Date 2025.03.19

Disjoint Set or Union Find

- Group n items into 1 to n sets.
  Initial: n items in n sets
  Overtime: Union sets together to get fewer sets] Current num sets 1-n
  'stopstate': 1 set

Tasks (ask of an 'item')

- 1. Which set do I you belong to?
- 2. Union 2 sets together

Union/Find

Make-Set(x) - make a new set with x as it's only member *(Subsumed into constructor to 'make set' all sets at once)

Data Structure

public (bounds check)
- Union(x,y) - union sets containing x and y into one new set *
- Find(x,y) - (aka FindSet(x)) return unique representative of set that contains x

bool Same-Component(x,y) returns Find(x)==Find(y) //possibly useful helper func
int numSets() return # of current sets

How to Implement

- Linked List for each set] SLOW BAD (O(n)) wc

Good:
Disjoint Set Forest — two arrays

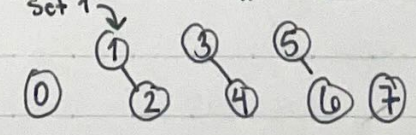
Well same speed wc
Expect/Average O(1)
"amortized"

Heuristics

- "union by rank"
- path compression

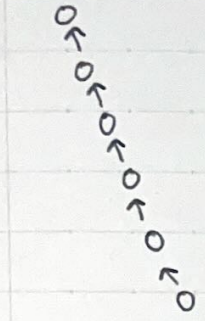
Both attempts to reduce the amount of time used by Find & Union funcs.

Sets are named after the root of the tree (that tracks the set).



*Longer paths worst performance
Shorter paths better performance

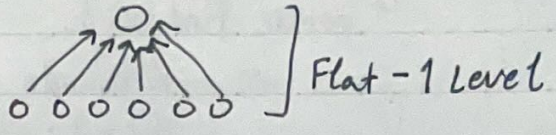
WC (Worst Case) (like a Linked List)



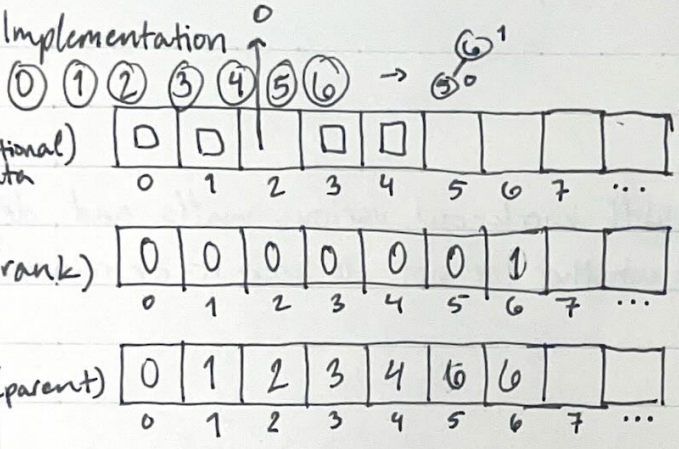
Path Compression

=>

BC (Best Case)



Implementation



private/helper function

```
Make-Set(x)
    parent[x] = x
    rank[x] = 0
```

```
Constructor. (int n)
    for (int i=0; i<n; ++i) {
        make-Set(i);
    } (or just add code to constructor)
```

PSEUDO CODE ints

```
public Union(x,y) {
    //bounds check
    Link(find(x), find(y));
}
```

trying to prevent the worst case

"Union by Rank"

```
private Link(x,y)
    //checks! - for example x==y
    if (rank[x] > rank[y]){
        parent[y] = x;
    }else{
        parent[x] = y;
        if (rank[x] == rank[y]) {
            rank[y]++;
        }
    }
```

"Find with path compression"

PSEUDO CODE CONT.

wc: O(h) = O(n) (h: height)

```
public Find(x) {
        //bounds check
        if(x != parent[x]) {
            parent[x] = find(parent[x]);
        }
        return parent[x];
    }
```

GENERATE MAZE

start

3x3

0 0,0	1 0,1	2 0,2
3 1,0	4 1,1	5 1,2
6 2,0	7 2,1	8 2,2

nxn

end

- Will knock-out various walls and decide whether or not to keep it or remove it

Basic Idea:

- have maze class & disjoint set DON'T COMBINE THEM TOGETHER
- Will come up with 1D or 2D scheme
  - ↳ 1D is better
- Actual maze representation is us tracking each square in hexadecimals & tracks walls