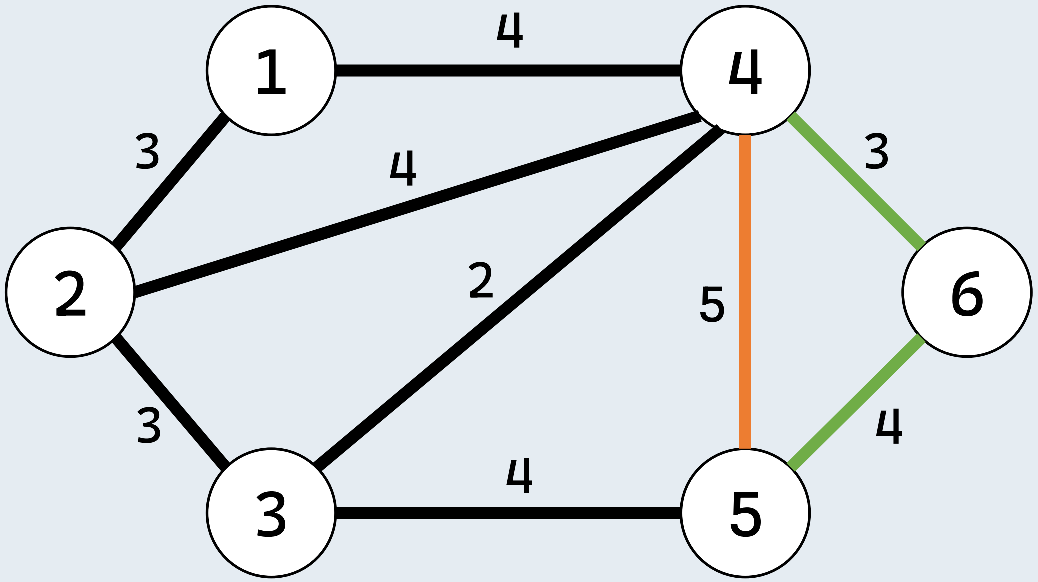
Implementation:

- Start from one node. Push all neighboring edges of MST into a **min heap**.
- Pop the minimum edge, include it in MST, and repeat the process.



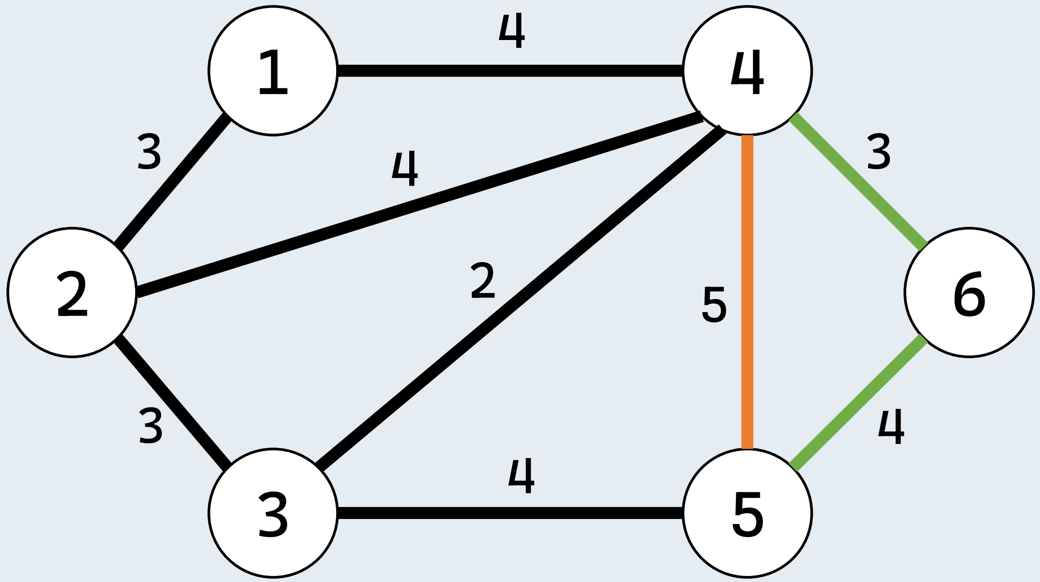
Minimum Spanning Tree

Cycle property: For every cycle in the graph, the edge with the maximum weight is not in the MST. (Why?)



Kruskal's Algorithm

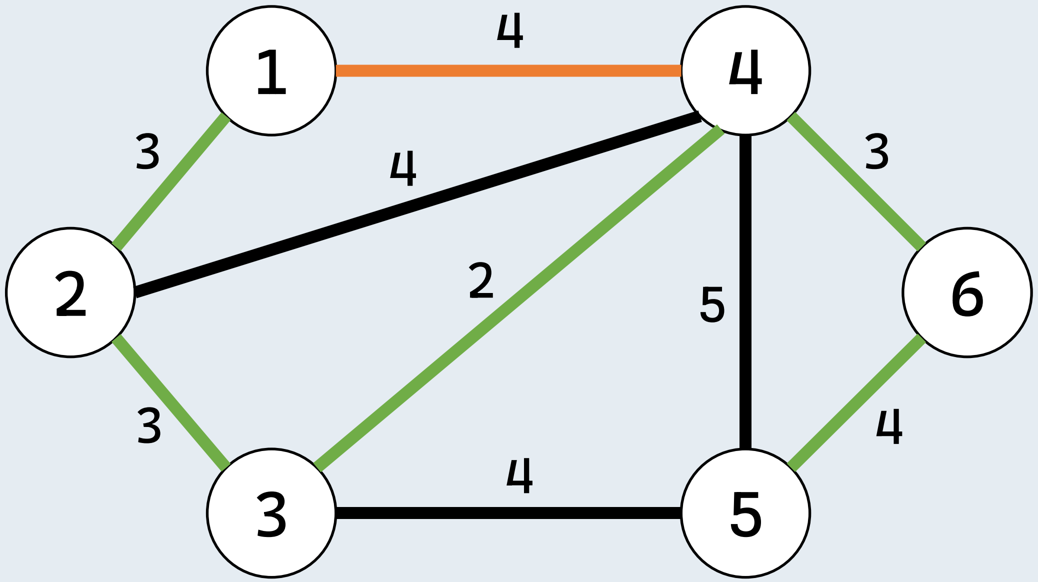
Cycle property: For every cycle in the graph, the edge with the maximum weight is not in the MST. (Why?)



Kruskal's Algorithm

Idea:

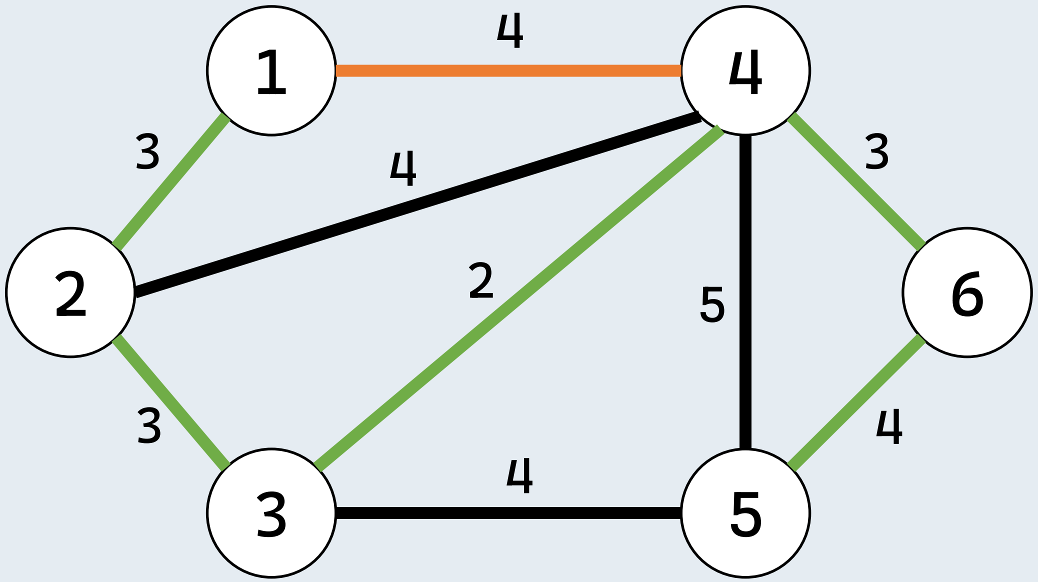
- Repeatedly add the smallest weighted edges into MST.
- Discard if the added edge form a cycle within the MST.



Kruskal's Algorithm

Implementation:

- Keep a **UFDS** of all vertices: each set is a connected component in MST.
- If both ends of an edge are in the same set, adding this edge would form a cycle. (Why?)



Problem 2.a

True or false: *The MST is always a connected, undirected graph.*

True. This is by the property of MST. It is coming from an undirected graph. Since it also connects all vertices, it must be connected.

Problem 2.b

True or false: *The MST will always have $V-1$ edges.*

True. This is by the property of trees, number of edges in trees are always equal to $V-1$.

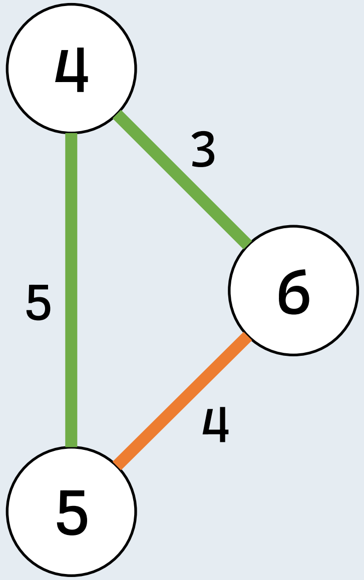
Problem 2.c

True or false: *For a graph with unique edge weights, the edge with the largest weight in any cycle of the graph can be included in the MST.*

False. This can be proved with contradiction.

