MORE GRAPH ALGORITHMS

David Kauchak
CS 140 – Spring 2024

1

Admin

Assignment 8 out: don't reinvent the wheel

Assignment schedule updated for the rest of the semester

Groups optional this week

2

Connectedness

Given an undirected graph, for every node $u \in V$, can we reach all other nodes in the graph?
Algorithm + running time

Run BFS or DFS-Visit (one pass) and mark nodes as we visit them. If we visit all nodes, return true, otherwise false.

Running time: $O(|V| + |E|)$

3

Strongly connected

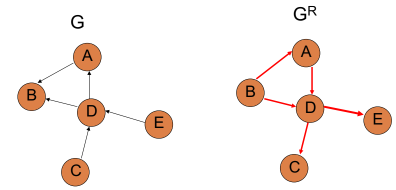
Given a directed graph, can we reach any node v from any other node u?

Can we do the same thing?

4

Transpose of a graph

Given a graph G, we can calculate the transpose of a graph G^R by reversing the direction of all the edges



Running time to calculate G^R? $\theta(|V| + |E|)$

5

Strongly connected

Strongly-Connected(G)
- Run DFS-Visit or BFS from some node u
- If not all nodes are visited: return false
- Create graph G^R
- Run DFS-Visit or BFS on G^R from node u
- If not all nodes are visited: return false
- return true

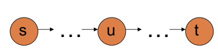
6

Is it correct?

What do we know after the first pass?
- Starting at u, we can reach every node

What do we know after the second pass?
- All nodes can reach u. Why?
- We can get from u to every node in G^R, therefore, if we reverse the edges (i.e. G), then we have a path from every node to u

Which means that any node can reach any other node. Given any two nodes s and t we can create a path through u



7

Runtime?

Strongly-Connected(G)
- Run DFS-Visit or BFS from some node u — $O(|V| + |E|)$
- If not all nodes are visited: return false — $O(|V|)$
- Create graph G^R — $\theta(|V| + |E|)$
- Run DFS-Visit or BFS on G^R from node u — $O(|V| + |E|)$
- If not all nodes are visited: return false — $O(|V|)$
- return true

$O(|V| + |E|)$

8

Minimum spanning trees

What are they?

What do you remember about them?

What algorithms do you remember?

9

Minimum spanning trees

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

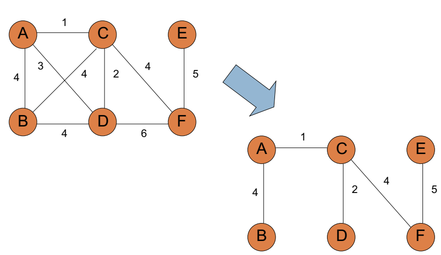
Input: An undirected, positive weight graph, G=(V,E)

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

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

10

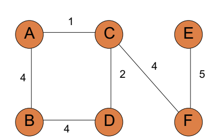
MST example



11

MSTs

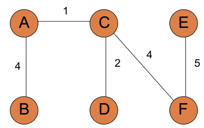
Can an MST have a cycle?



12

MSTs

Can an MST have a cycle?



13

Applications?

Connectivity

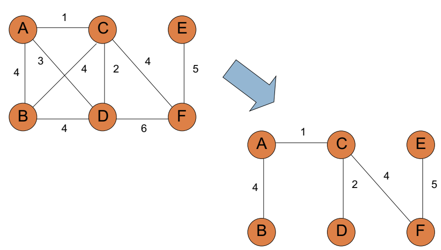
- Networks (e.g. communications)
- Circuit design/wiring

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

Traveling salesman problem?

14

Algorithm ideas?

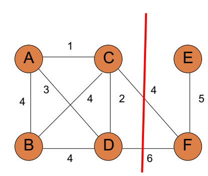


15

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

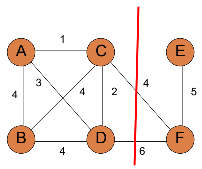


16

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.

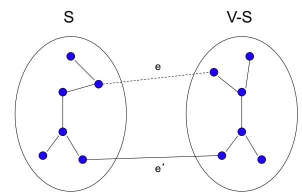
Proof by contradiction?



