Minimum Spanning Trees

Dr. Navjot Singh

Design and Analysis of Algorithms

Minimum spanning trees

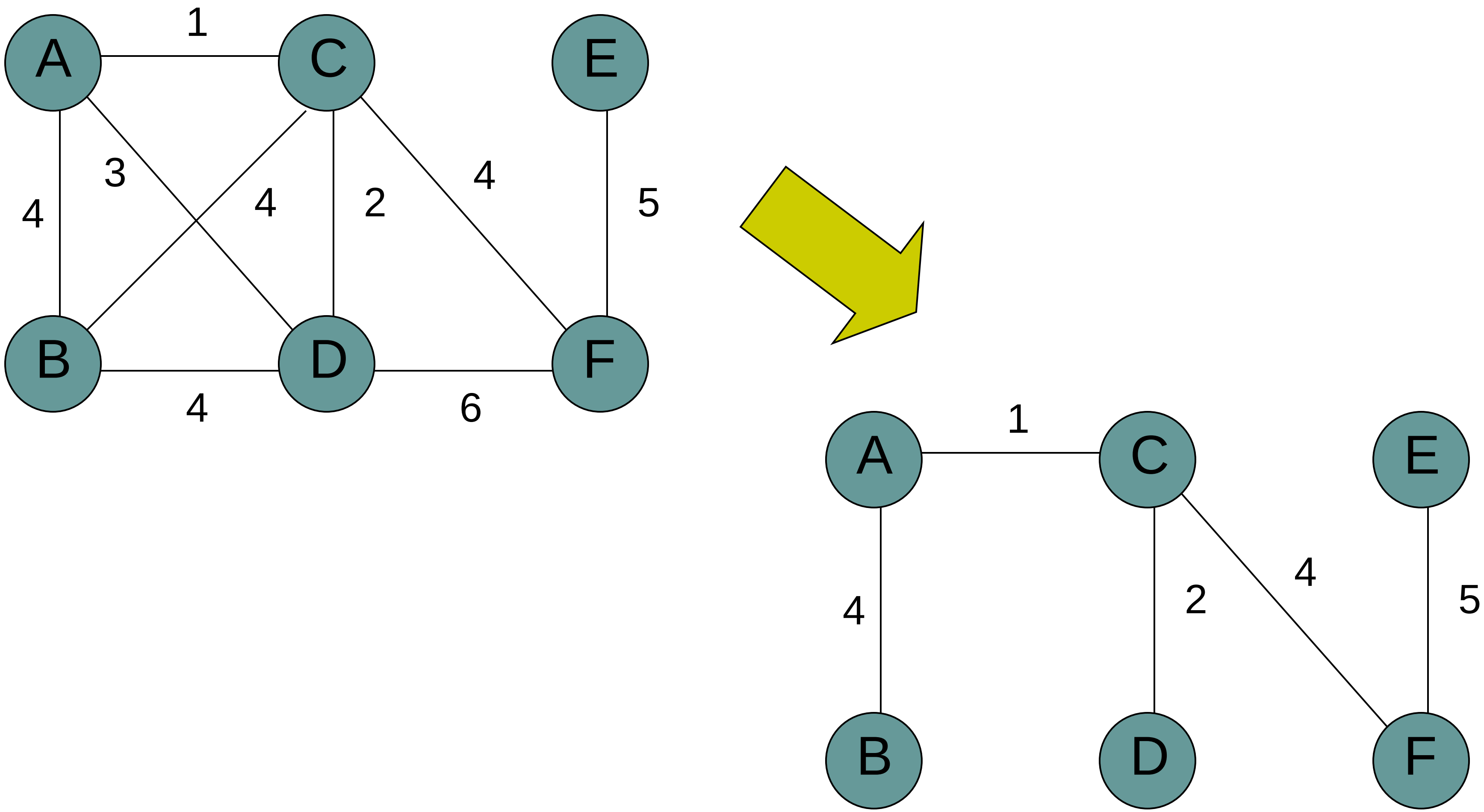
What is the lowest weight set of edges that connects all vertices of an undirected graph with positive weights

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

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

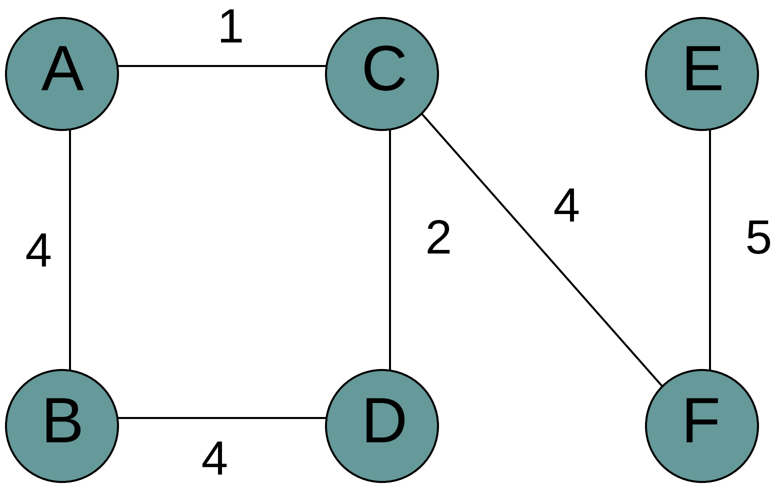
$$weight(T) = \sum_{e \in E'} w_e$$

MST example



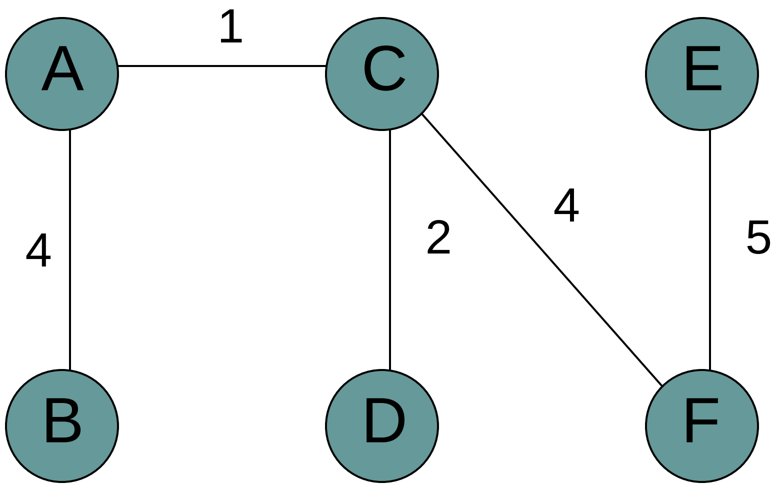
MSTs

Can an MST have a cycle?



MSTs

Can an MST have a cycle?



Applications

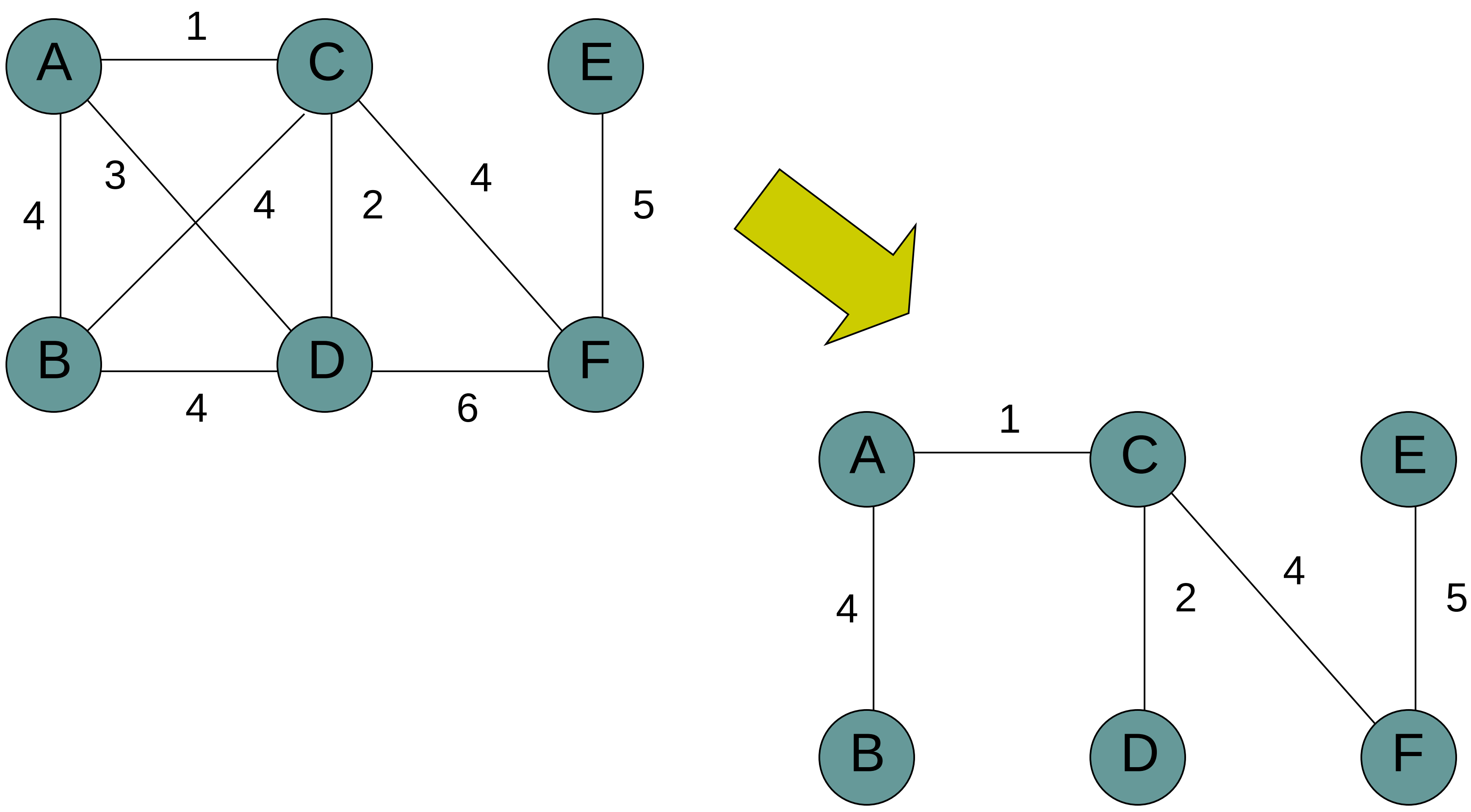
Connectivity

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

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

Traveling salesman problem?