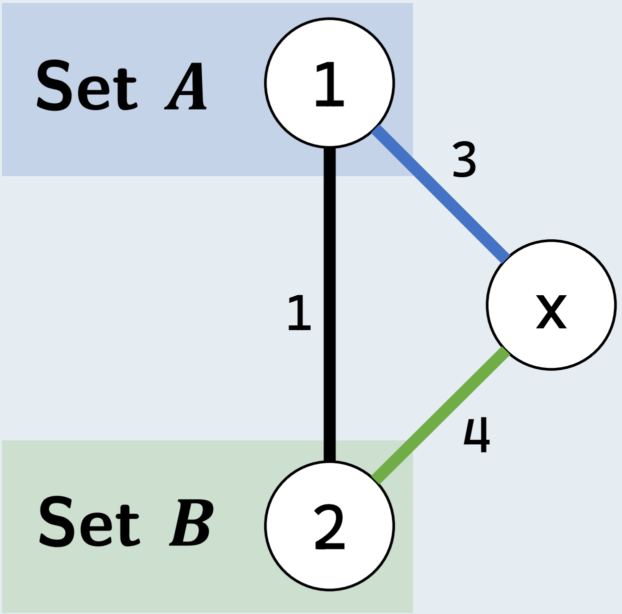
Suppose MST contains the largest-weighted edge e, there must be an edge e' of smaller weight in the cycle that is not in MST.

Then substituting e with e' will decrease total weight.



Problem 2.d

True or false: *For a graph with two disjoint sets of vertices A and B (vertices in A are not in B and vice versa), and another vertex x not inside both the sets, the combined MST of $A \cup x$ and $B \cup x$ is a MST of the original graph.*



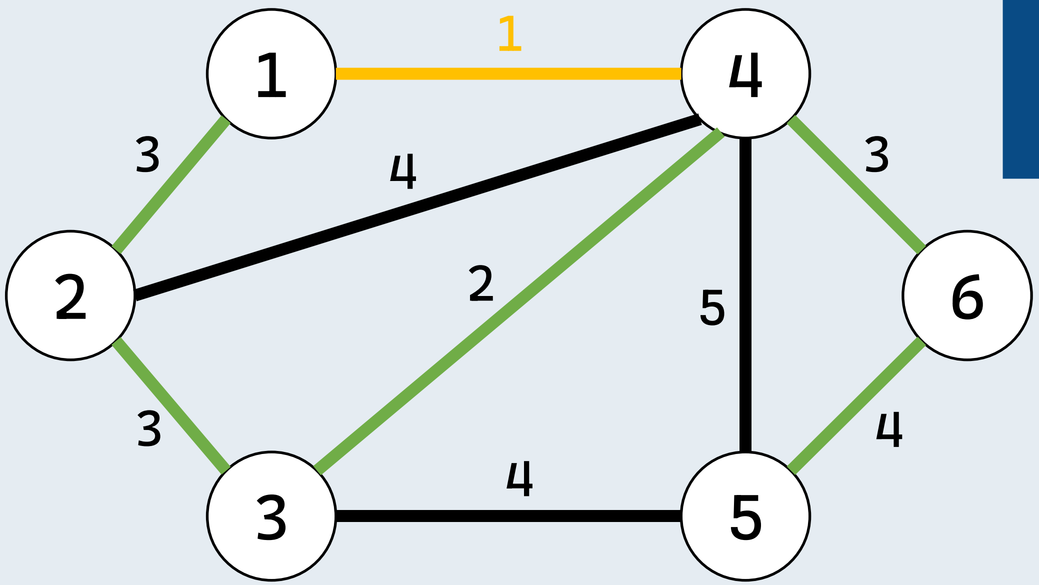
False. A counter example can be drawn.

The combined MST of $A \cup x$ and $B \cup x$ has total weight $3 + 4 = 7$, while the minimum weight should be $1 + 3 = 4$.

Problem 3.a

Given MST of a graph, find the new MST if another edge is added.

Trivial Solution: Re-run Prim's or Kruskal's algorithm on the new graph, $O(E \log V)$ time.

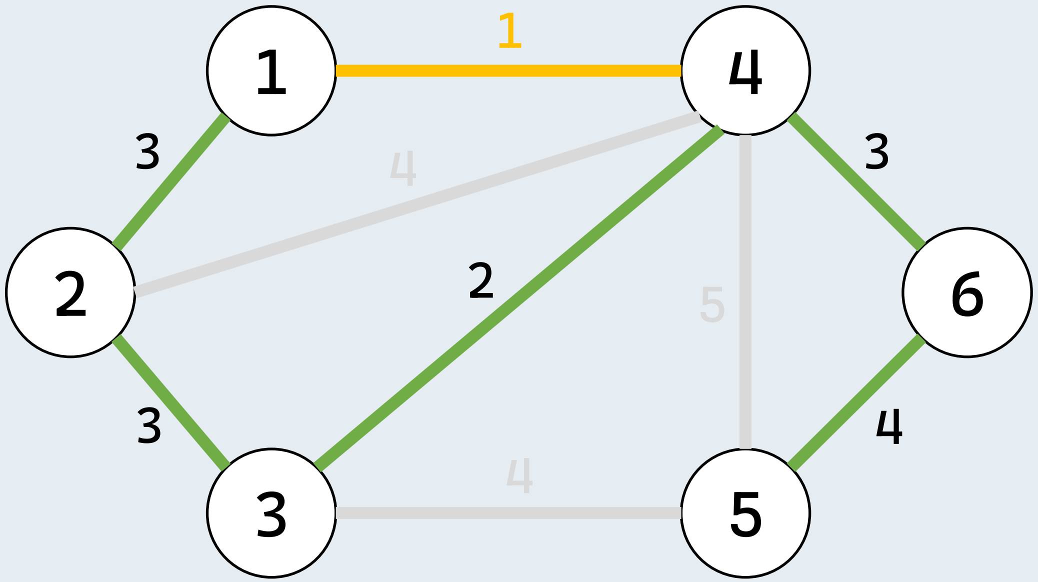


Do we still need to consider all edges?

Problem 3.a

Observation: the edges not in the original MST will not be in the new MST.

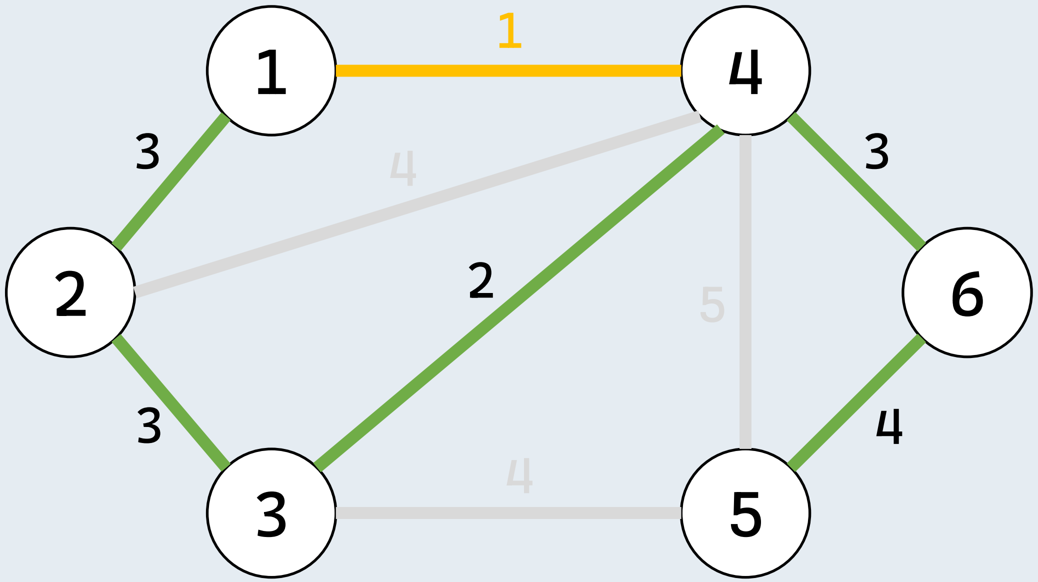
Better Solution: We don't need to consider those edges that are originally not in MST. $O(V \log V)$ time.



Problem 3.a

Observation: the newly added edge forms a cycle with the old MST.