17

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.

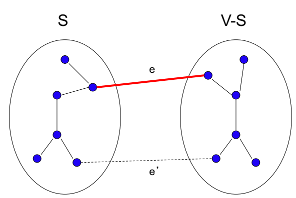


Consider an MST with edge e' that is not the minimum edge

18

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.

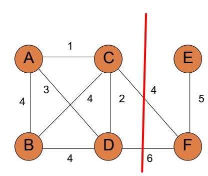


Using e instead of e', still connects the graph, but produces a tree with smaller weights

19

Minimum cut property

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



20

Kruskal's algorithm

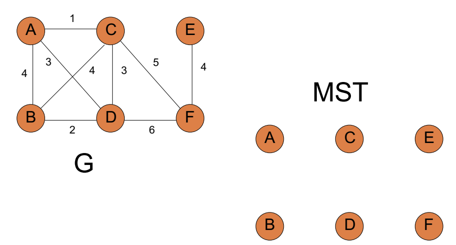
Given a partition S, let edge e be the minimum cost edge that **crosses** the partition. *Every* minimum spanning tree contains edge e.

```
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))
```

21

Kruskal's algorithm

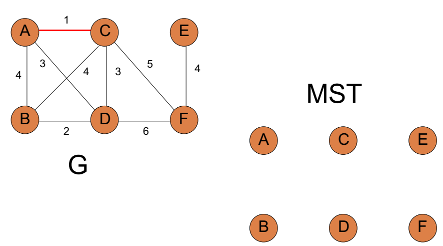
Add smallest edge that connects two sets not already connected



22

Kruskal's algorithm

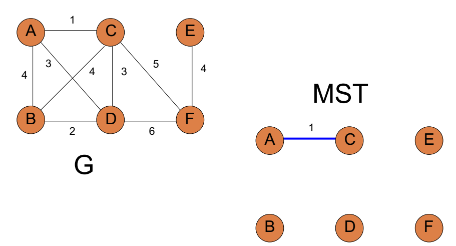
Add smallest edge that connects two sets not already connected



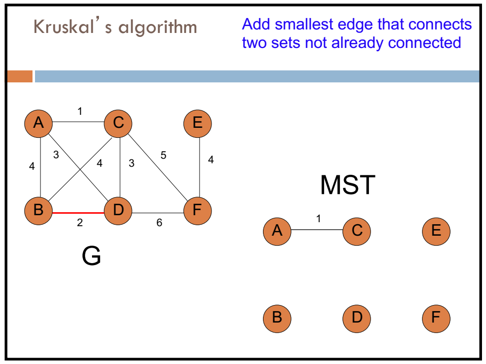
23

Kruskal's algorithm

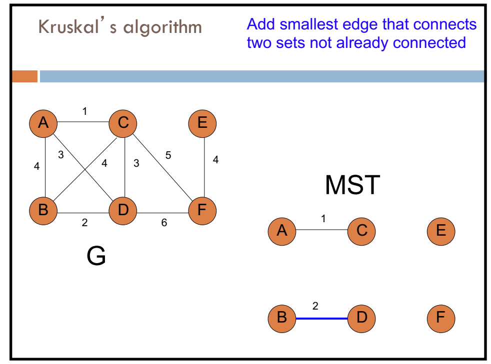
Add smallest edge that connects two sets not already connected



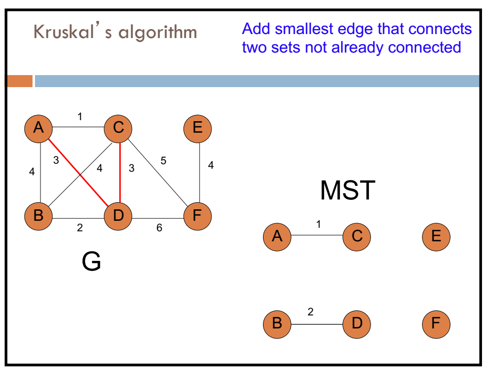
24


Kruskal's algorithm
Add smallest edge that connects two sets not already connected
G
MST

