CS 344

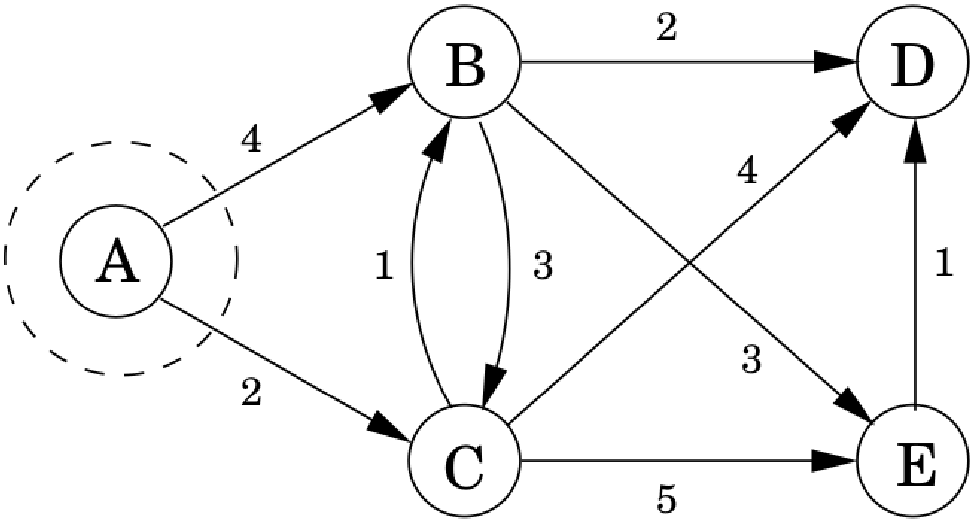
LECTURE 9

MINIMUM SPANNING TREES

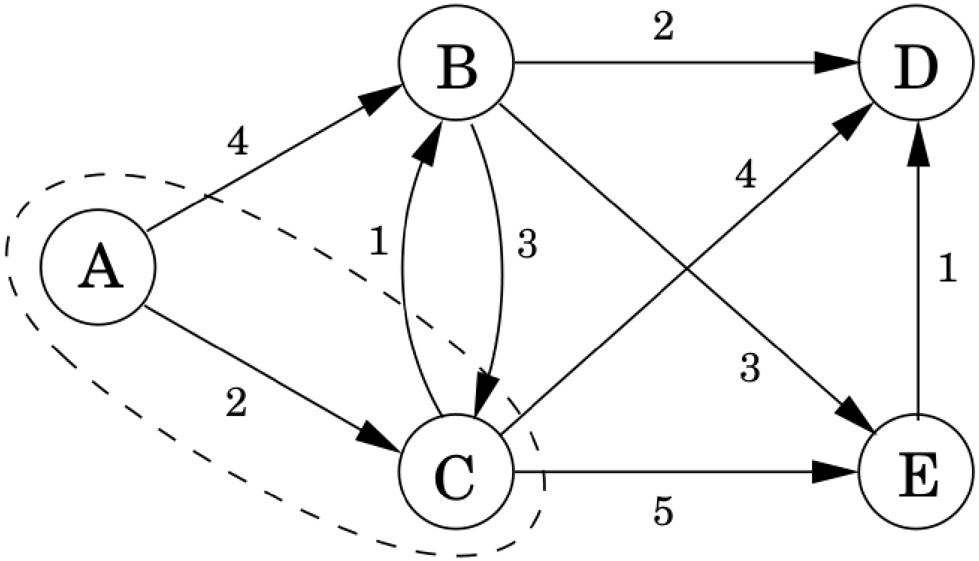
DIJKSTRA'S ALGORITHM

Given a graph G and a starting vertex s,
find shortest paths to all reachable vertices

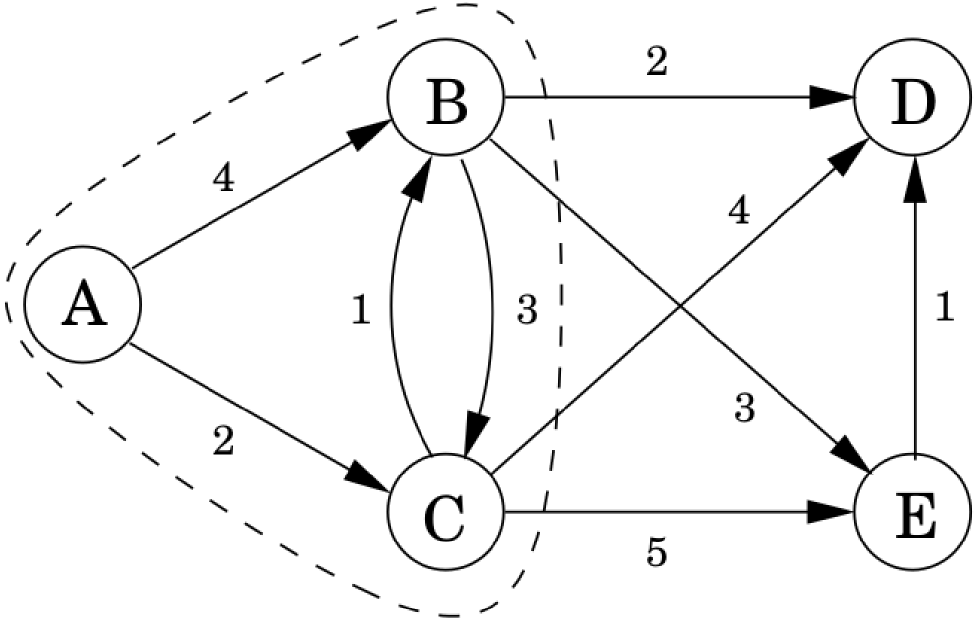
```
procedure dijkstra(G, l, s)
Input:    Graph G = (V, E), directed or undirected;
          positive edge lengths {l_e : e ∈ E}; vertex s ∈ V
Output:   For all vertices u reachable from s, dist(u) is set
          to the distance from s to u.

for all u ∈ V:
   dist(u) = ∞
   prev(u) = nil
dist(s) = 0

H = makequeue(V)  (using dist-values as keys)
while H is not empty:
   u = deletemin(H)
   for all edges (u, v) ∈ E:
      if dist(v) > dist(u) + l(u, v):
         dist(v) = dist(u) + l(u, v)
         prev(v) = u
         decreasekey(H, v)
```



