# MINIMUM SPANNING TREES + COURSE RECAP

David Kauchak
CS62– Fall 2020

1

## Admin

Quiz 10 graded

Assignment 10 due Today

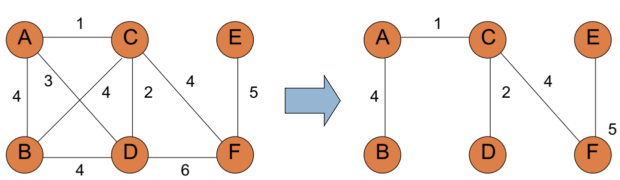
Last mentor hours tonight
- Will update office hours/mentor hours for next week

Final exam 12/4, 2-5pm (PST)
- 4 pages of notes (2 double-sided)
- sakai
- Will post zoom link
- If you can't take it then, let me know and you can take it any time that day

2

## Minimum spanning trees (MST)

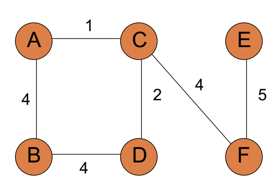
The lowest weight set of edges that connects all vertices of an undirected graph with positive weights




3

## MSTs

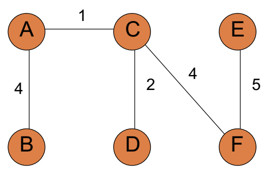
Can an MST have a cycle?




4

## MSTs

Can an MST have a cycle?




5

## Kruskal's algorithm

Given a partition S, let edge e be the minimum cost edge that **crosses** the partition. *Every* minimum spanning tree contains edge e.

Kruskals:
- Sort edges by increasing weight
- for each edge (by increasing weight):
  - check if adding edge to MST creates a cycle
  - if not, add edge to MST

6

## Why does Kruskal's work?

Never adds an edge that creates a cycle

Therefore, always adds lowest cost edge to connect two connected components. By min cut property, that edge must be part of the MST

Kruskals:
- Sort edges by increasing weight
- for each edge (by increasing weight):
  - check if adding edge to MST creates a cycle
  - if not, add edge to MST

Run-time: $O(VE+E^2)$ do this E times

7

## Kruskal's details

We can do better!

Uses a data structure called "disjoint set" to efficiently check whether adding an edge creates a cycle

Run-time: O(E log E) (bounded by the sort)

8

## Prim's algorithm

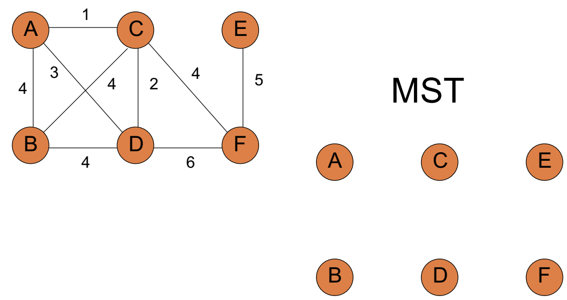
Greedily grow the MST starting at a vertex:

- Start with a random vertex and count that vertex as connected by the MST
- Add the edge with the smallest weight that connects a vertex not connected by the MST
- Repeat until we've added V-1 edges

9

## Prim's

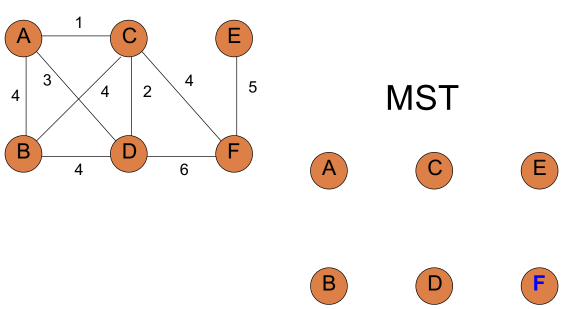
Add the edge with the smallest weight that connects a vertex not connected by the MST




10

## Prim's

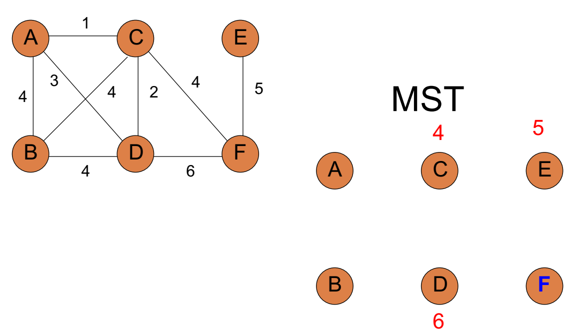
Add the edge with the smallest weight that connects a vertex not connected by the MST