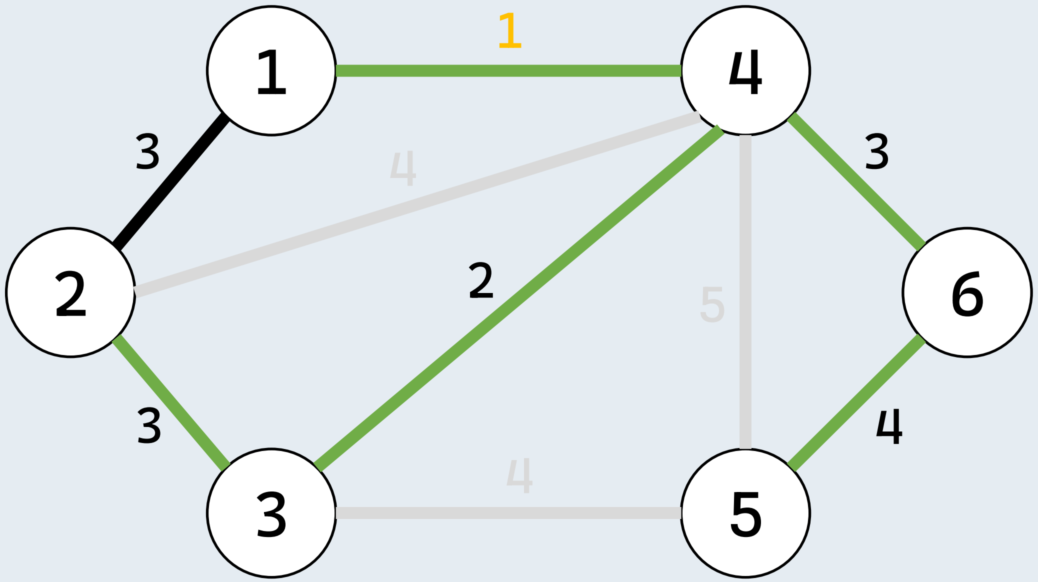
Idea: By the cycle property, simply remove the largest edge in the cycle!



Problem 3.a

Better solution: given a new edge `(u, v)`

- Do a DFS in the old MST from **u** to **v**, and find the largest edge e in the path.
- Compare e with the new edge. If the new edge have smaller weight, replace e with the new edge.

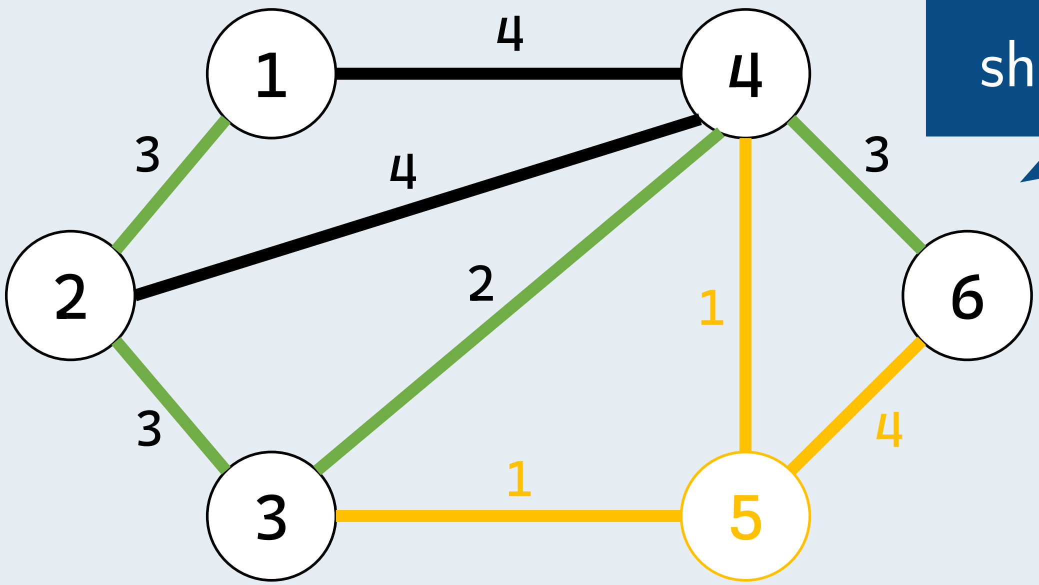


Problem 3.b

Given MST of a graph, find the new MST if a new vertex Y and a set of edges connecting Y are added.

Question: Can we simply run one step of Prim's algorithm, and simply pick the smallest edge among the newly added edges?

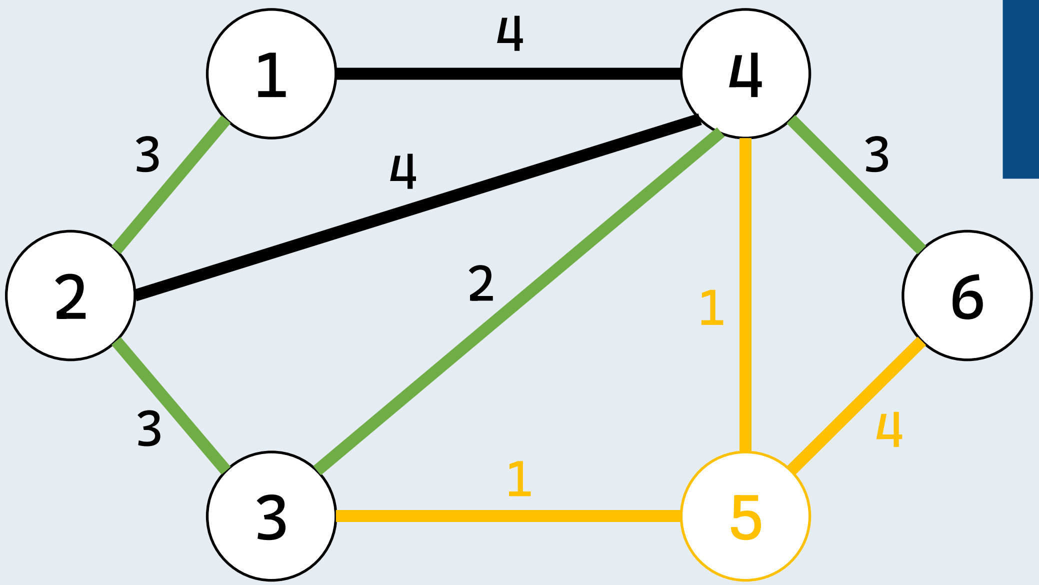
The edge `(3, 4)` shouldn't be in new MST!



Problem 3.b

Given MST of a graph, find the new MST if a new vertex Y and a set of edges connecting Y are added.

Trivial Solution: Re-run Prim's or Kruskal's algorithm on the new graph, $O(E \log V)$ time.

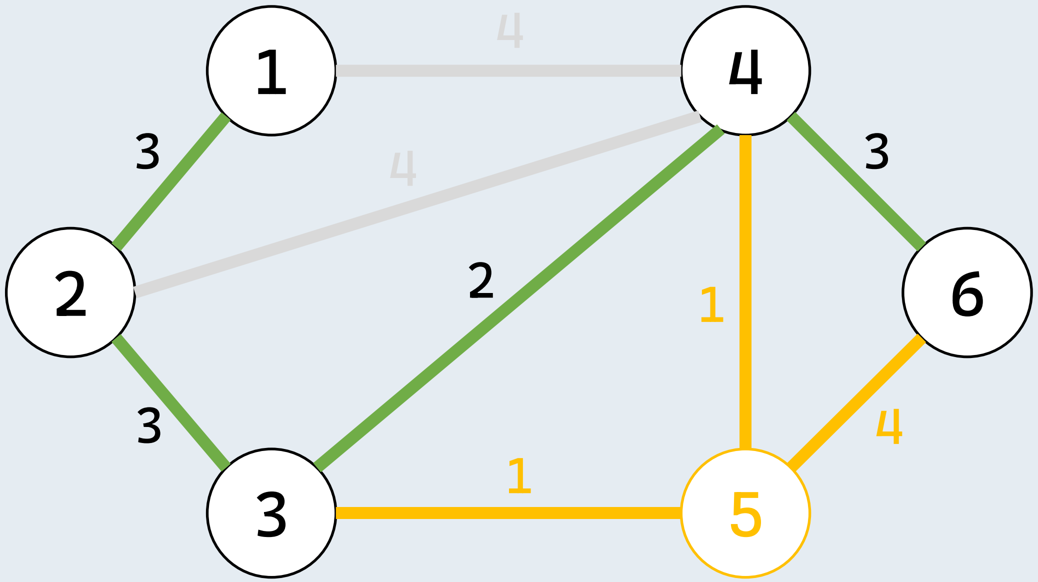


Do we still need to consider all edges?

Problem 3.b

Observation: the edges not in the original MST will not be in the new MST.

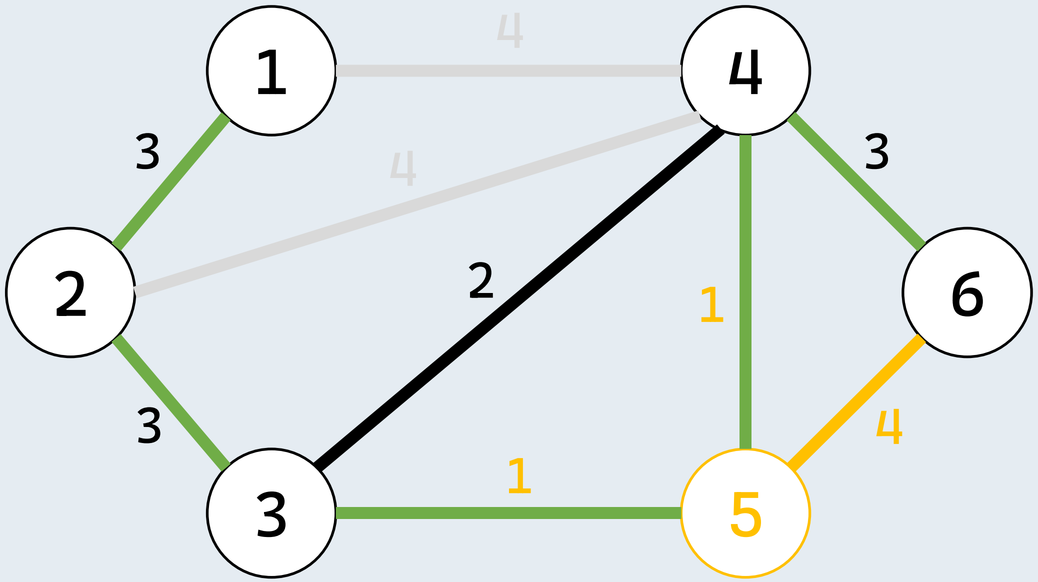
Better Solution: We don't need to consider those edges that are originally not in MST. $O(V \log V)$ time.