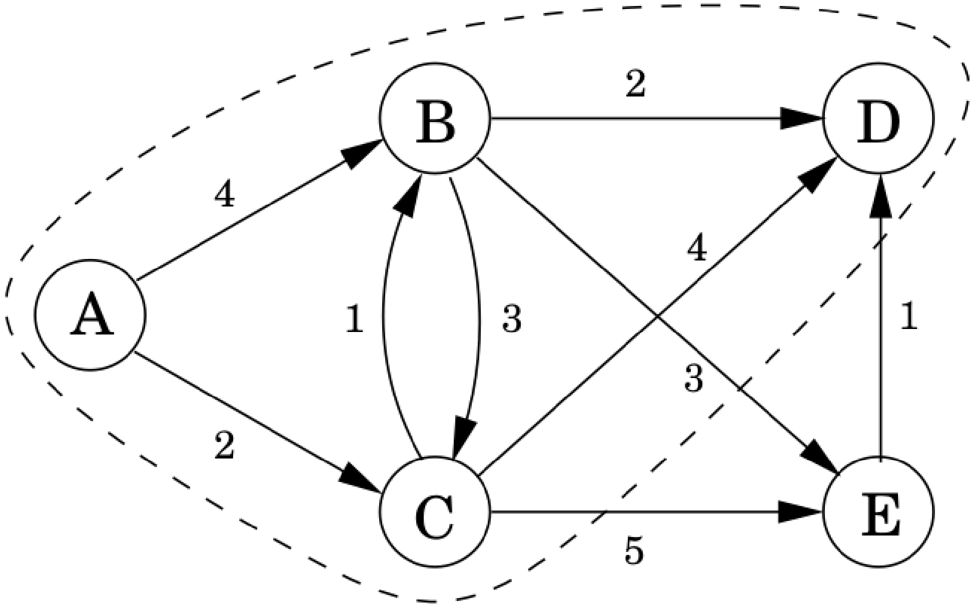
A: 0	**D**: ∞
B: 4	**E**: ∞
C: 2	



A: 0	**D**: 6
B: 3	**E**: 7
C: 2	

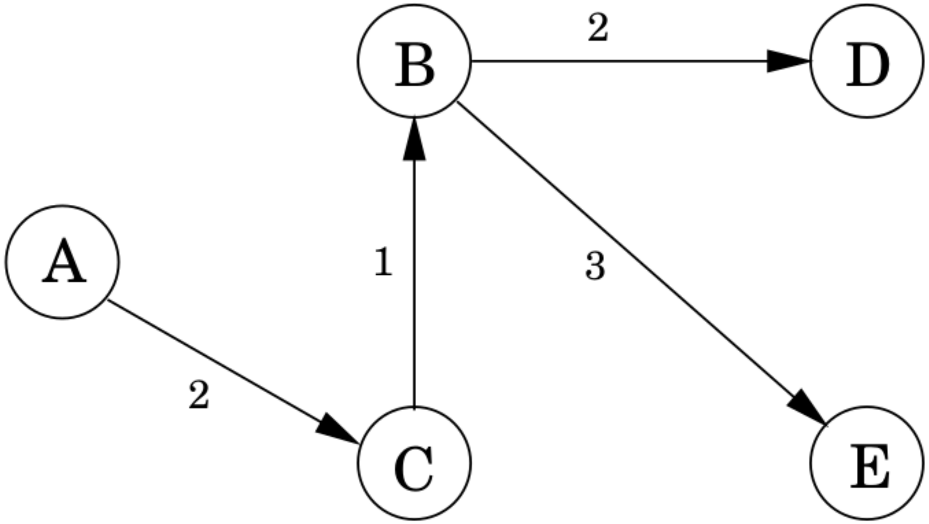


A: 0	D: 5
B: 3	E: 6
C: 2	

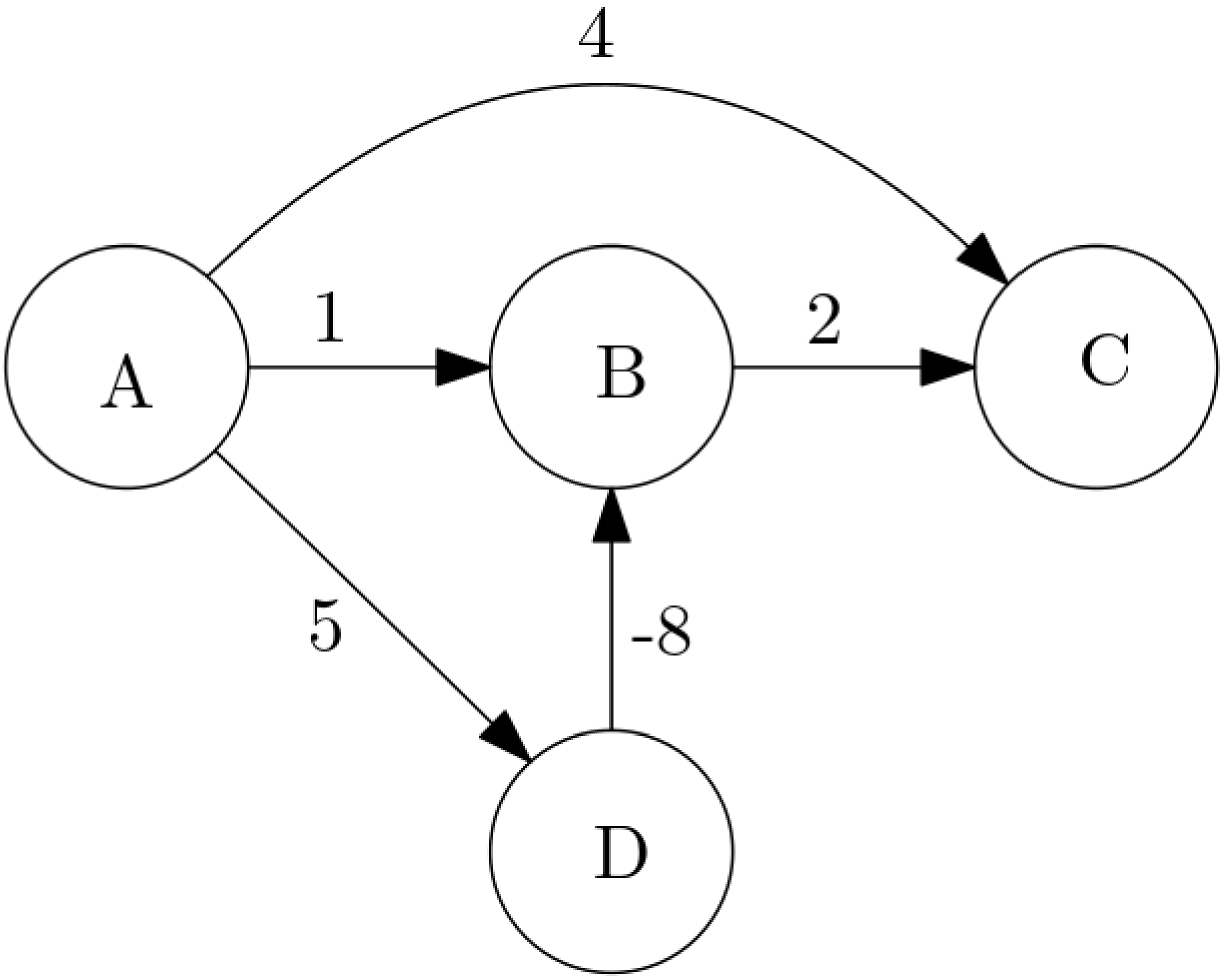


A: 0	**D**: 5
B: 3	**E**: 6
C: 2	

Finally, we get this tree for paths from A:



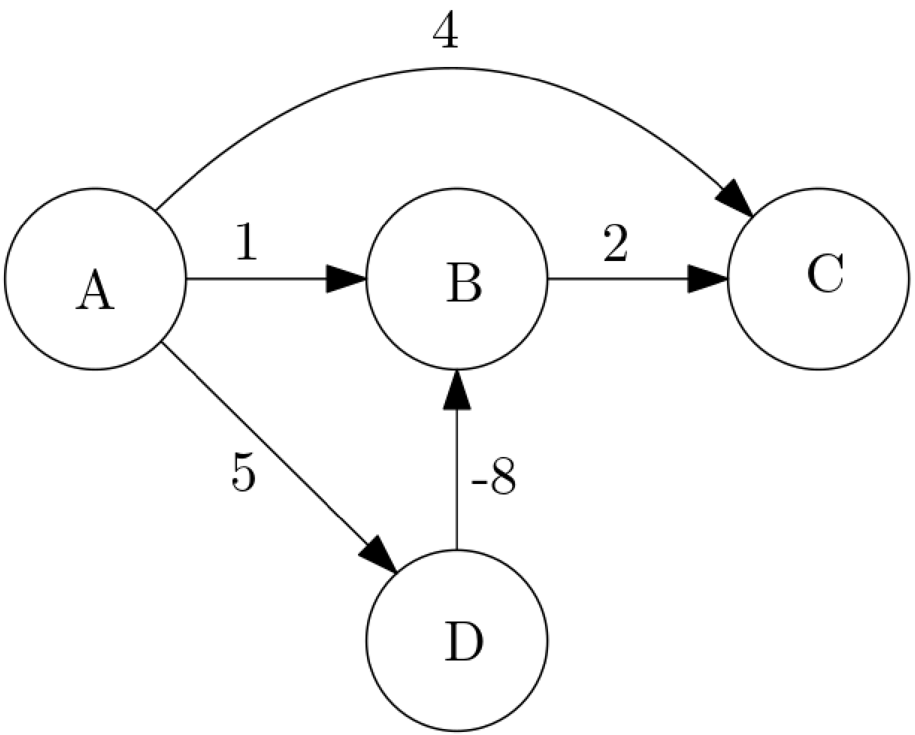
Dijkstra's algorithm with negative edges



Bellman-Ford algorithm:

```
procedure shortest-paths(G, l, s)
Input:    Directed graph G = (V, E);
          edge lengths {l_e : e ∈ E} with no negative cycles;
          vertex s ∈ V
Output:   For all vertices u reachable from s, dist(u) is set
          to the distance from s to u.

for all u ∈ V:
   dist(u) = ∞
   prev(u) = nil

dist(s) = 0
repeat |V| − 1 times:
   for all e ∈ E:
      update(e)
```

Bellman-Ford algorithm:



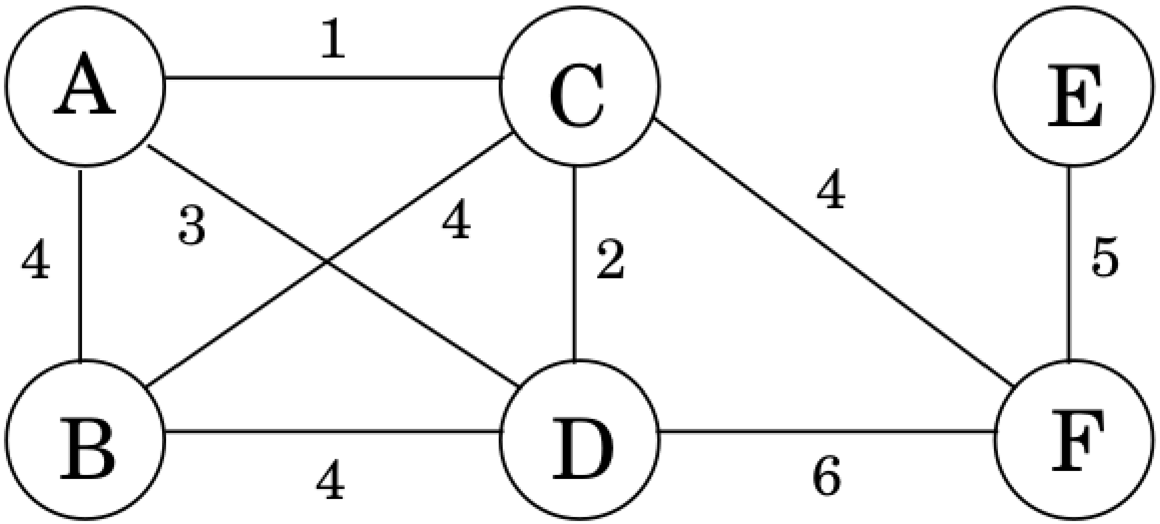
A	0	0	0	0
B	∞	1	-3	-3
C	∞	4	3	-1
D	∞	5	5	5

GREEDY ALGORITHM

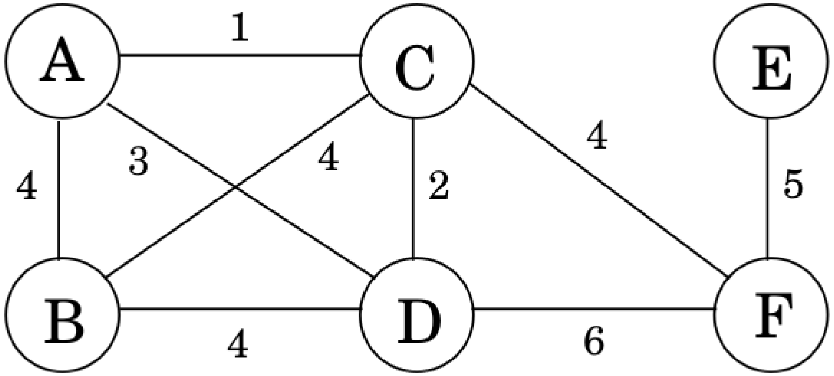
- Builds up a solution piece by piece
- Picks the best option at each stage