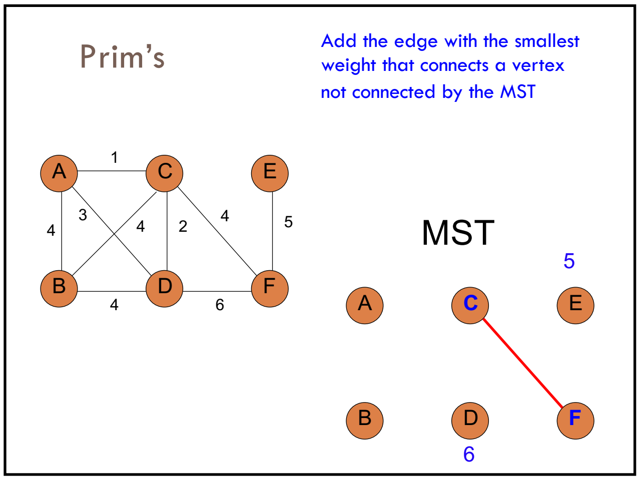

11

## Prim's

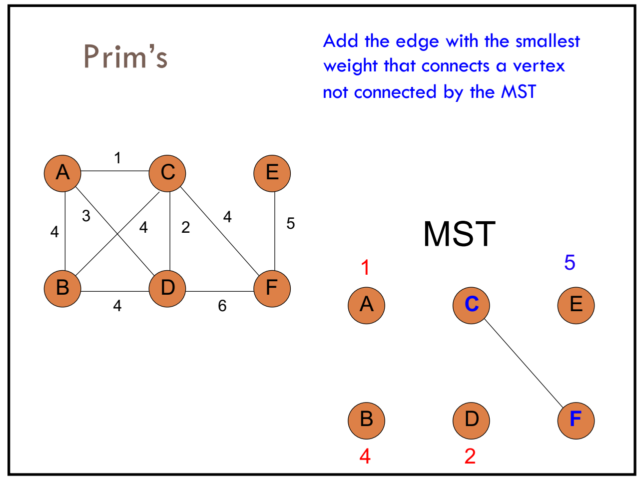
Add the edge with the smallest weight that connects a vertex not connected by the MST



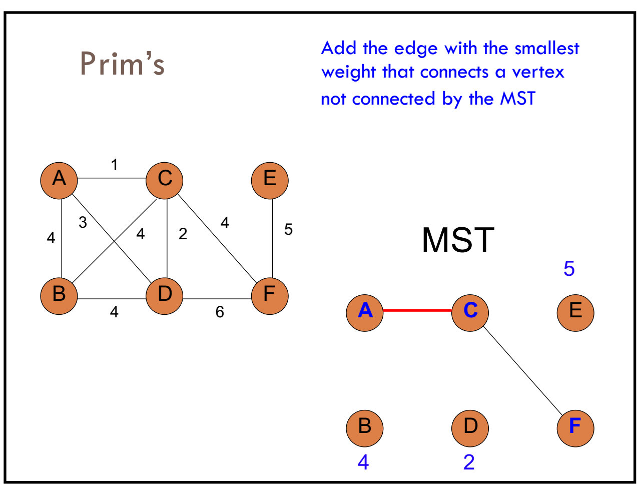

12


Prim's
Add the edge with the smallest weight that connects a vertex not connected by the MST
A C E B D F
1 3 4 4 2 4 5 4 6
MST
5
A C E
B D F
6

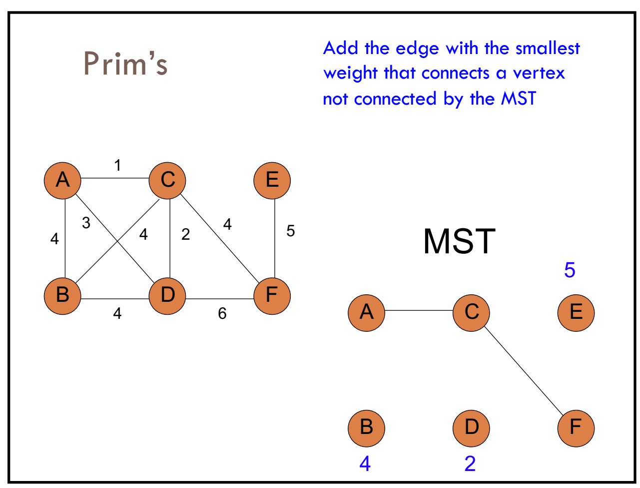
13


Prim's
Add the edge with the smallest weight that connects a vertex not connected by the MST
A C E B D F
1 3 4 4 2 4 5 4 6
MST
1 5
A C E
B D F
4 2

