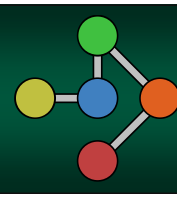
Graph Spanning Trees

Part 15

1

Spanning Trees

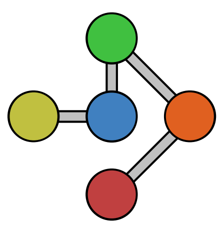


Connect the Dots (la la la la)

2

Spanning Trees

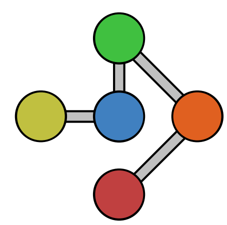
- In many cases, we want to make sure that every vertex is connected in a graph
- A *Spanning Tree* includes every vertex of the original graph (an no cycles, obviously)



3

Why Spanning Trees?

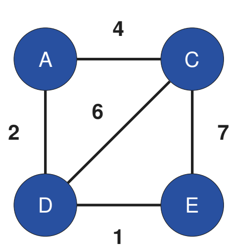
- Often, we want to make sure all vertices are connected
- These can be anything from cities (connected by roads) to computers (connected by cables)
- ... just to name a few…



4

Example Graph

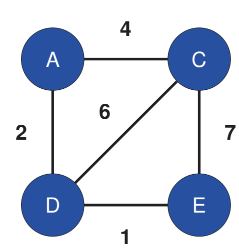
- The graph on the left is quite simple – containing just 4 vertices
- We need to create a tree that contains all 4 vertices



5

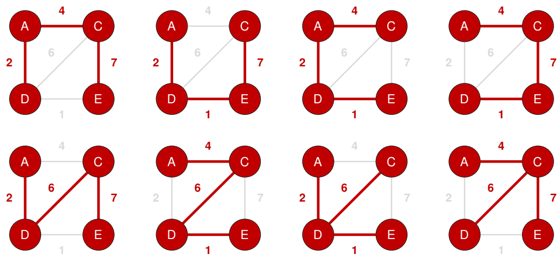
Example Graph

- As you can imagine, most graphs have multiple solutions
- Also note that this graph is weighted
- This can make some solutions better than others



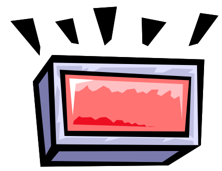
6

Example: Spanning Trees



7

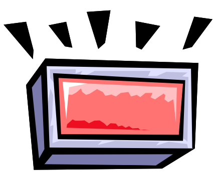
So How Many Spanning Trees?



- Let's look at the most complex graph possible – a complete one
- In this one, every vertex is connected to every other vertex
- *This results in a **large** number of possible solutions*

8

So How Many Spanning Trees?



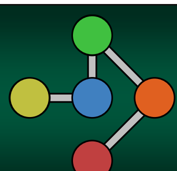
- Given a complete graph of *N* vertices, there are N^{N-2} possible spanning trees
- Yes, that is N to the power of N – which is even *worst* than exponential growth!

9

Complete Graph Spanning Trees

Vertices	Total Spanning Trees	Vertices	Total Spanning Trees
1	1	7	16,807
2	1	8	262,144
3	3	9	4,782,969
4	16	10	100,000,000
5	125	11	2,357,947,691
6	1,296	12	61,917,364,224

10



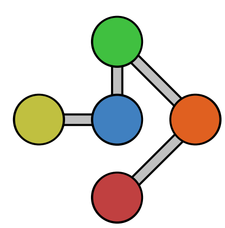
Minimum Spanning Tree

Getting the "Best" Tree

11

Minimum Spanning Tree

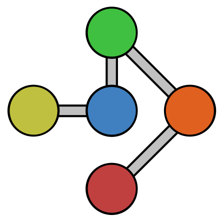
- *Minimum Spanning Tree (MST)* contains all the vertices of a graph with the minimum possible weight
- Naturally, depending on the weights, some solutions are more desirable than others



12

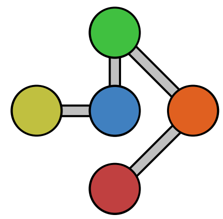
Minimum Spanning Tree

- Some uses (minimize cost)
  - building cable networks
  - building a road that joins cities
- So, creating an algorithm – that computes these trees - *is of vital importance*