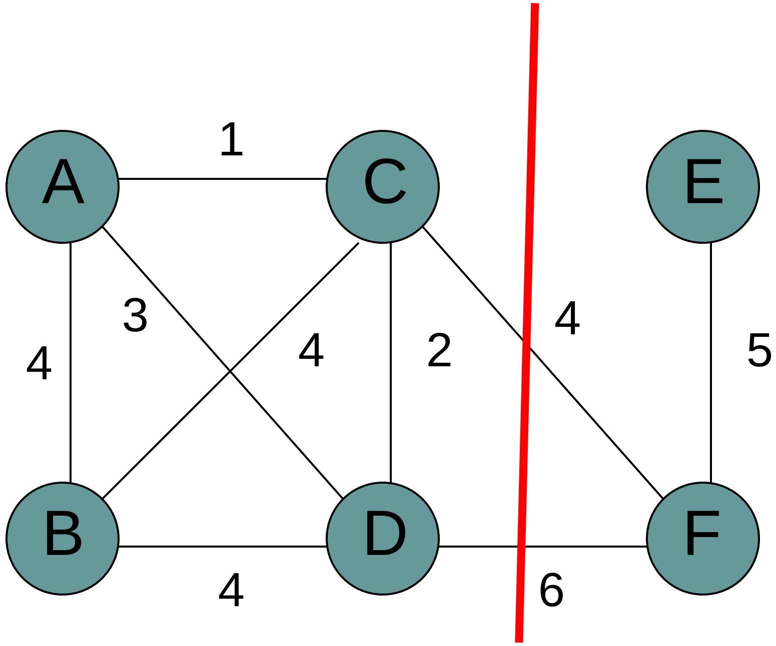
Algorithm ideas



Cuts

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

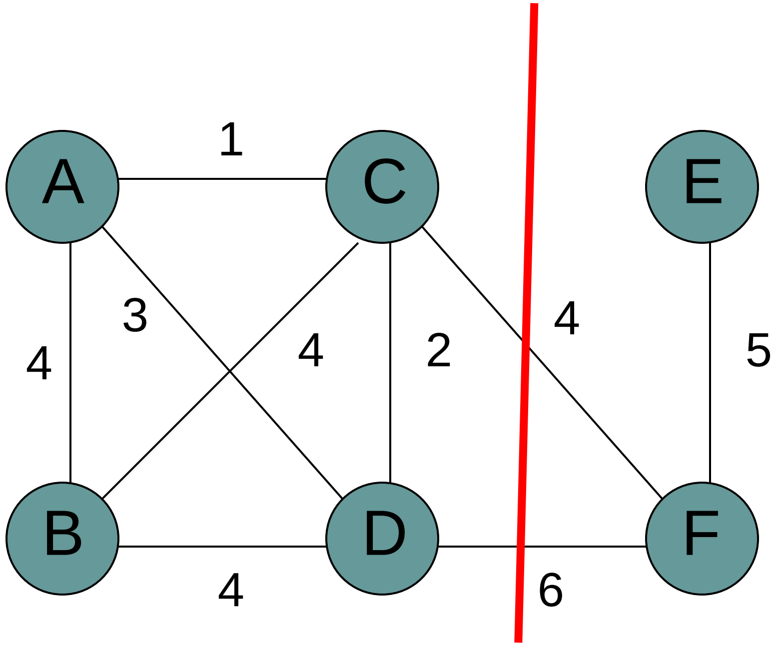
An edge "crosses" the cut if it connects a vertex u∈V and v∈V-S



Minimum cut property

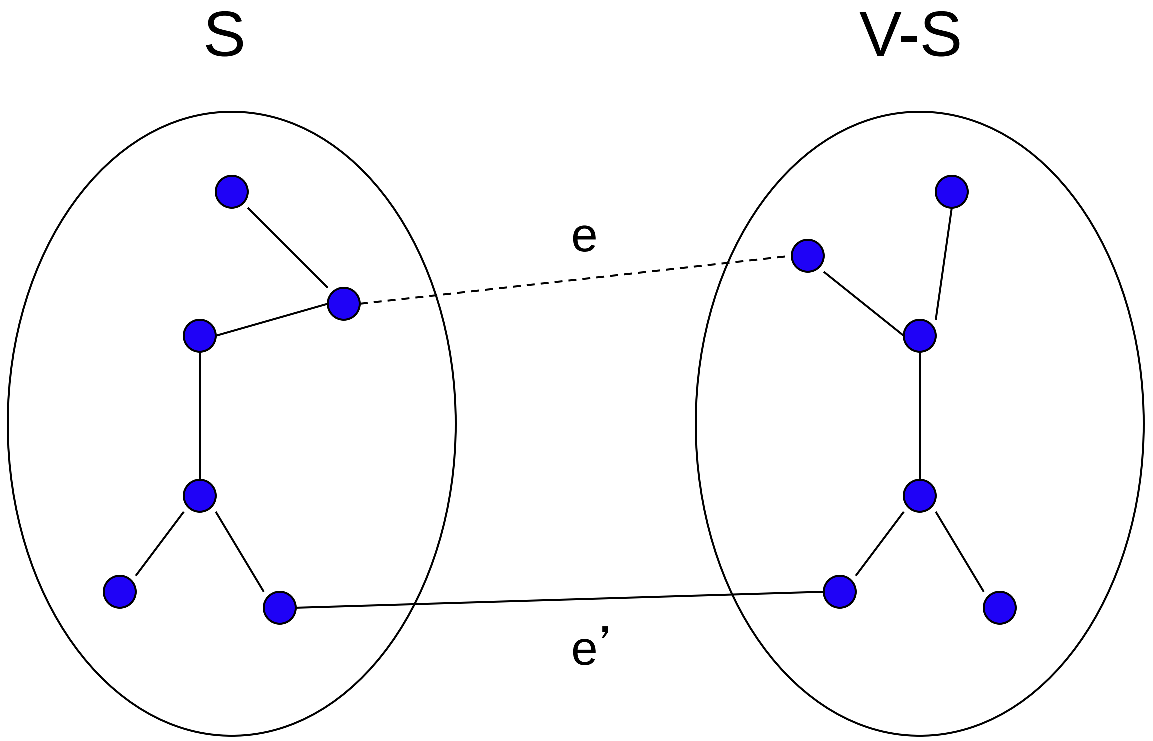
Given a partition S, let edge *e* be the minimum cost edge that **crosses** the partition. *Every* minimum spanning tree contains edge *e*.

Prove this!



Minimum cut property

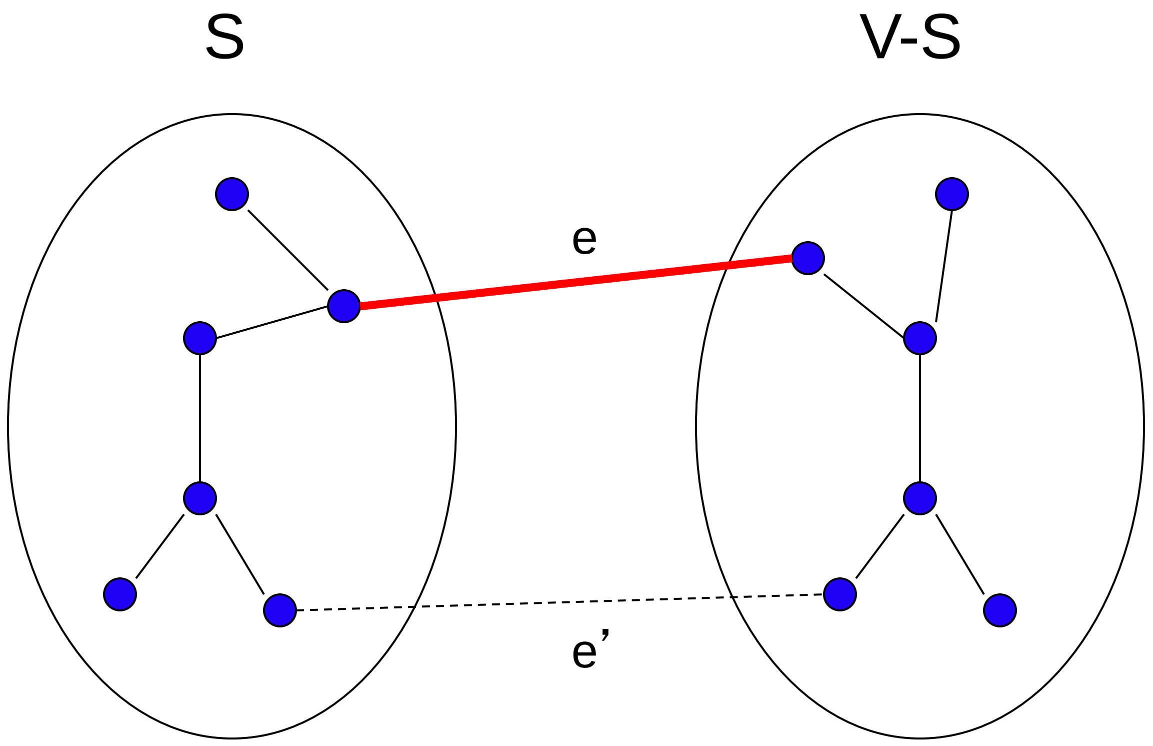
Given a partition S, let edge *e* be the minimum cost edge that **crosses** the partition. *Every* minimum spanning tree contains edge *e*.



