CS 106 INTRODUCTION TO DATA STRUCTURES

SPRING 2020

PROF. SARA MATHIESON

HAVERFORD COLLEGE

ADMIN

- Lab 7 due tomorrow
- Video on (if your internet will accommodate that)
- Let me know if you will be missing class or lab

Plan:

- Today: last class of material before the midterm
- Tuesday: review session
- Midterm 2: take during a 2 hour block next Thurs/Fri

Remind me to RECORD

REVISED TA/OFFICE HOURS

Sunday 7-9pm (Juvia)

Monday 8-midnight (Steve)

Tuesday 11:30-12:30pm (Lizzie)

Tuesday 4:30-6pm (Sara)

Wednesday 8-midnight (Steve)

Thursday 11:30-12:30pm (Lizzie)

Thursday 9-11pm (Will)

Friday 8-10pm (Gareth)

Today/Tomorrow

Saturday 4-6pm (Will)

Saturday 8-10pm (Gareth)

PLAN FOR REMAINING ASSIGNMENTS

- **Midterm 2: take during a 2-hour block next Thursday or Friday**
  - Flexible but I'd like to keep everyone around these dates)
  - Study guide posted, can create "cheat-sheet" like before

- **Recommendation: pause all lab work after tomorrow (Friday)**
  - If an extension would be helpful for Lab 7 I would recommend May 3 or a bit later

- **Final project due May 15 (last day of exam period)**
  - Extensions beyond this day usually require coordination with a Dean, although that may be different this semester

APR 23 OUTLINE

- **Capstone: Kruskal's algorithm; combines:**
  - Graphs
  - Trees
  - Sets
  - Heaps

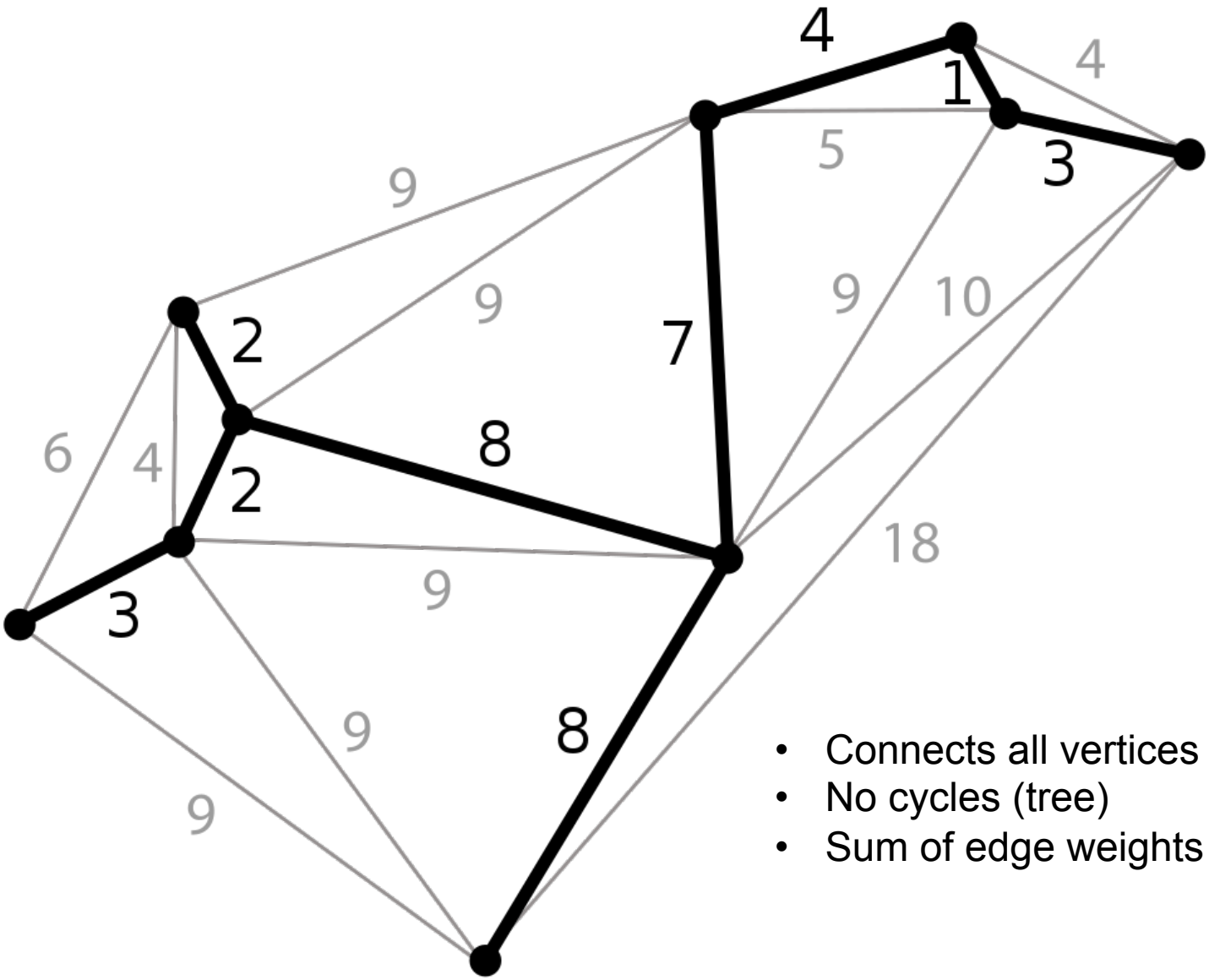
- **Highlight important concepts from this course and other takeaways**

APR 23 OUTLINE

- **Capstone: Kruskal's algorithm; combines:**
  - Graphs
  - Trees
  - Sets
  - Heaps

- **Highlight important concepts from this course and other takeaways**

MINIMUM SPANNING TREE



- Connects all vertices
- No cycles (tree)
- Sum of edge weights is minimized

https://en.wikipedia.org/wiki/Minimum_spanning_tree

MINIMUM SPANNING TREE: APPLICATION

Laying cable in a new neighborhood

- Nodes: houses
- Edges: cable
- Edge weights: difficulty/cost of laying cable
- Minimum spanning tree: best way to lay the cable so everyone is connected