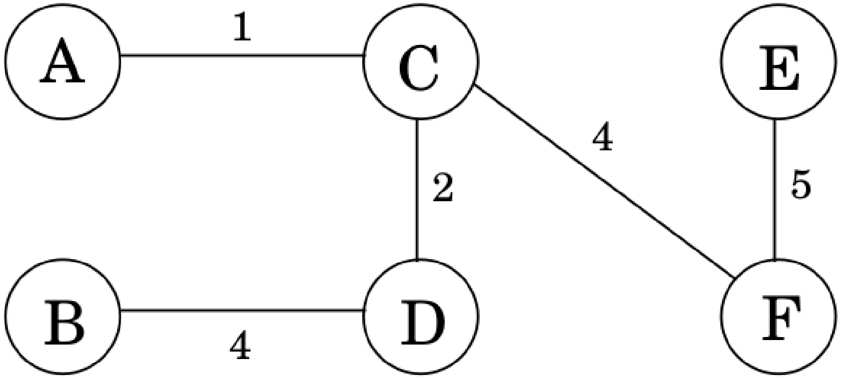
MINIMUM SPANNING TREE

- We want to connect all vertices in a graph with minimum cost



Note that removing a cycle edge cannot disconnect a graph


A
C
E
B
D
F
1
4
3
4
2
4
5
4
6


A
C
E
B
D
F
1
2
4
5
4

MINIMUM SPANNING TREE

- Tree (no cycles)
- Connects all vertices (spanning)
- Minimum cost

MINIMUM SPANNING TREE

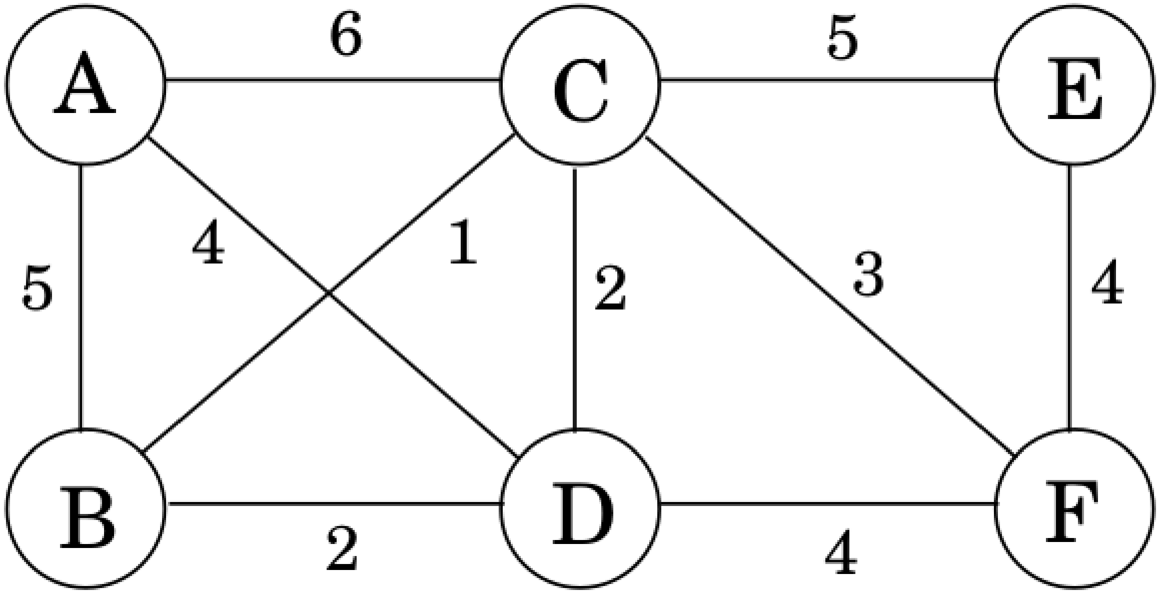
Given an undirected graph $G = (V, E)$ with edge weights w_e,
find a tree $T = (V, E')$, where $E' \subseteq E$,
that minimizes

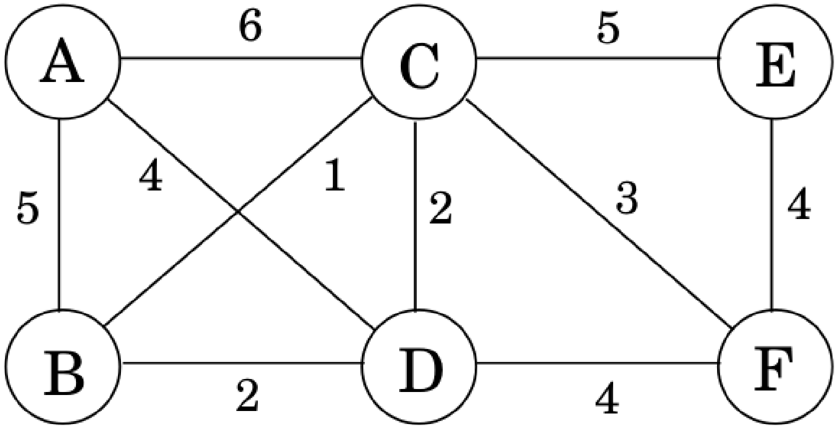
$$\text{weight}(T) = \sum_{e \in E'} w_e$$

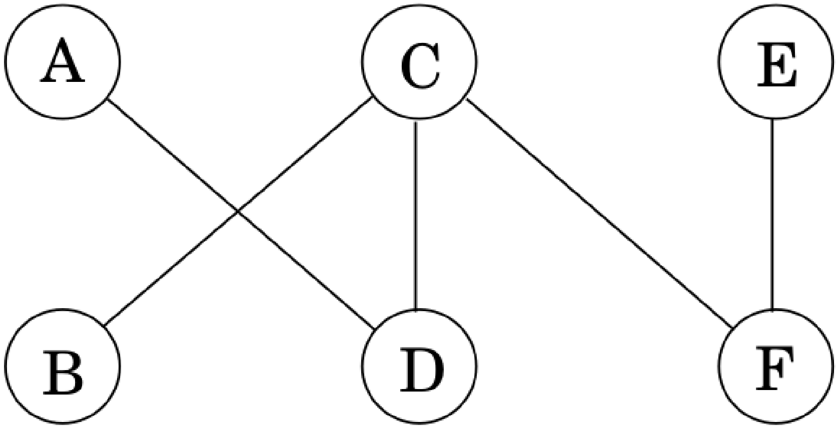
KRUSKAL'S ALGORITHM

Start with the empty set

Repeatedly add the next lightest edge that doesn't create a cycle


A
C
E
B
D
F
6
5
5
4
1
2
3
4
2
4


A
C
E
B
D
F
6
5
5
4
1
2
3
4
2
4


A
C
E
B
D
F

ASIDE: TREES

- undirected graph
- connected
- acyclic

NICE TREE PROPERTIES

A tree on n nodes has $n - 1$ edges

NICE TREE PROPERTIES

Any connected, undirected graph with $|E| = |V| - 1$ is a tree

NICE TREE PROPERTIES

An undirected graph is a tree iff there is a unique path between any pair of nodes

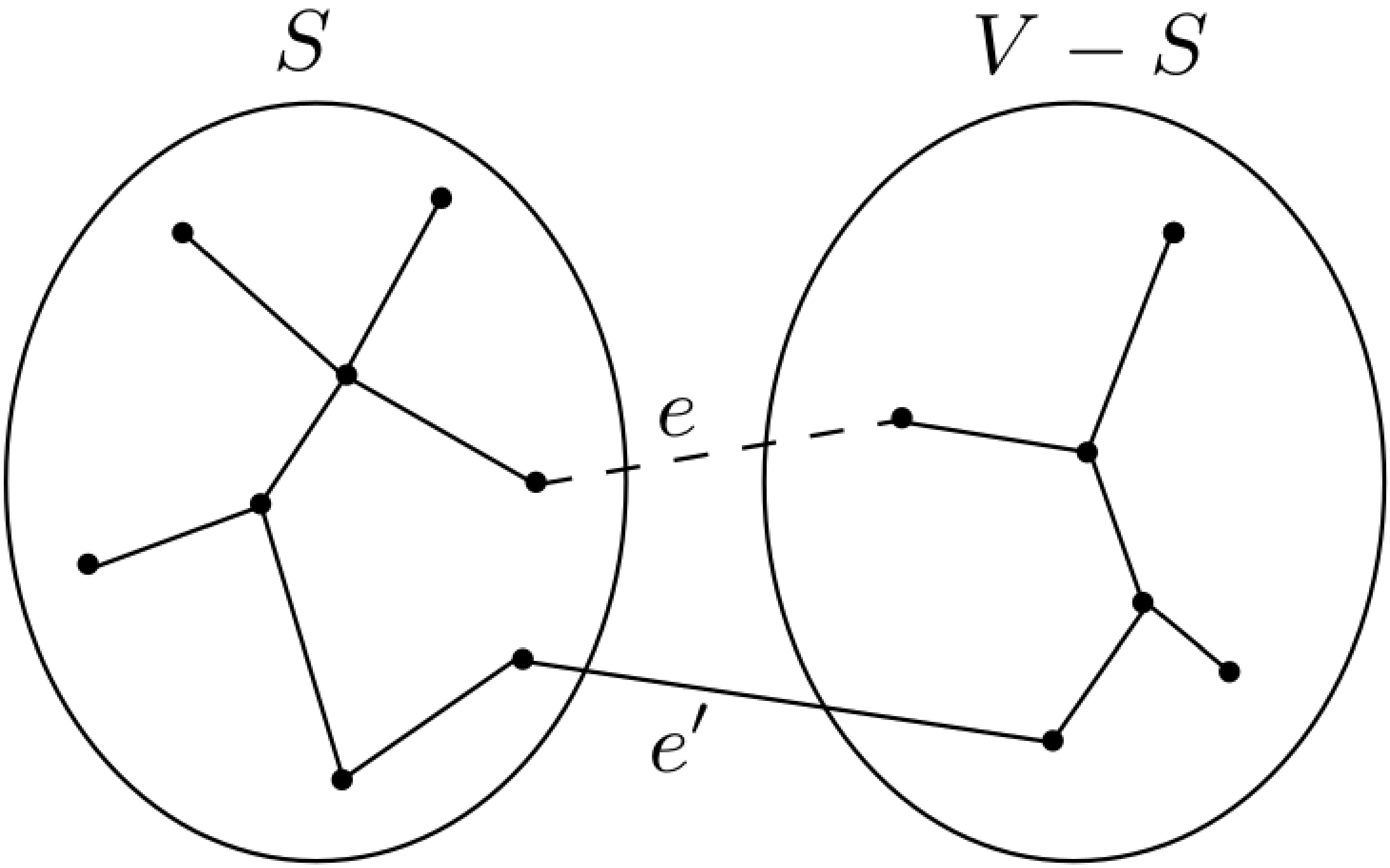
Is Kruskal's algorithm correct?

