MINIMUM SPANNING TREES

David Kauchak
CS 140 – Fall 2020

1

Admin

Assignment

Assignment 4.1graded

Mentor hours this week

Checkpoint

2

Minimum spanning trees

What are they?

What do you remember about them?

What algorithms do you remember?

3

Minimum spanning trees

What is the lowest weight set of edges that connects all vertices of an undirected graph with positive weights

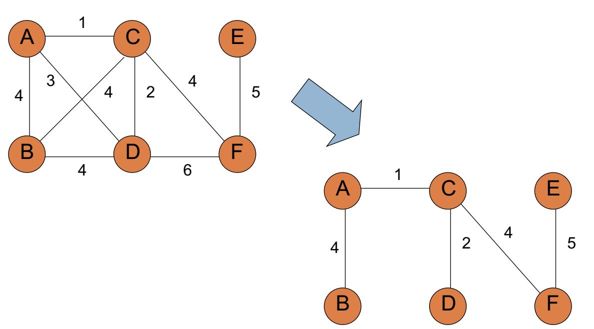
Input: An undirected, positive weight graph, G=(V,E)

Output: A tree T=(V,E') where E' $\subseteq$ E that minimizes

$$weight(T) = \sum_{e \in E'} w_e$$

4

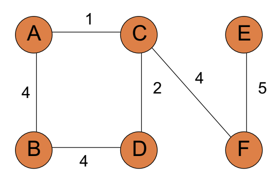
MST example



5

MSTs

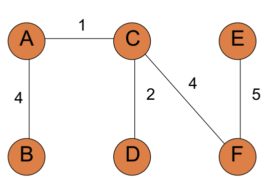
Can an MST have a cycle?



6

MSTs

Can an MST have a cycle?



7

Applications?

Connectivity

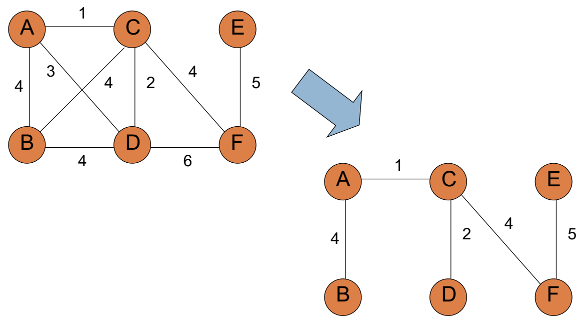
- Networks (e.g. communications)
- Circuit design/wiring

hub/spoke models (e.g. flights, transportation)

Traveling salesman problem?

8

Algorithm ideas?

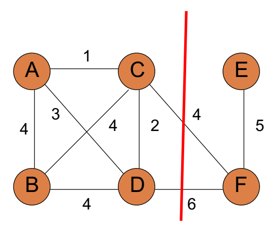


9

Cuts

A cut is a partitioning of the vertices into two sets S and V-S

An edge "crosses" the cut if it connects a vertex u∈V and v∈V-S

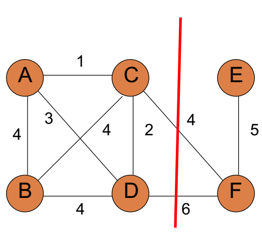


10

Minimum cut property

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

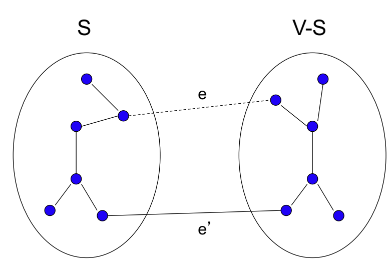
Prove this!



11

Minimum cut property

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

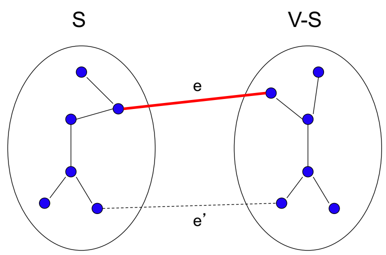


Consider an MST with edge e' that is not the minimum edge

12

Minimum cut property

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

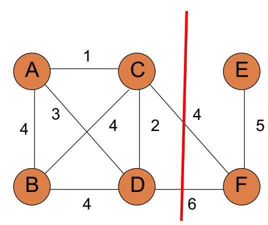


Using e instead of e', still connects the graph, but produces a tree with smaller weights

13

Minimum cut property

If the minimum cost edge that **crosses** the partition is not unique, then *some* minimum spanning tree contains edge e.