13

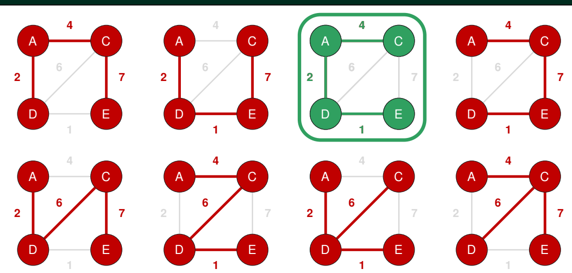
Uniformly Weighted Graph



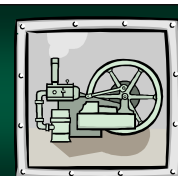
- If each edge has the same weight, then the computation of the minimum spanning tree is trivial
- A breadth-first or depth-first search can be used
- …but many are weighted

14

Example: Minimal Spanning Tree



15



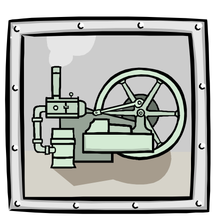
Brute Force

Doing in the hard way
(and impossible)

16

Brute Force

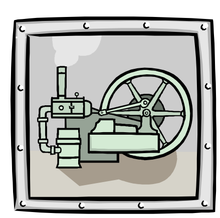
- One way to compute the minimum spanning tree is to simply ***try all of them!***
- Approach:
  - calculate every tree
  - keep track of the tree with the minimum total weight



17

Brute Force

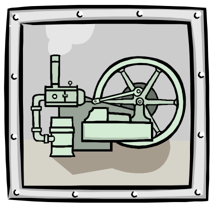
- Calculating just one tree will require O(n) time where *n* is the total number of vertices
- How many trees are there?
- For complete graphs, we know it's n^{n-2} – which is what we must use for Big-O



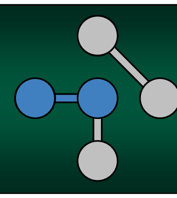
18

Brute Force

- So, there is a $n \times n^{n-2}$ computational requirement
- Which is $O(n^{n-1})$
- Naturally, this is a poor – and ***impracticable*** – solution



19



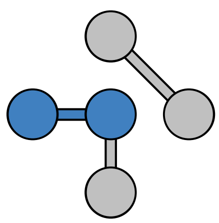
Kruskal's Algorithm

Computing a Minimum Spanning Tree

20

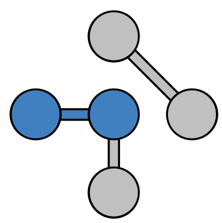
Kruskal's Algorithm

- *Kruskal's Algorithm* computes a minimum spanning tree
- It was invented in 1956 by *Joseph Kruskal* and published in *Proceedings of the American Mathematical Society*



21

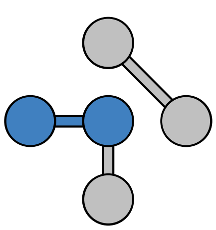
Kruskal's Algorithm



- It conceptually starts with a forest of single vertices
- It then adds edges – with the minimum weight – to connect the two vertices (and possible subtrees)

22

Kruskal's Algorithm

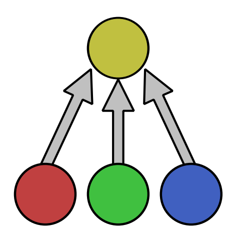


- So, it adds edges in sorted order of weight
- If an edge would cause a cycle, then it is rejected and not part of the solution
- Otherwise, add it.

23

Kruskal's Algorithm

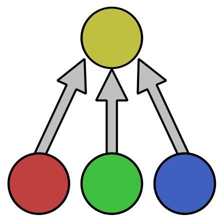
- Cycles are avoided by using a **union-find** to merge subtrees
- Remember: Union-Find allows objects to be unioned (grouped together)



24

Kruskal's Algorithm

- If any two vertices have the same "representative", then they are already connected.
- So, don't add the edge – it would create a cycle



25

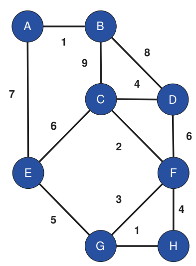
Kruskal's Algorithm

```
Create a list of edges sorted by weight
Add each vertex to a Union-Find
loop through the list
   if the vertices have different representatives
      Add the edge
      Union the two vertices (in the Union-Find)
   end if
end loop
```


