

MP6 released soon (Friday or by 5 today) Red Black \equiv AVL trees
Tries Huffman Encoding

Announcements:

Midterm exam 2, 4/5, 7-9p

rooms: 12p lecture in DCL 1320, 1p lecture in Everitt 151

email cinda with conflict

Class + Section cancelled Friday 4/6

Today:

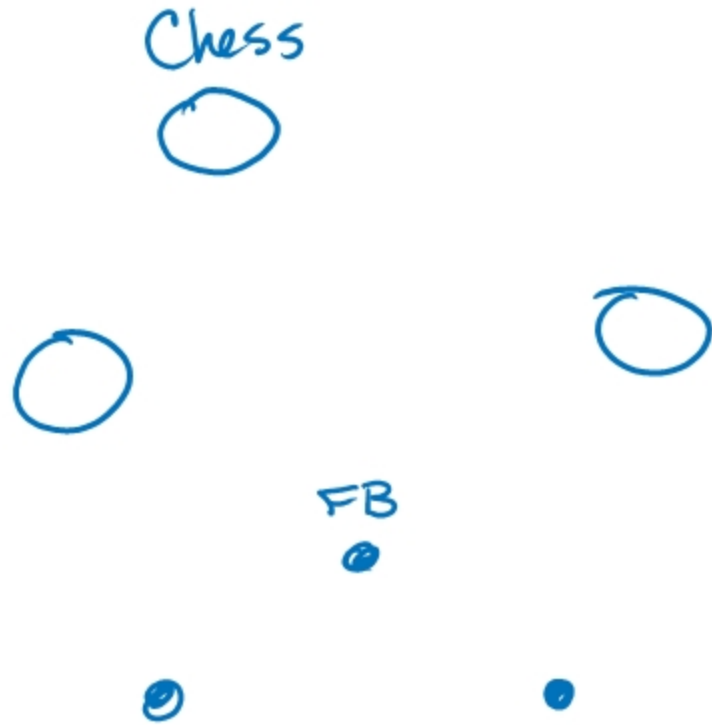
Disjoint Sets (something completely new)

3MT \rightarrow 2MT

Union :
Find :

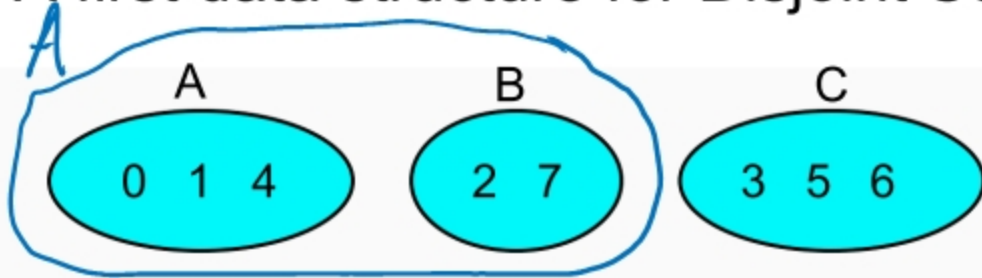
First an example:

Let R be an equivalence relation on the set of students in this room, where $(s,t) \in R$ if s and t have the same favorite game.



1. every student is in some set
2. no student is in more than 1 set.

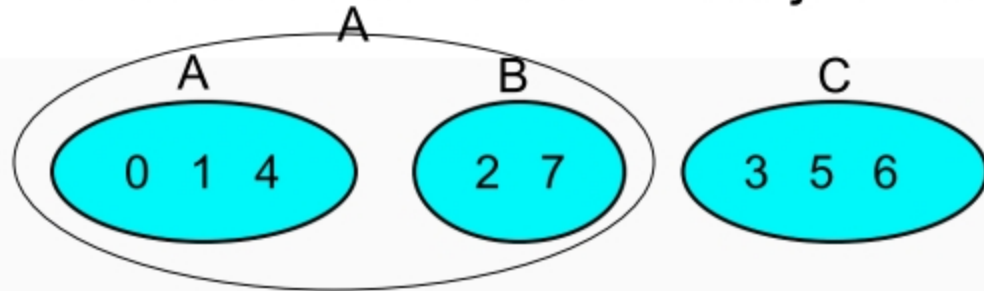
A first data structure for Disjoint Sets:



0	1	2	3	4	5	6	7
A	A	B A	C	A	C	C	B A

Find: $O(1)$ array lookup
Union: $O(n)$ for the scan (⊂)

A first data structure for Disjoint Sets:



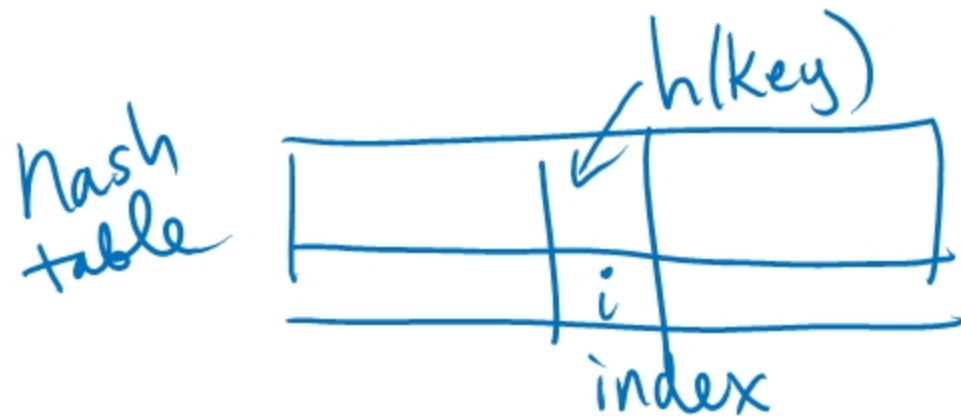
0	1	2	3	4	5	6	7
A	A	B _A	C	A	C	C	B _A

Find: $O(1)$ time

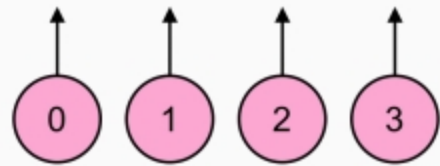
Union: $O(n)$ time

Indices used to keep track of relationships, but they're far from the data.

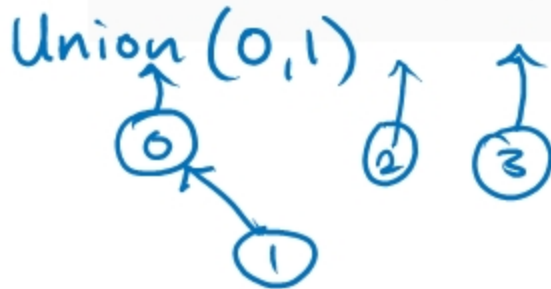
need a way to connect keys to indices (we want to use indices as an internal ID for our keys)



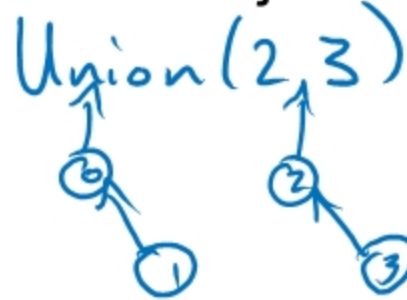
A second data structure for Disjoint Sets:



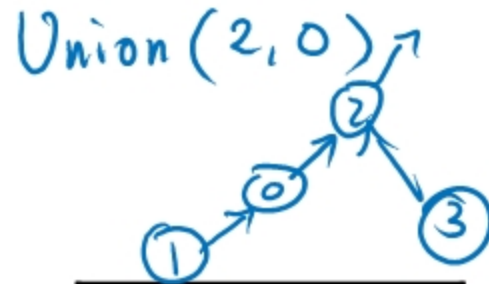
0	1	2	3
-1	-1	-1	-1



0	1	2	3
-1	0	-1	-1

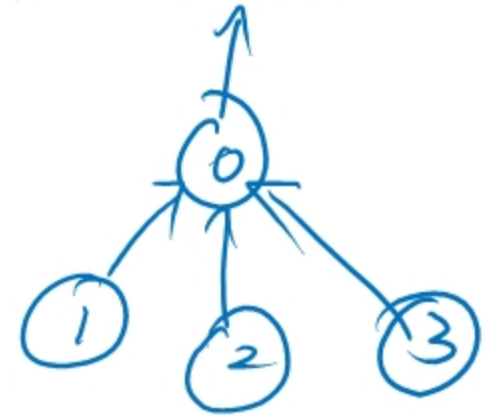


0	1	2	3
-1	0	-1	2



0	1	2	3
2	0	-1	2

$U(0,1) \cup U(0,2) \cup U(0,3)$



"Up Trees"

not ordered
not BST

not even nec. binary

Find(1) \rightarrow 2

3 array lookups

worst case $O(n)$ find(0)

A second data structure for Disjoint Sets:

```
int DS::Find(int i) {  
    if (s[i] < 0) return i;  
    else return Find(s[i]);  
}
```

$O(\text{tree height})$

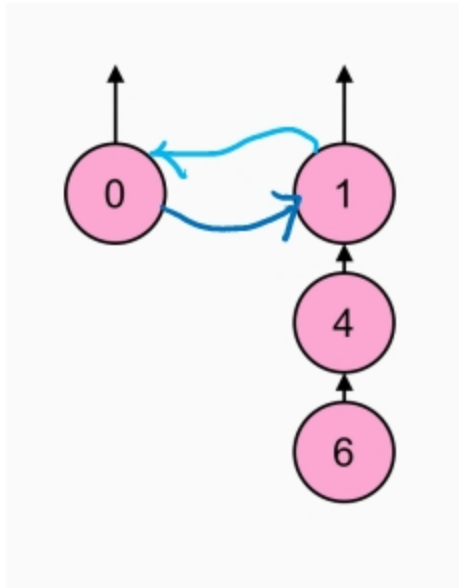
→ if (root2 != root1)

```
void DS::Union(int root1, int root2) {  
    s[root2] = root1;  
}
```