Problem 3.b

Observation: the edges not in the original MST will not be in the new MST.

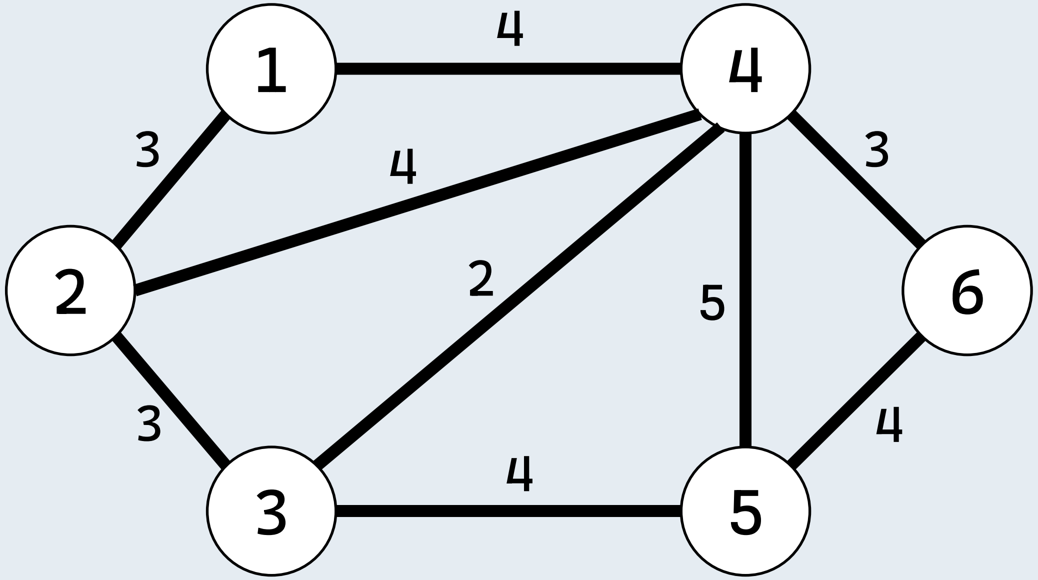
Better Solution: We don't need to consider those edges that are originally not in MST. $O(V \log V)$ time.



Problem 4

Given a graph, find edges with sum at most b that minimizes number of connected components k.

Example: $b = 15$.

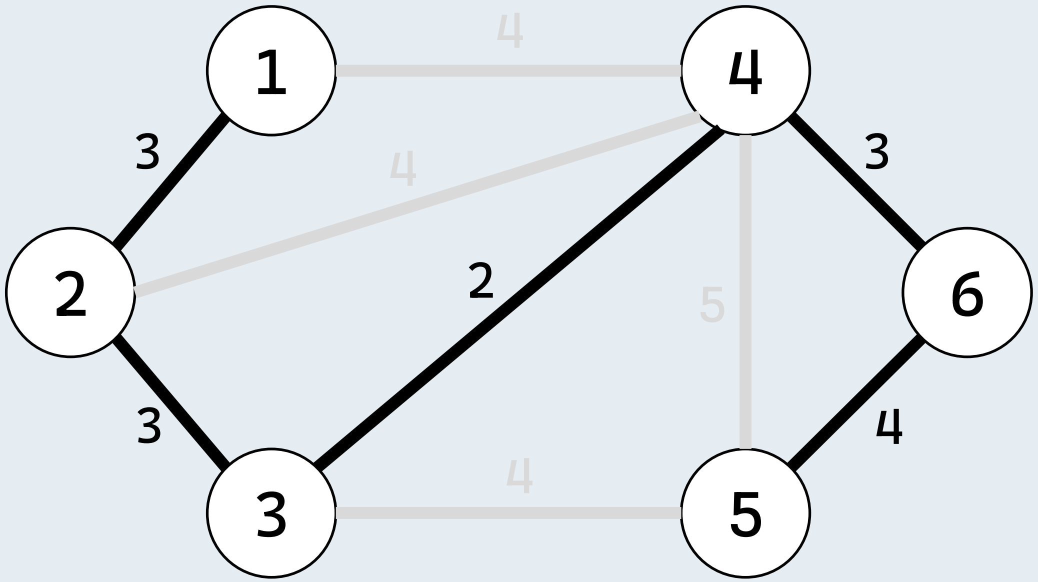


Problem 4

Given a graph, find edges with sum at most b that minimizes number of connected components k.

Example: $b = 15$.

We can simply choose all edges in the MST. $k = 1$.

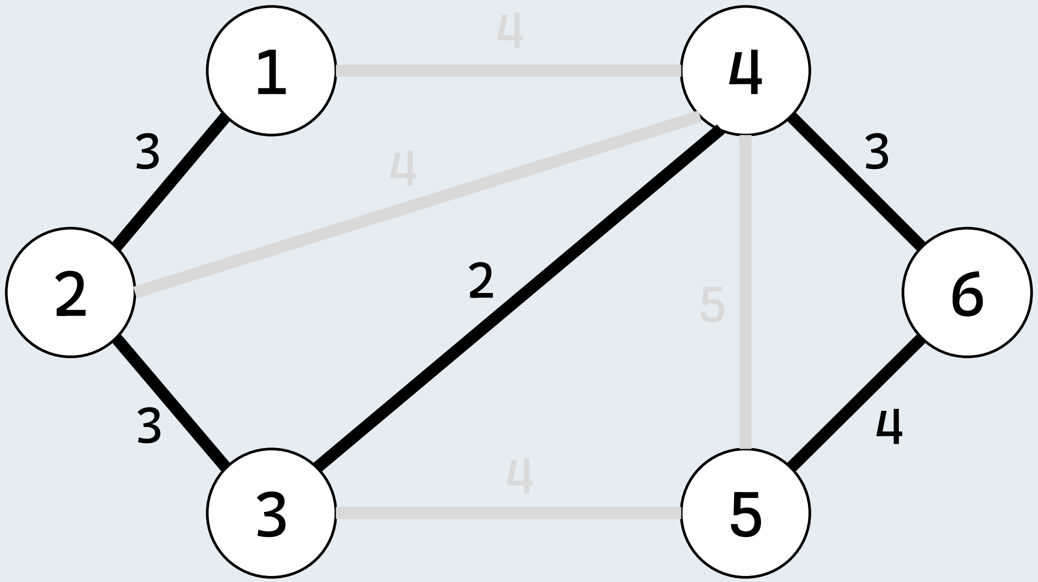


Problem 4

Given a graph, find edges with sum at most b that minimizes number of connected components k.

Example: $b = 8$.

We need to delete some edges...

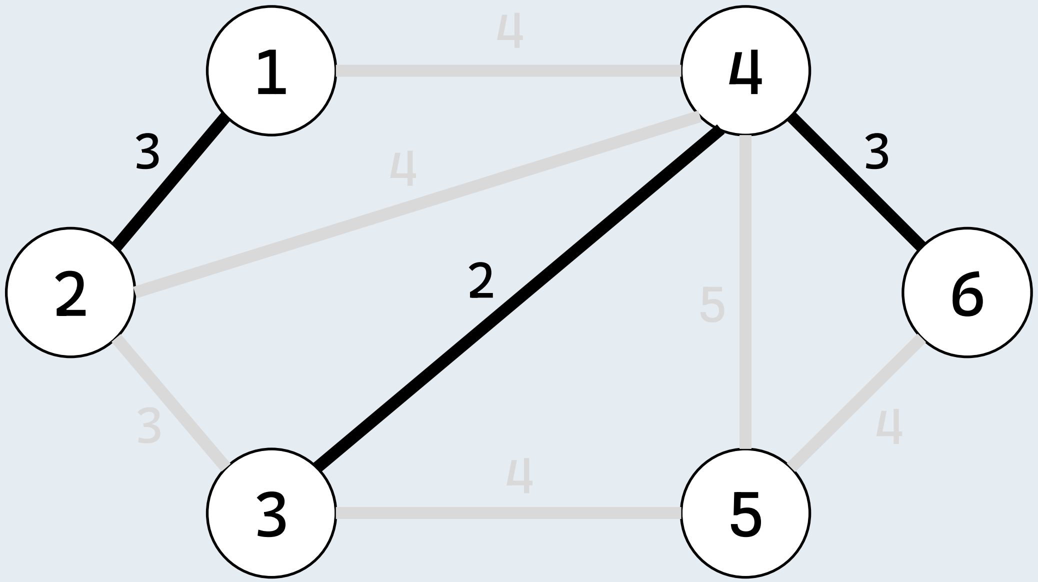


Problem 4

Given a graph, find edges with sum at most b that minimizes number of connected components k.

Example: $b = 8$.

We need to delete at least 2 edges... leaving $k = 3$.