26

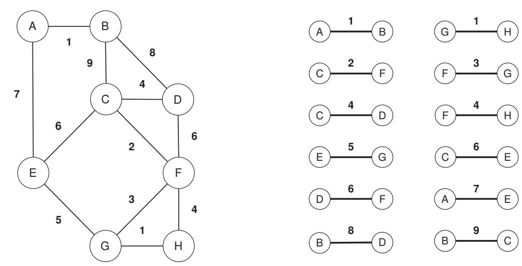
Kruskal's Algorithm Example

- The following is a graph with multiple weights
- Kruskal's Algorithm will create a list of the edges sorted by weight



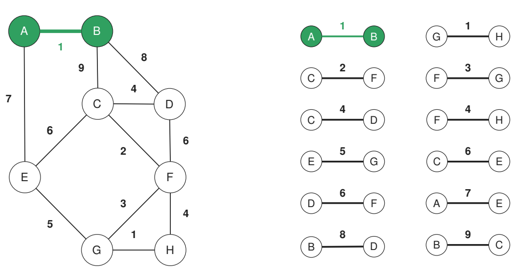
27

Kruskal's Algorithm Example



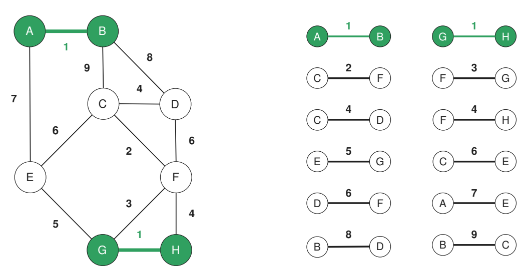
28

Add Edge

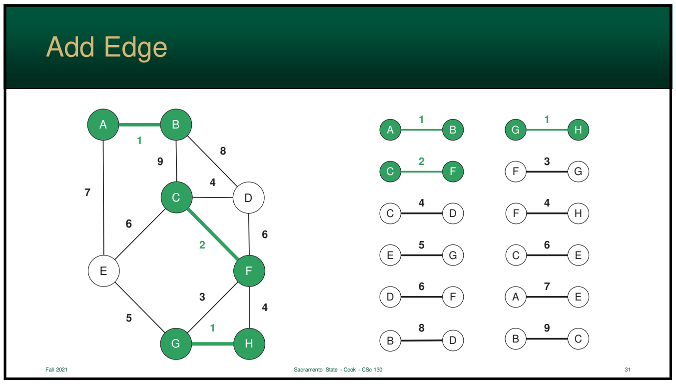


29