Cut property: Suppose edges X are part of a MST of G.

Pick any subset of nodes S, where none of the edges in X cross between S and $V - S$

Let e be the lightest edge across this cut.

Then $X \cup \{e\}$ is part of some MST.

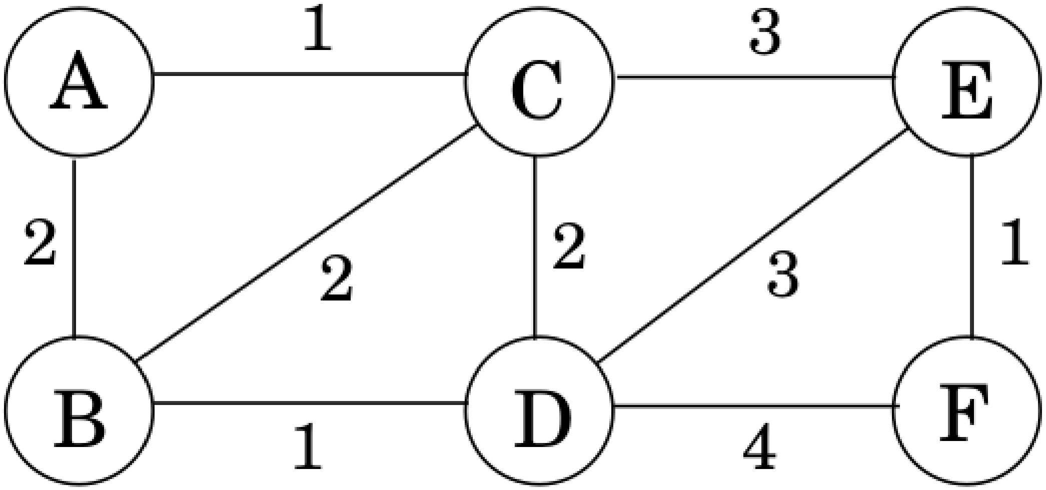

S
$V - S$
e
e'

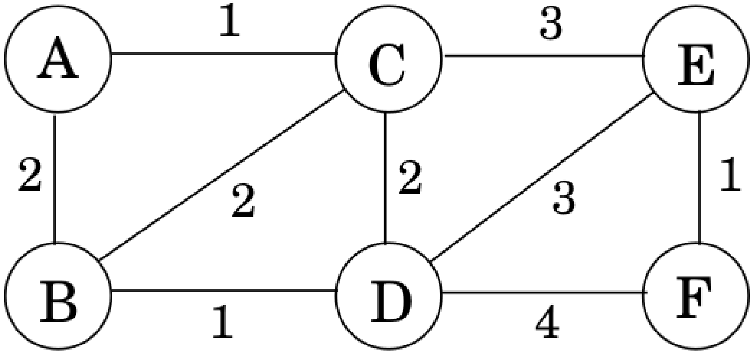
Then

$$\textbf{weight}(T') = \textbf{weight}(T) + w(e) - w(e')$$

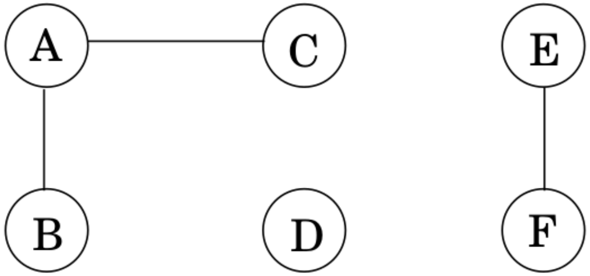
But $w(e) \leq w(e')$, so weight(T') $\leq$weight(T).

Since T is an MST, the weights must be equal, and T' is also an MST.

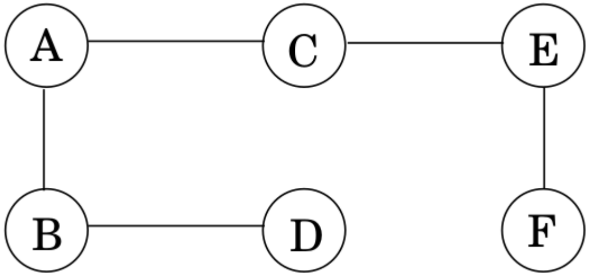

A
B
C
D
E
F
1
3
2
2
2
3
1
1
4

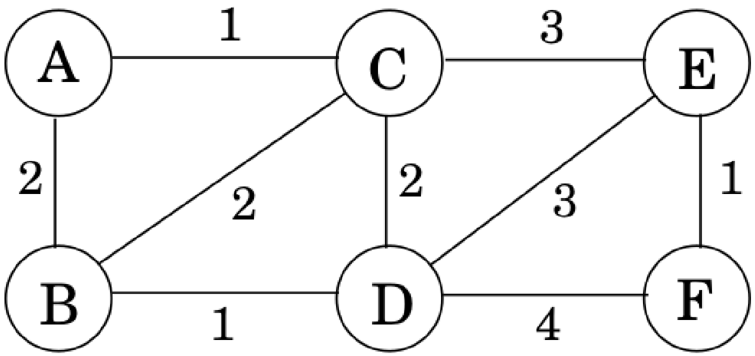


Edges X:

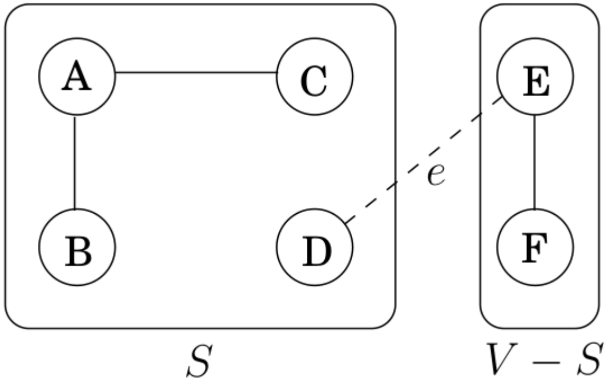


MST T:

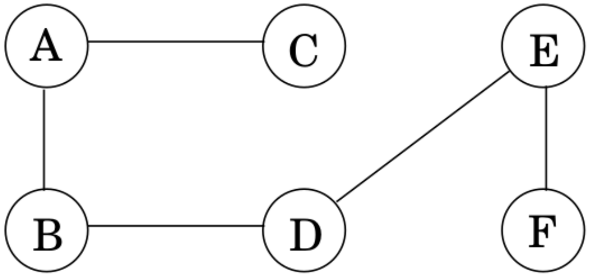




The cut:



MST T':



So Kruskal's algorithm starts with n trees (single vertices)

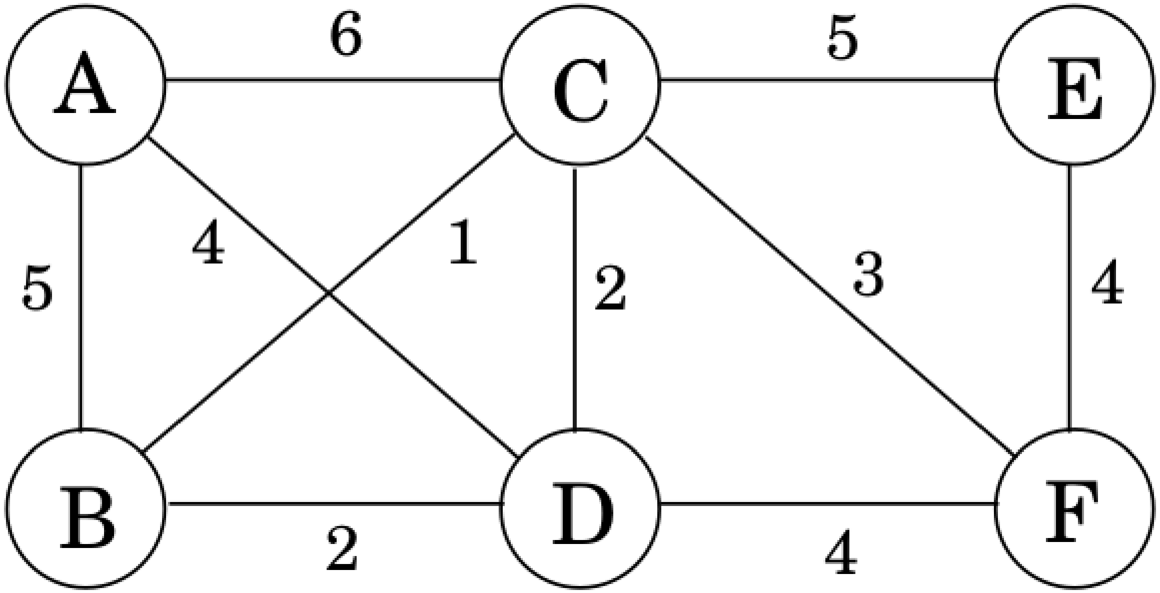
At each stage it adds an edge to connect two trees T_1 and T_2

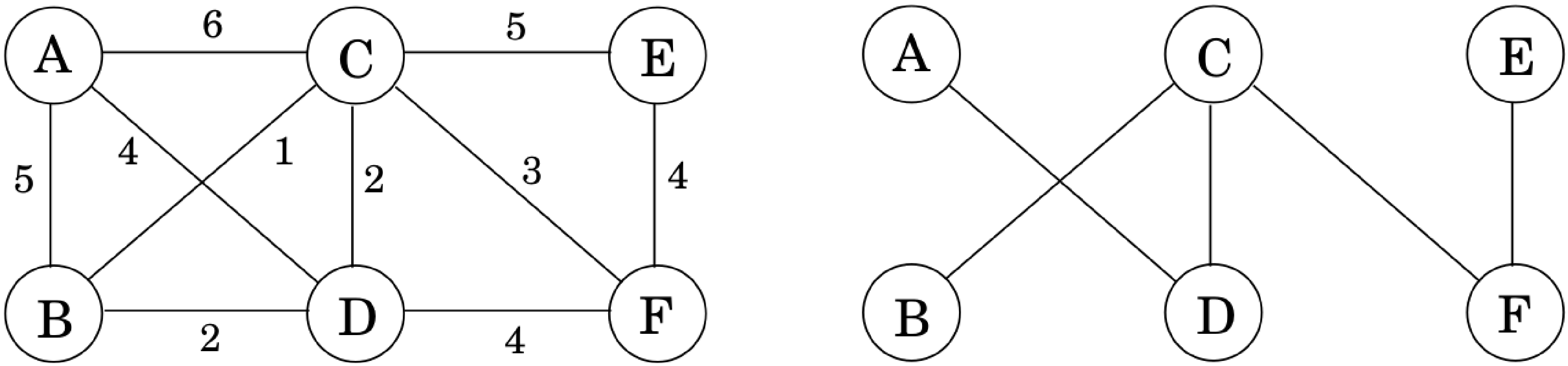
Let the cut be T_1 and $V - T_1$.