14

Kruskal's algorithm

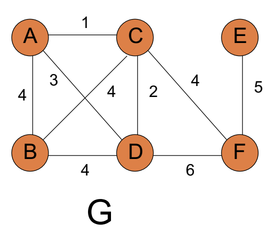
Given a partition S, let edge e be the minimum cost edge that **crosses** the partition. *Every* minimum spanning tree contains edge e.

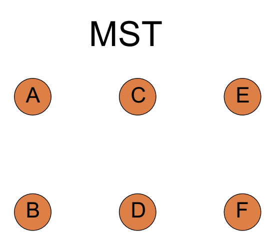
```
KRUSKAL(G)
for all v ∈ V
        MAKESET(v)
T ← {}
sort the edges of E by weight
for all edges (u, v) ∈ E in increasing order of weight
        if FIND-SET(u) ≠ FIND-SET(v)
                add edge to T
                UNION(FIND-SET(u),FIND-SET(v))
```

15

Kruskal's algorithm

Add smallest edge that connects two sets not already connected

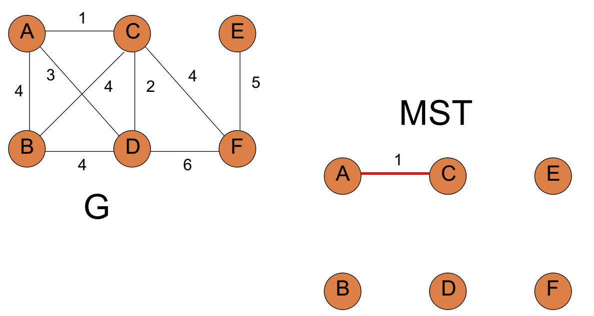




16

Kruskal's algorithm

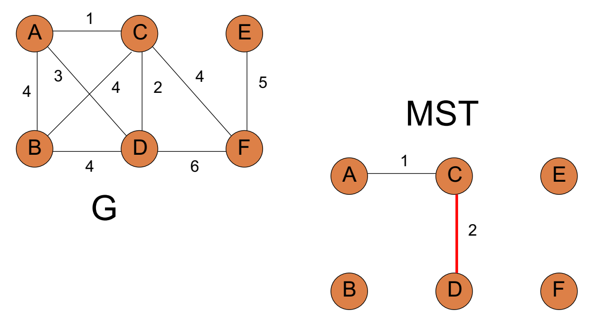
Add smallest edge that connects two sets not already connected



17

Kruskal's algorithm

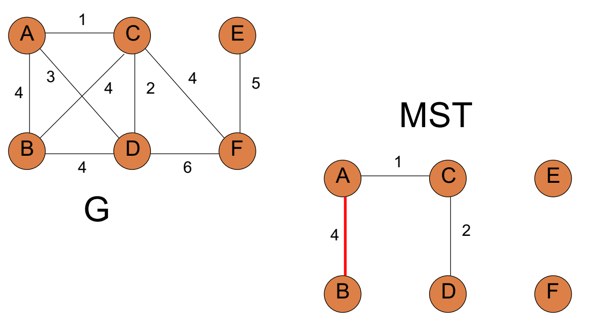
Add smallest edge that connects two sets not already connected



18

Kruskal's algorithm

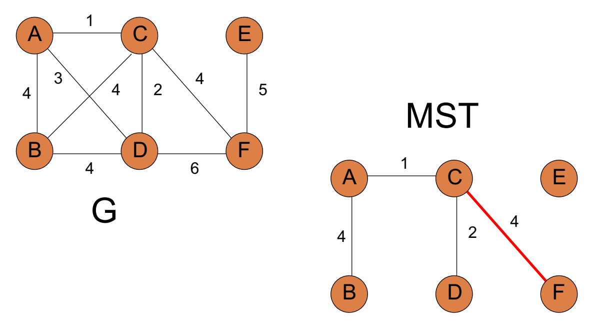
Add smallest edge that connects two sets not already connected