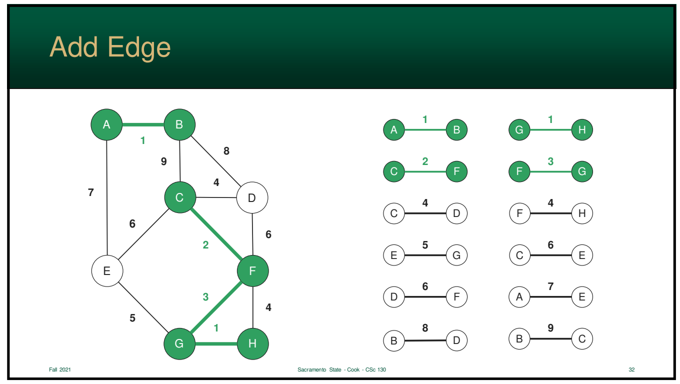
Add Edge



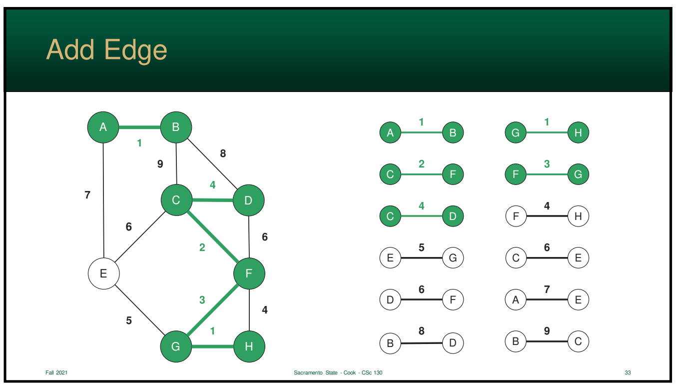
30



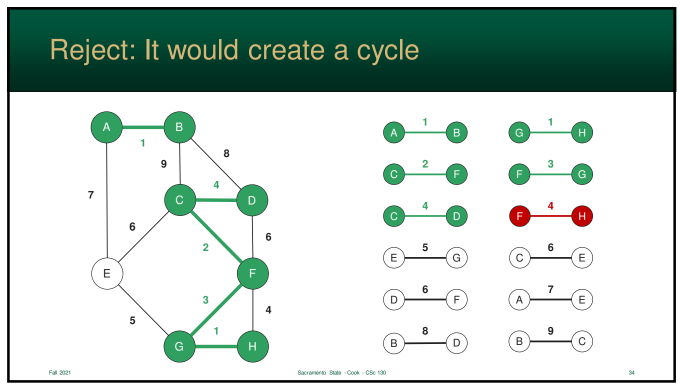
31



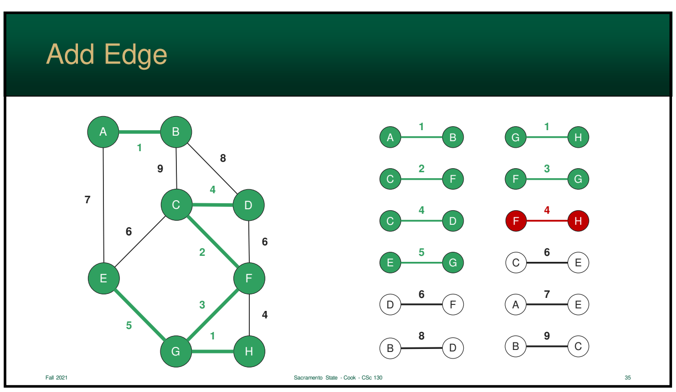
32



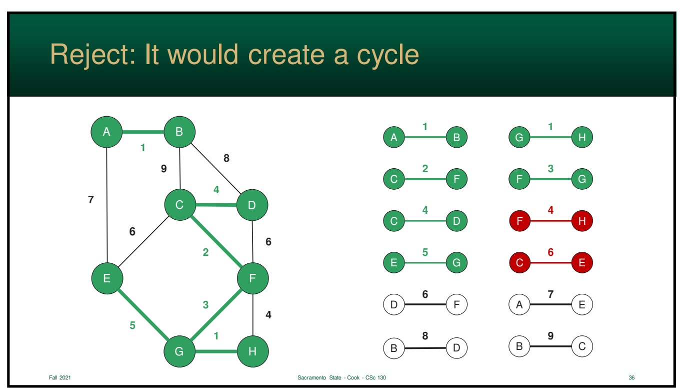
33



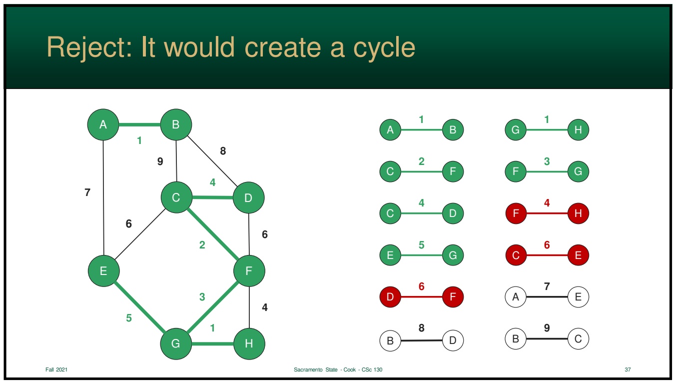
34



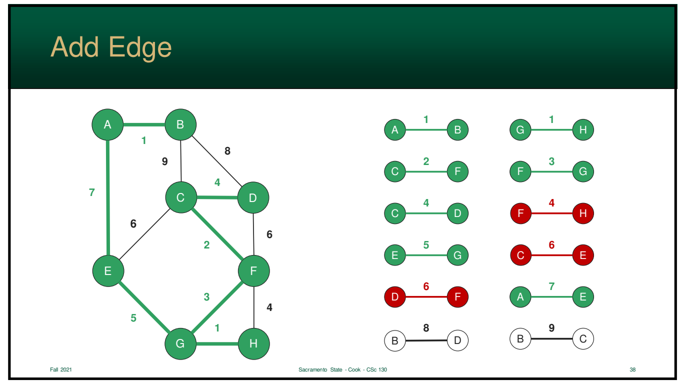
35



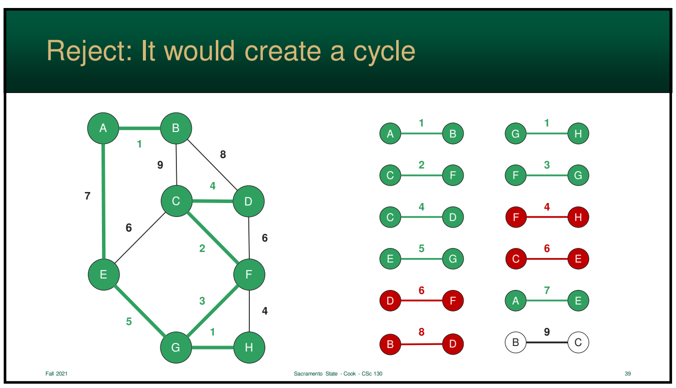
36