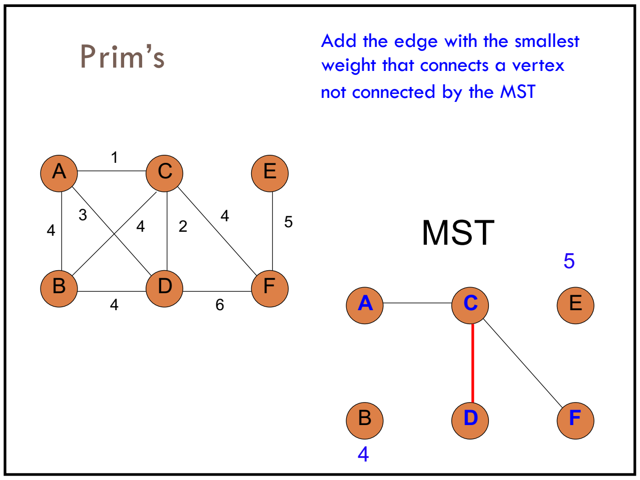
14


Prim's
Add the edge with the smallest weight that connects a vertex not connected by the MST
A C E B D F
1 3 4 4 2 4 5 4 6
MST
5
A C E
B D F
4 2

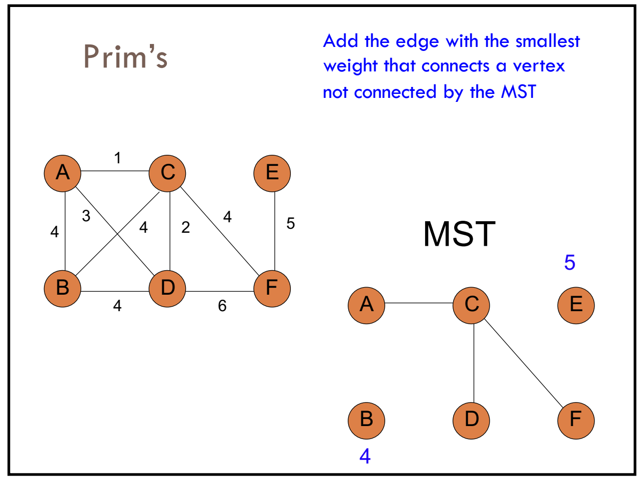
15


Prim's
Add the edge with the smallest weight that connects a vertex not connected by the MST
A C E B D F
1 3 4 4 2 4 5 4 6
MST
5
A C E
B D F
4 2

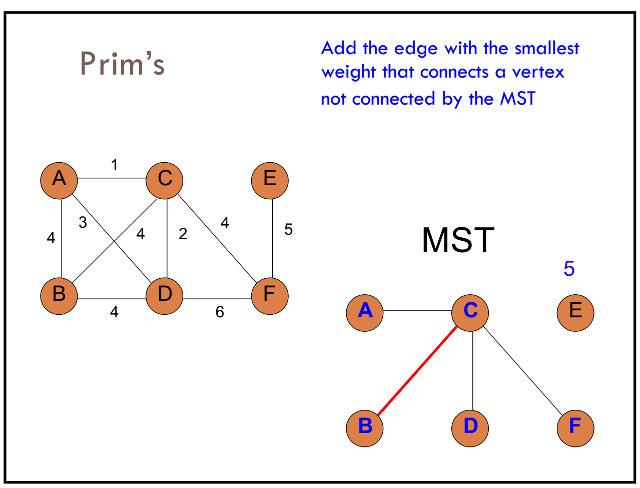
16


Prim's
Add the edge with the smallest weight that connects a vertex not connected by the MST
A
B
C
D
E
F
1
3
4
4
2
4
5
4
6
MST
5
4