Since e is the lightest edge in the graph (that doesn't create a cycle), it must be the lightest edge in this cut.

- makeset(x) -- create a singleton set
- find(x) -- find which set x belongs to
- union(x, y) -- merge sets containing x and y

```
procedure kruskal(G, w)
Input:    A connected undirected graph G = (V, E) with edge weights w_e
Output:   A minimum spanning tree defined by the edges X

for all u ∈ V:
   makeset(u)

X = {}
Sort the edges E by weight
for all edges {u, v} ∈ E, in increasing order of weight:
   if find(u) ≠ find(v):
      add edge {u, v} to X
      union(u, v)
```

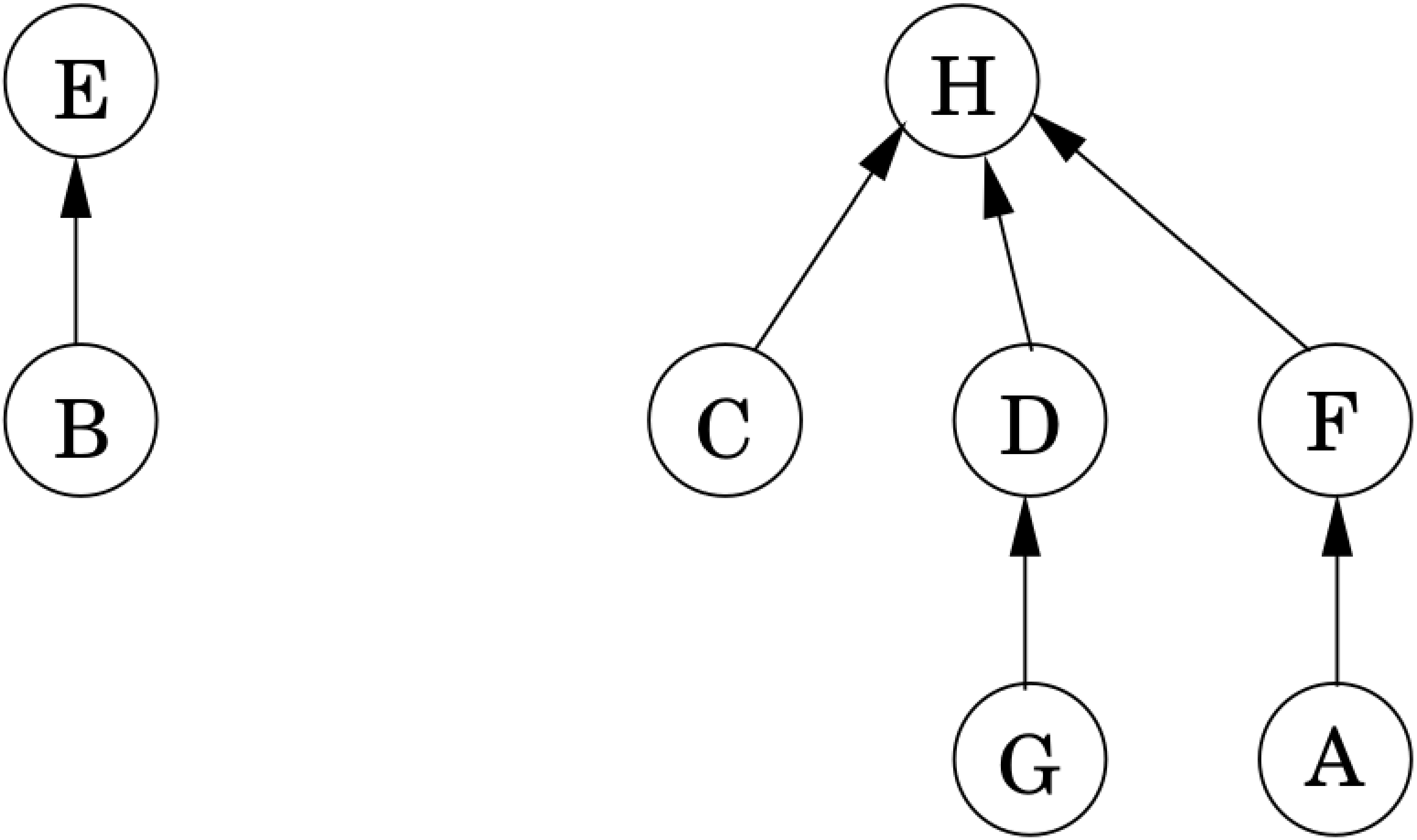

A
C
E
B
D
F
6
5
5
4
1
2
3
4
2
4


A
C
E
B
D
F
6
5
5
4
1
2
3
4
2
4
A
C
E
B
D
F

UNION BY RANK

- Use directed trees
- Each node has a parent pointer π
- Each node has a rank (height of its subtree)

Directed tree representation of sets $\{B, E\}$ and $\{A, C, D, F, G, H\}$:



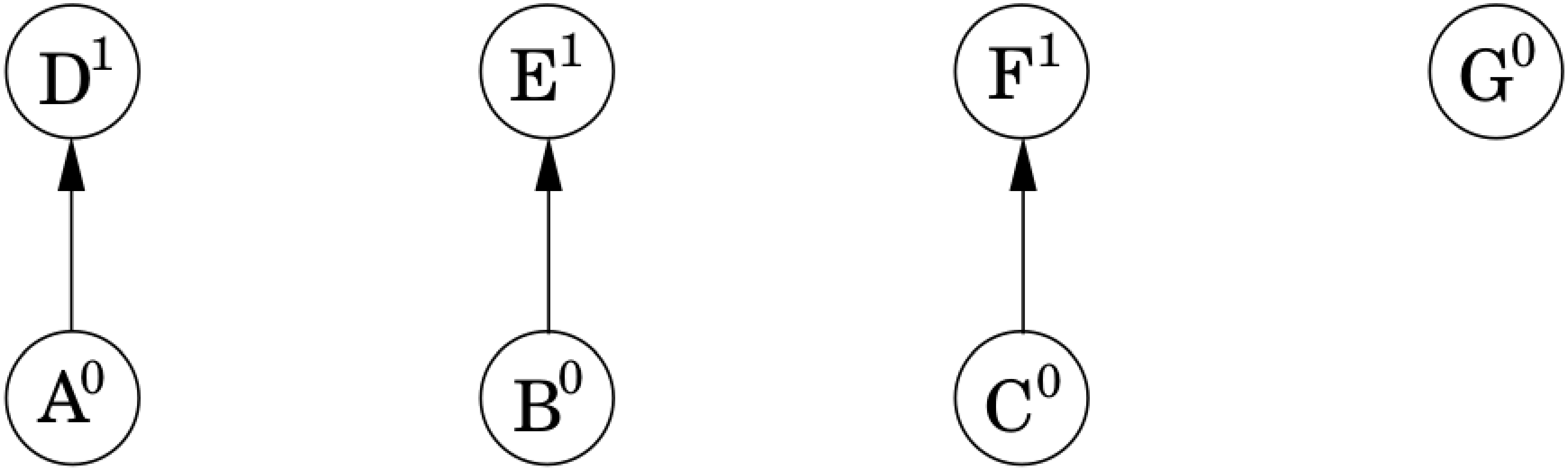
$\underline{\texttt{procedure makeset}}(x)$
$\pi(x) = x$
$\texttt{rank}(x) = 0$

$\underline{\texttt{function find}}(x)$
$\texttt{while } x \neq \pi(x): \quad x = \pi(x)$
$\texttt{return } x$

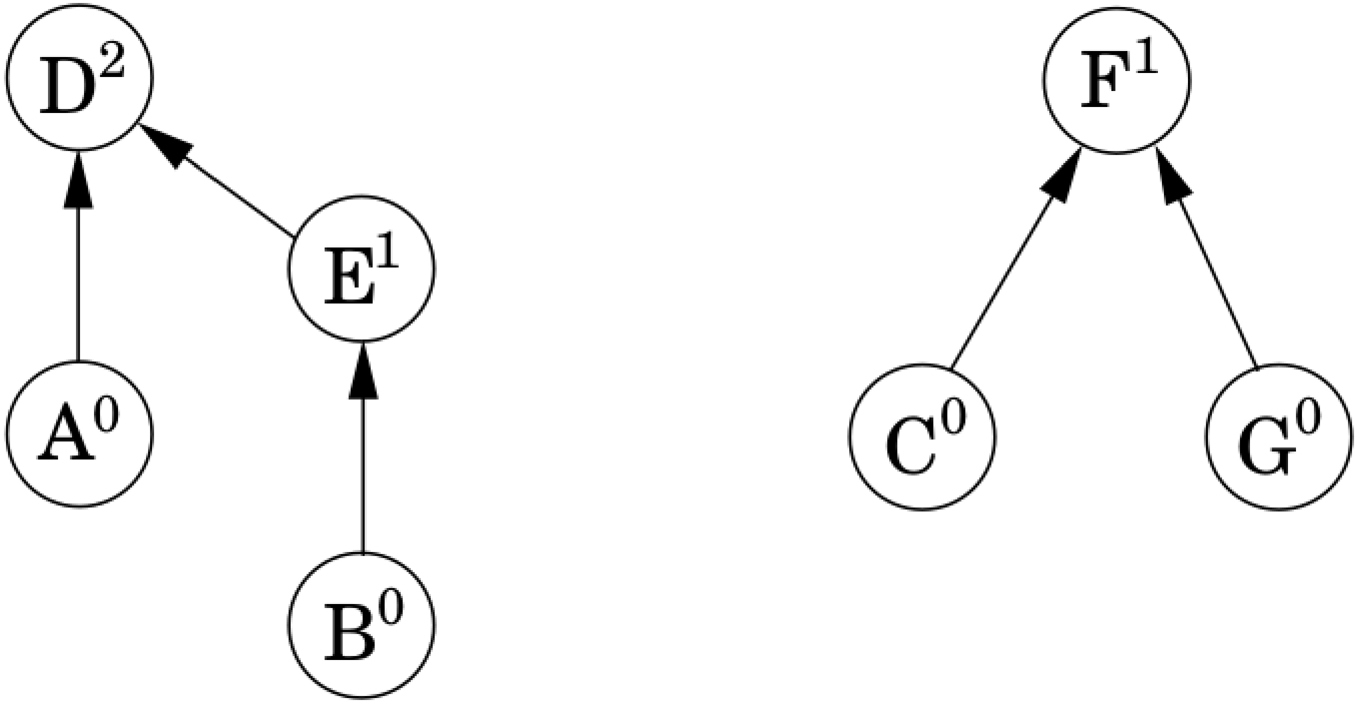
```
procedure union(x, y)
r_x = find(x)
r_y = find(y)
if r_x = r_y:   return
if rank(r_x) > rank(r_y):
    π(r_y) = r_x
else:
    π(r_x) = r_y
    if rank(r_x) = rank(r_y):  rank(r_y) = rank(r_y) + 1
```