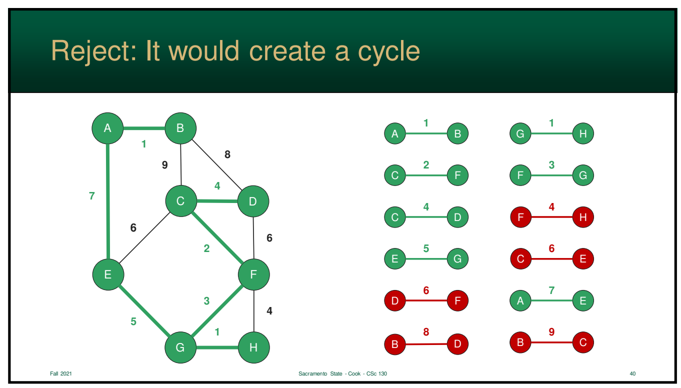
37



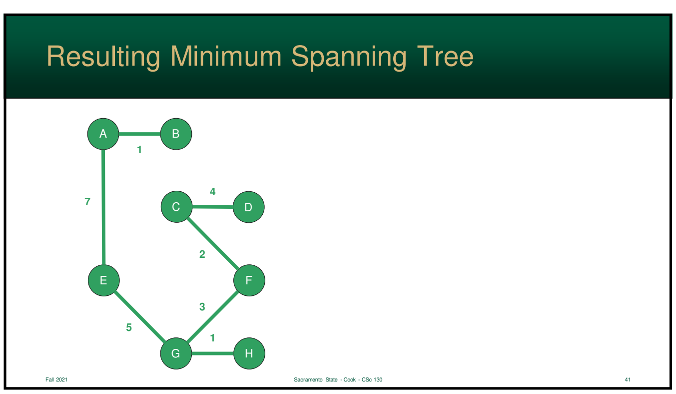
38



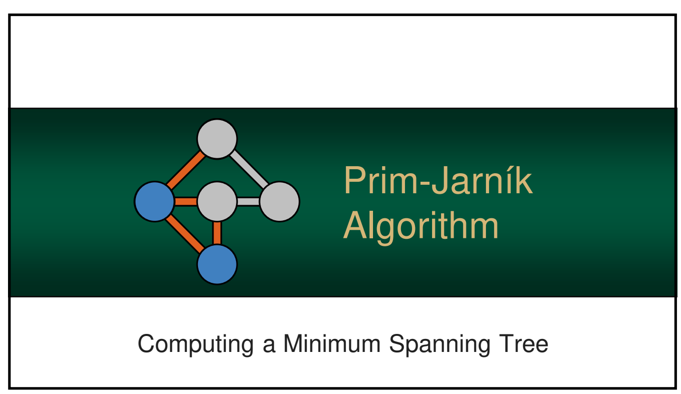
39



40



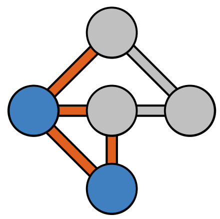
41



42

Prim-Jarník Algorithm

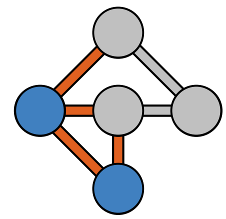
- *Prim's Algorithm* computes a minimum spanning tree
- It was invented in 1956 by computer scientist *Robert C. Prim*
- … same year as Kruskal