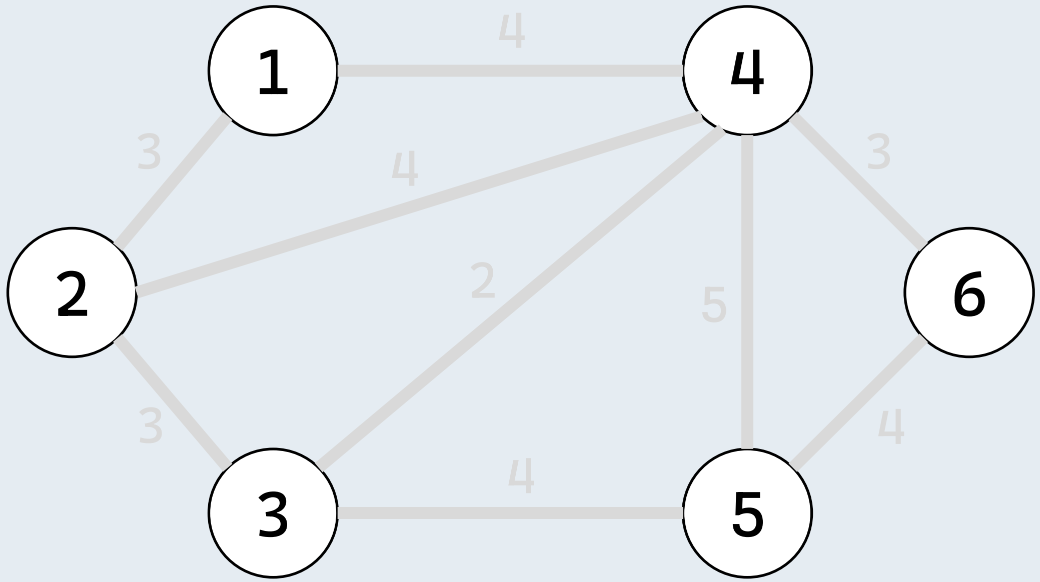


Problem 4

Observation: each deletion from MST adds one more connected component.

Idea: keep as many edges from MST as possible...

- Use Kruskal's algorithm!
- greedily add the smallest edges, until total weight reaches b!

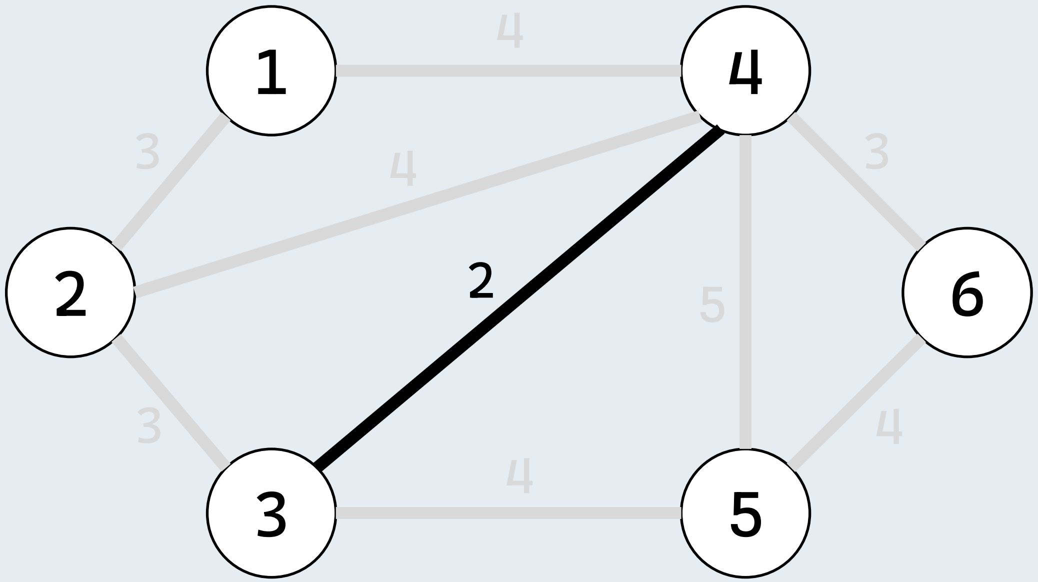


Problem 4

Observation: each deletion from MST adds one more connected component.

Idea: keep as many edges from MST as possible...

- Use Kruskal's algorithm!
- greedily add the smallest edges, until total weight reaches b!

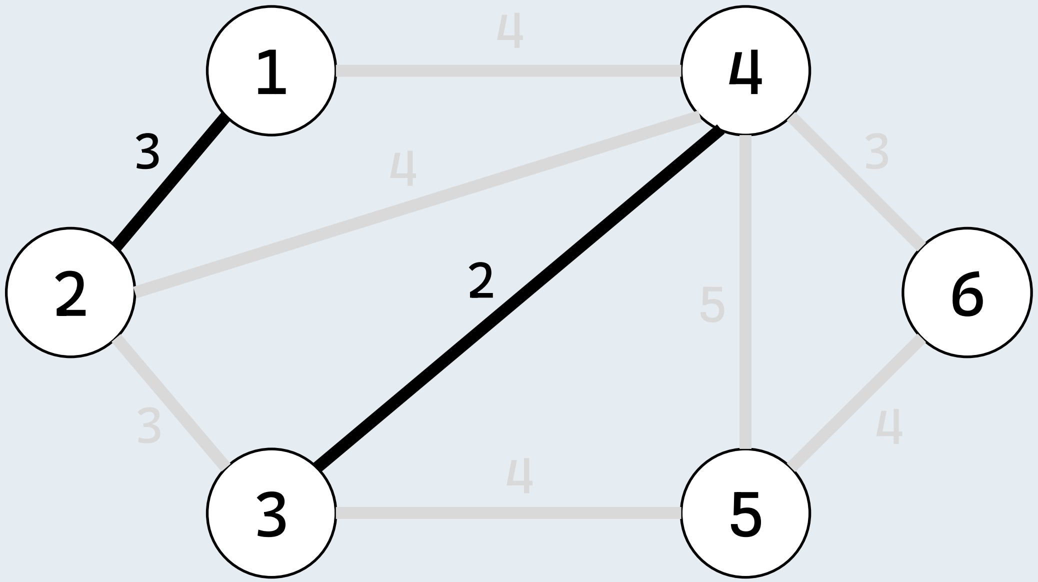


Problem 4

Observation: each deletion from MST adds one more connected component.

Idea: keep as many edges from MST as possible...

- Use Kruskal's algorithm!
- greedily add the smallest edges, until total weight reaches b!

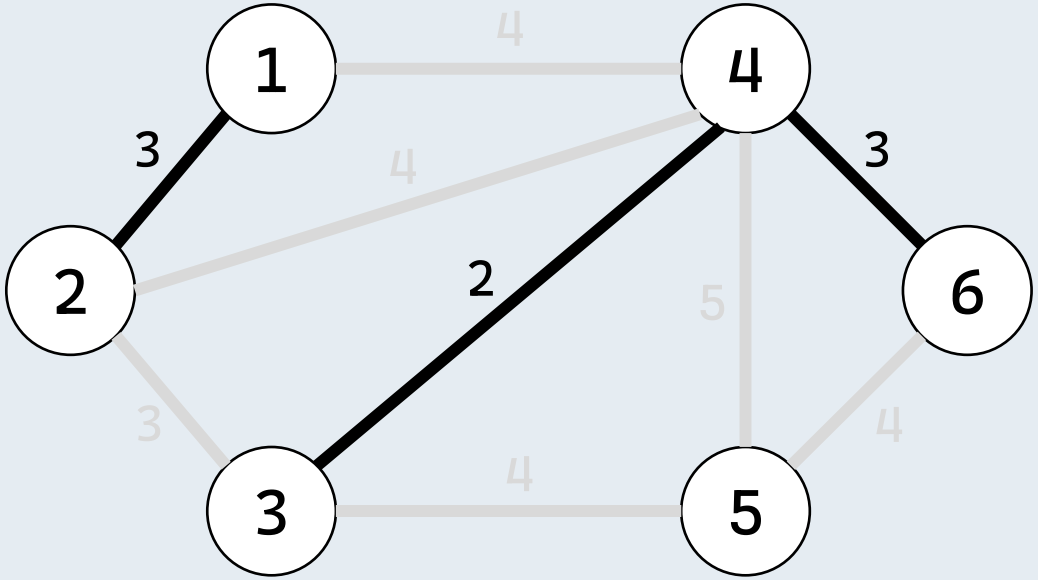


Problem 4

Observation: each deletion from MST adds one more connected component.

Idea: keep as many edges from MST as possible...

- Use Kruskal's algorithm!
- greedily add the smallest edges, until total weight reaches b!



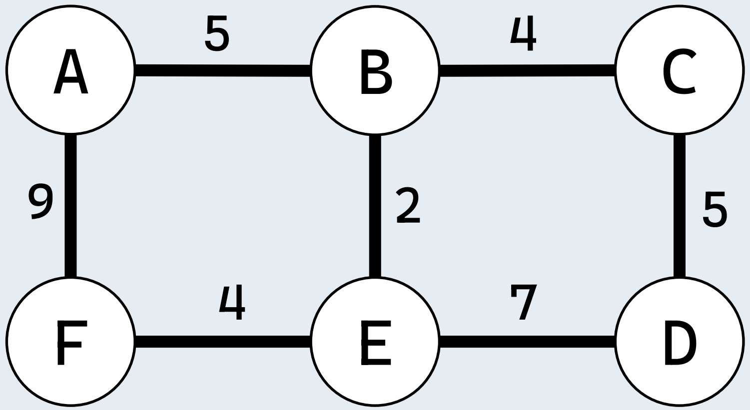
Applications of MST

How to utilize MST and its variants?

