UNION-FIND!

WHAT WE'VE COVERED SO FAR

Prior to Spring Break we covered: Linear and Binary Search. Bubble, Selection, Insertion, Merge and Quick Sort. Since Spring Break we have covered Breadth-First Search, Depth-First Search, Hashing, Dijkstras, and Dynamic Programming!

WHAT WE ARE COVERING TODAY

Today we are covering Union-Find!

WHAT IS UNION-FIND?

The Union-Find (or Disjoint-Set) algorithm is a data structure used to efficiently manage a collection of disjoint sets and track their connectivity.

WHAT IS UNION-FIND?

It is a data structure that keeps track of elements split into groups or sets. It supports two main operations: Union and Find.

It’s used as Cycle detection in graphs

WHAT?

Think of it like social networking app.

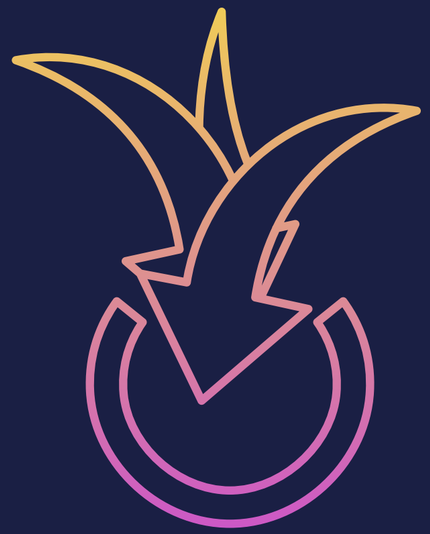
When a user goes to a person’s profile it will try to “Find” if they are friends. If they aren’t, it will ask if they want to connect with them (Union).

UNION-FIND

Union-Find has two functions:

Union:

Combines two sets



Find:

Takes an element and finds what subset it is in

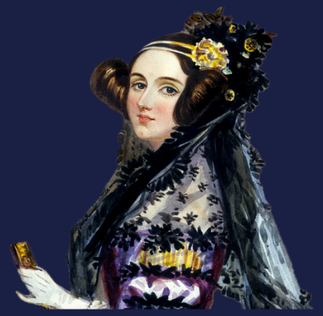
Lets consider 5 people:

Ada Lovelace
Grace Hopper
Katherine Johnson
Joan Clarke
Hedy Lamarr

Each person starts in it’s own group.

We track this using a parent array where:
parent[person] = person → each element is its own parent

Lets consider 5 people:



Ada Lovelace

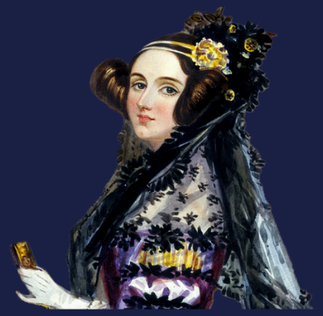
Grace Hopper

Katherine Johnson

Joan Clarke

Hedy Lamarr

And we'll make them their own parent.



Ada Lovelace

Grace Hopper

Katherine Johnson

Joan Clarke

Hedy Lamarr

And we'll make them their own parent.

A.L.

Ada Lovelace

Grace Hopper

Katherine Johnson

Joan Clarke

Hedy Lamarr

And we'll make them their own parent.

A.L.

Ada Lovelace

G.H.

Grace Hopper

Katherine Johnson

Joan Clarke

Hedy Lamarr

And we'll make them their own parent.

A.L.

Ada Lovelace

G.H.

Grace Hopper

K.J.

Katherine Johnson

J.C.

Joan Clarke

H.L.

Hedy Lamarr

Lets start with Union.

A.L.

Ada Lovelace

G.H.

Grace Hopper

K.J.

Katherine Johnson

J.C.

Joan Clarke

H.L.

Hedy Lamarr

Lets join Ada and Grace:

Union(Ada, Grace)

Union(Ada, Grace)

Ada Lovelace

Grace Hopper

Katherine Johnson

Joan Clarke

Hedy Lamarr

Now Grace points to Ada, and Ada is the parent of both Ada and Grace

parent[Grace] = Ada

Next: Union(Katherine, Joan)

A.L.

Ada Lovelace

A.L.

Grace Hopper

K.J.

Katherine Johnson

J.C.

Joan Clarke

H.L.

Hedy Lamarr

Joan points to Katherine, and Katherine is the parent of both Katherine and Joan

Next: Union(Katherine, Joan)

Ada Lovelace

Grace Hopper

Katherine Johnson

Joan Clarke

Hedy Lamarr

Joan points to Katherine, and Katherine is the parent of both Katherine and Joan

parent[Joan] = Katherine

The other part of Union-Find is Find

A.L.

Ada Lovelace

A.L.

Grace Hopper

K.J.

Katherine Johnson

K.J.

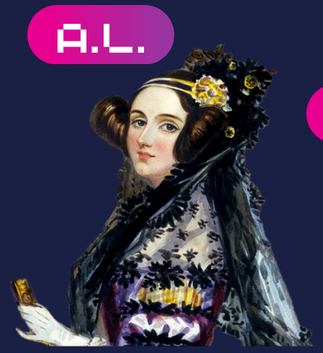
Joan Clarke

H.L.

Hedy Lamarr

Find takes the array of parents, and the person, and returns the name of the parent.

So if we are looking for Hedy Lamarr...



Ada Lovelace

Grace Hopper

Katherine Johnson

Joan Clarke

Hedy Lamarr

Our call to the find function would look something like this:

```
Find(parents, Hedy)
```

Find(parents, Hedy)

A.L.

Ada Lovelace

A.L.

Grace Hopper

K.J.

Katherine Johnson

K.J.

Joan Clarke

H.L.

Hedy Lamarr

the parents array would look like: [Ada, Katherine, Hedy]

And the Find call would return Hedy

If we changed it to:
Find(parents, Joan)

A.L.

Ada Lovelace

A.L.

Grace Hopper

K.J.

Katherine Johnson

K.J.

Joan Clarke

H.L.

Hedy Lamarr

the parents array would look like: [Ada, Katherine, Hedy]

And the Find call would return Katherine

THE PSEUDOCODE

```
function FIND(parent, x):
    if parent != x then:
        parent[x] = FIND(parent, parent[x])
    RETURN parent[x]

function UNION(parent, rank, x, y):
    rootX = FIND(parent, x)
    rootY = FIND(parent, y)

    if rootX != rootY then:
        // Add smaller tree to larger tree
        if rank[rootX] > rank[rootY] then:
            parent[rootY] = rootX
        else if rank[rootX] < rank[rootY] then:
            parent[rootX] = rootY
        else:
            parent[rootY] = rootX
            rank[rootX] = rank[rootX] + 1
```

EXAMPLES REPLIT AND GITHUB! PLEASE GO TO:

HTTPS://REPLIT.COM/@RIKKIEHRHART/GRABABYTE

HTTPS://GITHUB.COM/RIKKITOMIKOEHRHART/GRABABYTE

UP NEXT

May 7 - Kruskal's Algorithm
May 14 - Prim's Algorithm

Questions? - rikki.ehrhart@g.ausitncc.edu

If you'd like the opportunity to run a Grab a Byte algorithm workshop, please let me know!