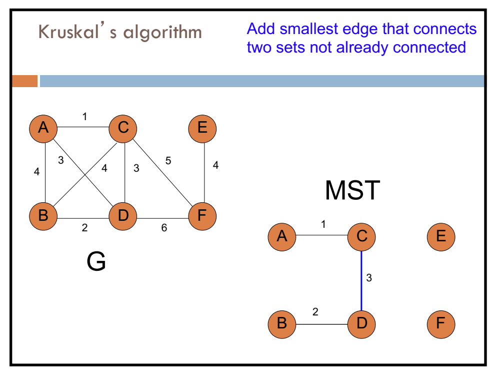
25


Kruskal's algorithm
Add smallest edge that connects two sets not already connected
G
MST

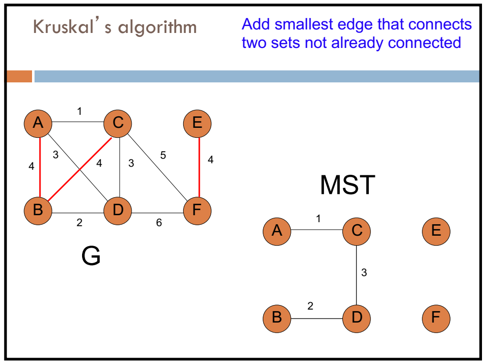
26


Kruskal's algorithm
Add smallest edge that connects two sets not already connected
G
MST

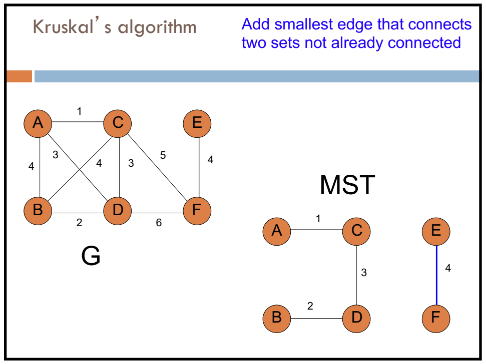
27


Kruskal's algorithm
Add smallest edge that connects two sets not already connected
G
MST

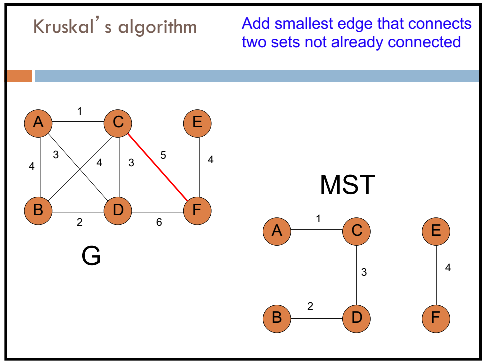
28


Kruskal's algorithm
Add smallest edge that connects two sets not already connected
G
MST

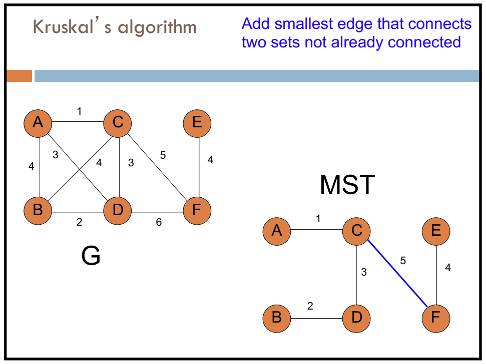
29


Kruskal's algorithm
Add smallest edge that connects two sets not already connected
G
MST

30


Kruskal's algorithm
Add smallest edge that connects two sets not already connected
G
MST

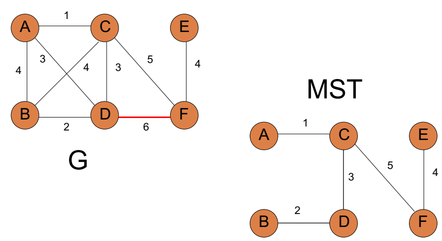
31


Kruskal's algorithm
Add smallest edge that connects two sets not already connected
G
MST