After makeset on A through G:

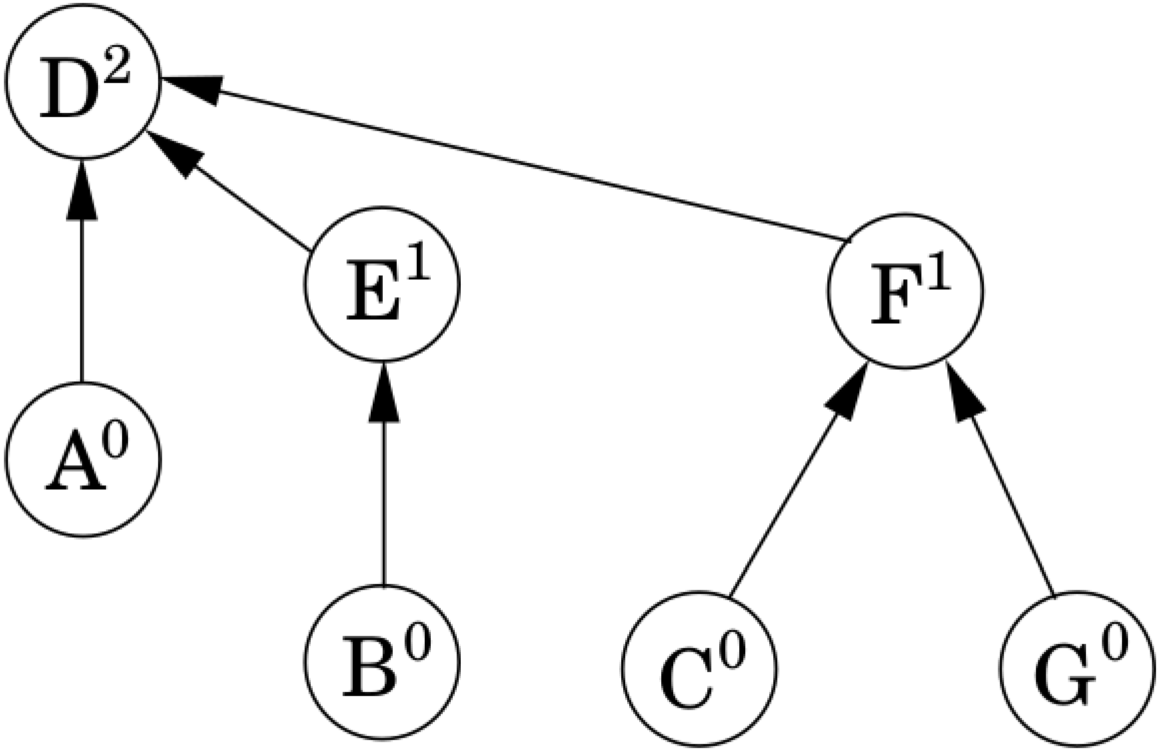
After union(A, D), union(B, E), union(C, F):



After union(C, G), union(E, A):



After union(B, G):



$$\mathrm{rank}(x) < \mathrm{rank}(\pi(x))$$

A root node of rank k is created by merging two trees with rank $k - 1$, so

a root node of rank k has at least 2^k nodes in its tree

(internal nodes too!)

If there are n elements, there can be at most $n/2^k$ nodes of rank k.

So the maximum rank is $\log n$

```
procedure kruskal(G, w)
Input:    A connected undirected graph G = (V, E) with edge weights w_e
Output:   A minimum spanning tree defined by the edges X

for all u ∈ V:
   makeset(u)

X = {}
Sort the edges E by weight
for all edges {u, v} ∈ E, in increasing order of weight:
   if find(u) ≠ find(v):
      add edge {u, v} to X
      union(u, v)
```

Time for Kruskal's

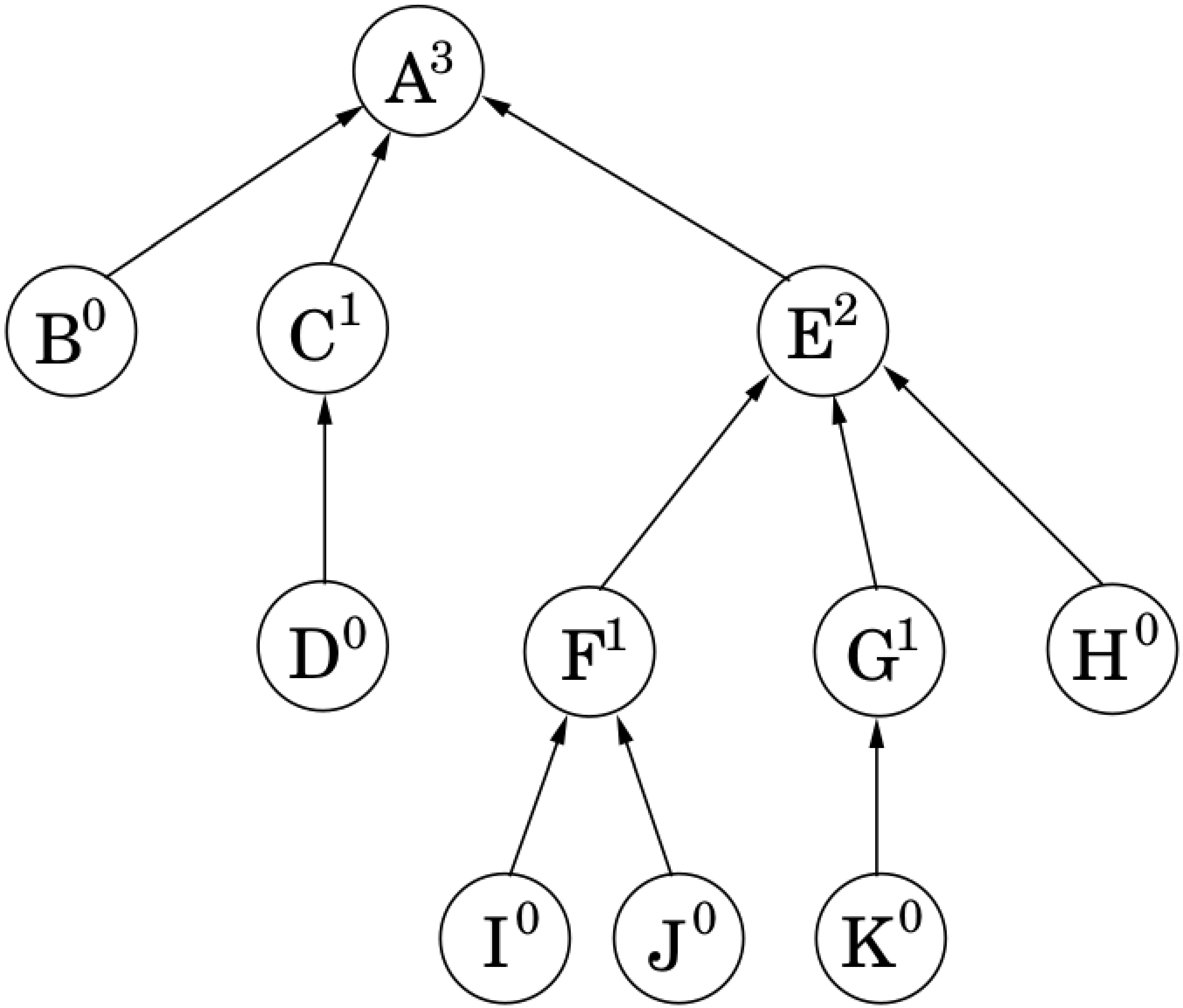
- $|V|$ makeset operations
- $2|E|$ find operations
- $|V| - 1$ union operations