19

Kruskal's algorithm

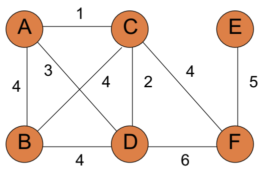
Add smallest edge that connects two sets not already connected



20

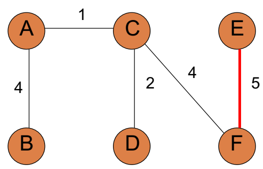
Kruskal's algorithm

Add smallest edge that connects two sets not already connected



G

MST



21

Correctness of Kruskal's

Never adds an edge that connects already connected vertices

Always adds lowest cost edge to connect two sets. By min cut property, that edge must be part of the MST

KRUSKAL(G)
for all $v \in V$
  MAKESET(v)
$T \leftarrow \{\}$
sort the edges of E by weight
for all edges $(u, v) \in E$ in increasing order of weight
  if FIND-SET(u) $\neq$ FIND-SET(v)
    add edge to T
    UNION(FIND-SET(u),FIND-SET(v))

22

Running time of Kruskal's

KRUSKAL(G)
for all $v \in V$
  MAKESET(v)
$T \leftarrow \{\}$
sort the edges of E by weight
for all edges $(u, v) \in E$ in increasing order of weight
  if FIND-SET(u) $\neq$ FIND-SET(v)
    add edge to T
    UNION(FIND-SET(u),FIND-SET(v))

23

Running time of Kruskal's

KRUSKAL(G)
for all $v \in V$
  MAKESET(v) — |V| calls to MakeSet
$T \leftarrow \{\}$
sort the edges of E by weight — O(|E| log |E|)
for all edges $(u, v) \in E$ in increasing order of weight
  if FIND-SET(u) $\neq$ FIND-SET(v) — 2 |E| calls to FindSet
    add edge to T
    UNION(FIND-SET(u),FIND-SET(v)) — |V| calls to Union

24

Running time of Kruskal's

Disjoint set data structure

O(|E| log |E|) +

	MakeSet	FindSet |E| calls	Union |V| calls	Total
Linked lists				

25

Disjoint set

26

Disjoint set: union

27

Disjoint set: union

28

Disjoint set: union

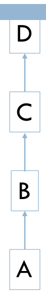
29

Disjoint set: union

Running time?

30

Disjoint set: union



O(1)

31

Disjoint set: find-set

Search each linked list

32

Disjoint set: find-set

Running time?

33

Disjoint set: find-set

O(n) -- n = number of things in set

34

Running time of Kruskal's

Disjoint set data structure

	MakeSet	FindSet |E| calls	Union |V| calls	Total
		O(|E| log |E|) +		
Linked lists	|V|	O(|V| |E|)	|V|	O(|V||E| + |E| log |E|) O(|V| |E|)
Linked lists + heuristics	|V|	O(|E| log |V|)	|V|	O(|E| log |V|+ |E| log |E|) O(|E| log |E|)