32

Kruskal's algorithm

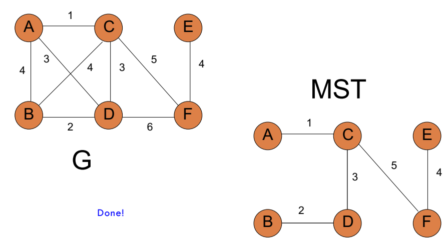
Add smallest edge that connects two sets not already connected



33

Kruskal's algorithm

Add smallest edge that connects two sets not already connected



34

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 ∈ 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))
```

35

Running time of Kruskal's

```
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))
```

36

Running time of Kruskal's

```
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))
```

|V| calls to MakeSet

O(|E| log |E|)

2 |E| calls to FindSet

|V| calls to Union

37

Disjoint set data structures

Represents a collection of one or more sets

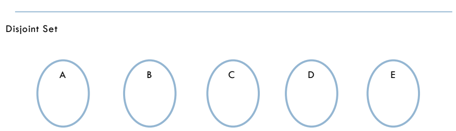
Operations:

- MakeSet: Add a new value to the collections and make the value it's own set
- FindSet: Given a value, return the set the value is in
- Union: Merge two sets into a single set

38

Disjoint set data structure

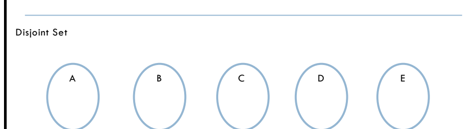
MakeSet(A), MakeSet(B), MakeSet(C), MakeSet(D), MakeSet(E)



39

Disjoint set data structure

FindSet(A)?

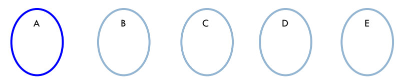


40

Disjoint set data structure

FindSet(A)?

Disjoint Set

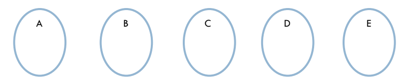


41

Disjoint set data structure

Union(FindSet(A), FindSet(E))

Disjoint Set

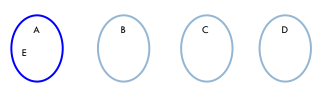


42

Disjoint set data structure

Union(FindSet(A), FindSet(E))

Disjoint Set

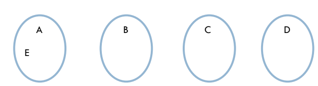


43

Disjoint set data structure

Union(FindSet(C), FindSet(D))

Disjoint Set

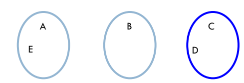


44

Disjoint set data structure

Union(FindSet(C), FindSet(D))

Disjoint Set

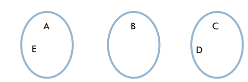


45

Disjoint set data structure

FindSet(D)?

Disjoint Set

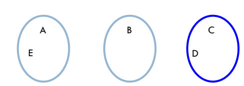


46

Disjoint set data structure

FindSet(D)?

Disjoint Set

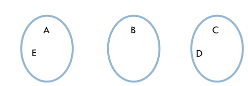


47

Disjoint set data structure

Union(FindSet(D), FindSet(B))

Disjoint Set



48

Disjoint set data structure

Union(FindSet(D), FindSet(B))

Disjoint Set

49

Disjoint set data structure

How would we implement it with a list of linked lists?

MakeSet?

FindSet?

Union?

Disjoint Set

50

Disjoint set

A B C D

51

Disjoint set: union

A B C D

52

Disjoint set: union

53

Disjoint set: union

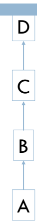
54

Disjoint set: union

Running time?

55

Disjoint set: union



O(1)

56

Disjoint set: find-set

Search each linked list

57

Disjoint set: find-set

Running time?

58

Disjoint set: find-set

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

59

Running time of Kruskal's

Disjoint set data structure

$O(\lvert E\rvert \log \lvert E\rvert) +$

	MakeSet (V calls)	FindSet ($\lvert E\rvert$ calls)	Union ($\lvert V\rvert$ calls)	Total
Linked lists	$\lvert V\rvert$	$O(\lvert V\rvert \lvert E\rvert)$	$\lvert V\rvert$	$O(\lvert V\rvert\lvert E\rvert + \lvert E\rvert \log \lvert E\rvert)$ $O(\lvert V\rvert \lvert E\rvert)$
Linked lists + heuristics	$\lvert V\rvert$	$O(\lvert E\rvert \log \lvert V\rvert)$	$\lvert V\rvert$	$O(\lvert E\rvert \log \lvert V\rvert + \lvert E\rvert \log \lvert E\rvert)$ $O(\lvert E\rvert \log \lvert E\rvert)$

60

Prim's algorithm

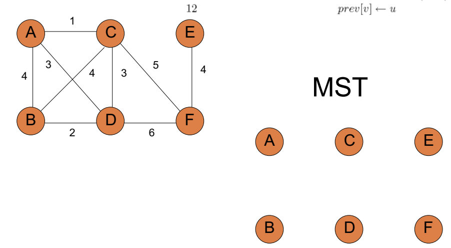
Start at some root node and build out the MST by adding the lowest weighted edge at the frontier