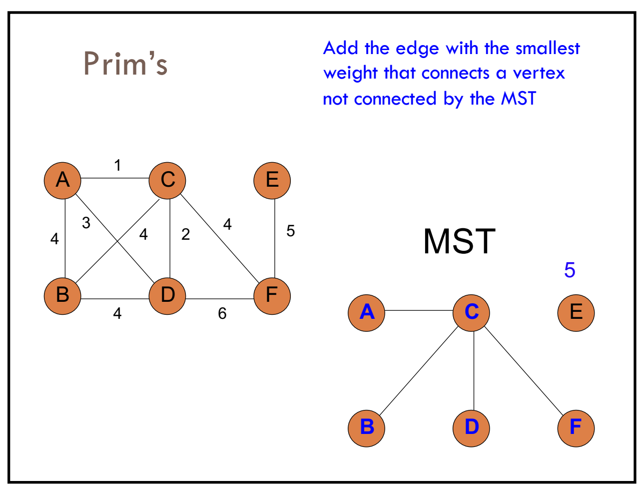
17


Prim's
Add the edge with the smallest weight that connects a vertex not connected by the MST
A
B
C
D
E
F
1
3
4
4
2
4
5
4
6
MST
5
4

18


Prim's
Add the edge with the smallest weight that connects a vertex not connected by the MST
A
B
C
D
E
F
1
3
4
4
2
4
5
4
6
MST
5

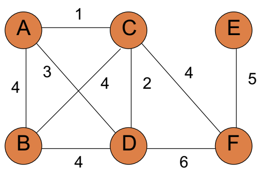
19


Prim's
Add the edge with the smallest weight that connects a vertex not connected by the MST
A
B
C
D
E
F
1
3
4
4
2
4
5
4
6
MST
5

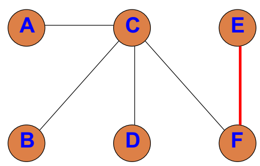
20

## Prim's

Add the edge with the smallest weight that connects a vertex not connected by the MST

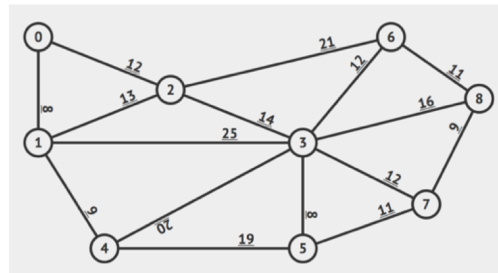



MST



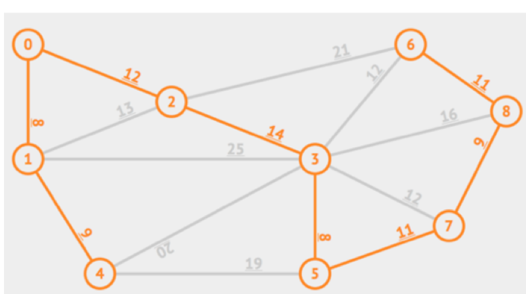

21

## Practice: start at vertex 0




22

## Solution




Sum = 8 + 8 + 9 + 9 + 11 + 11 + 12 + 14 = 82

23

## Why does Prim's work?

Given a partition S, let edge e be the minimum cost edge that **crosses** the partition. *Every* minimum spanning tree contains edge e.

Let S be the set of vertices visited so far

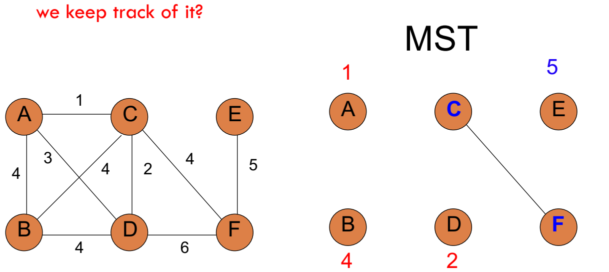
The only time we add a new edge is if it's the lowest weight edge from S to V-S

24

## Prim's

Add the edge with the smallest weight that connects a vertex not connected by the MST

How do we find the smallest weight edge? Or, how could we keep track of it?



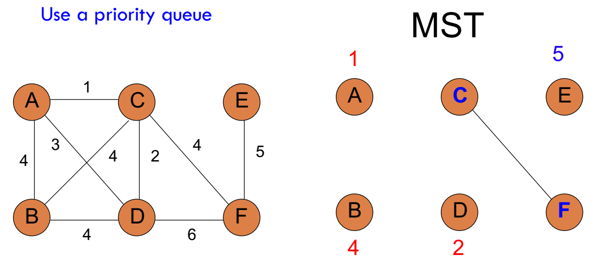

25

## Prim's

Add the edge with the smallest weight that connects a vertex not connected by the MST

Very similar implementation to Dijksra's!