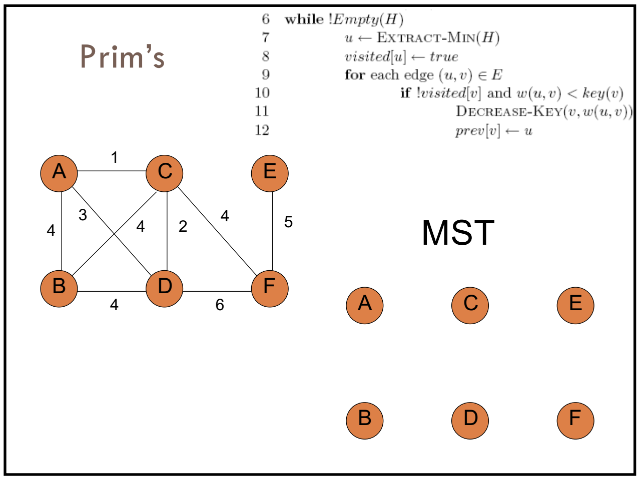
35

Prim's algorithm

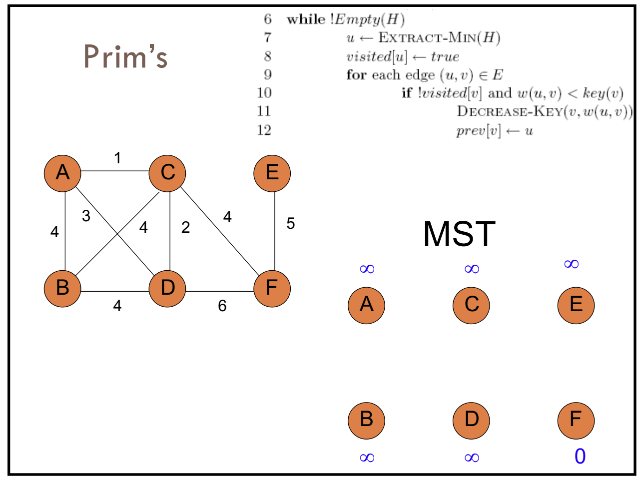
Start at some root node and build out the MST by adding the lowest weighted edge at the frontier

```
PRIM(G, r)
for all v ∈ V
        key[v] ← ∞
        prev[v] ← null
key[r] ← 0
H ← MAKEHEAP(key)
while !Empty(H)
        u ← EXTRACT-MIN(H)
        visited[u] ← true
        for each edge (u, v) ∈ E
                if !visited[v] and w(u, v) < key(v)
                        DECREASE-KEY(v, w(u, v))
                        prev[v] ← u
```

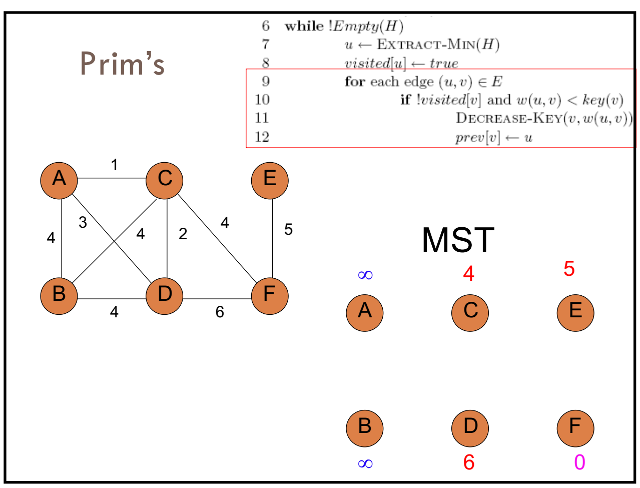
39


Prim's
while !Empty(H)
u ← Extract-Min(H)
visited[u] ← true
for each edge (u, v) ∈ E
if !visited[v] and w(u, v) < key(v)
Decrease-Key(v, w(u, v))
prev[v] ← u
MST
A
B
C
D
E
F
1
3
4
4
2
4
5
4
6

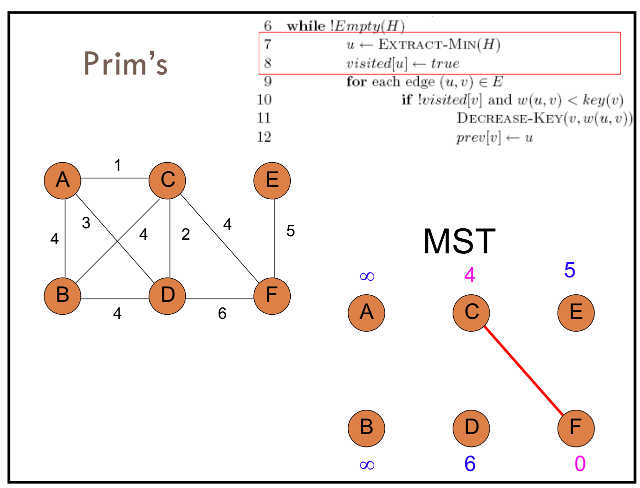
40


Prim's
while !Empty(H)
u ← Extract-Min(H)
visited[u] ← true
for each edge (u, v) ∈ E
if !visited[v] and w(u, v) < key(v)
Decrease-Key(v, w(u, v))
prev[v] ← u
MST
∞
∞
∞
∞
∞
0