```
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
```

64

Prim's

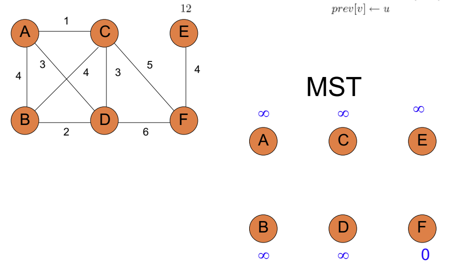
```
while !Empty(H)
    u ← EXTRACT-MIN(H)
    visited[u] ← true
    for each edge (u, v) ∈ E
        if !visited[v] and w(u, v) < key(v)
            DECREASE-KEY(v, w(u, v))
            prev[v] ← u
```



65

Prim's

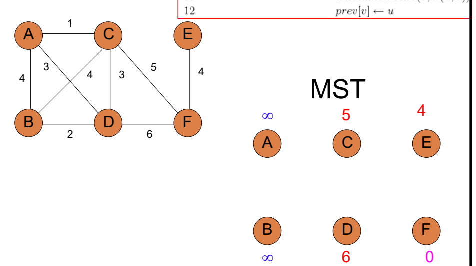
```
while !Empty(H)
    u ← EXTRACT-MIN(H)
    visited[u] ← true
    for each edge (u, v) ∈ E
        if !visited[v] and w(u, v) < key(v)
            DECREASE-KEY(v, w(u, v))
            prev[v] ← u
```



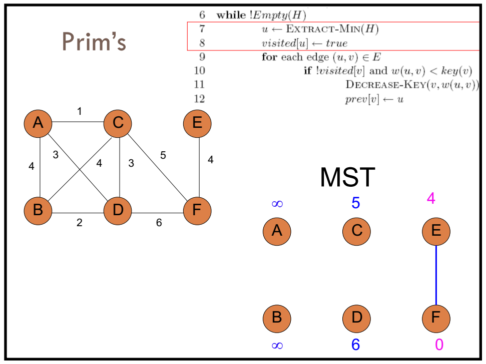
66

Prim's

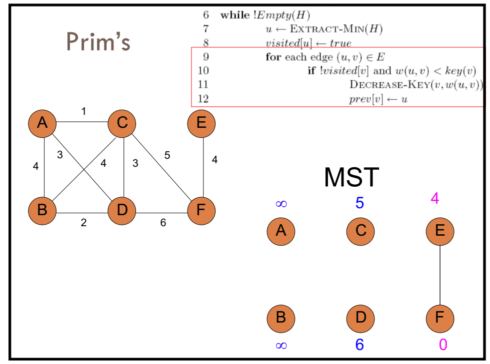
```
while !Empty(H)
    u ← EXTRACT-MIN(H)
    visited[u] ← true
    for each edge (u, v) ∈ E
        if !visited[v] and w(u, v) < key(v)
            DECREASE-KEY(v, w(u, v))
            prev[v] ← u
```



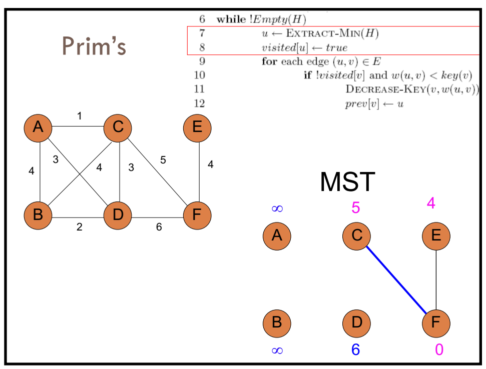
67


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
∞ 5 4
∞ 6 0

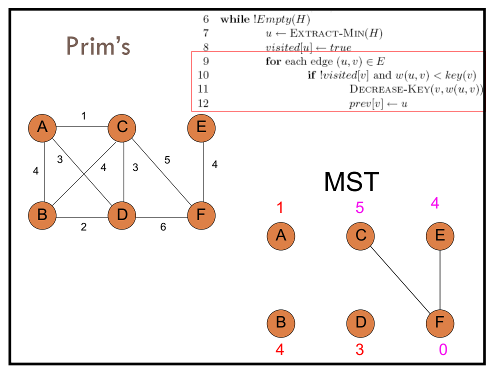
68


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
∞ 5 4
∞ 6 0

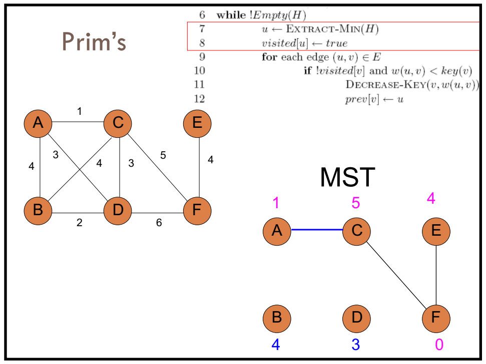
69