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

Minimum cut property

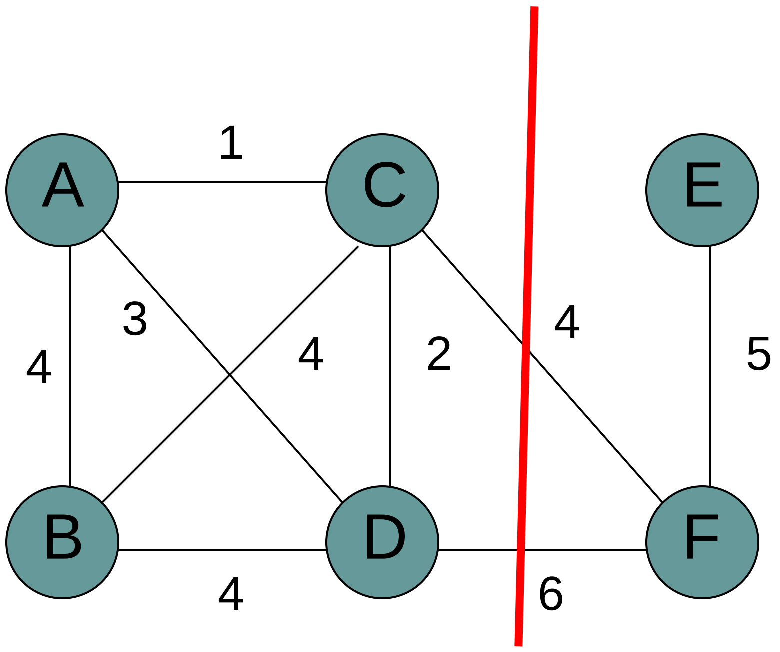
Given a partition S, let edge *e* be the minimum cost edge that **crosses** the partition. *Every* minimum spanning tree contains edge *e*.



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

Minimum cut property

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



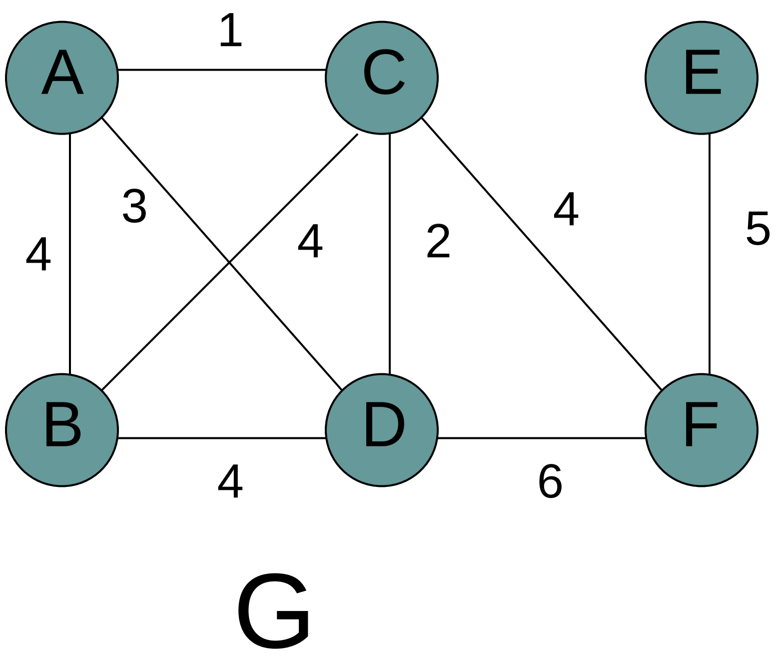
Kruskal's algorithm

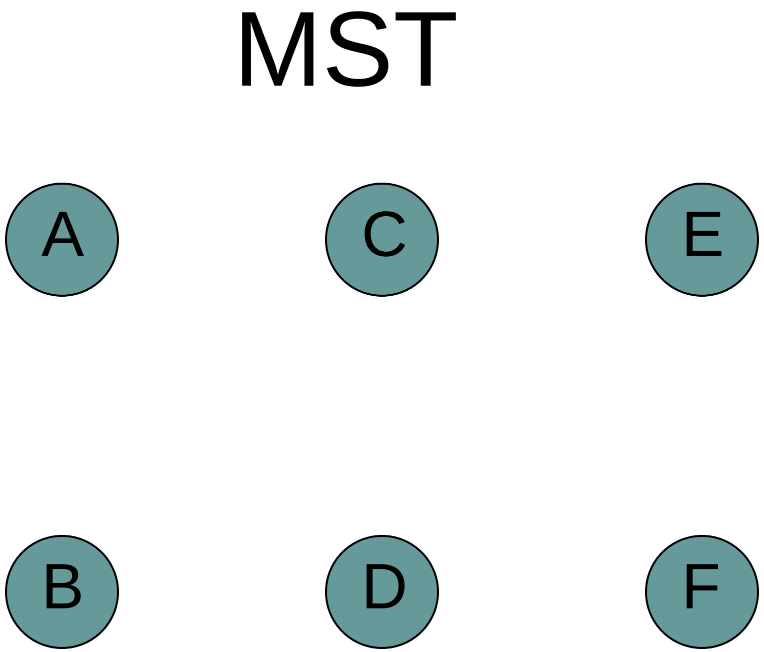
Given a partition S, let edge *e* be the minimum cost edge that **crosses** the partition. *Every* minimum spanning tree contains edge *e*.

```
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))
```

Kruskal's algorithm

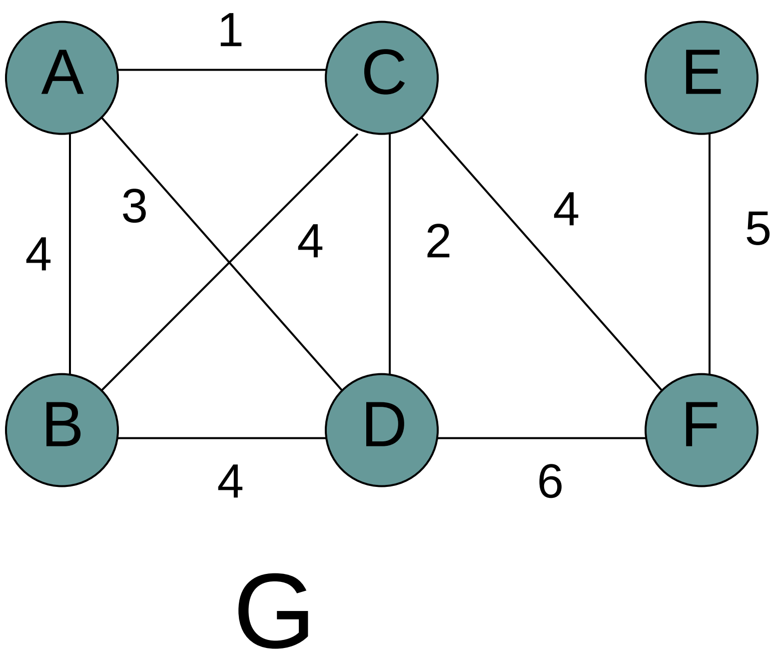
Add smallest edge that connects two sets not already connected

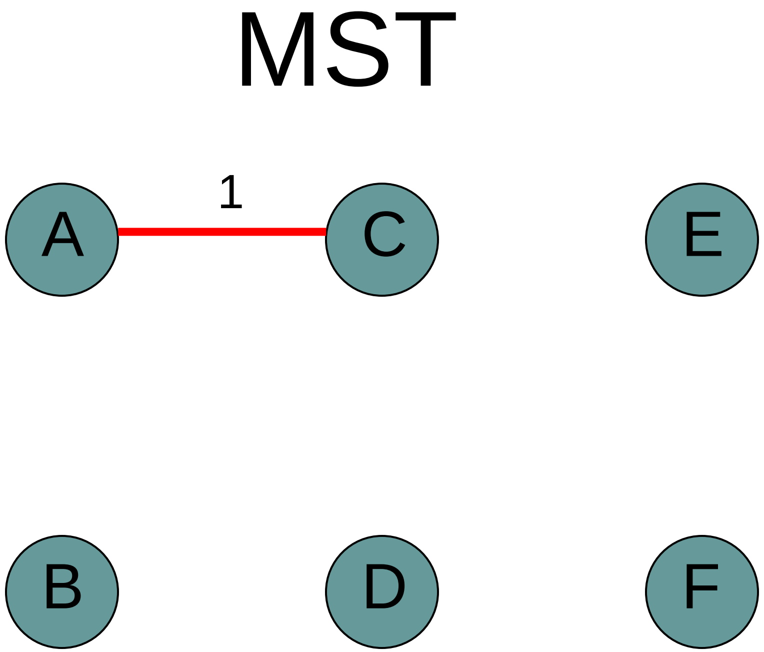




Kruskal's algorithm

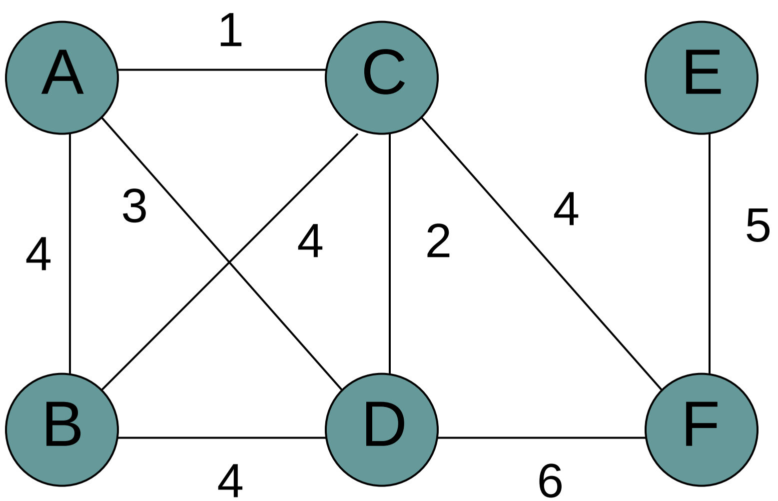
Add smallest edge that connects two sets not already connected