Use a priority queue




26

## Running time of Prim's

Varies depending on the priority queue implementation

Practical version: O(E log V)

27

## Sample question 1: True or False

- A binary tree with *n* nodes has at most height $O(\log_2 n)$
- Heapsort is always $O(n \log_2 n)$
- A complete graph is connected.
- The following code will print out 1:

```
String[] strings = new String[10];
strings[0] = "banana";
System.out.println(strings.length);
```

28

## Sample question 2

Write a method for that takes a BinaryTree, a value, and an ArrayList and populates the ArrayList with all data items in the tree that are less than the value passed in. Assume that the data item stored in the tree is a string and that the strings are unique.

```
public static <E implements comparable<E>> void findLessThan
    (BinaryTree<E> t, E value, ArrayList<E> list)
```

29

## Sample question 3

What is a hashtable collision? Why do they happen? What are two ways of dealing with them?

30

## Sample question 4

Explain the difference between weak and strong connectivity in a directed graph.

31

## Sample question 5

Explain the problem with the following code and suggest how to fix it:

```
public void printEven(Iterator<Integer> iter) {
  while (iter.hasNext()) {
    if (iter.next() % 2 == 0) {
      System.out.println(iter.next());
    }
  }
}
```

32

## Sample question 6

In a graph where the shortest path from A to B is 3 edges, while the shortest path from A to C is 7 edges, **will breadth-first search** starting at A explore B or C first, or does it depend on the structure of the graph? **Justify your answer.**

33

## Sample question 7

a. **Draw a diagram** of a doubly-linked list with both head and tail pointers that contains the elements 17, 23, and 31. **Use arrows to indicate** pointers, including the head and tail pointers, and the next and previous pointers of each node. **For null pointers,** draw an arrow pointing to the word "NULL."

b. Assuming we wanted to insert the value 47 between the 23 and 31 in our list, **write a list of which pointers** would have to be changed. **Include pointers** that are part of the new node.

34

## Sample question 8

**Draw** a directed, acyclic graph with 5 nodes which is **not** a tree, but which has at least one spanning tree. *Label your nodes with the letters A through E.*

35

## Sample question 9

Assuming that BinaryTree t is a full *binary search tree* and that it is not a leaf, what does the method below do?

```
public static String mystery(BinaryTree t){
  BinaryTree temp = t.right();

  while( !temp.left().isEmpty() ){
    temp = temp.left();
  }

  return temp.data();
}
```

36

## Sample question 1 solution

- False. At most O(n)
- True.
- True. It has edges between every pair of vertices.
- False. Even though it only has one entry in it, strings will always have length 10 since it is an arrays length never changes.

37

## Sample question 3: solution

Collisions happen when you have two **different** objects (i.e., objects that are not equals) that have the same hashCode

They happen because hash codes are mapping from a larger space of possibilities (e.g., all strings) down to a finite set of hash codes (e.g., 64-bit integers)

Collision resolution by chaining or open addressing

38

## Sample question 4: solution

A strongly-connected directed graph is one in which for any pair of nodes A and B, there is both a path from A to B and a path from B to A. In a weakly-connected graph, this would be true if edges could be traversed either direction, but it isn't necessarily true with the edges as-is. A weakly-connected graph is simply any graph which *cannot* be cut into two pieces without cutting through at least one edge.

39

## Sample question 5: solution

This code calls next once in the condition of the while loop, and then again in the body, getting two different values each time. It needs to call next only once, and store the result.

40

## Sample question 6: solution

It will explore B first, because breadth-first search always explores closer nodes (in terms of path length) before farther ones.

41

## Sample question 7: solution

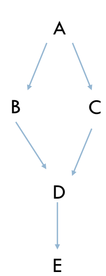
a.

```
     head                               tail
      |                                  |
      |    /-------\        /-------\    |
      v    v       |        v       |    v
/------------\   /-|----------\   /-|----------\
| o | 17 | o |   | o | 23 | o |   | o | 31 | o +
\-|--------|-/   \----------|-/   \----------|-/
  v        |       ^        |       ^        v
 NULL      \-------/        \-------/       NULL
```

b. 23.next
31.previous
47.previous
47.next

42

## Sample question 8: solution




43

## Sample question 9: solution

Finds the value that comes immediately after the root in sorted order

44