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
∞ 5 4
∞ 6 0

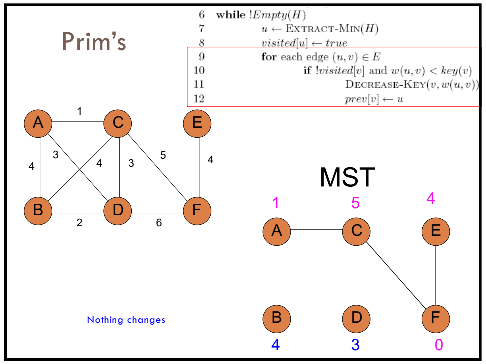
70


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
1 5 4
4 3 0

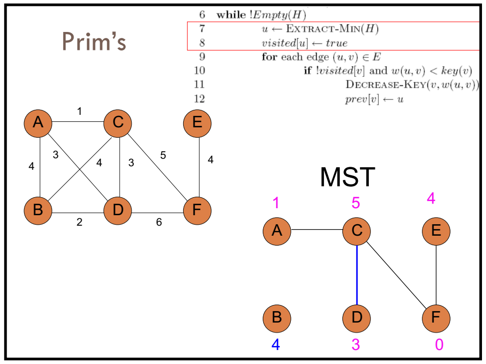
71


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

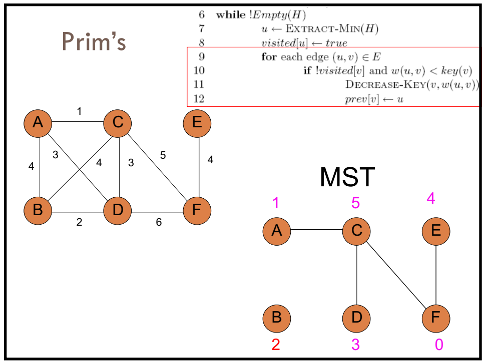
72


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
Nothing changes

73


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

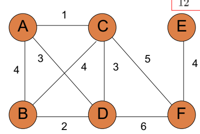
74


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

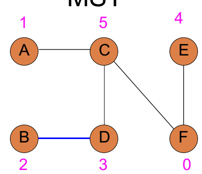
75

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
```



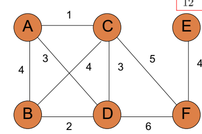
MST



76

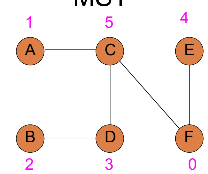
Prim's

```
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

Done!



77

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

78

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
```

79

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

80

Running time of Prim's

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

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

81