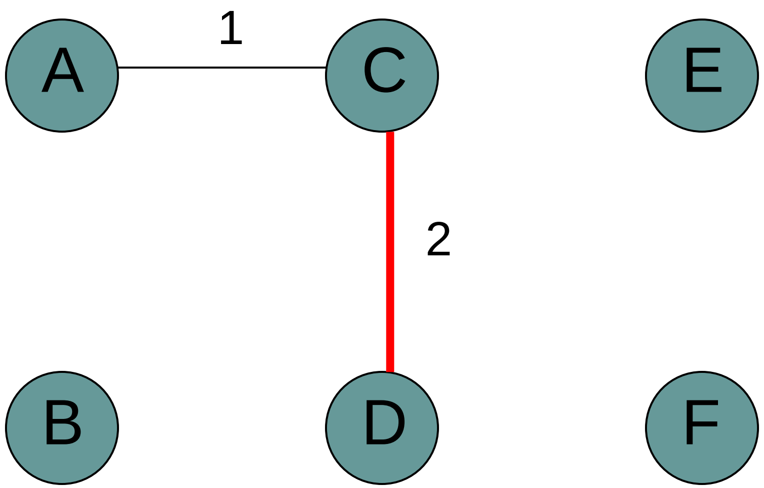
Kruskal's algorithm

Add smallest edge that connects two sets not already connected



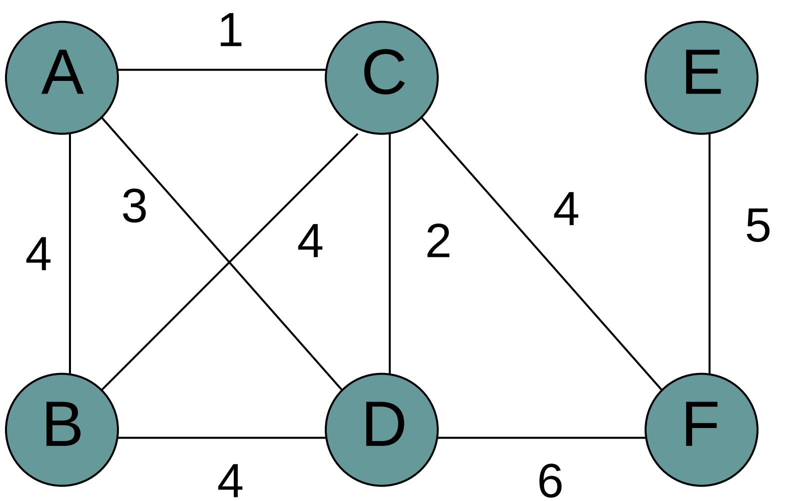
G

MST



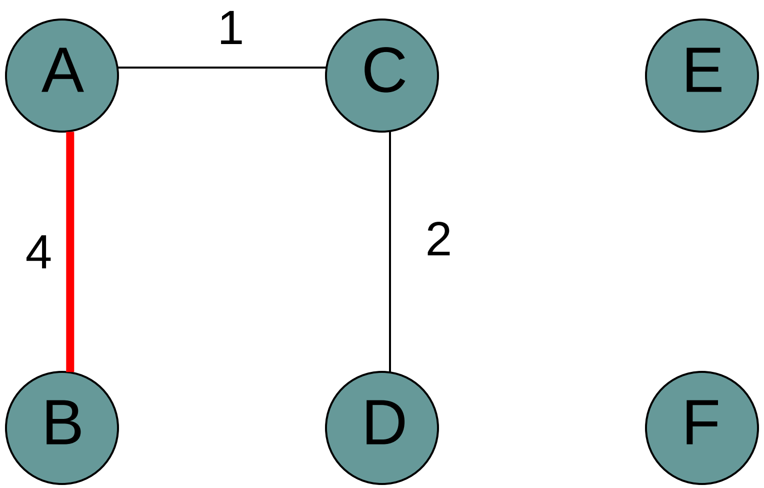
Kruskal's algorithm

Add smallest edge that connects two sets not already connected



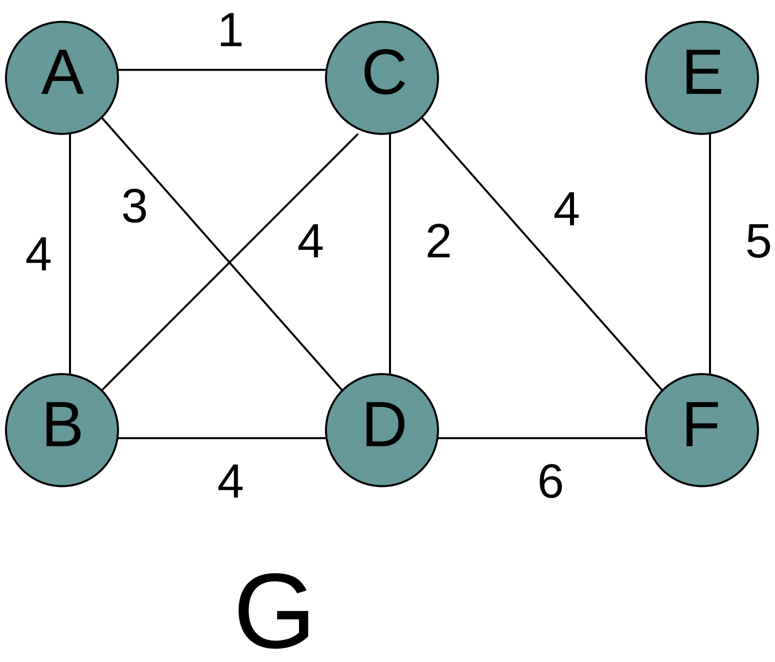
G

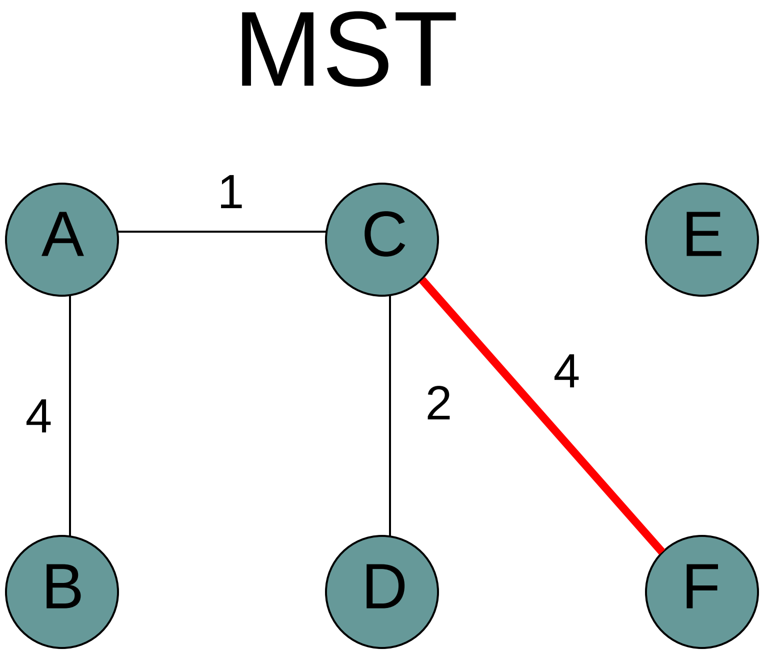
MST



Kruskal's algorithm

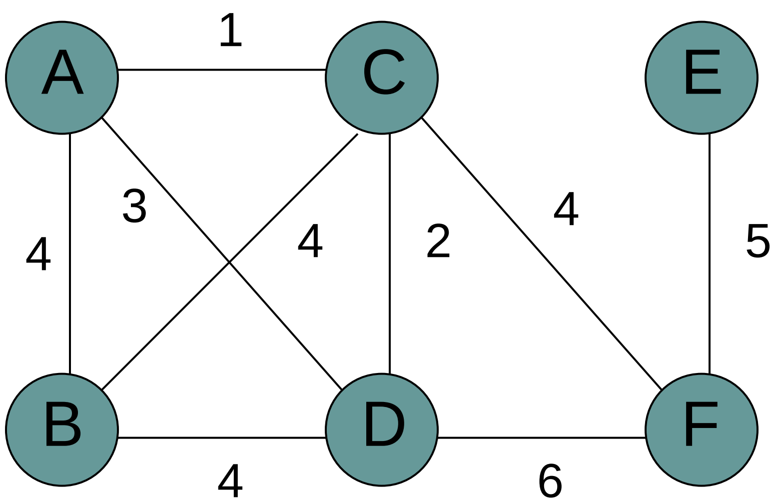
Add smallest edge that connects two sets not already connected