43

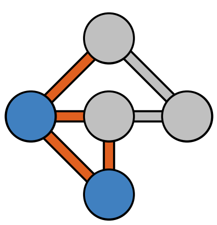
Prim-Jarník Algorithm

- However, it was discovered that computer scientist *Vojtěch Jarník* had also invented it in 1930
- So, it is also called the *Prim-Jarník Algorithm*



44

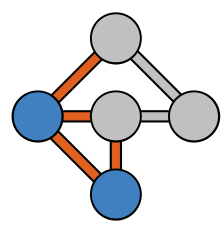
Prim-Jarník Algorithm



- This algorithm starts with an *arbitrarily chosen*, vertex
- The algorithm then adds the minimal edge – that the entire current tree can currently "see"

45

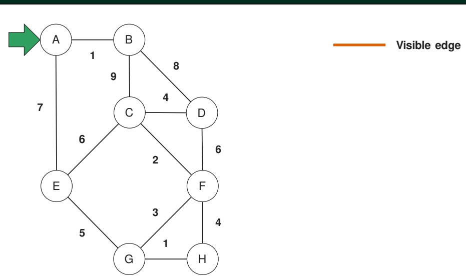
Prim's Algorithm



- So, it checks all the edges connect to unvisited vertices
- The algorithm is *greedy* – always following the local optimal path – to grow the tree