Time for Kruskal's

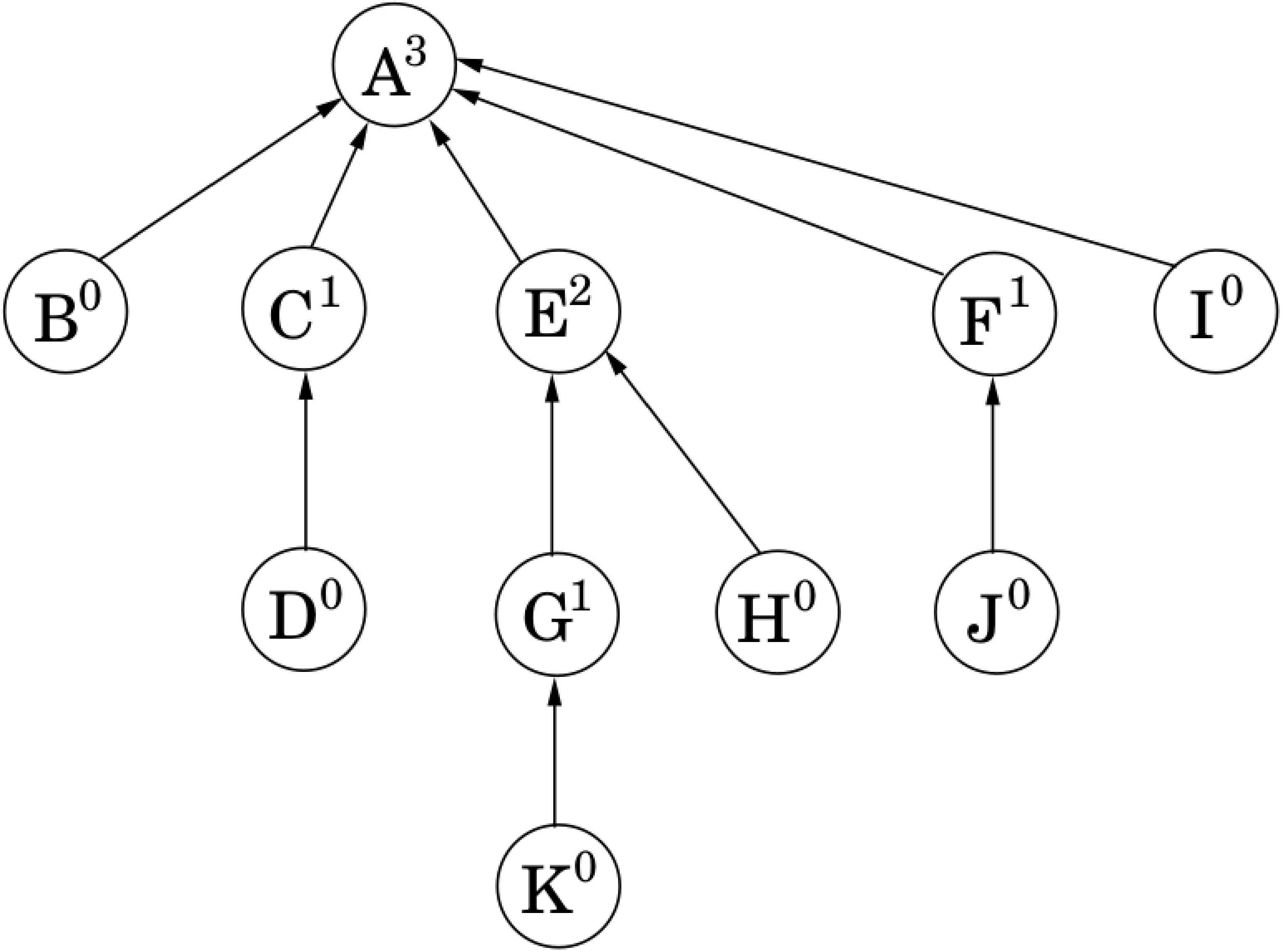
- $O(|E| \log |V|)$ for sorting edges
- $O(|E| \log |V|)$ for union and find

Can we do better?

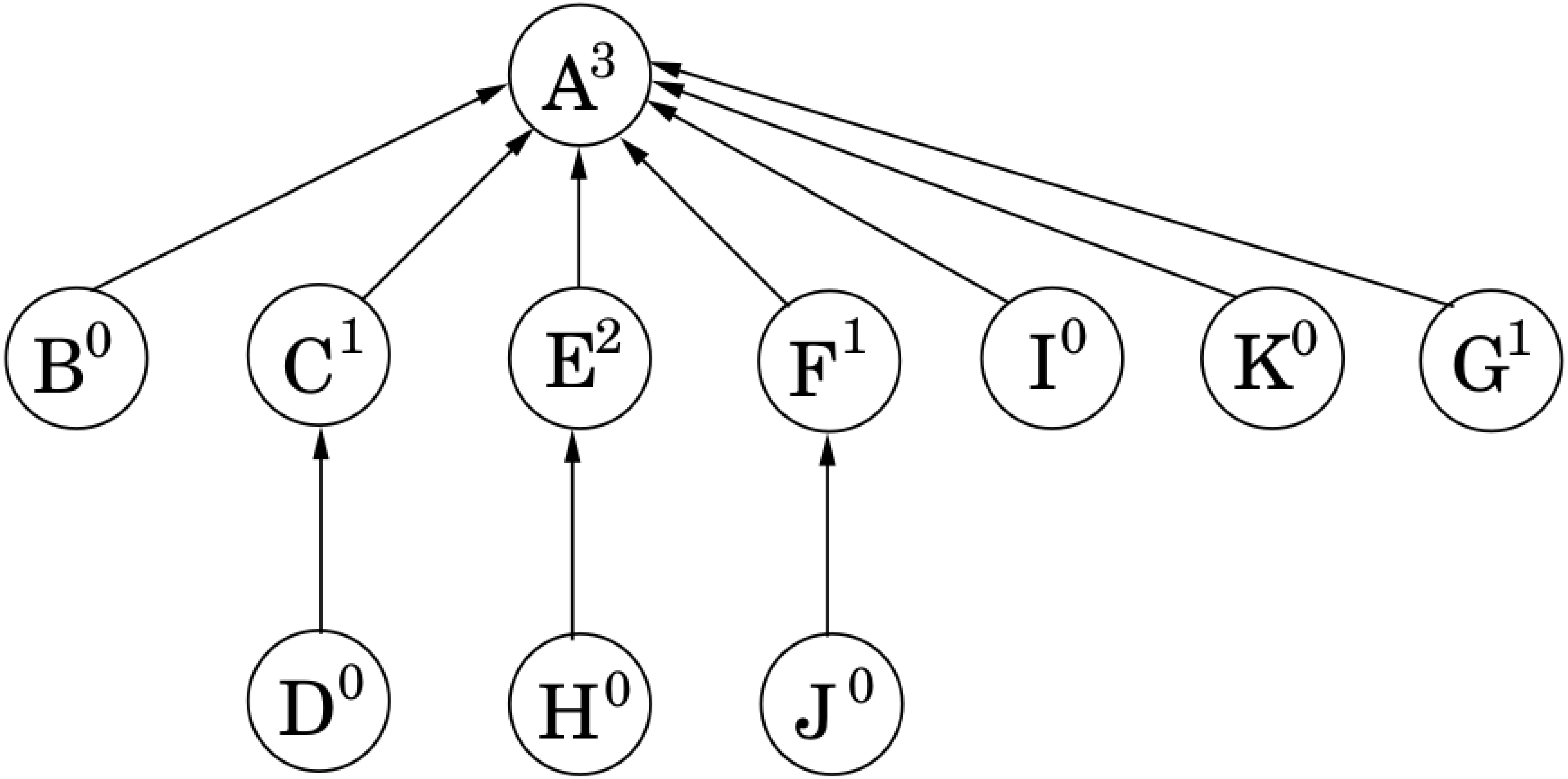
```
function find(x)
if x ≠ π(x):  π(x) = find(π(x))
return π(x)
```


A3
B0
C1
E2
D0
F1
G1
H0
I0
J0
K0

After find(I)



After find(K)



- find operations look at the inside of the tree
- union operations look at the root
- path compression doesn't affect union or change ranks
- rank properties still hold

Ranks can vary between 0 and $\lg n$

- $\{1\}$
- $\{2\}$
- $\{3, 4\}$
- $\{5, 6, \ldots, 16\}$
- $\{17, 18, 2^{16} = 65536\}$
- $\{65537, 65538, \ldots, 2^{65536}\}$
- ...

That is, $\{k+1, k+2, \ldots, 2^k\}$

Let $\lg^* n$ be the number of lgs to bring it down to 1:

$$\lg^* 1000 = 4$$

$$\lg 1000 \approx 9.97$$
$$\lg 9.97 \approx 3.32$$
$$\lg 3.32 \approx 1.73$$
$$\lg 1.73 \approx 0.79$$

Some find operations may take longer than others.

Give each node some cash, so the total amount is $\leq n \lg^* n$ dollars.

Then find takes $O(\lg^* n)$ steps plus a bit (covered by the cash)

Node gets its cash when it is no longer a root (then rank is fixed)

If rank is in $\{k+1, k+2, \ldots, 2^k\}$, it gets 2^k dollars.

Number of nodes with rank $> k$ is bounded by

$$\frac{n}{2^{k+1}} + \frac{n}{2^{k+2}} + \cdots \leq \frac{n}{2^k}$$

So total amount given to nodes in this interval is at most n

Total amount given to nodes in this interval is at most n

There are $\lg^* n$ intervals

Total amount given out is $\leq n \lg^* n$

Time for a specific find operation -- number of pointers followed

Nodes x on the chain to the root are in one of two categories:

- rank of $\pi(x)$ is in a higher interval than rank of x
- rank of $\pi(x)$ is in the same interval

- rank of $\pi(x)$ is in a higher interval than rank of x
- rank of $\pi(x)$ is in the same interval

At most $\lg^* n$ nodes of the first type: $O(\lg^* n)$ work

Remaining nodes pay a dollar for their processing

We require that each node has enough to cover this.

- Each time x pays a dollar, parent's rank increases
- If x's rank is in $\{k+1, k+2, \ldots, 2^k\}$, then it pays at most 2^k dollars