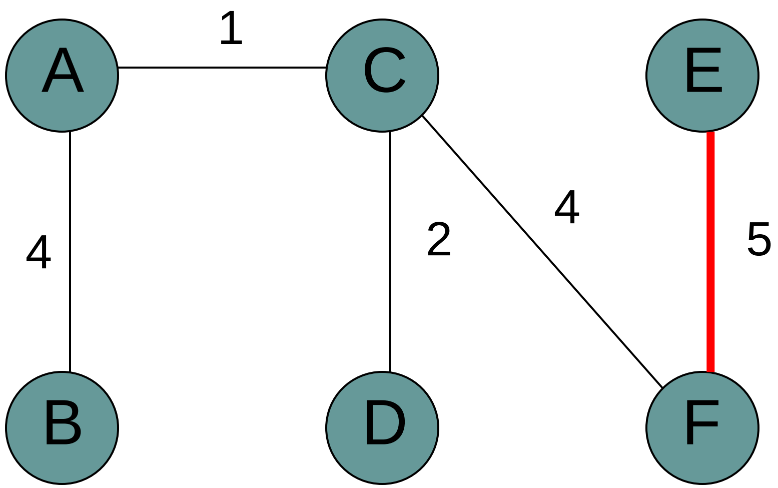
Kruskal's algorithm

Add smallest edge that connects two sets not already connected



G

MST



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))
```

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))
```

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))

|V| calls to MakeSet

O(|E| log |E|)

2 |E| calls to FindSet

|V| calls to Union

Running time of Kruskal’s

Disjoint set data structure

O(|E| log |E|) +

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

Disjoint set

A B C D

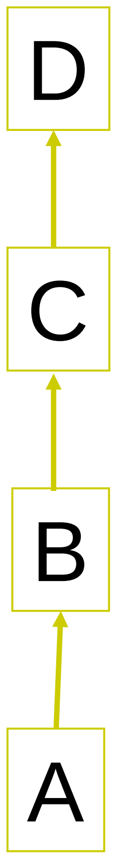
Disjoint set: union

A B C D

Disjoint set: union

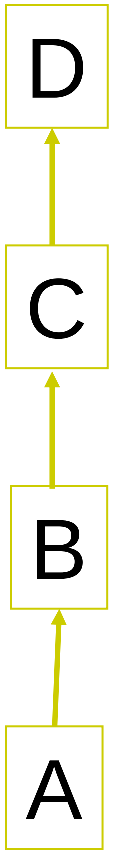
Disjoint set: union

Disjoint set: union



Running time?

Disjoint set: union



O(1)

Disjoint set: find-set

Search each linked list

Disjoint set: find-set

Running time?

Disjoint set: find-set

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

Running time of Kruskal's

Disjoint set data structure

O(|E| log |E|) +

	MakeSet	FindSet |E| calls	Union |V| calls	Total
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|)

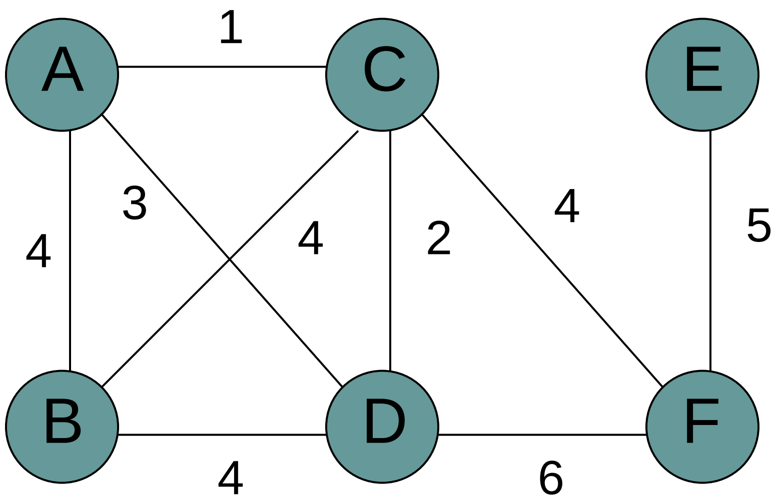
Prim’s algorithm

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

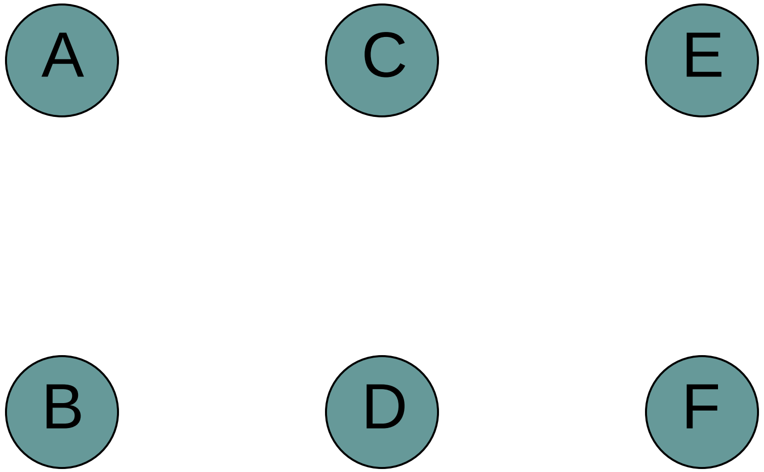
```
Prim(G, r)
for all v ∈ V
        key[v] ← ∞
        prev[v] ← null
key[r] ← 0
H ← MakeHeap(key)
while !Empty(H)
        u ← Extract-Min(H)
        visited[u] ← true
        for each edge (u, v) ∈ E
                if !visited[v] and w(u, v) < key(v)
                        Decrease-Key(v, w(u, v))
                        prev[v] ← u
```

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
```

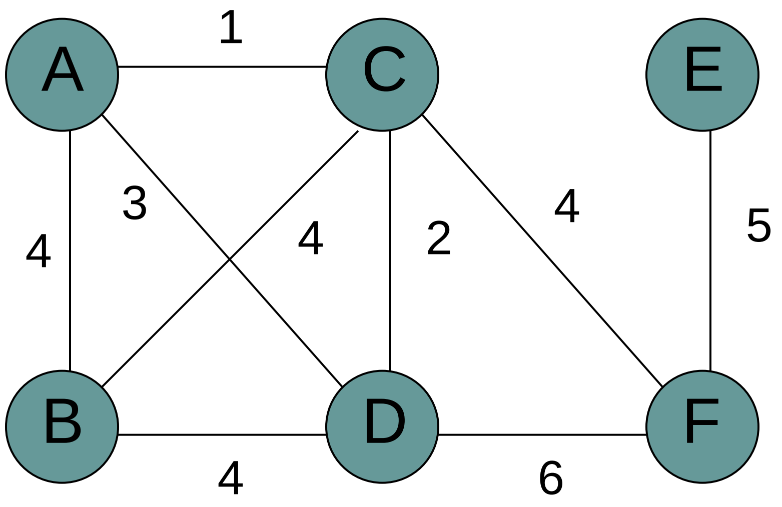


MST

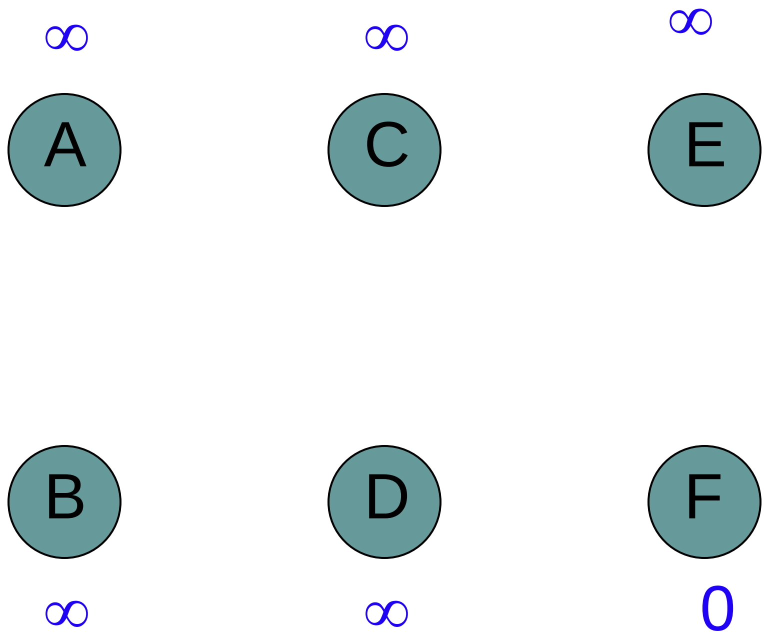


Prim's