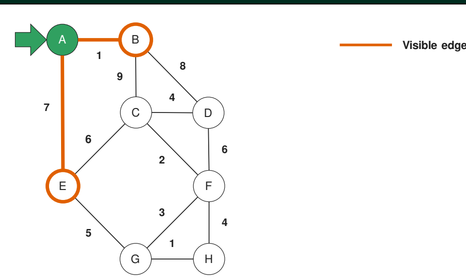
46

Prim's Algorithm Example

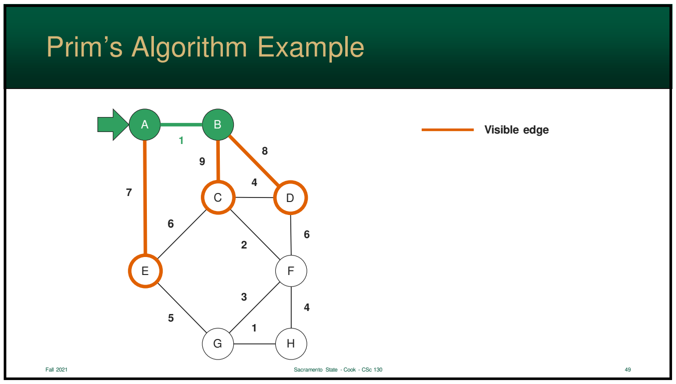


47

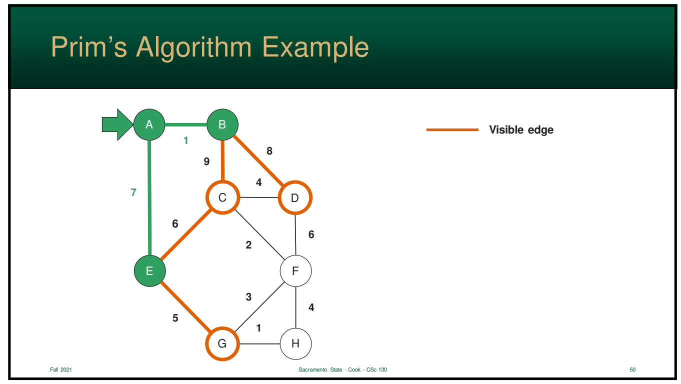
Prim's Algorithm Example



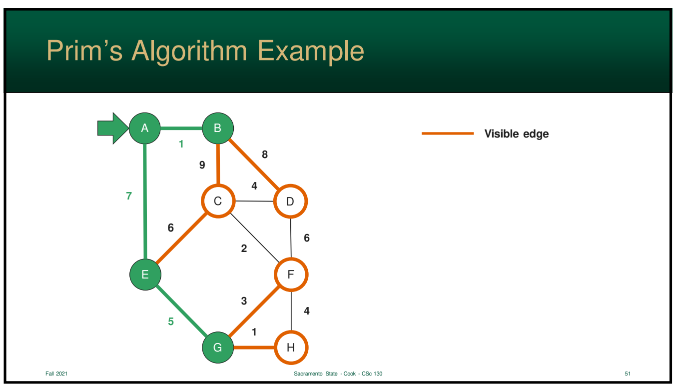
48



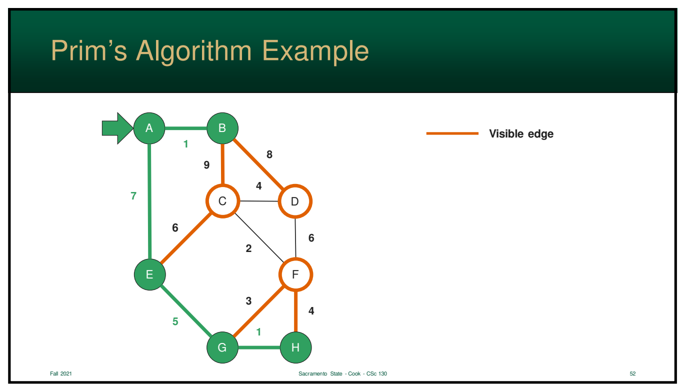
49



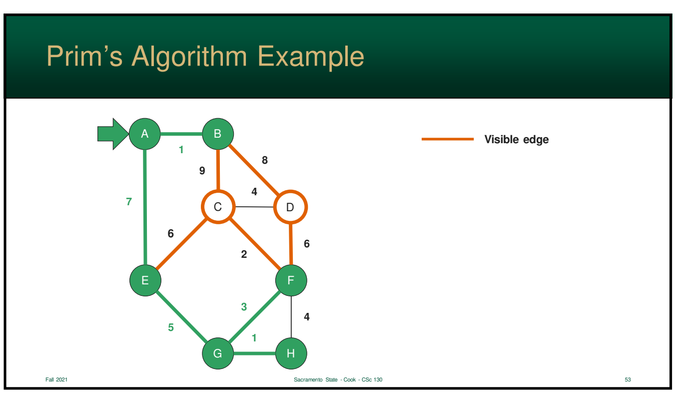
50



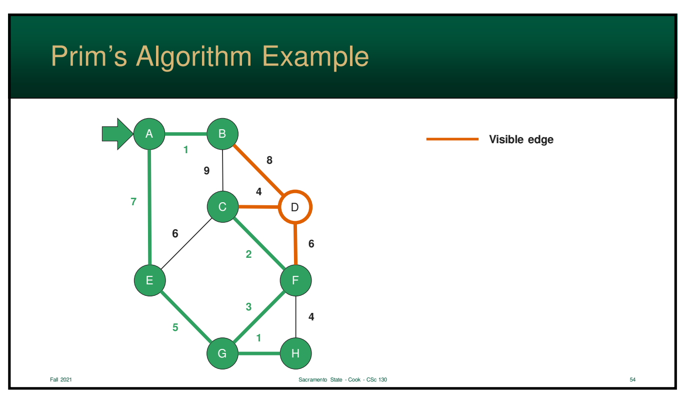
51



52

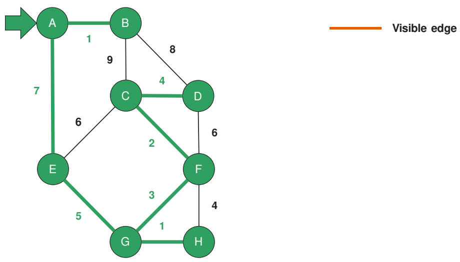


53



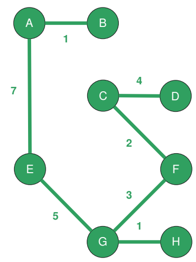
54

Prim's Algorithm Example



55

Resulting Minimum Spanning Tree



56

Kruskal vs. Prim-Jarník

- Both algorithms find the minimum spanning tree
  - *though not necessarily identical*
  - *… if multiple, equal, solutions exist (they might)*
- Both are O(|E| log |V|) where |E| is the number of edges and |V| is the number of vertices
- Kruskal is far easier to conceptualize, but Prim is far easier to implement

57