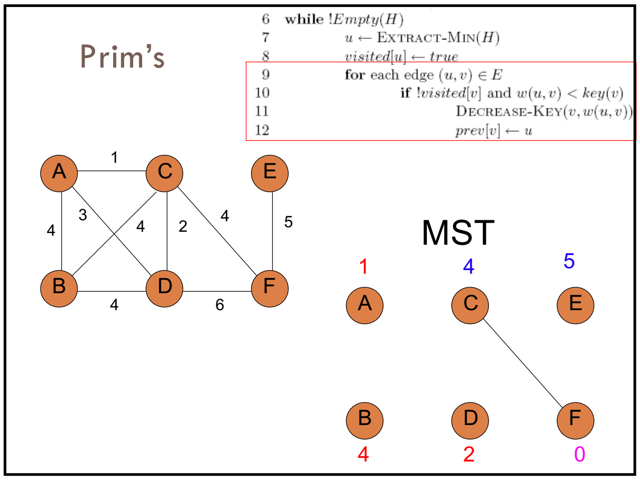
41


Prim's
while !Empty(H)
u ← Extract-Min(H)
visited[u] ← true
for each edge (u, v) ∈ E
if !visited[v] and w(u, v) < key(v)
Decrease-Key(v, w(u, v))
prev[v] ← u
MST
∞
4
5
∞
6
0

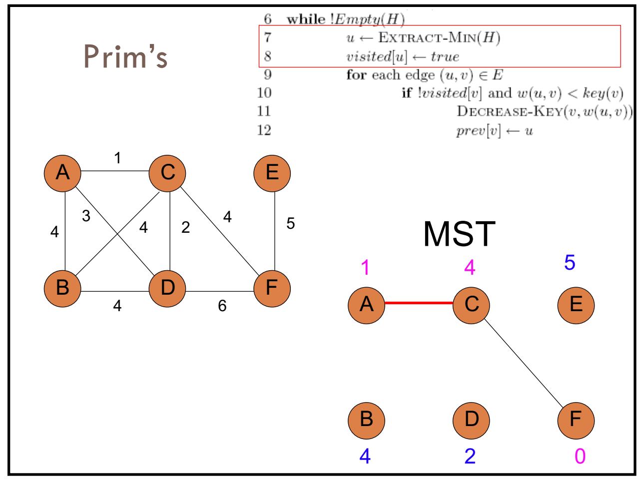
42


Prim's
while !Empty(H)
u ← Extract-Min(H)
visited[u] ← true
for each edge (u, v) ∈ E
if !visited[v] and w(u, v) < key(v)
Decrease-Key(v, w(u, v))
prev[v] ← u
MST
∞
4
5
∞
6
0

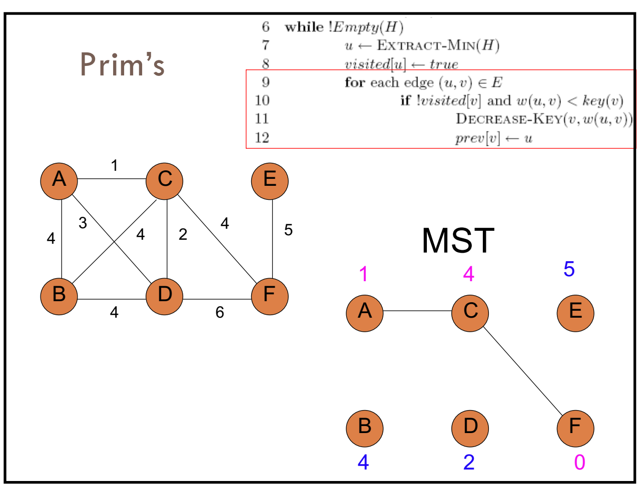
43


Prim's
while !Empty(H)
u ← Extract-Min(H)
visited[u] ← true
for each edge (u, v) ∈ E
if !visited[v] and w(u, v) < key(v)
Decrease-Key(v, w(u, v))
prev[v] ← u
MST