```
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
```

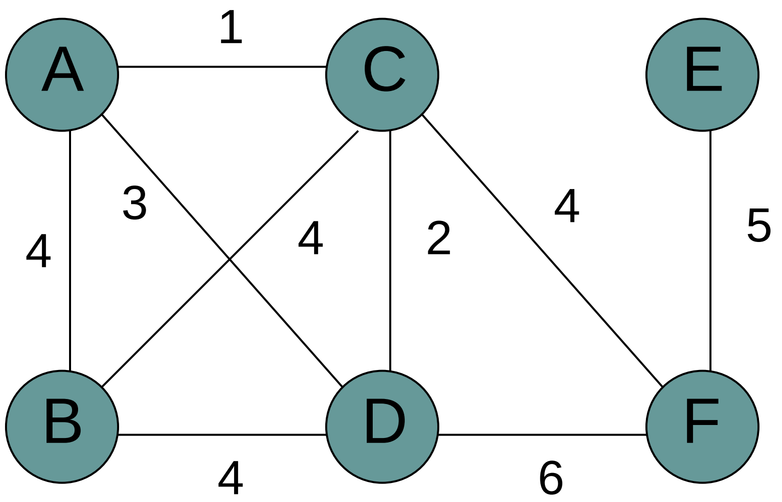


MST

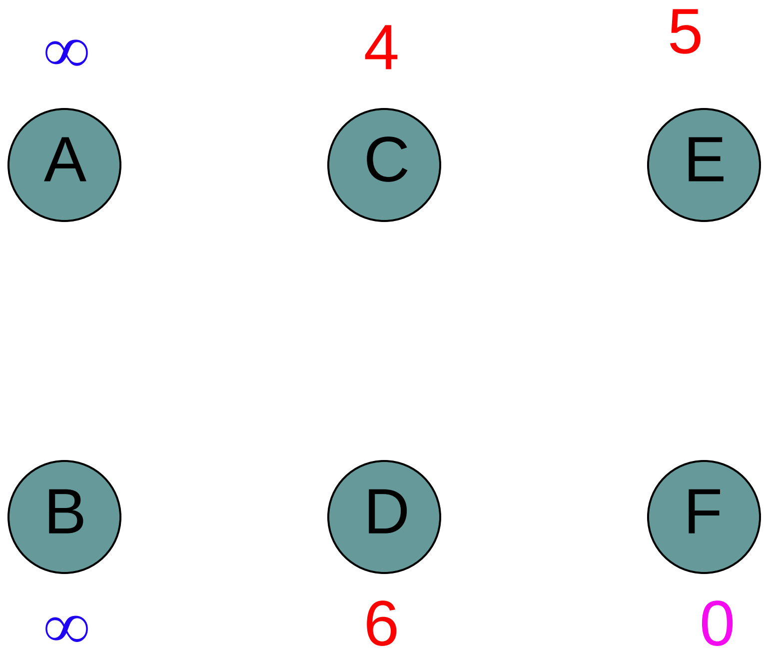


Prim's

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

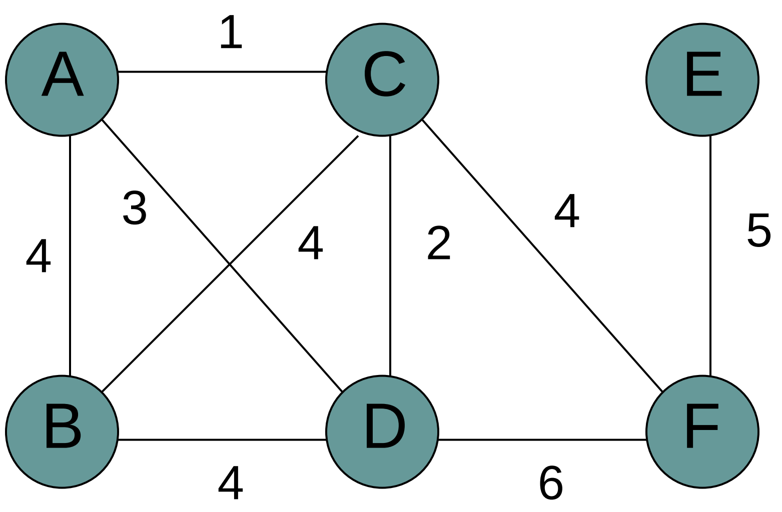


MST

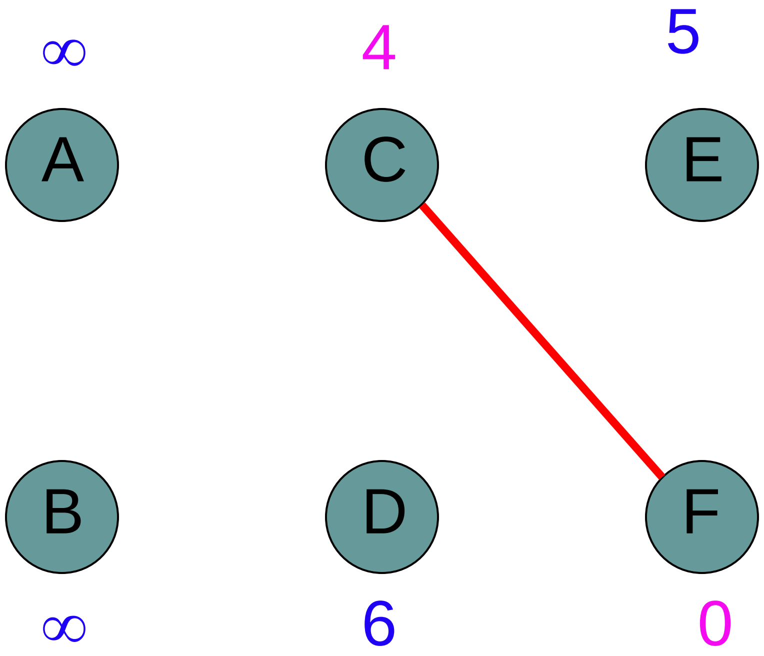


Prim's

```
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
```

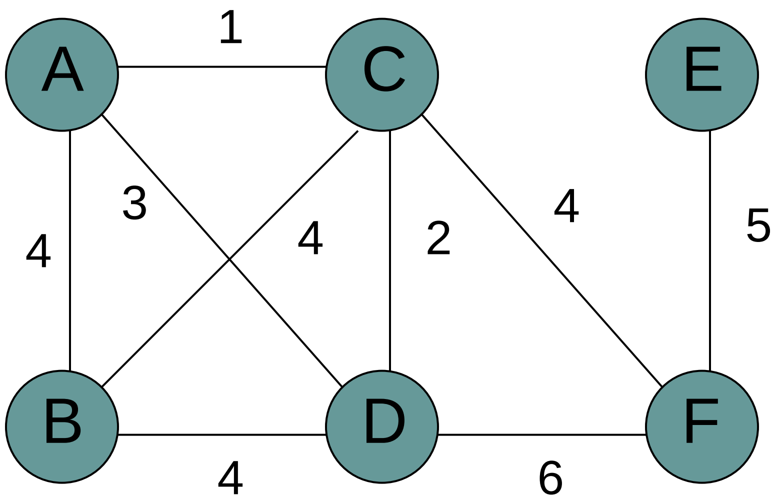


MST

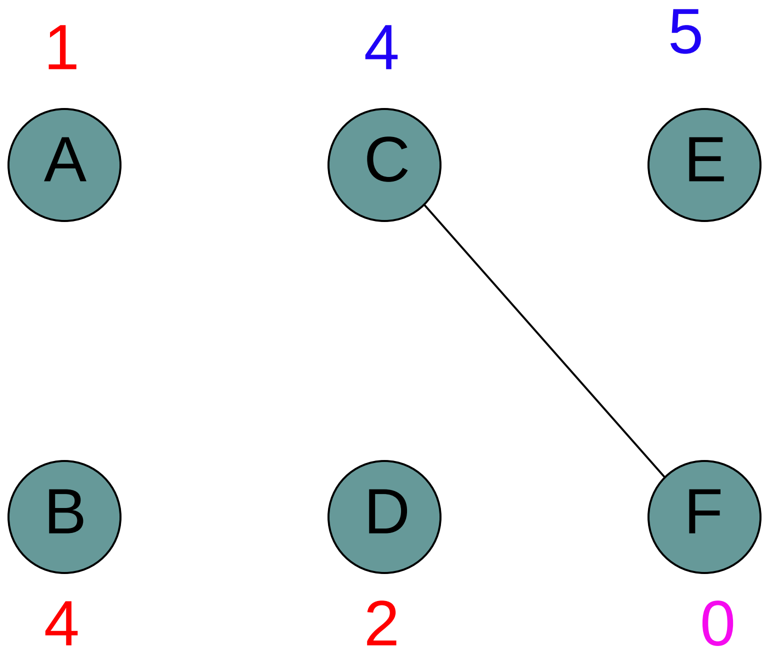


Prim's

```
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
```

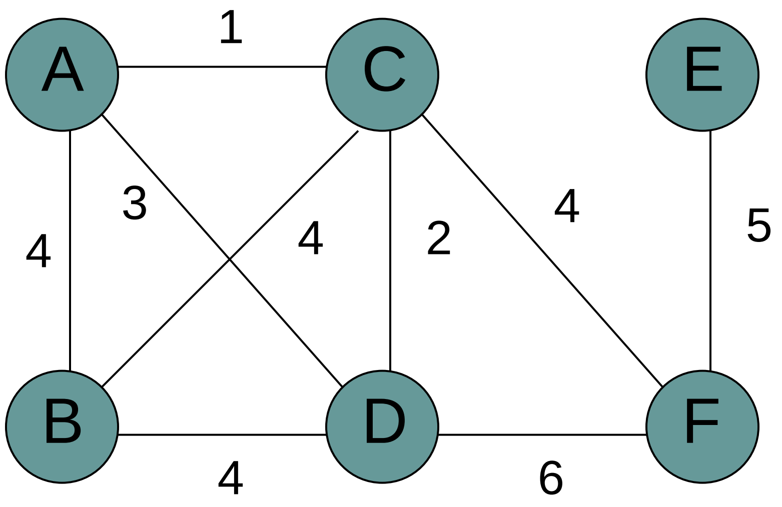


MST

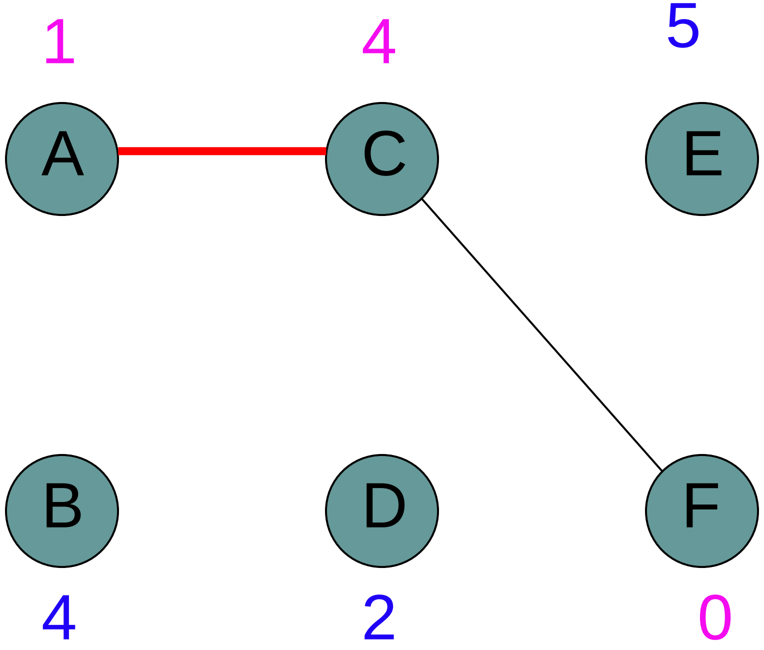


Prim's