Problem 5

Given an undirected graph, find a set of edges such that for every cycle in the graph, at least one edge in the set, and the total weight of the selected edges is minimized.

Example: Edge `(B, E)`, `(B, C)` would be covering all cycles and total weight 6 is minimized.



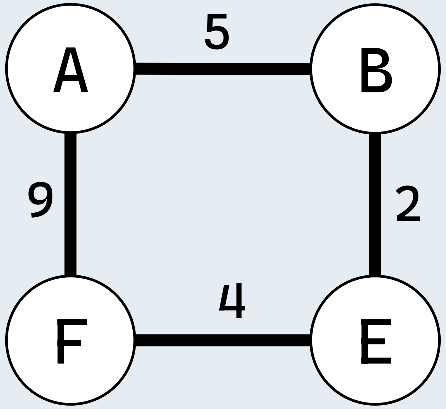
Problem 5

Simplified Example:

Suppose we have only one cycle...

We can simply pick the edge with minimum weight.

Idea: for each cycle in graph, select the edge with minimum weight!

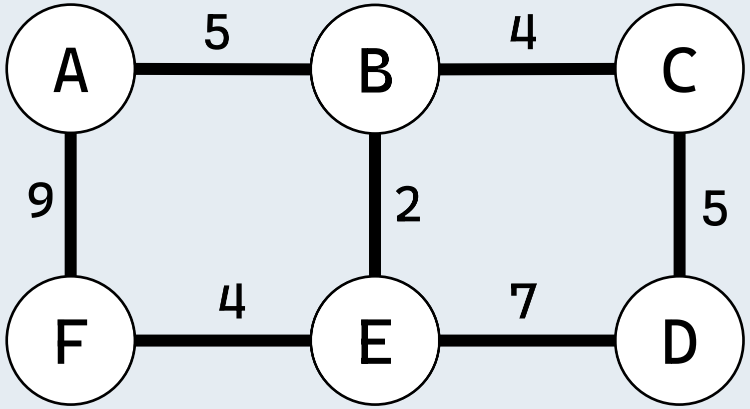


Problem 5

Trivial Answer:

- Run cycle detection algorithm, e.g. DFS.
- When a cycle is detected, find smallest edge within the cycle.
- delete the edge and repeat.

Will take $O\big(|E|(|V| + |E|)\big)$ time.

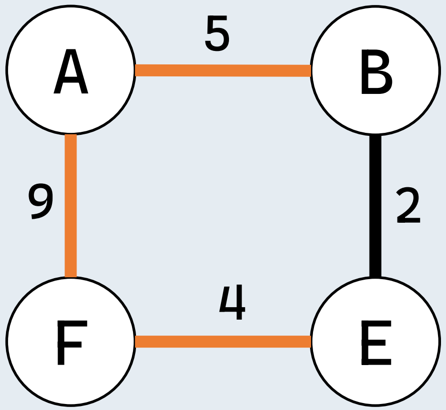


Problem 5

Recall: in minimum spanning tree: in every cycle, the largest edge is not included (cycle property).

In ***maximum spanning tree***: in every cycle, the smallest edge is not included!

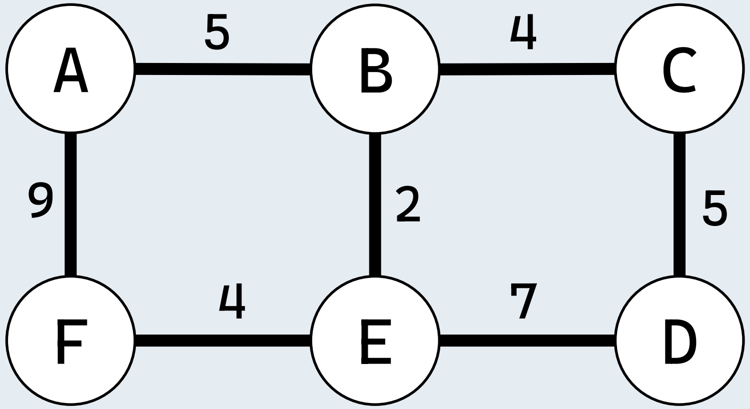
Idea: find the maximum spanning tree instead. The edges not included are what we need.



Problem 5

Better Answer:

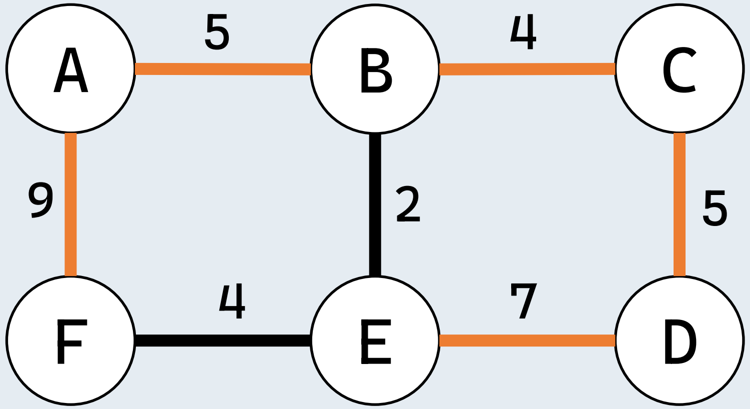
- Run Kruskal's algorithm, with edges in descending order (from largest to smallest)



Problem 5

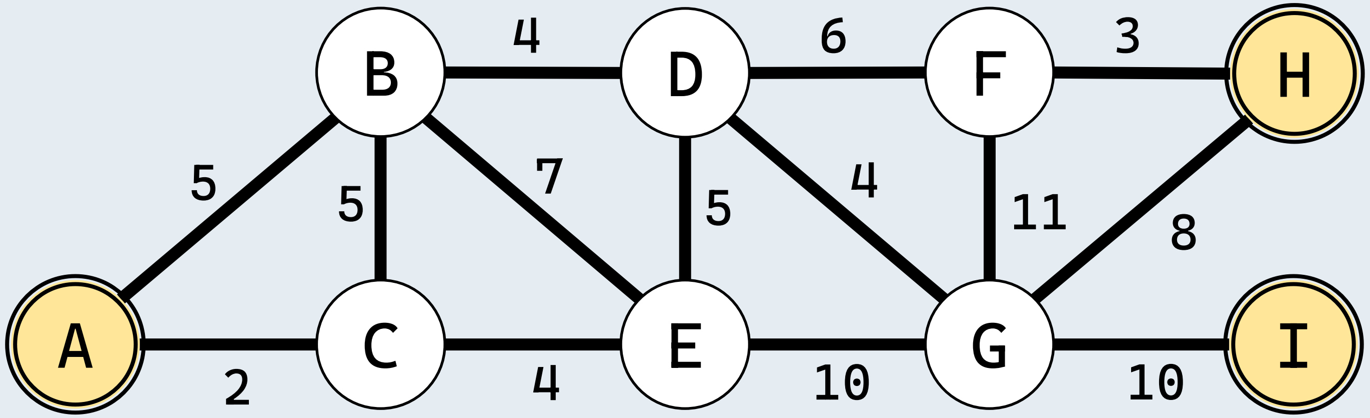
Better Answer:

- Run Kruskal's algorithm, with edges in descending order (from largest to smallest)
- The edges that are **not** in the maximum spanning tree are selected.



Problem 6

- We have a graph, some vertices are “power plants”.
- Find a set of edges that connects all vertices to at least one “power plant”.

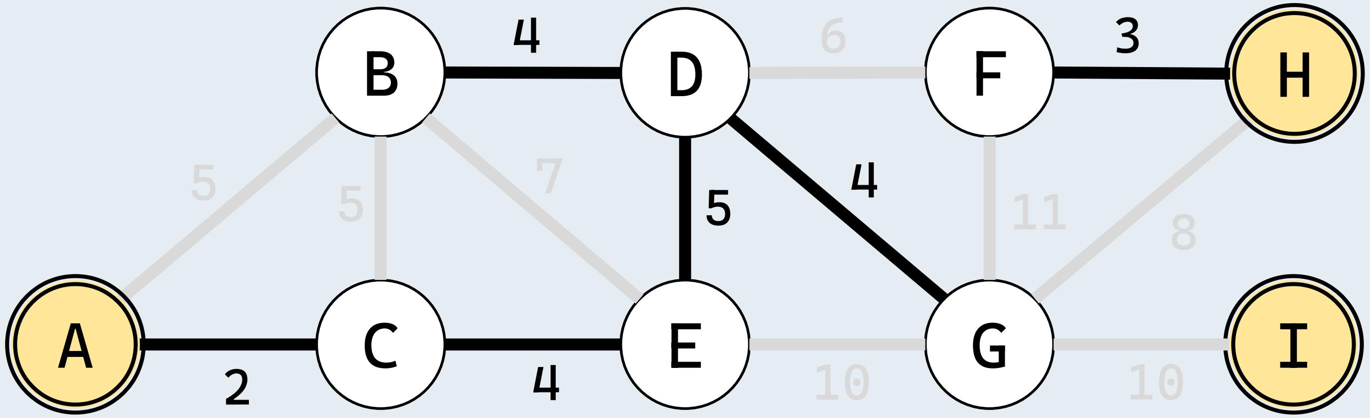


Problem 6

- We have a graph, some edges are “power plants”.
- Find a set of edges that connects all vertices to at least one “power plant”.

