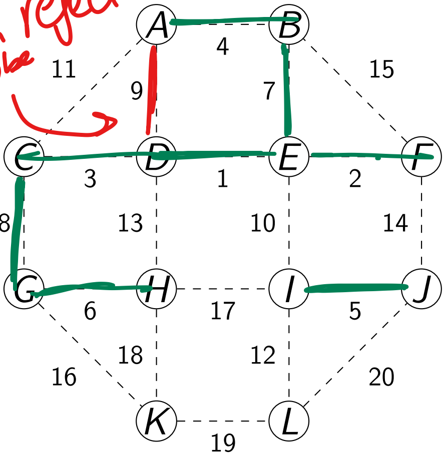
CompSci 161
Winter 2023 Lecture 21:
Greedy Algorithms:
Kruskal's Algorithm, Union-Find Data Structure

Finding a MST (Kruskal)

1st rejected
to be



Disjoint Sets

Goal: Quickly determine a, b connected?
Answer: Disjoint-set data structure

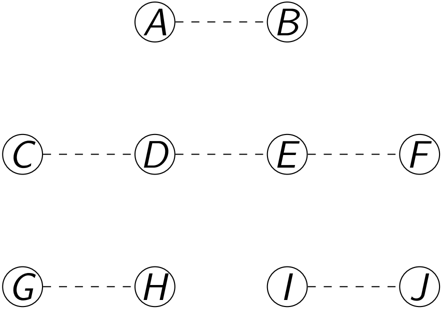
- Maintain a collection of disjoint dynamic sets
- Identify each set by a representative
- Support the following operations:
  - `Make-Set` : constructor
  - Given a vertex, in which tree?
    - x, y same tree? Same answer.
  - Two vertices, merge
    - Different trees now connected

interface

"find"

find(x) = find(y) iff x, y same tree

An Application: Counting Disjoint Sets



Edges	Collection
initial	$\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g\}, \{h\}, \{i\}, \{j\}$
(d, e)	$\{a\}, \{b\}, \{c\}, \{d, e\}, \{f\}, \{g\}, \{h\}, \{i\}, \{j\}$
(e, f)	$\{a\}, \{b\}, \{c\}, \{d, e, f\}, \{g\}, \{h\}, \{i\}, \{j\}$

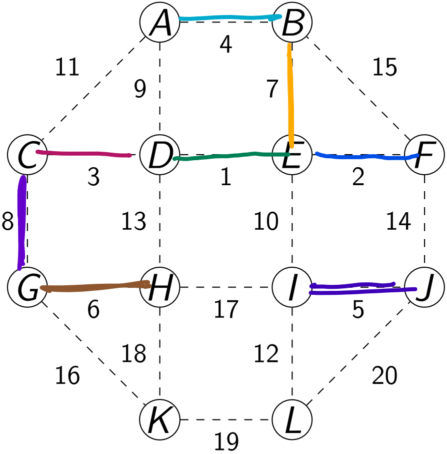
Linked List Representation

- Objects represented by linked list
  - Each set has head, tail
  - Each element has member, next, pointer back
- Implementation of Union(x, y)
  - Append y's list onto end of x's
  - Convenient use of tail pointer

- How long does a series of operations take?

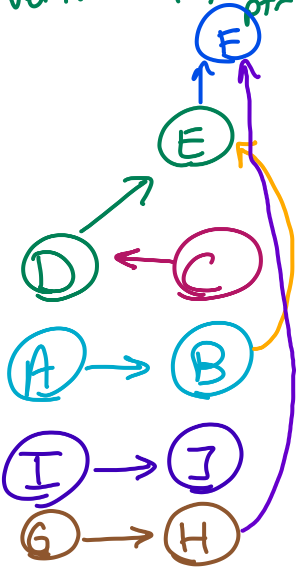
Disjoint-set Forests

- Represent sets by rooted trees
- Each tree is one set
- Each node has one member