```
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
```

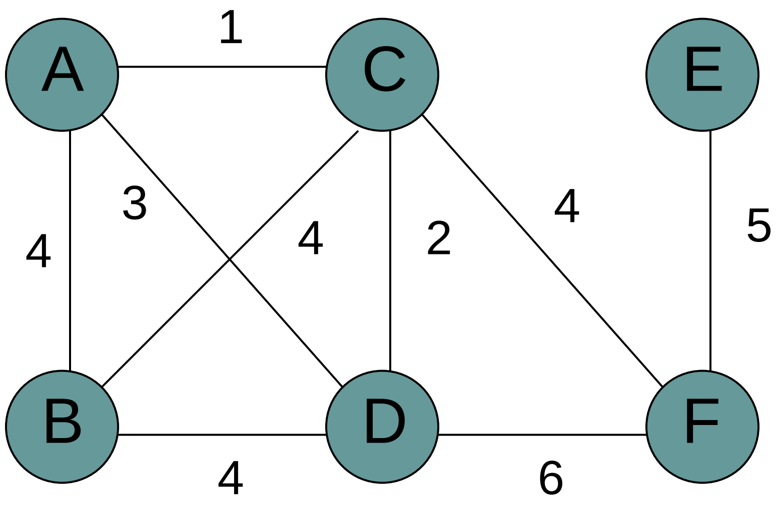


MST

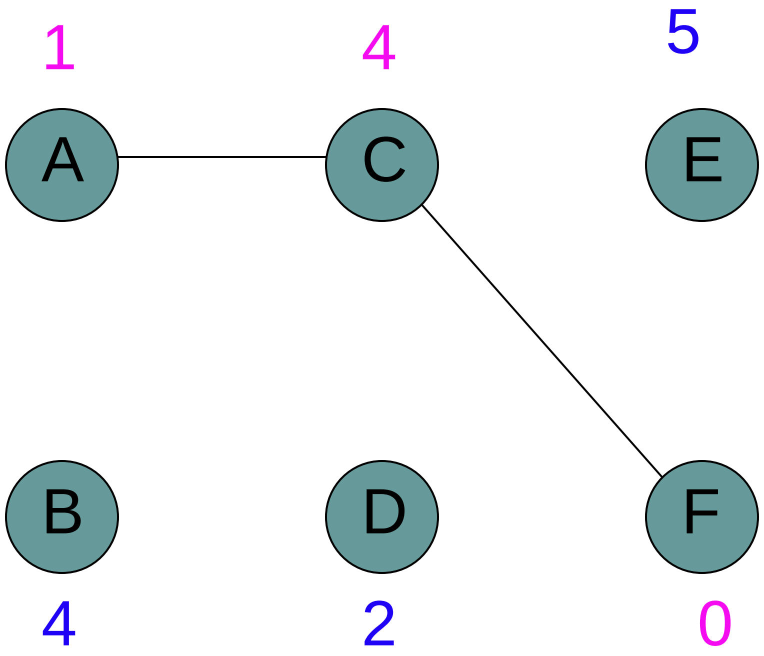


Prim’s

```
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
```

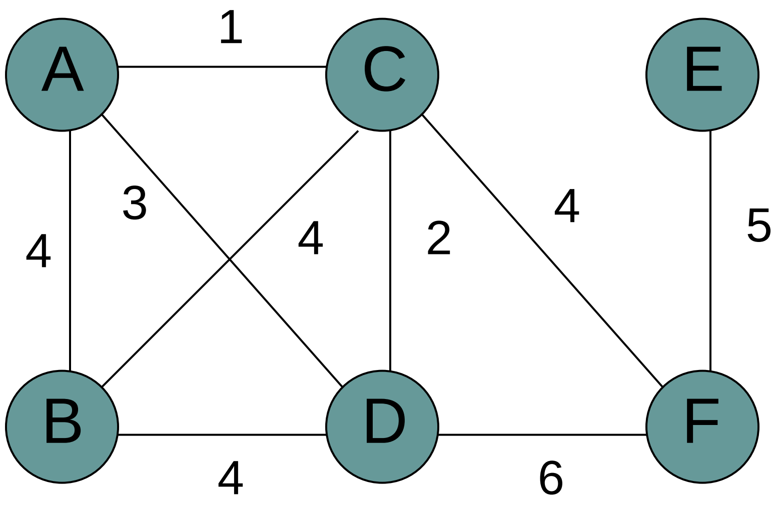


MST

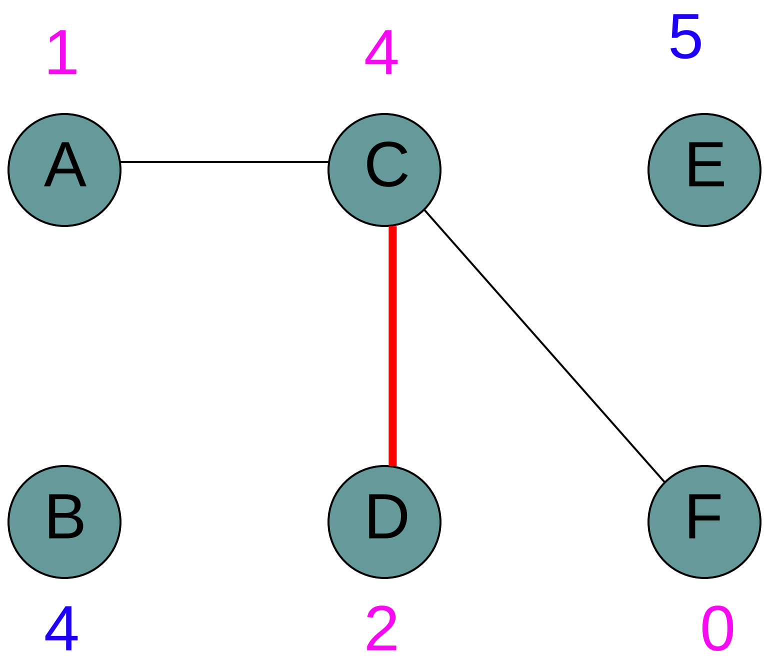


Prim's

```
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
```

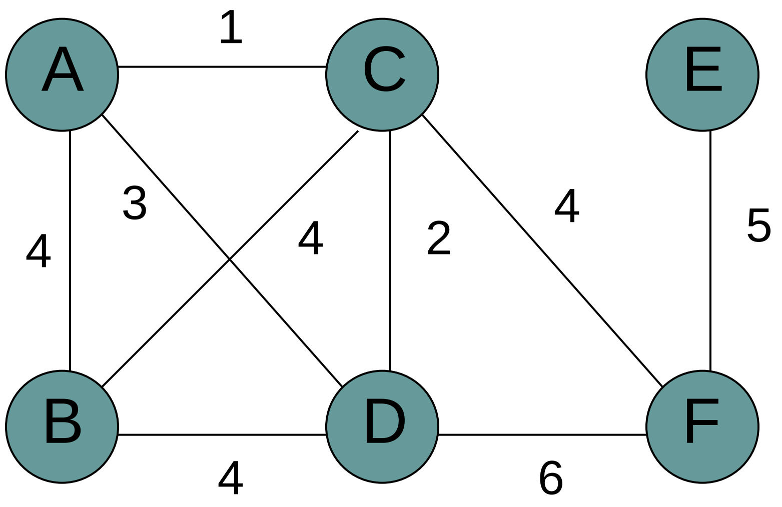


MST

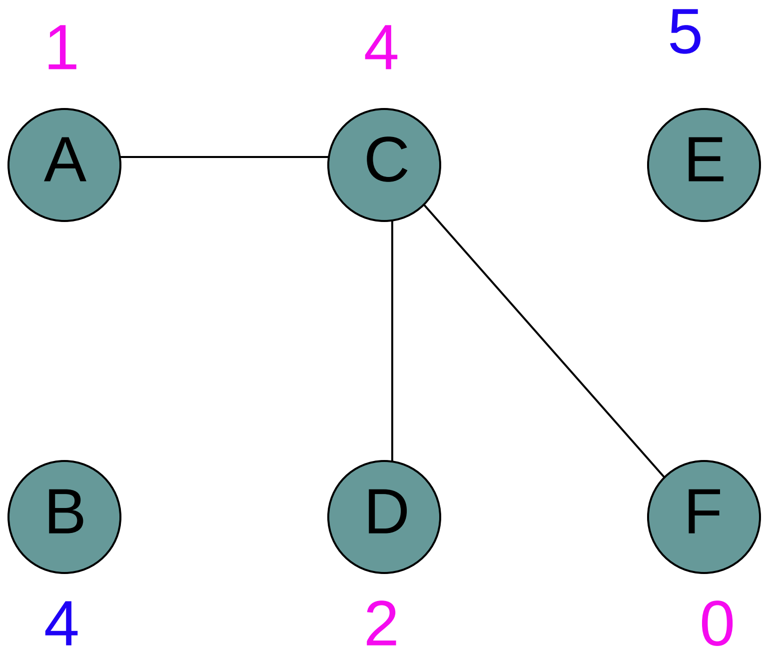


Prim's