Goal: find ***minimum spanning forest***, with each connected component containing at least one “power plant”.

How to find a MST containing A?

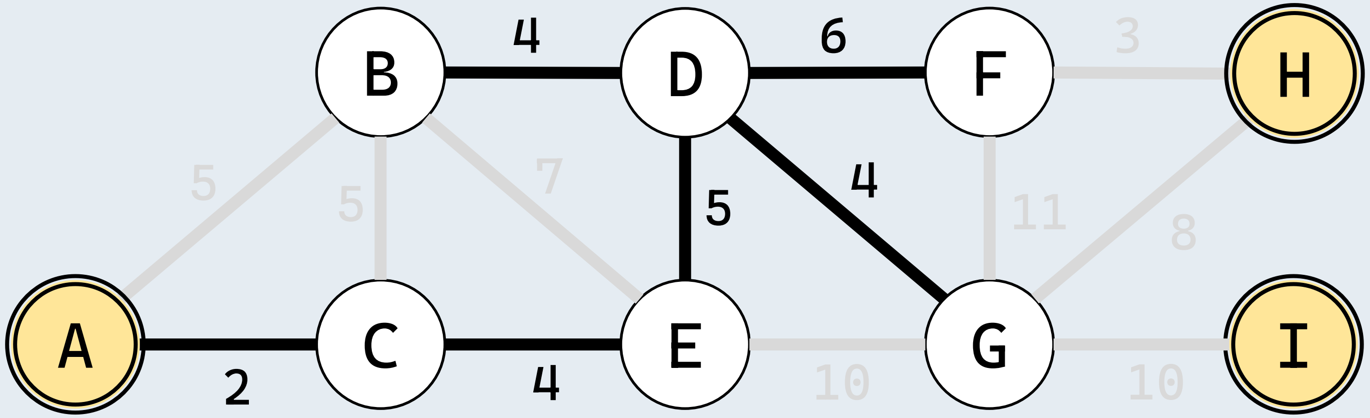


Problem 6

Attempt:

- Use Prim's algorithm, starting from a power plant. (e.g. vertex `A`)
- We can get a MST starting from A...

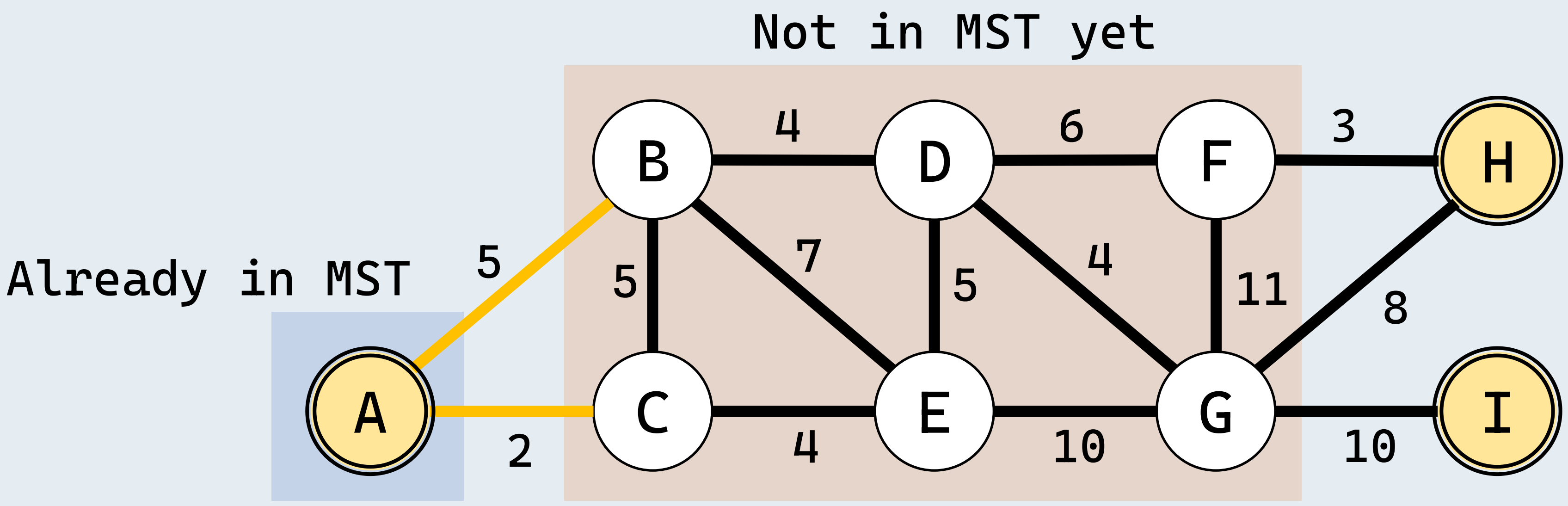
This is not optimal!



Problem 6

What we are essentially doing is:

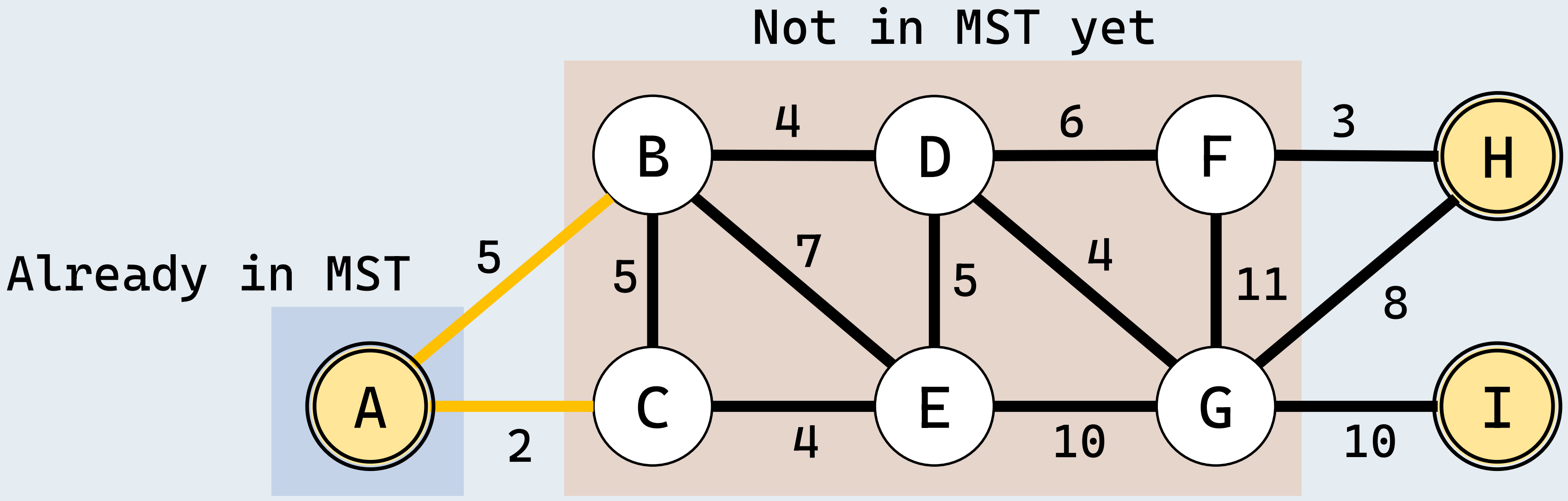
- partitioning A and remaining vertices into two sets,
- find the smallest edge in the cut,



Problem 6

What we are essentially doing is:

- partitioning A and remaining vertices into two sets,
- find the smallest edge in the cut,

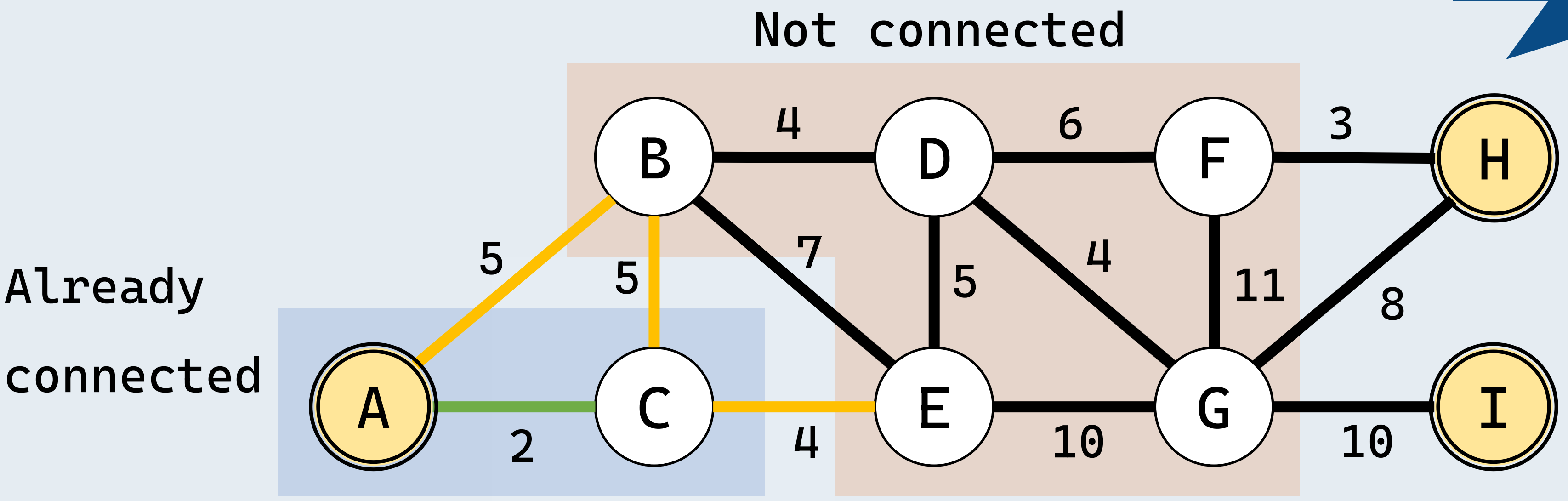


Problem 6

What we are essentially doing is:

- partitioning A and remaining vertices into two sets,
- find the smallest edge in the cut,
- include that edge in MST... then continue.