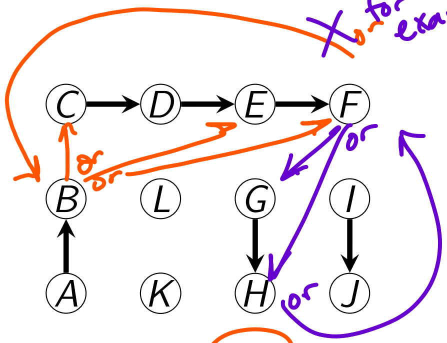
Support data structure

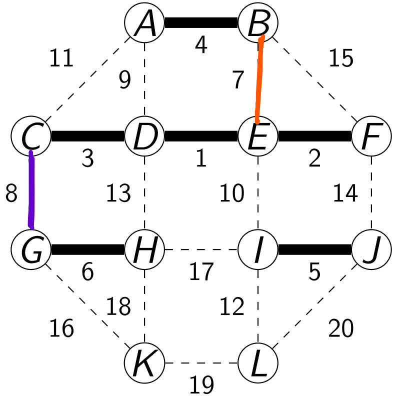
vertices w/ parent ptr



Continuing the structure

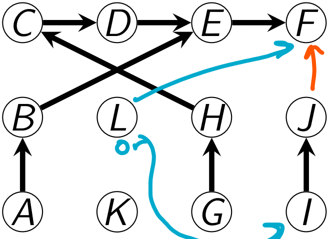
for example, no



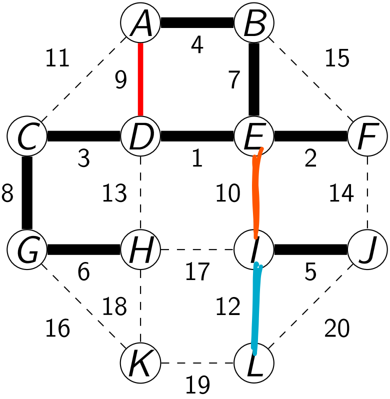


- Edge *BE*?
- Edge *CG*?
- Edge *AD*?

Continuing the structure



- Edge EI?
- ~~Edge AC?~~ rejected
- Edge IL?
- ~~Edge DH?~~ rejected



Running time for Operations

if X.count > Y.count
swap X, Y before adj: parent/count

Union(A,B)

$X \leftarrow$ find(A)
$Y \leftarrow$ find(B)
if $X \neq Y$ **then**
* X.count += Y.count
X.parent $\leftarrow Y$

find(A)

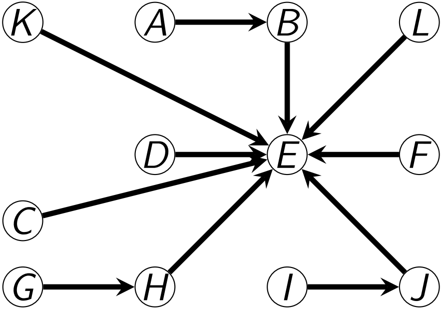
if A.parent $\neq$ nullptr
then
return find(A.parent)
return A

2xfind + O(1)

- If there are n elements, times? O(n) for find
- Can we improve the **worst-case** for one?

Union by rank.

Example of Union by Rank



- This is the result of *Union by Rank*
- Ties are broken alphabetically
  Earlier letter $\rightarrow$ later letter (when tied)
- Running time for operations? $O(\log n)$ find

Running time for Operations

Union(A,B)
  $X \leftarrow$ find(A)
  $Y \leftarrow$ find(B)
  if $X \neq Y$ **then**
    if X.count $>$ Y.count **then**
      Swap X and Y
    X.parent $\leftarrow Y$
    Y.count $+=$ X.count

find(A)
  if A.parent $\neq$ nullptr **then**
    return find(A.parent)
  return A

- If there are n elements, find is $\mathcal{O}(\log n)$
- Union takes time $2\times$ find $+\ \mathcal{O}(1)$

Improving Find

??

- Can we improve **find** further?

find(const Key &k) const

find(A)

if A.parent $\neq$ nullptr **then**

A.parent $\leftarrow$ find(A.parent)

path compression

return A.parent

return A

Mutable

Private member data is

Path Compression

- Path compression change worst-case of find?

- Does path compression improve over time?

- Suppose we have m union and f find operations