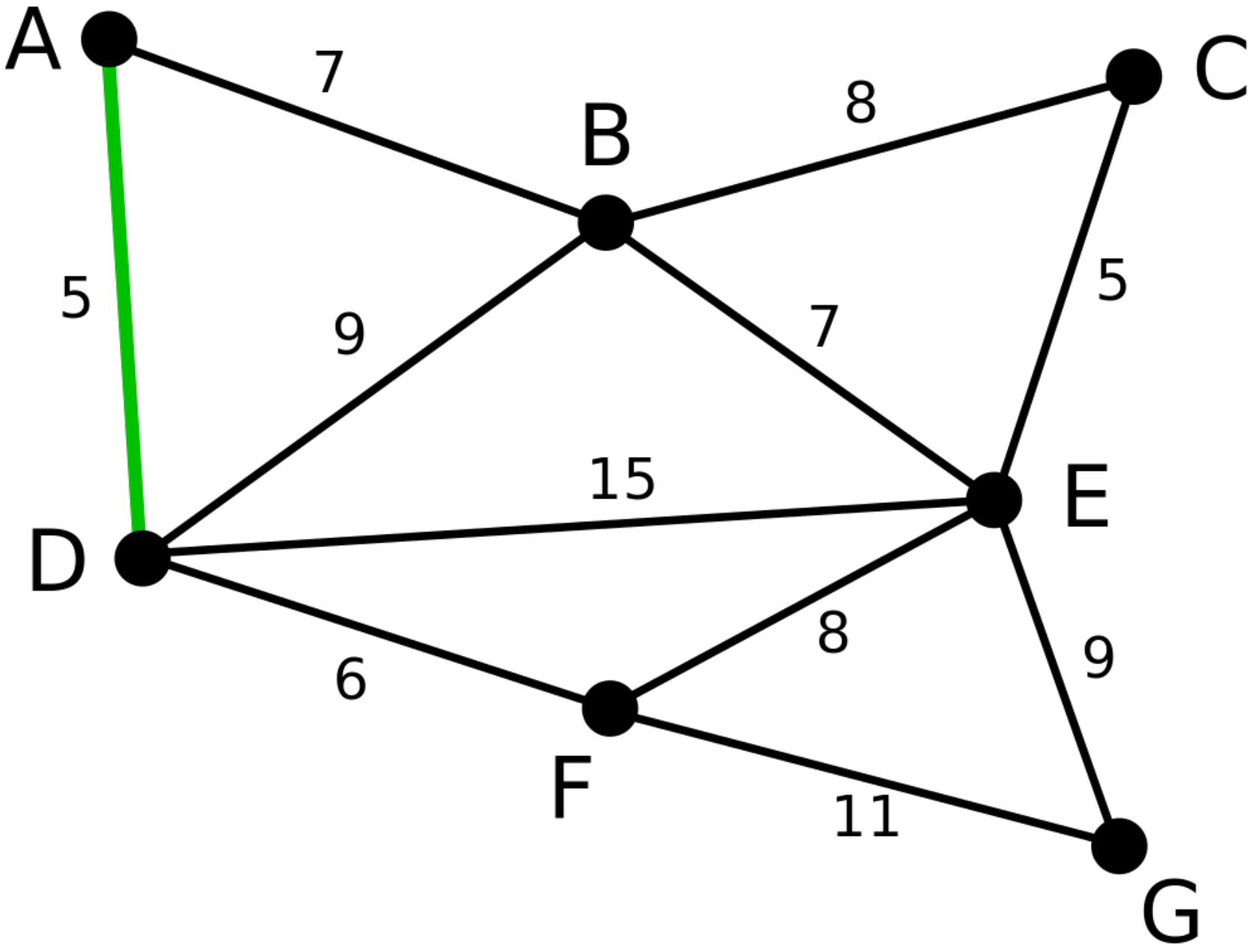
https://en.wikipedia.org/wiki/Minimum_spanning_tree

KRUSKAL'S ALGORITHM

Goal: find minimum spanning tree

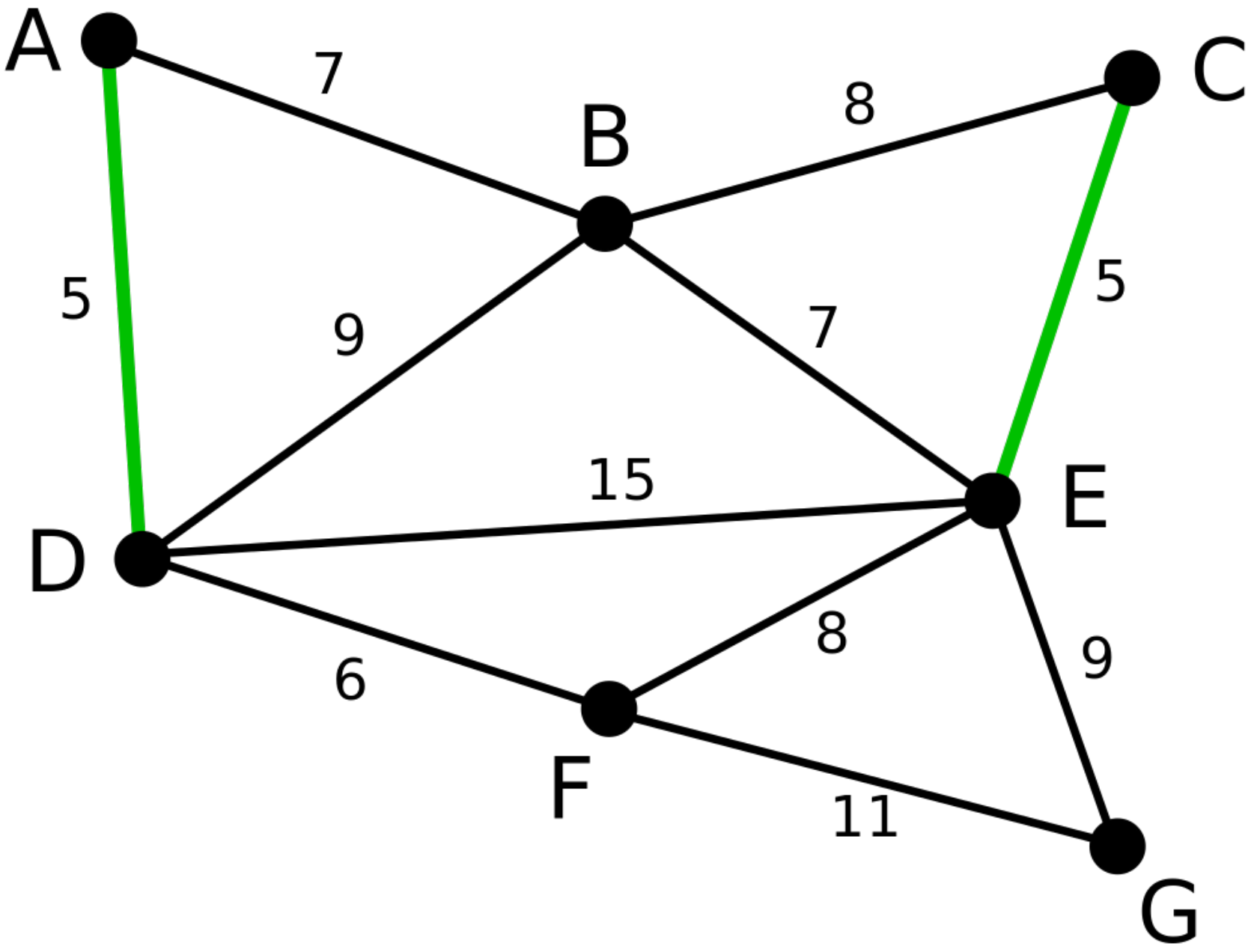
```
Put each node into its own set, creating forest F

Put each edge into a min heap H based on edge weight

While H is not empty and F not spanning:
    edge (u,v) = removeMin(H)
    if u and v belong to different sets:
        combine them into one and add to F
```

https://en.wikipedia.org/wiki/Kruskal%27s_algorithm



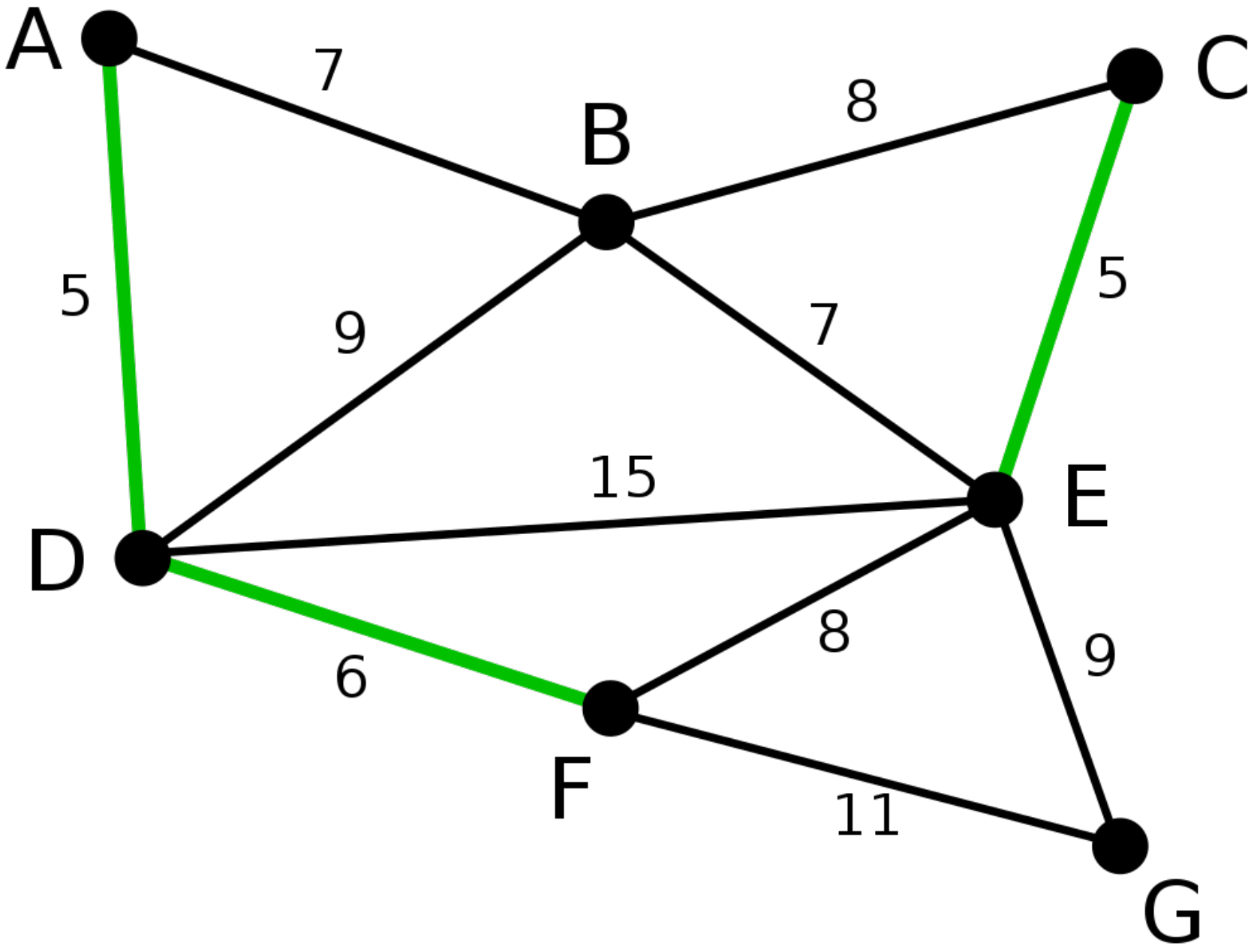
Kruskal's algorithm example

https://en.wikipedia.org/wiki/Kruskal%27s_algorithm



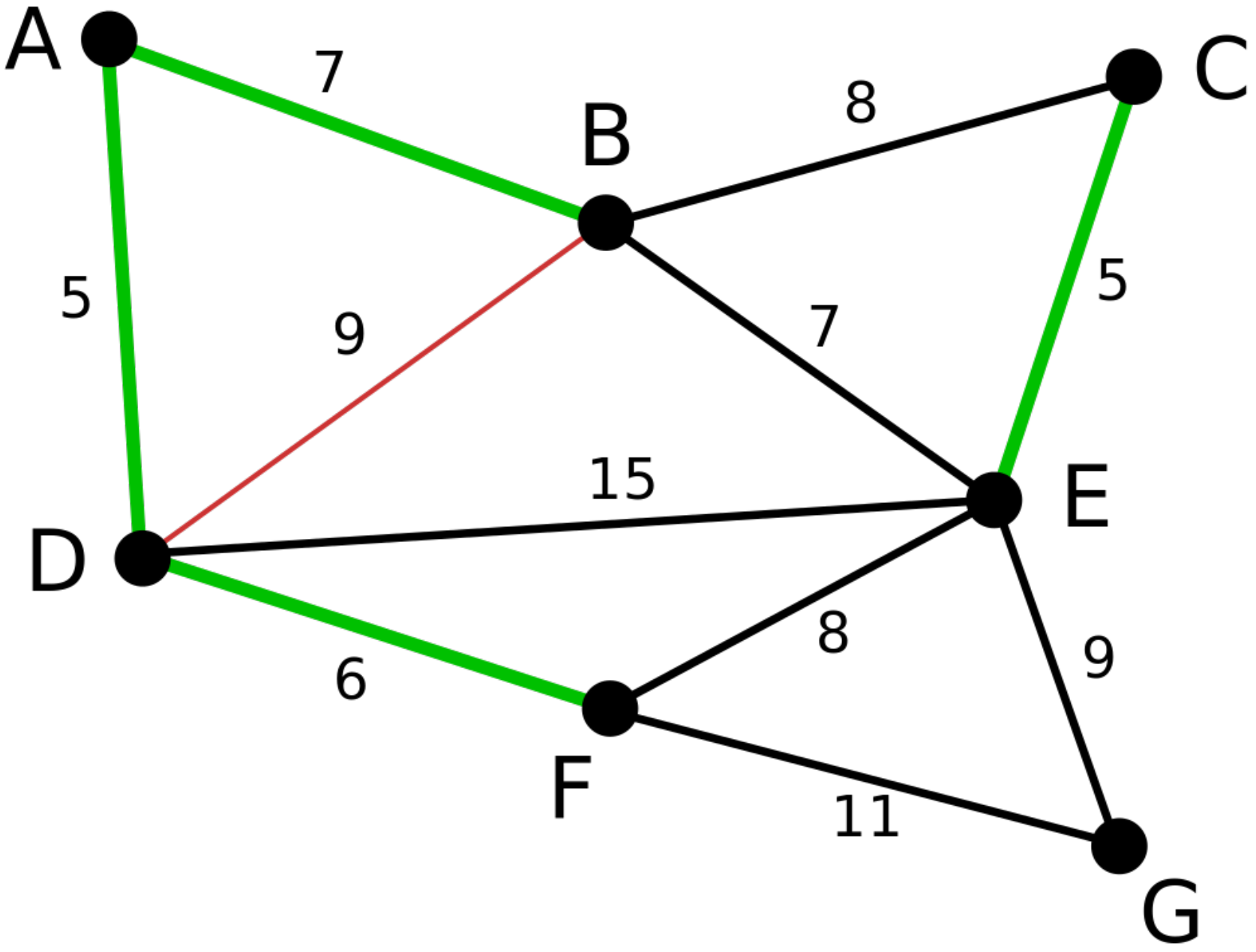
Kruskal's algorithm example

https://en.wikipedia.org/wiki/Kruskal%27s_algorithm



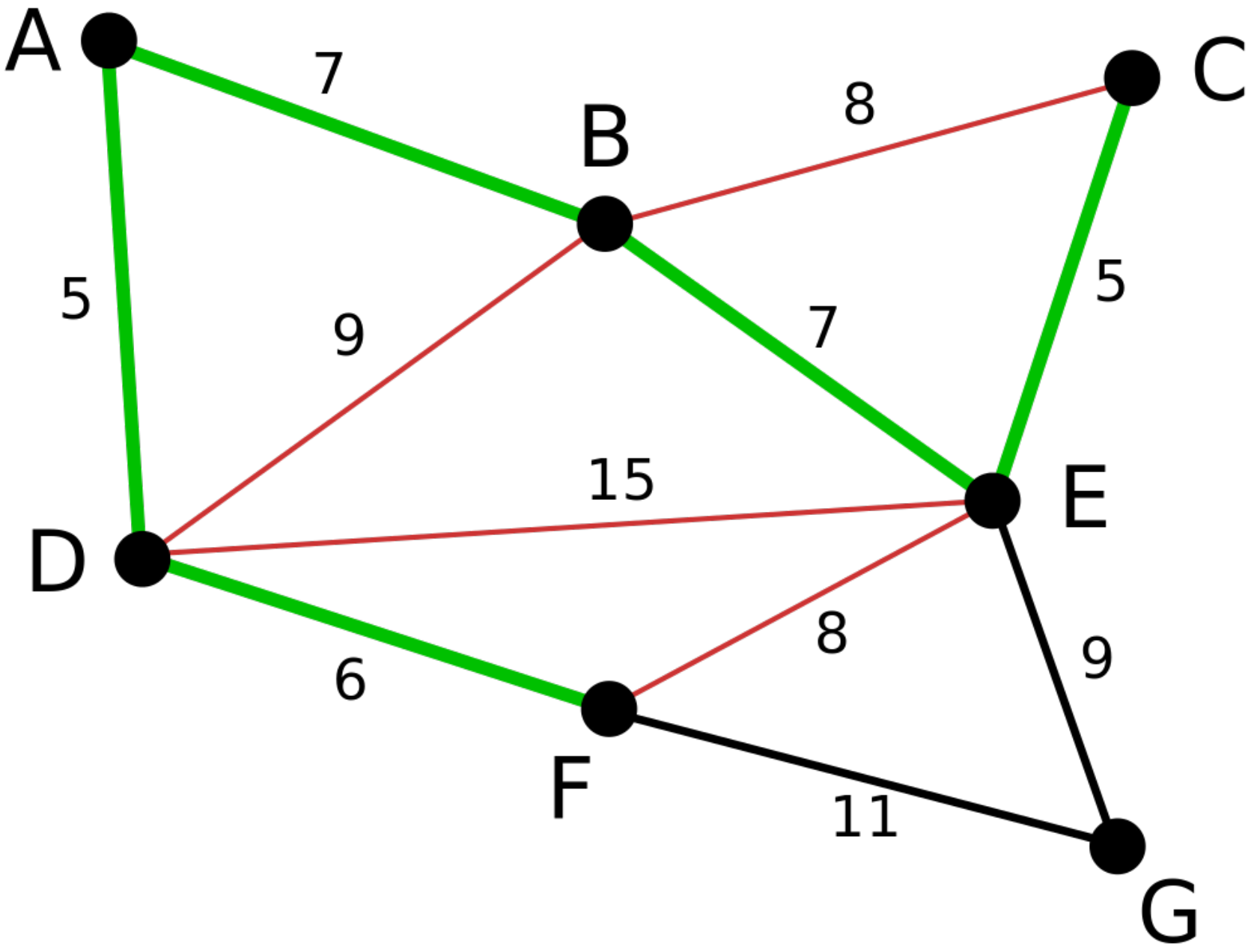
Kruskal's algorithm example

https://en.wikipedia.org/wiki/Kruskal%27s_algorithm



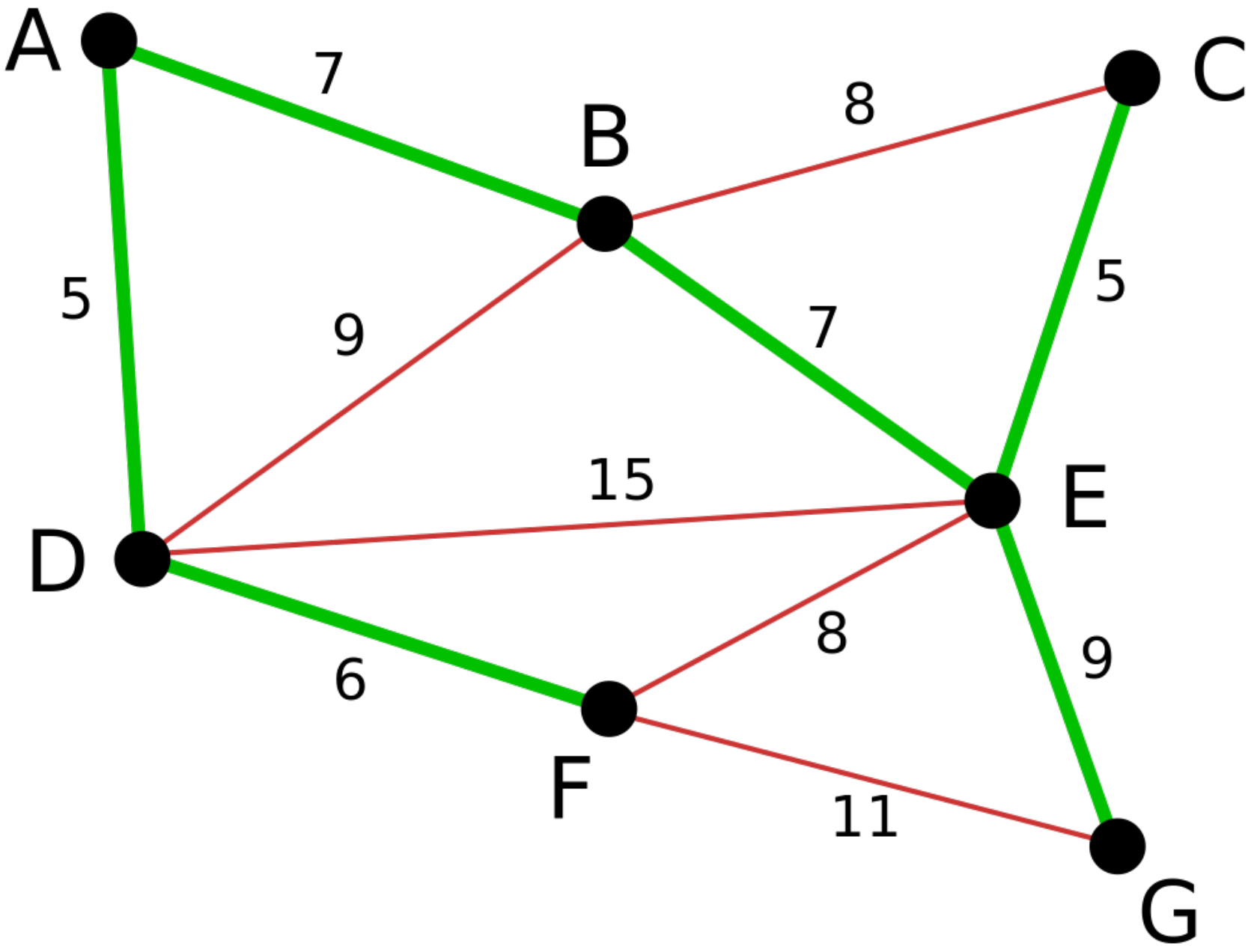
Kruskal's algorithm example

https://en.wikipedia.org/wiki/Kruskal%27s_algorithm



Kruskal's algorithm example

https://en.wikipedia.org/wiki/Kruskal%27s_algorithm



Kruskal's algorithm example

https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

SET ADT

add(e): Adds the element e to S (if not already present).

remove(e): Removes the element e from S (if it is present).

contains(e): Returns whether e is an element of S.

iterator( ): Returns an iterator of the elements of S.

There is also support for the traditional mathematical set operations of ***union***, ***intersection***, and ***subtraction*** of two sets S and T:

$$\begin{aligned} S \cup T &= \{e:\ e \text{ is in } S \text{ or } e \text{ is in } T\}, \\ S \cap T &= \{e:\ e \text{ is in } S \text{ and } e \text{ is in } T\}, \\ S - T &= \{e:\ e \text{ is in } S \text{ and } e \text{ is not in } T\}. \end{aligned}$$

addAll(T): Updates S to also include all elements of set T, effectively replacing S by $S \cup T$.

retainAll(T): Updates S so that it only keeps those elements that are also elements of set T, effectively replacing S by $S \cap T$.

removeAll(T): Updates S by removing any of its elements that also occur in set T, effectively replacing S by $S - T$.

IMPLEMENTATION

Recall that maps do not allow duplicate keys

A set is simply a map in which keys have no associated values (or null)

Examples in Java:

- `java.util.HashSet`
- `java.util.TreeSet`

UNION-FIND (DISJOINT SET FORESTS)

A data structure that keeps track of a set of elements partitioned into disjoint subsets.

- `create(u)`: create a set containing a single item u
- `find(u)`: find the set that contains u
- `union(u,v)`: merge the set containing u and the one containing v

Time complexity depends heavily on implementation

HOW COULD WE IMPLEMENT UNION-FIND USING TREES?

Basic idea: the root node holds the name of a set

Find: keep traversing up the tree via parent pointers until the root is reached

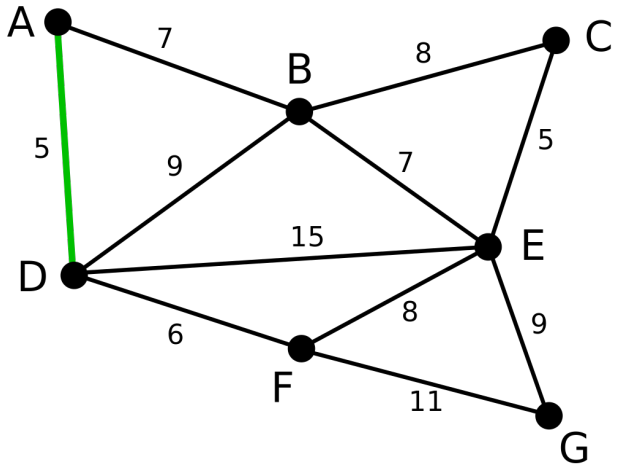
Union: make one root the child of another root

Which one? Make the smaller set the child of the larger so as to minimize the final "height."

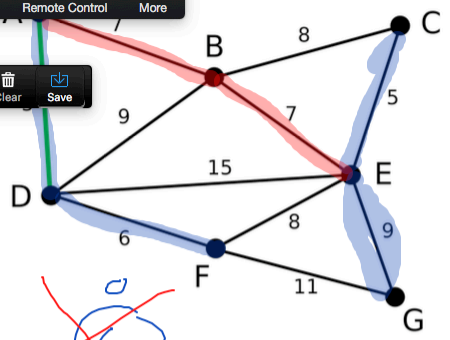
Actually keep track of *rank* which is updated only during union.

Note: these are not binary trees!

EXAMPLE

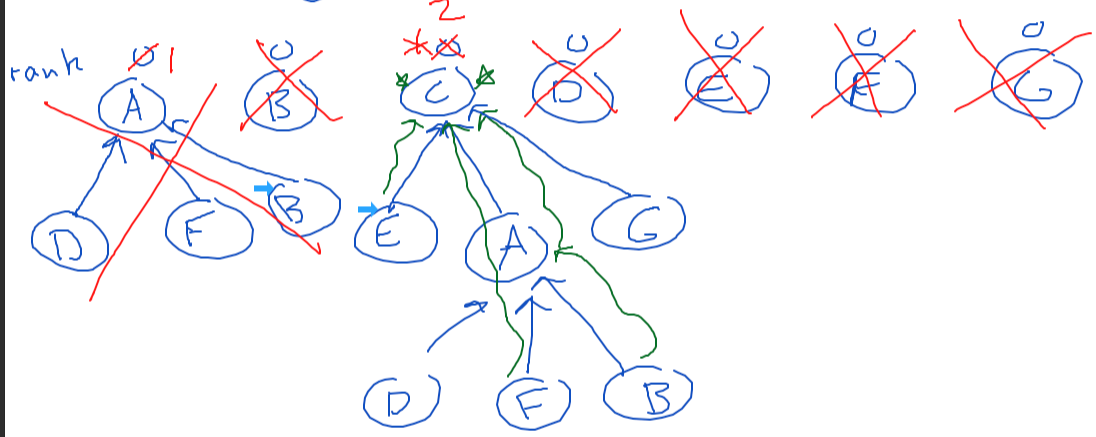


EXAMPLE



n = #nodes (7)

m = #edges (11)



(A, D)

(C, E)

(D, F)

(A, B)

(B, E)

(B, C)

(E, F)

(E, G)

PSEUDOCODE

```
create(u):
    // check if already present
    add u as root of new tree
    u.parent = u
    u.rank = 0


find(u):
    if u.parent != u:
        u.parent = find(u.parent)
    return u.parent
```

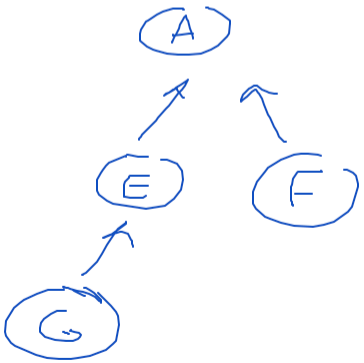
https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

PSEUDOCODE

```
create(u):
    // check if already present
    add u as root of new tree
    u.parent = u
    u.rank = 0

find(u):
    if u.parent != u:
        u.parent = find(u.parent)
    return u.parent
```

root

return A



https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

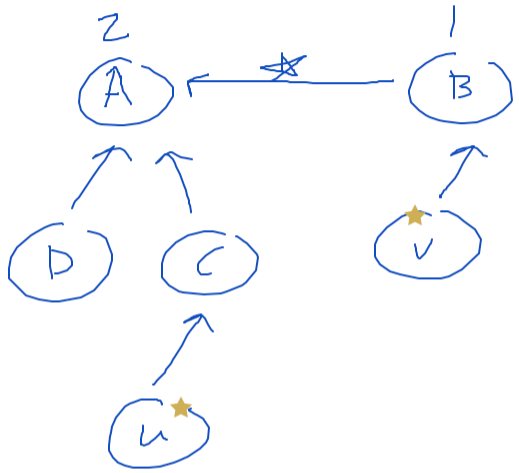
PSEUDOCODE

```
union(u,v):
    uRoot = find(u)
    vRoot = find(v)

    if uRoot == vRoot:

    min = min(uRoot,vRoot)
    max = max(uRoot,vRoot)

    min.parent = max
    if min.rank == max.rank:
        max.rank += 1
```

https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

PSEUDOCODE

```
union(u,v):
    uRoot = find(u)
    vRoot = find(v)

    // cycle
    if uRoot == vRoot:
        return

    min = min(uRoot,vRoot)
    max = max(uRoot,vRoot)

    min.parent = max
    if min.rank == max.rank:
        max.rank += 1
```

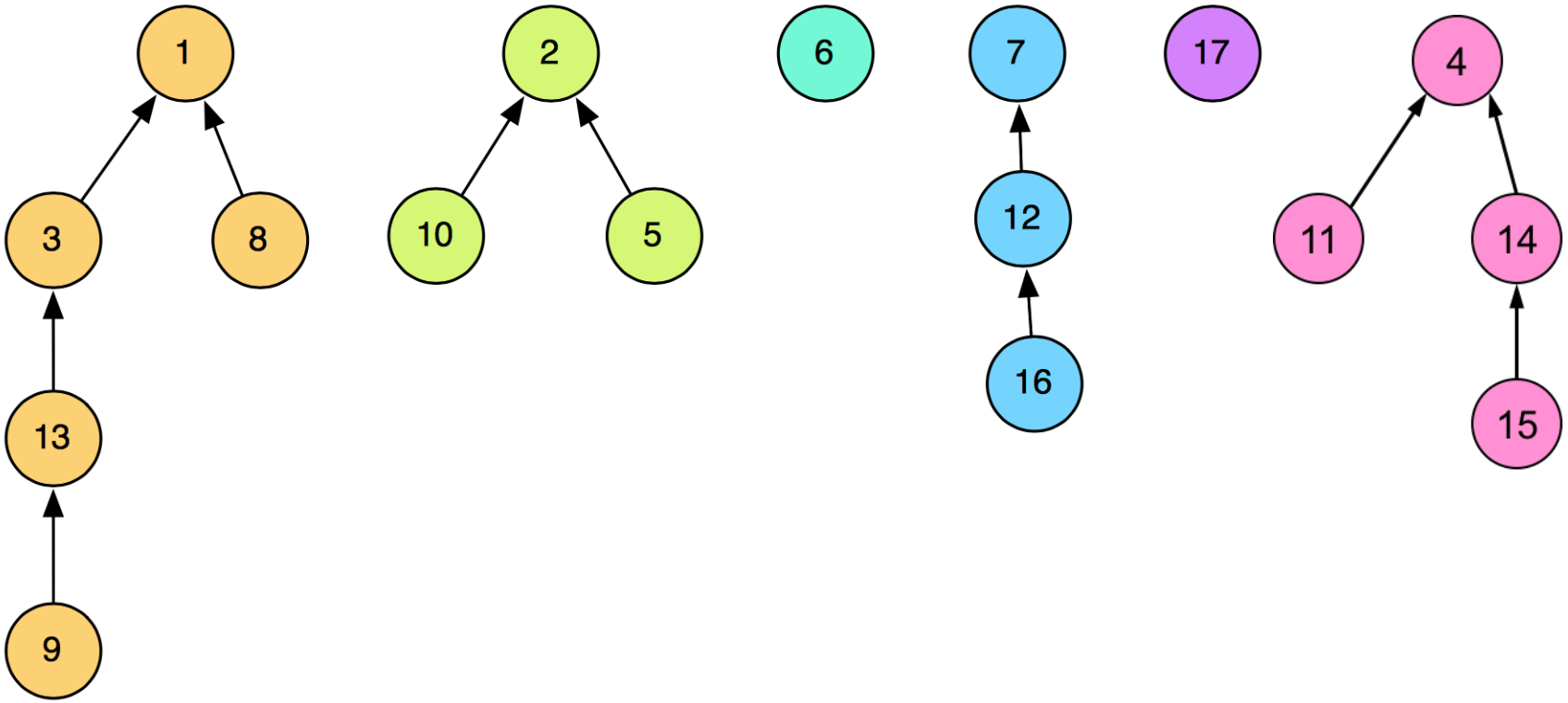


by rank

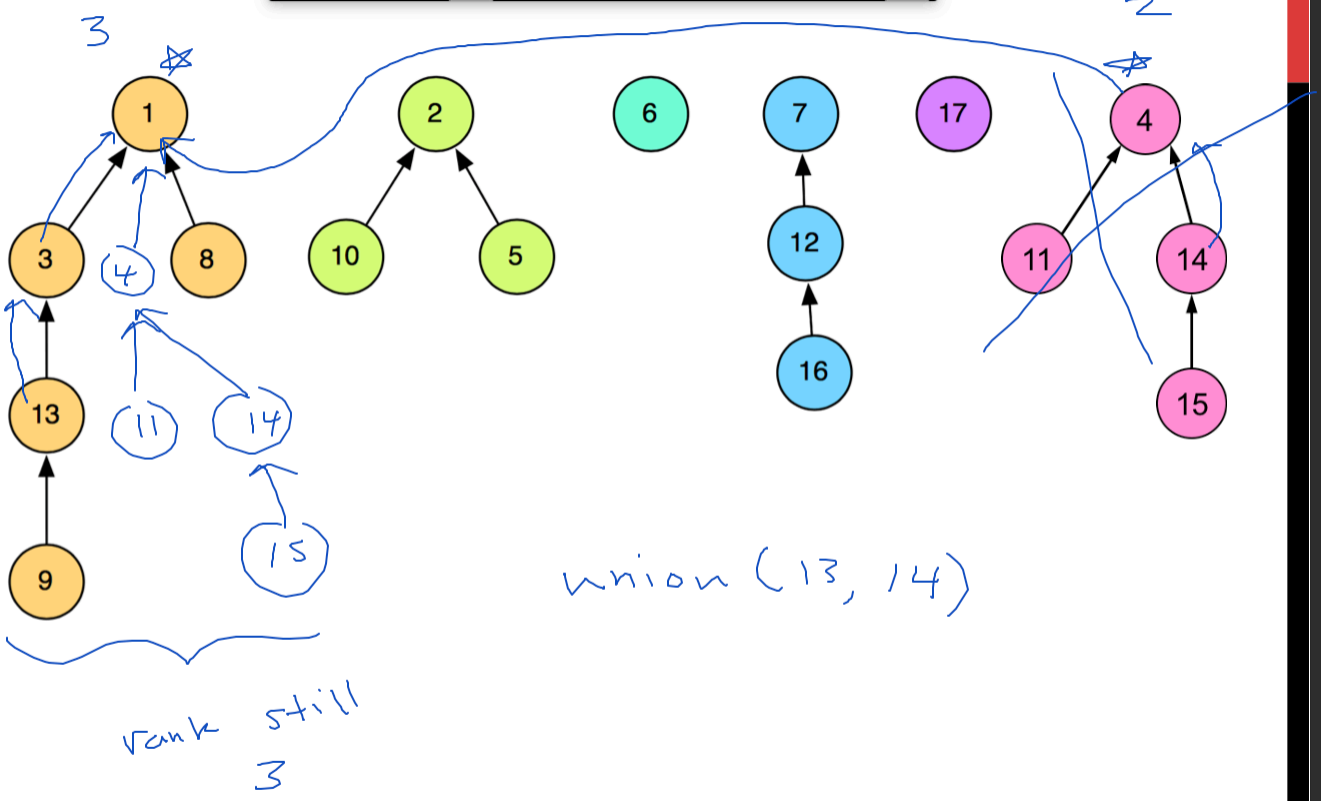
combine → O(1)