Then m find opertions take at most $O(m \lg^* n)$ plus at most $O(n \lg^* n)$

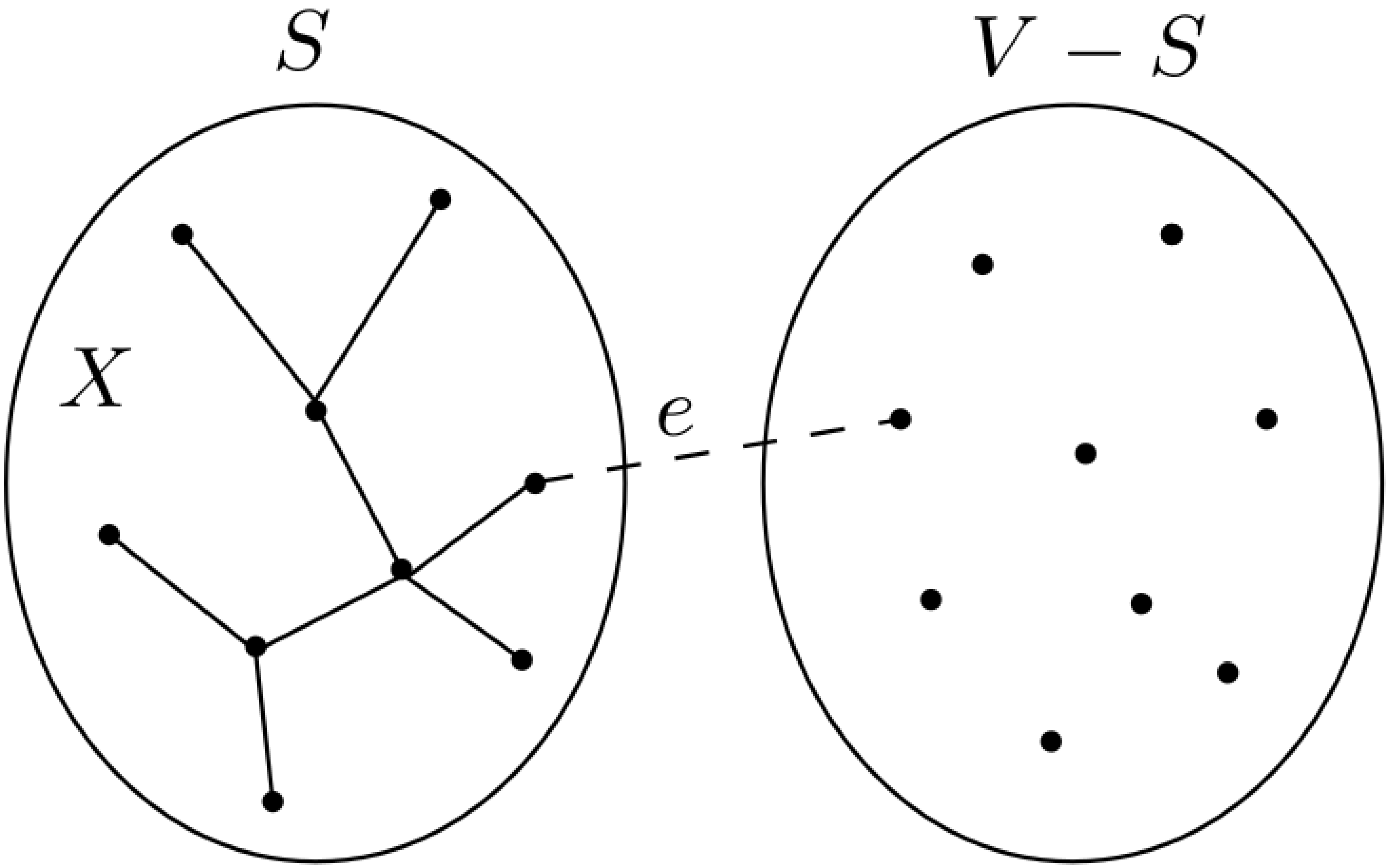
In general, this form of greedy scheme will work:

```
X = { } (edges picked so far)
repeat until |X| = |V| - 1:
  pick a set S ⊂ V for which X has no edges between S and V - S
  let e ∈ E be the minimum-weight edge between S and V - S
  X = X ∪ {e}
```

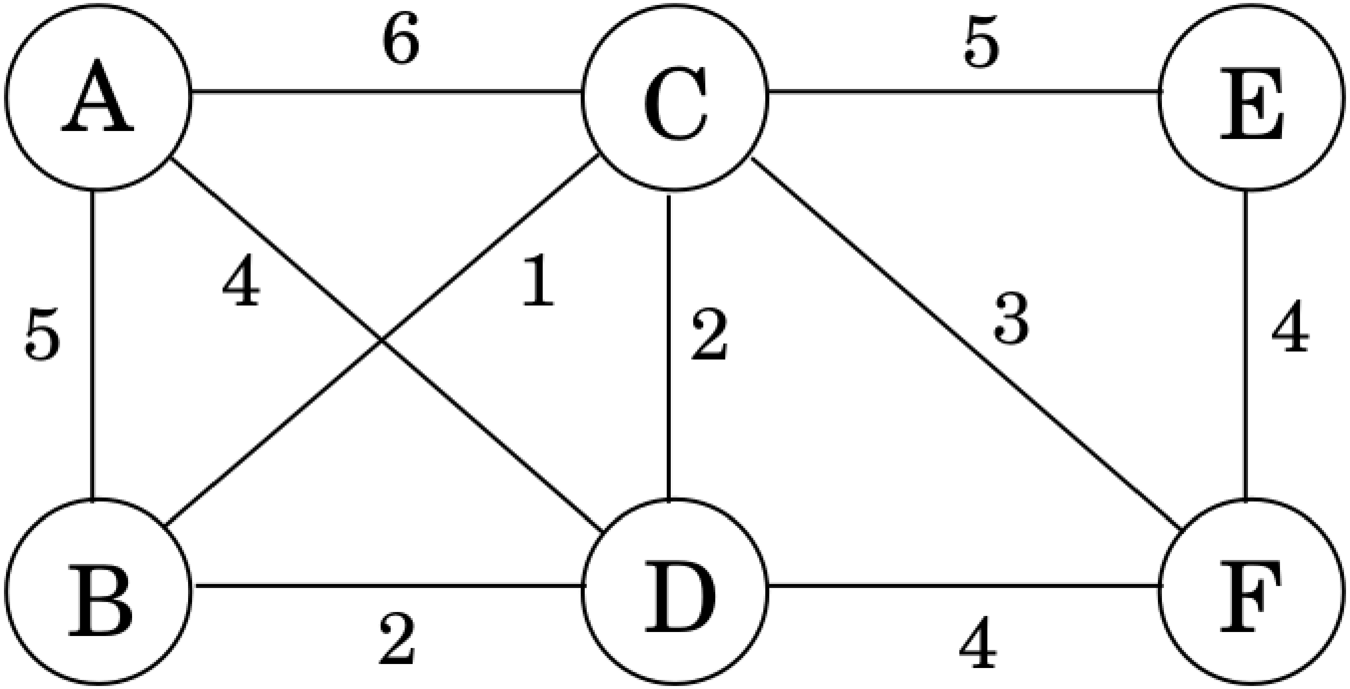
PRIM'S ALGORITHM

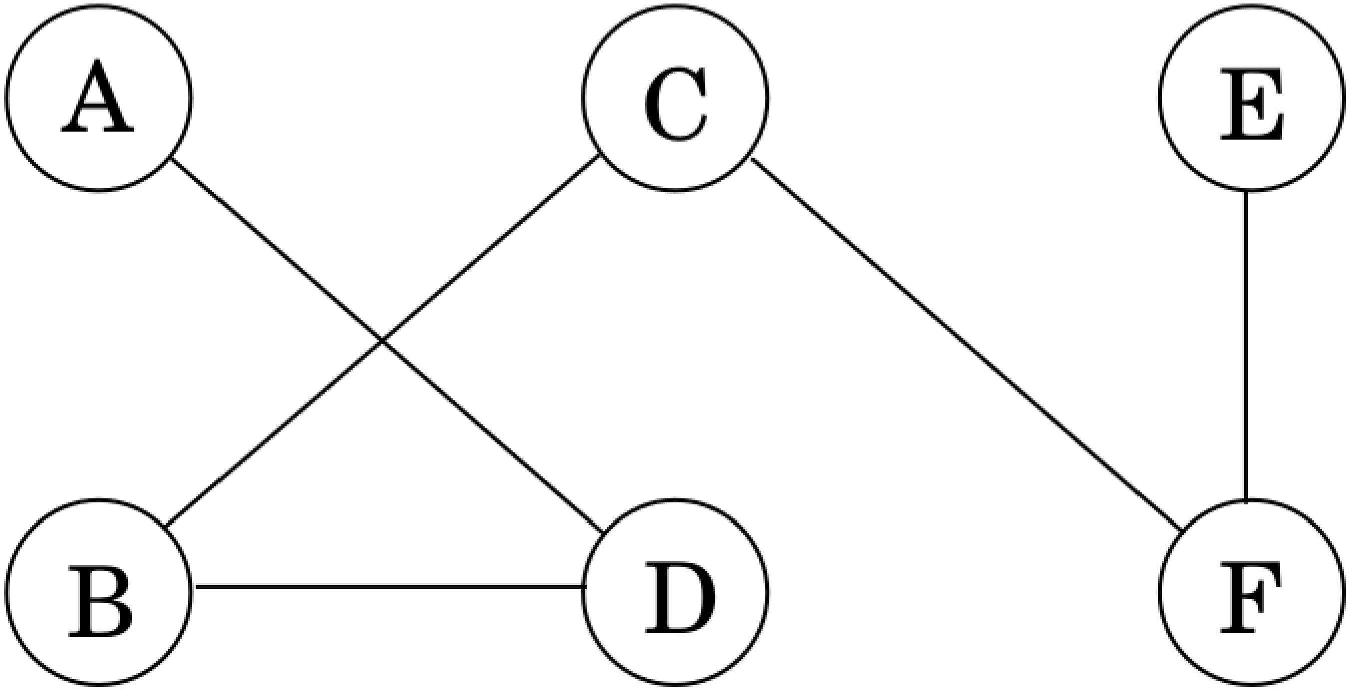
- Edge set X is always a subtree of G
- X grows by one (lightest) edge each time

(or think of S as growing to include the cheapest next vertex)


S
$V - S$
X
e

```
procedure prim(G, w)
Input:    A connected undirected graph G = (V, E) with edge weights w_e
Output:   A minimum spanning tree defined by the array prev

for all u ∈ V:
   cost(u) = ∞
   prev(u) = nil
Pick any initial node u_0
cost(u_0) = 0

H = makequeue(V)    (priority queue, using cost-values as keys)
while H is not empty:
   v = deletemin(H)
   for each {v, z} ∈ E:
      if cost(z) > w(v, z):
         cost(z) = w(v, z)
         prev(z) = v
         decreasekey(H, z)
```


A
C
E
B
D
F
6
5
5
4
1
2
3
4
2
4


A
C
E
B
D
F

Set S	A	B	C	D	E	F
$\{\}$	$0/\textbf{nil}$	$\infty/\textbf{nil}$	$\infty/\textbf{nil}$	$\infty/\textbf{nil}$	$\infty/\textbf{nil}$	$\infty/\textbf{nil}$
A		$5/A$	$6/A$	$4/A$	$\infty/\textbf{nil}$	$\infty/\textbf{nil}$
A, D		$2/D$	$2/D$		$\infty/\textbf{nil}$	$4/D$
A, D, B			$1/B$		$\infty/\textbf{nil}$	$4/D$
A, D, B, C					$5/C$	$3/C$
A, D, B, C, F					$4/F$	