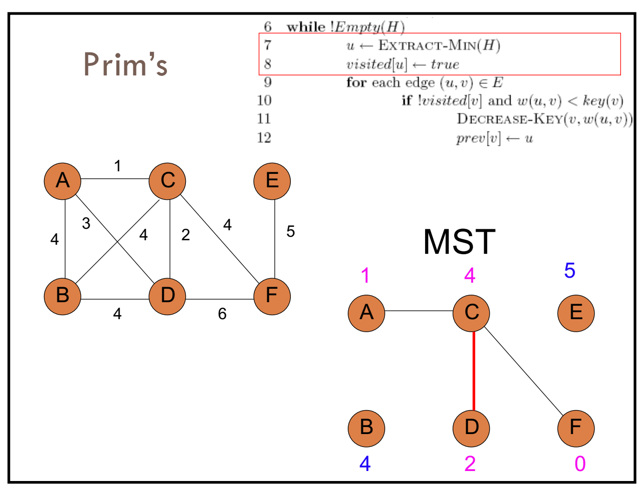
44


Prim's
while !Empty(H)
u ← Extract-Min(H)
visited[u] ← true
for each edge (u, v) ∈ E
if !visited[v] and w(u, v) < key(v)
Decrease-Key(v, w(u, v))
prev[v] ← u
MST

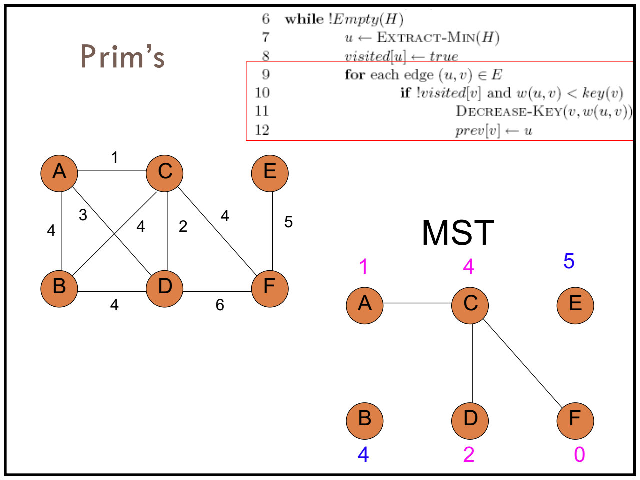
45


Prim's
while !Empty(H)
u ← Extract-Min(H)
visited[u] ← true
for each edge (u, v) ∈ E
if !visited[v] and w(u, v) < key(v)
Decrease-Key(v, w(u, v))
prev[v] ← u
MST

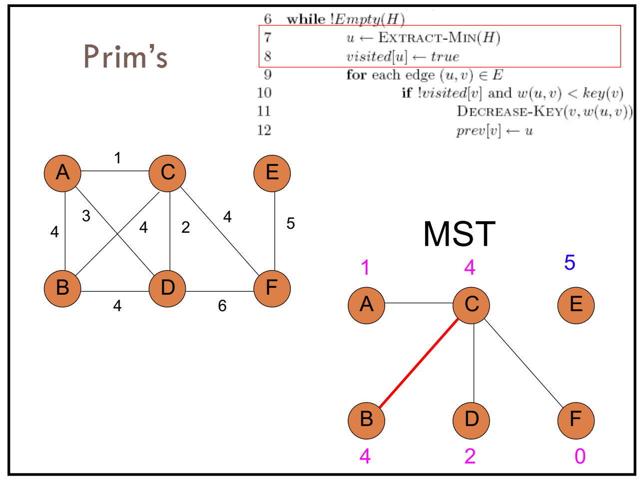
46


Prim's
while !Empty(H)
u ← Extract-Min(H)
visited[u] ← true
for each edge (u, v) ∈ E
if !visited[v] and w(u, v) < key(v)
Decrease-Key(v, w(u, v))
prev[v] ← u
MST