https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

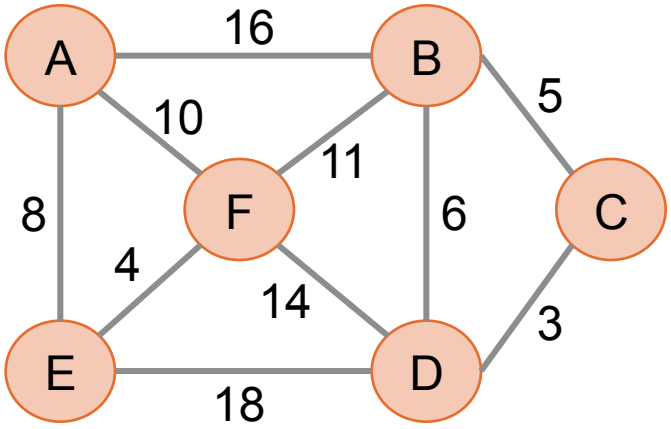
EXAMPLE



EXAMPLE

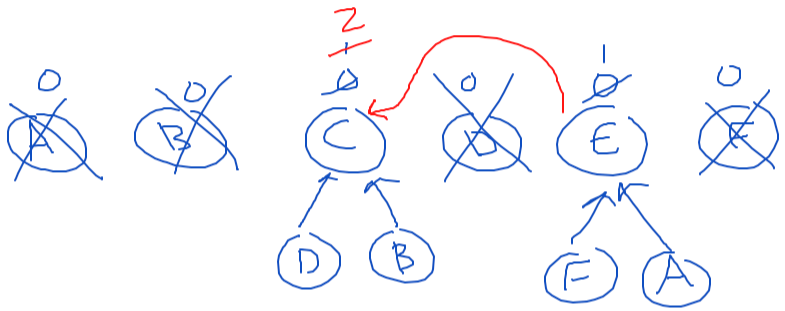

3
1
2
6
7
17
4
2
3
4
8
10
5
12
11
14
13
11
14
16
15
9
15
union (13, 14)
rank still
3

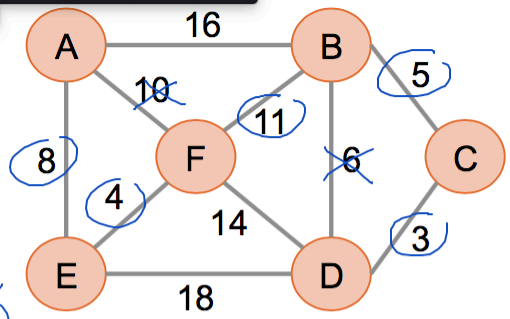
GROUP EXERCISE



Graph: modified from Chegg

GROUP EXERCISE





(C, D)

(E, F)

(B, C)

(B, D)

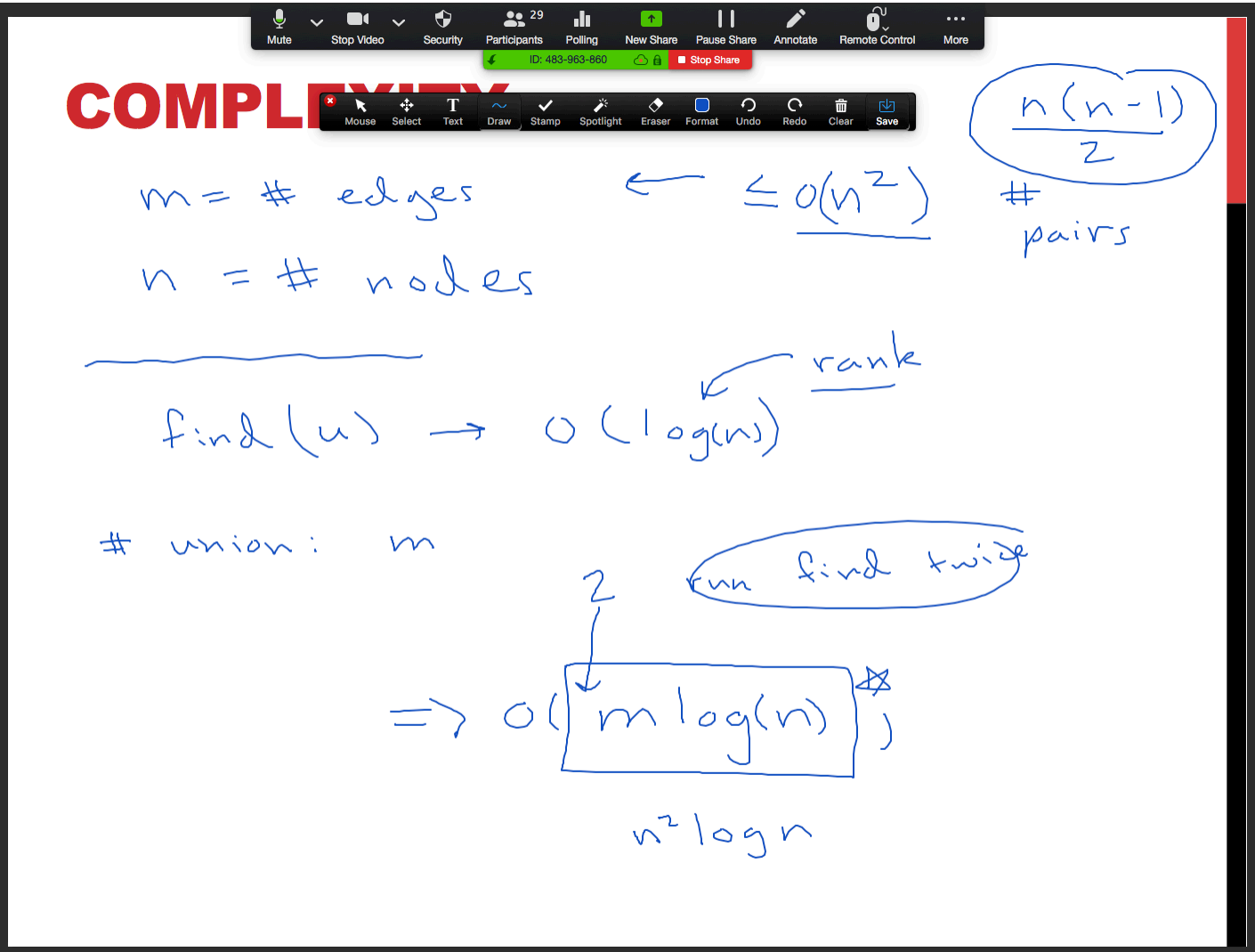
(A, E)

(A, F)

(B, F)

Graph: modified from Chegg

COMPLEXITY


Mute
Stop Video
Security
Participants
29
Polling
New Share
Pause Share
Annotate
Remote Control
More
ID: 483-963-860
Stop Share
Mouse
Select
Text
Draw
Stamp
Spotlight
Eraser
Format
Undo
Redo
Clear
Save
COMPLEXITY
m = # edges
n = # nodes
≤ O(n²)
n(n-1)/2
# pairs
find(u) → O(log(n))
rank
union: m
run find twice
2
⇒ O(m log(n))
n² log n

APR 23 OUTLINE

- Capstone: Kruskal's algorithm; combines:
  - Graphs
  - Trees
  - Sets
  - Heaps

- **Highlight important concepts from this course and other takeaways**

Next time!