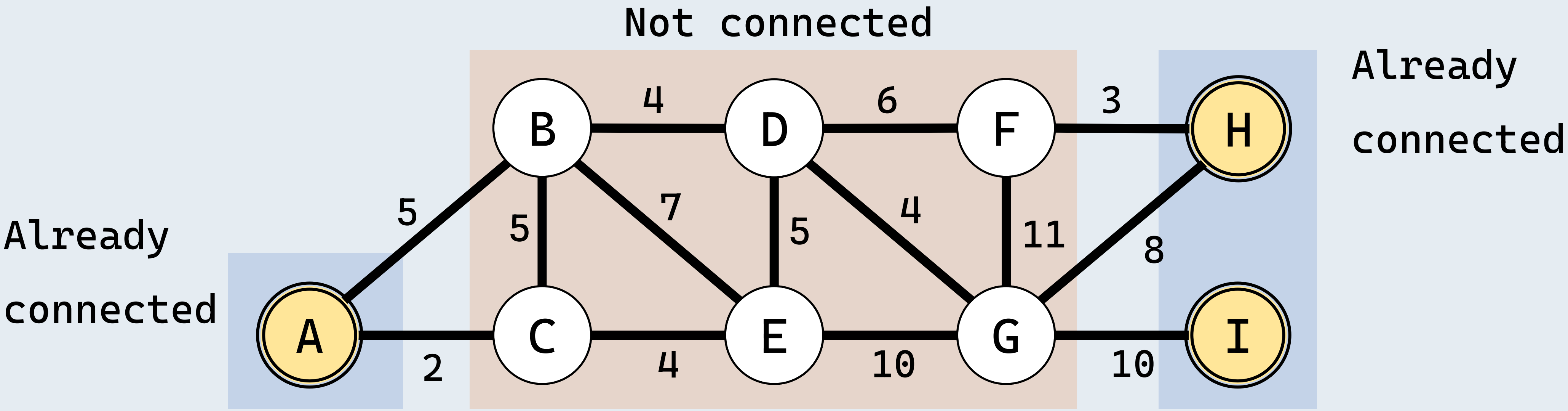
How to take those into account?



Problem 6

Idea:

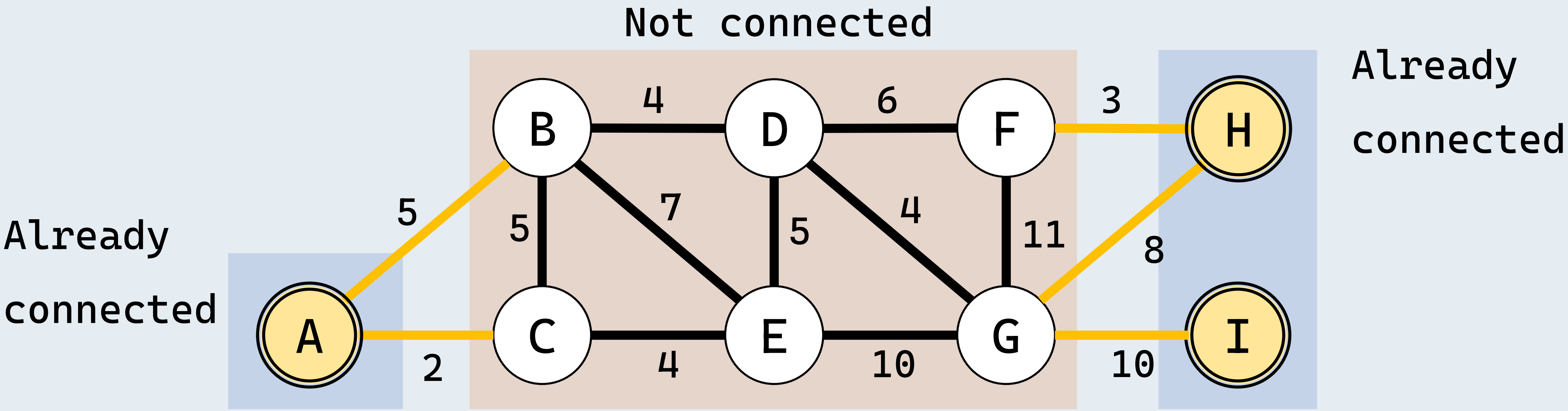
- We can consider all "power plants" as connected.



Problem 6

Idea:

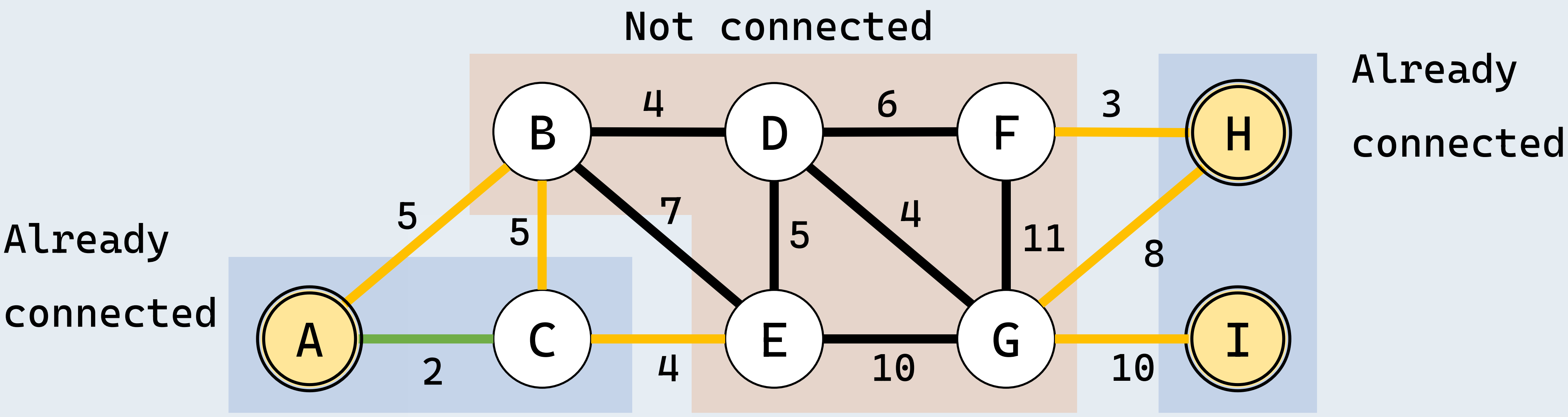
- We can consider all "power plants" as connected.
- Find the min edge in the larger cut instead!



Problem 6

Idea:

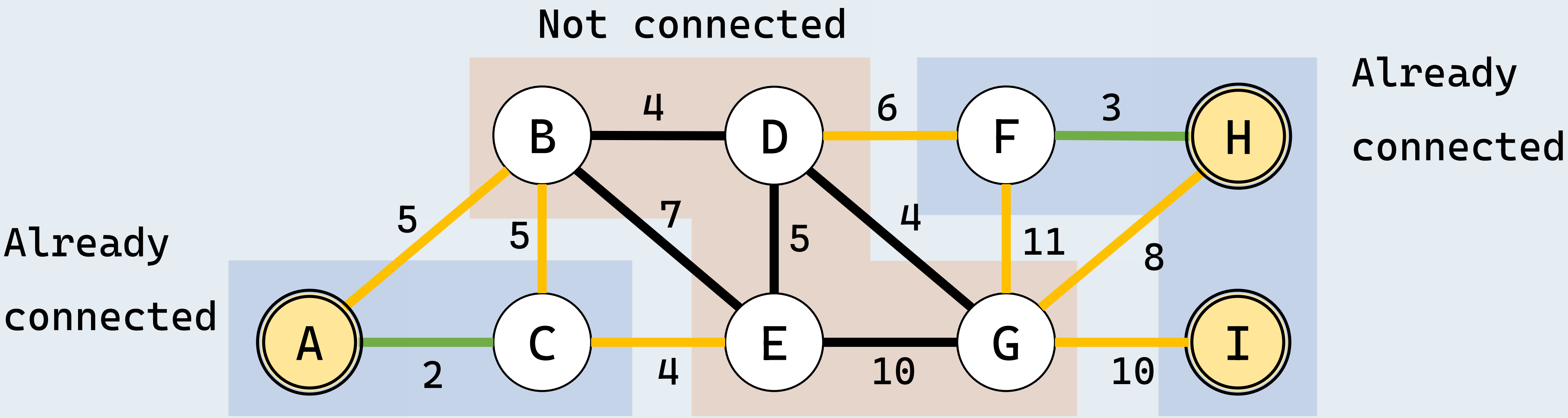
- We can consider all "power plants" as connected.
- Find the min edge in the larger cut instead!



Problem 6

Idea:

- We can consider all "power plants" as connected.
- Find the min edge in the larger cut instead!



End of File

Thank you very much for your attention :-)