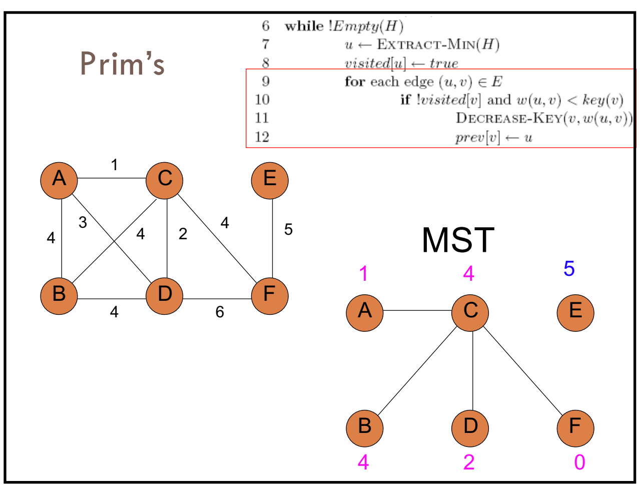
47


Prim's
while !Empty(H)
u ← Extract-Min(H)
visited[u] ← true
for each edge (u, v) ∈ E
if !visited[v] and w(u, v) < key(v)
Decrease-Key(v, w(u, v))
prev[v] ← u
MST

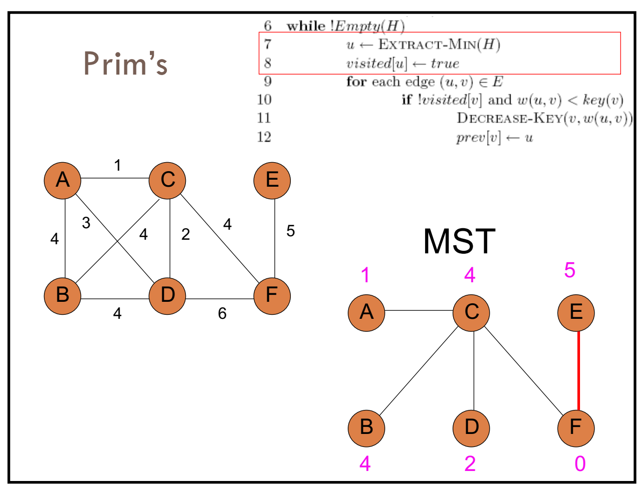
48


Prim's
while !Empty(H)
u ← Extract-Min(H)
visited[u] ← true
for each edge (u, v) ∈ E
if !visited[v] and w(u, v) < key(v)
Decrease-Key(v, w(u, v))
prev[v] ← u
MST

49


Prim's
while !Empty(H)
u ← Extract-Min(H)
visited[u] ← true
for each edge (u, v) ∈ E
if !visited[v] and w(u, v) < key(v)
Decrease-Key(v, w(u, v))
prev[v] ← u
MST

50


Prim's
while !Empty(H)
u ← Extract-Min(H)
visited[u] ← true
for each edge (u, v) ∈ E
if !visited[v] and w(u, v) < key(v)
Decrease-Key(v, w(u, v))
prev[v] ← u
MST

51

Correctness of Prim's?

Can we use the min-cut property?

- Given a partion 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

52

Running time of Prim's

```
PRIM(G, r)
for all v ∈ V
        key[v] ← ∞
        prev[v] ← null
key[r] ← 0
H ← MAKEHEAP(key)
while !Empty(H)
        u ← EXTRACT-MIN(H)
        visited[u] ← true
        for each edge (u, v) ∈ E
                if !visited[v] and w(u, v) < key(v)
                        DECREASE-KEY(v, w(u, v))
                        prev[v] ← u
```

53

Running time of Prim's

```
PRIM(G, r)
for all v ∈ V
        key[v] ← ∞
        prev[v] ← null
key[r] ← 0
H ← MAKEHEAP(key)
while !Empty(H)
        u ← EXTRACT-MIN(H)
        visited[u] ← true
        for each edge (u, v) ∈ E
                if !visited[v] and w(u, v) < key(v)
                        DECREASE-KEY(v, w(u, v))
                        prev[v] ← u
```

Θ(|V|)

1 call to MakeHeap

|V| calls to Extract-Min

|E| calls to Decrease-Key

54

Running time of Prim's

	1 MakeHeap	|V| ExtractMin	|E| DecreaseKey	Total
Array	θ(|V|)	O(|V|^2)	O(|E|)	O(|V|^2)
Bin heap	θ(|V|)	O(|V| log |V|)	O(|E| log |V|)	O((|V|+|E|) log |V|) O(|E| log |V|)
Fib heap	θ(|V|)	O(|V| log |V|)	O(|E|)	**O(|V| log |V| + |E|)**

Kruskal's: O(|E| log |E|)

55