```
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
```

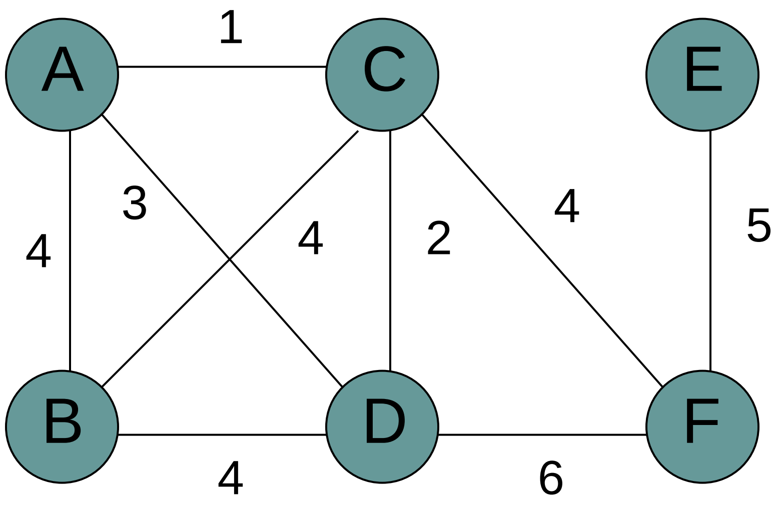


MST

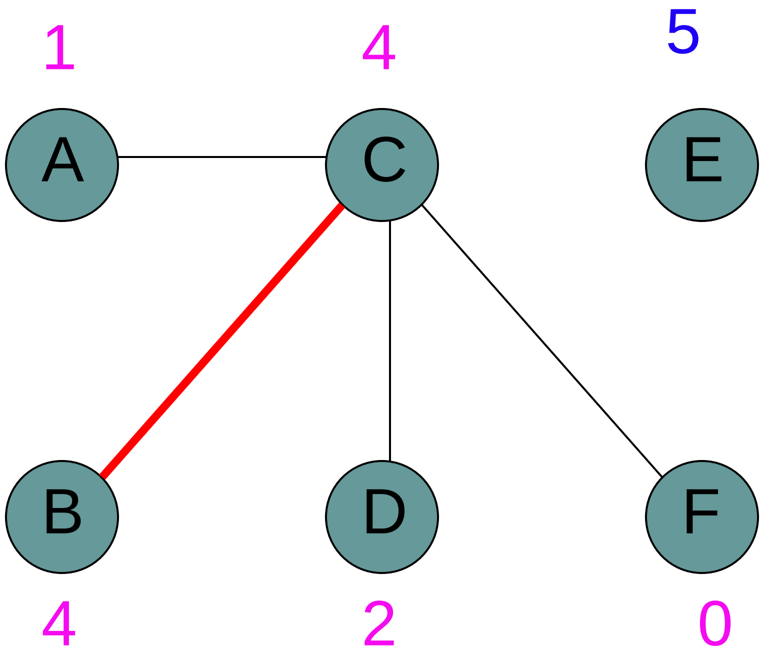


Prim's

```
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
```

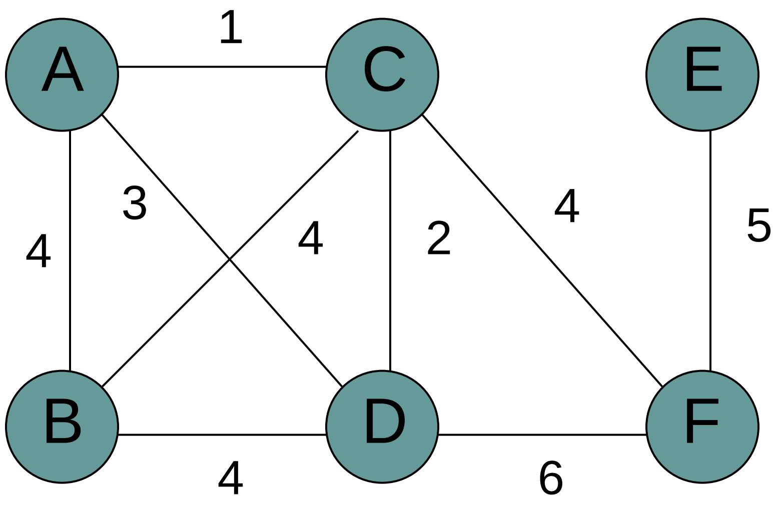


MST

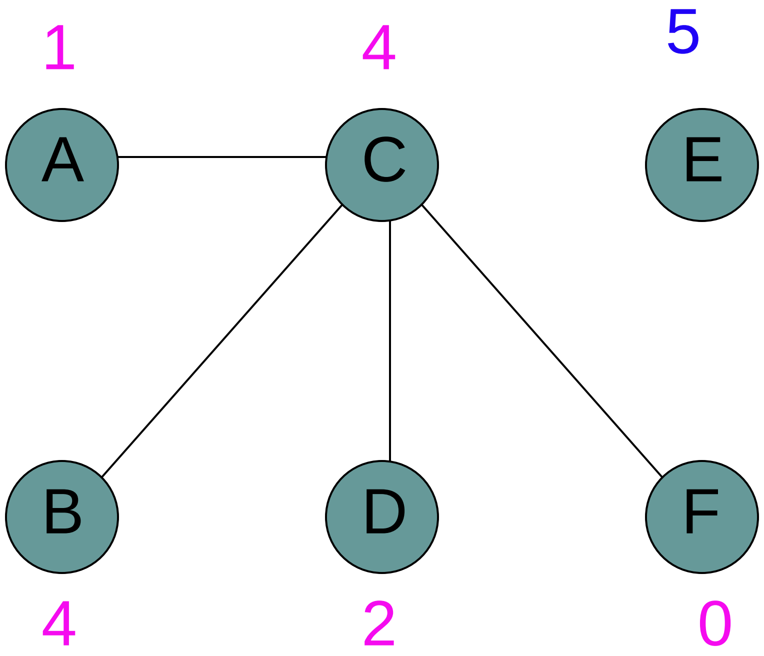


Prim's

```
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
```

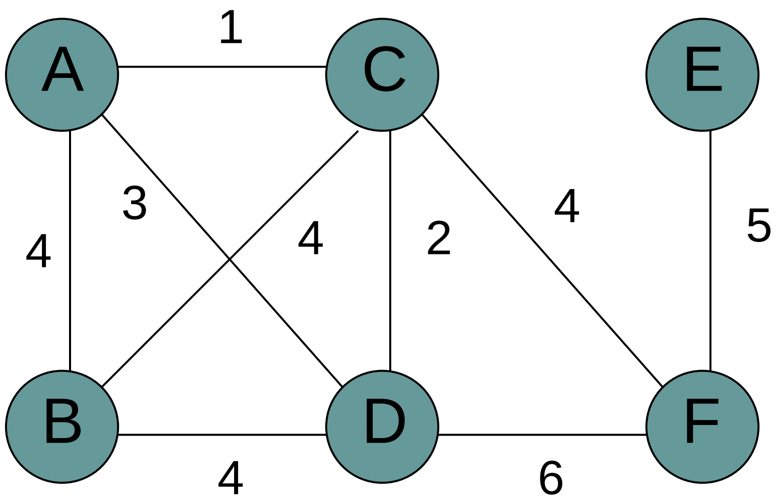


MST

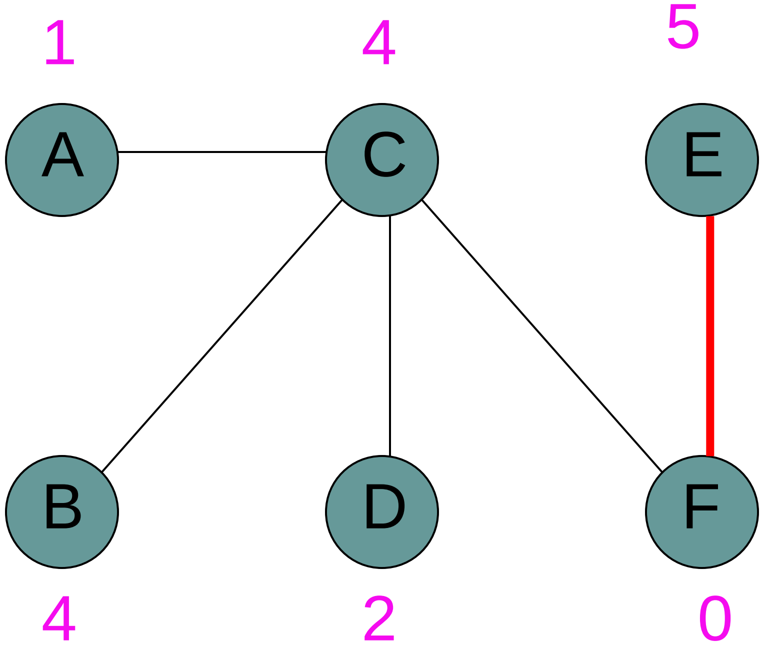


Prim's

```
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



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

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
```

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
```

$\Theta(|V|)$

1 call to MakeHeap

|V| calls to Extract-Min

|E| calls to Decrease-Key

Running time of Prim's

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

Kruskal's: $O(\lvert E\rvert \log \lvert E\rvert)$

Acknowledgements

- Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C., Introduction to algorithms. MIT press, 2009
- Dr. David Kauchak, Pomona College
- Prof. David Plaisted, The University of North Carolina at Chapel Hill