$O(1)$

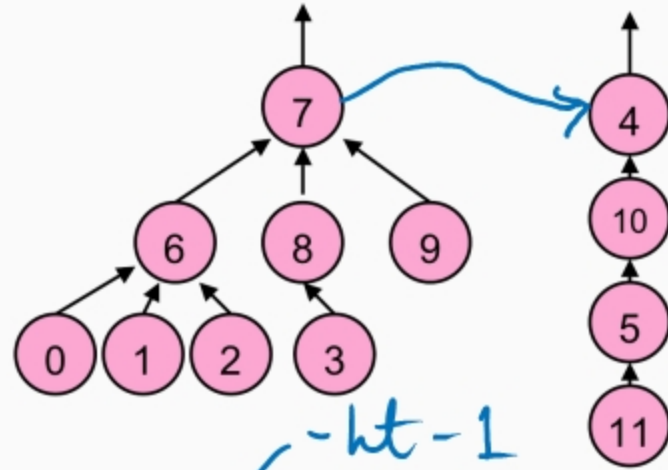
Find(4) == Find(8) they're in same set.

something to consider...



m is better
height doesn't increase
find time increases for
fewest # of nodes

Smart unions:



Union by height:

0	1	2	3	4	5	6	7	8	9	10	11
6	6	6	8	-5	10	7	4	7	7	4	5

Union by size:

0	1	2	3	4	5	6	7	8	9	10	11
6	6	6	8	7	10	7	-12	7	7	4	5

~ (# of nodes)

shorter points to taller

smaller points to larger
 ↗ (# of nodes)

Both schemes bound ht of tree to $O(\log n)$ worst case.

Smart unions:

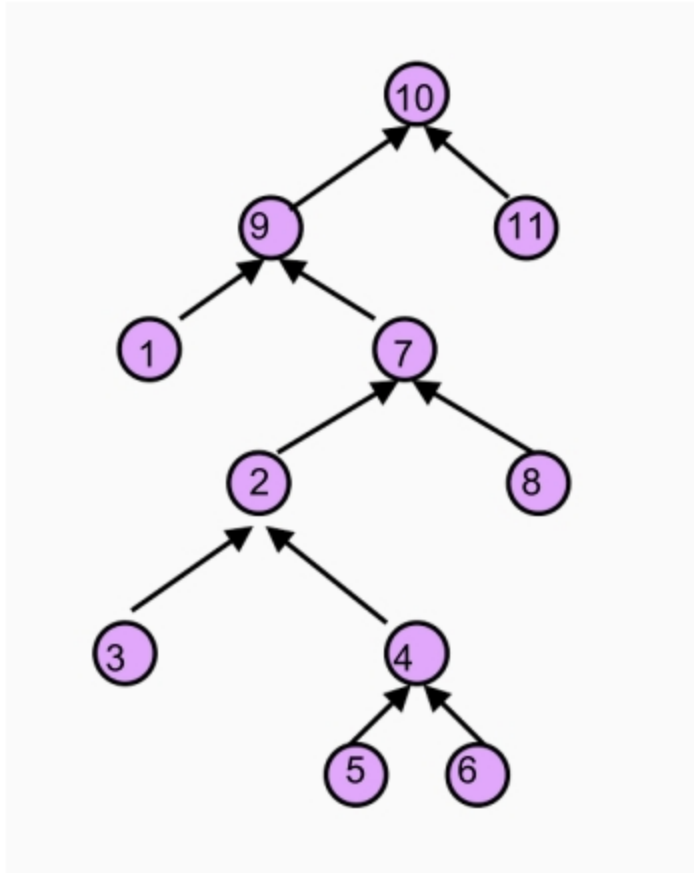
```
int DS::Find(int i) {  
    if (s[i] < 0) return i;  
    else return Find(s[i]);  
}
```

$O(\log n)$

```
void DS::UnionBySize(int root1, int root2) {  
    int newSize = s[root1]+s[root2];  
    if (isBigger(root1,root2)) {  
        s[root2]= root1;  
        s[root1]= newSize;  
    }  
    else {  
        s[root1] = root2;  
        s[root2]= newSize;  
    }  
}
```

$O(1)$

Path Compression:



Path Compression:

```
int DS::Find(int i) {  
    if (s[i] < 0) return i;  
    else return      Find(s[i]);  
}
```

```
void DS::UnionBySize(int root1, int root2) {  
    int newSize = s[root1]+s[root2];  
    if (isBigger(root1,root2)) {  
        s[root2]= root1;  
        s[root1]= newSize;  
    }  
    else {  
        s[root1] = root2;  
        s[root2]= newSize;  
    }  
}
```

Analysis: