CS201
Data Structures and Algorithms
Revision Session 8

disjoint set

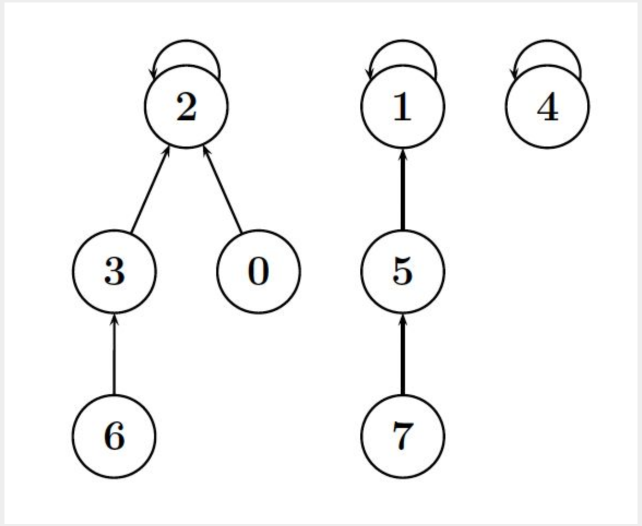
Disjoint set (Union-Find)

basics:
definition
implementation

operations:
finding set representative of an element (find)
merging two disjoint sets (union)

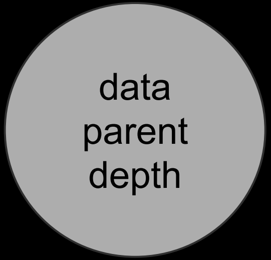
definition

- number of elements grouped into separate set which do not overlap
- each tree represents a set
- each tree has a representative (the root of the tree)
- each node is an element
- each node has a parent (except for the root node whose parent is itself)
- nodes point up to their parents (reduces the complexity of operations)

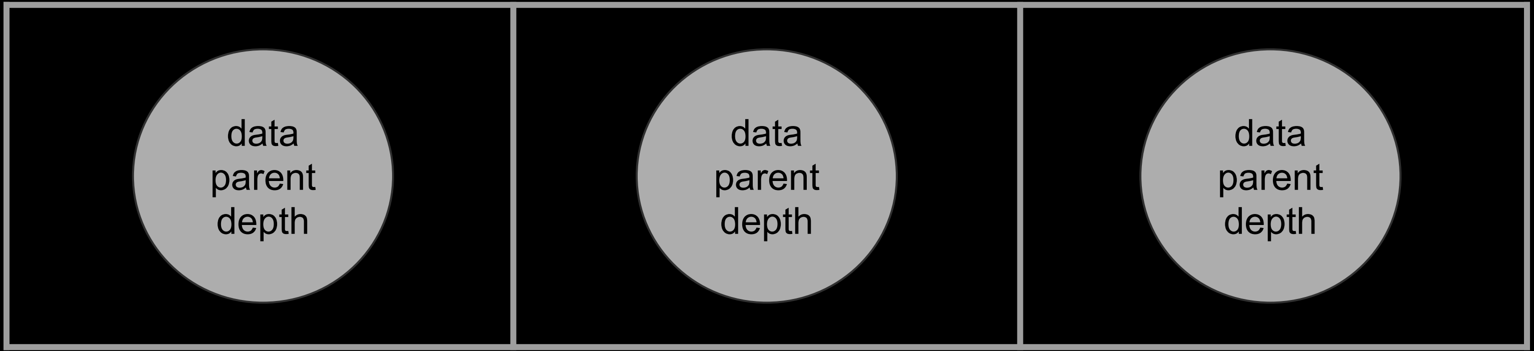


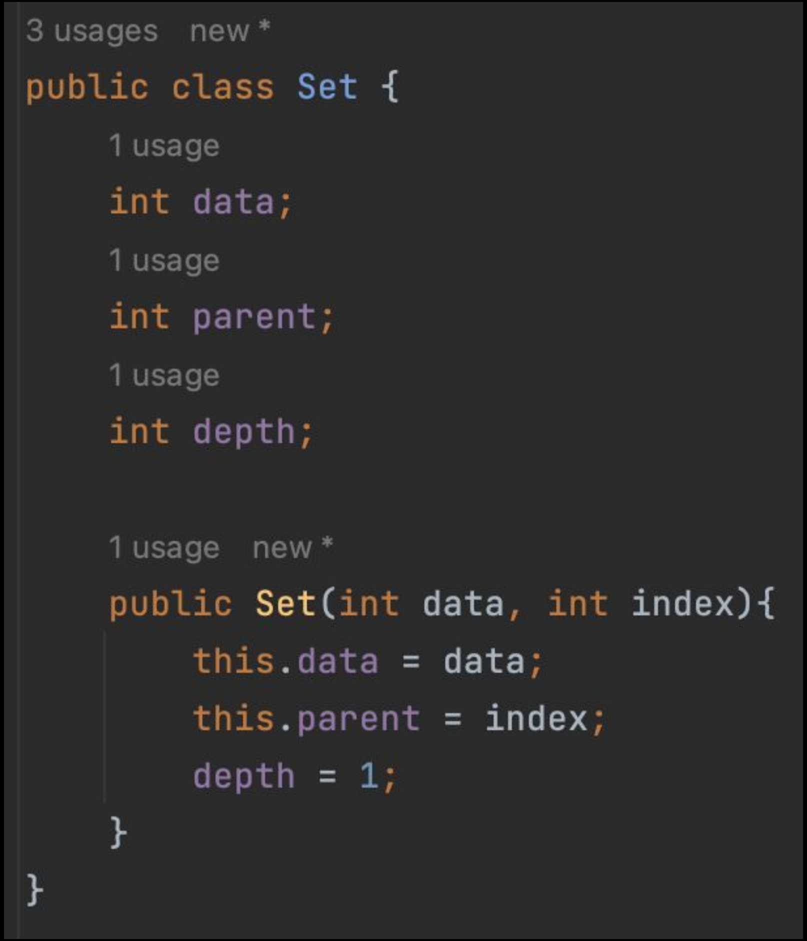
implementation

sets : starts as single nodes -> the nodes are bundles of 3 values (data, parent, depth)

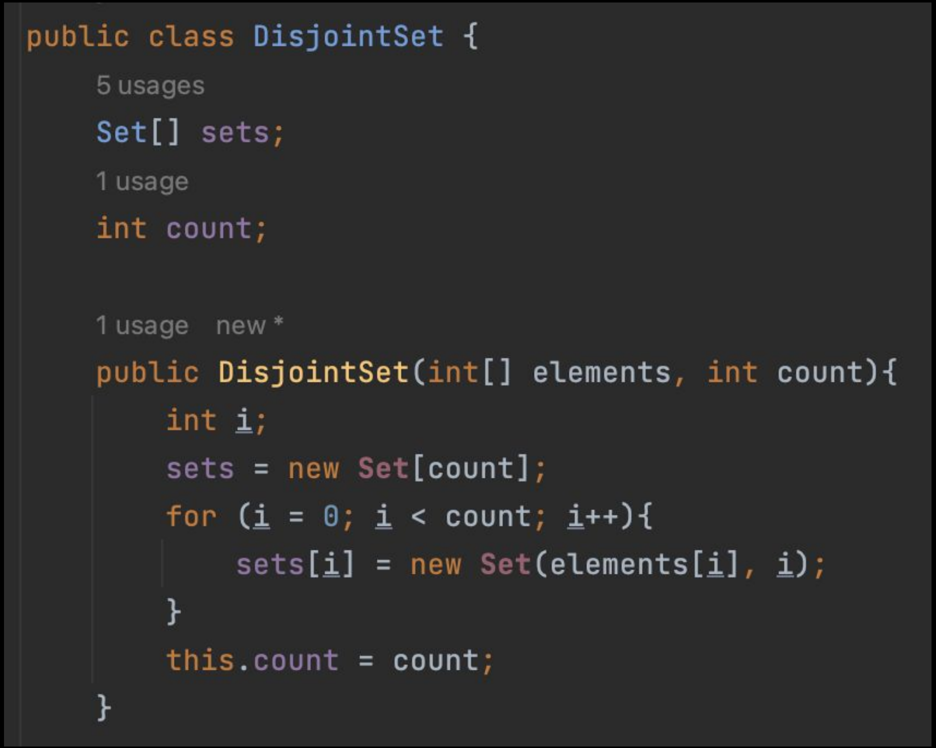


disjoint set : an array of sets



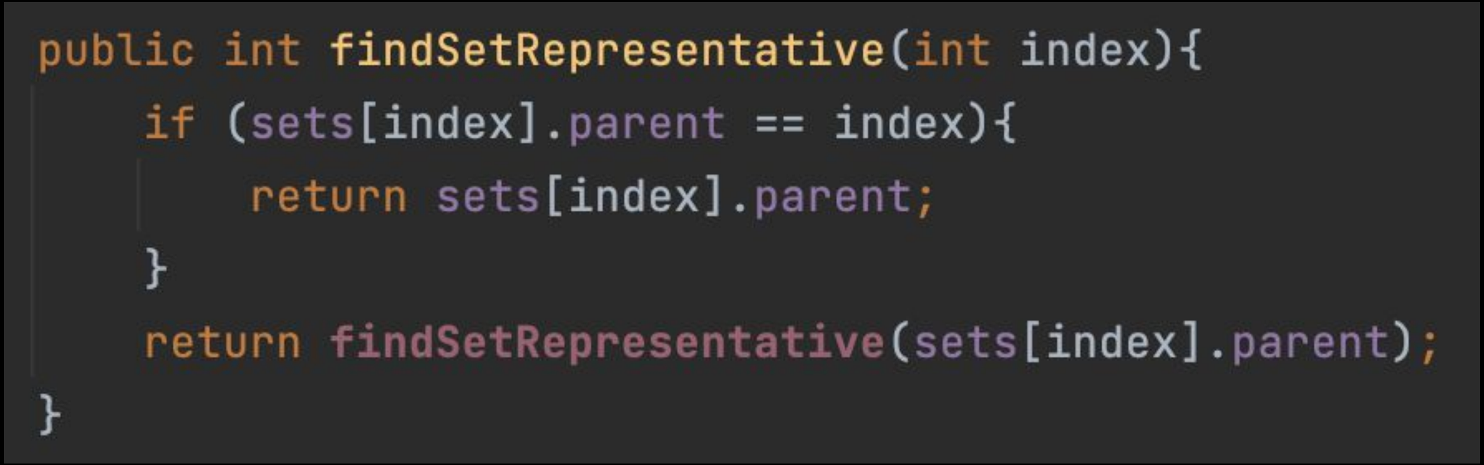


```
3 usages   new *
public class Set {
    1 usage
    int data;
    1 usage
    int parent;
    1 usage
    int depth;

    1 usage   new *
    public Set(int data, int index){
        this.data = data;
        this.parent = index;
        depth = 1;
    }
}
```



```
public class DisjointSet {
    5 usages
    Set[] sets;
    1 usage
    int count;

    1 usage   new *
    public DisjointSet(int[] elements, int count){
        int i;
        sets = new Set[count];
        for (i = 0; i < count; i++){
            sets[i] = new Set(elements[i], i);
        }
        this.count = count;
    }
```

finding the set representative of an element

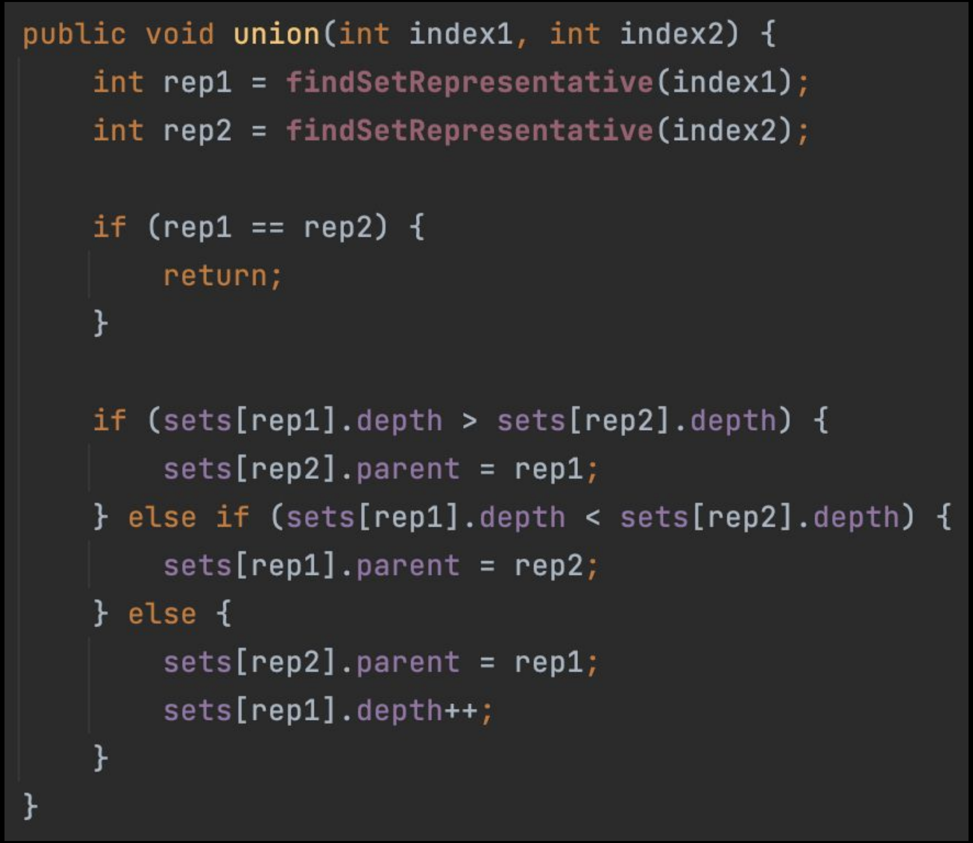


```java
public int findSetRepresentative(int index){
    if (sets[index].parent == index){
        return sets[index].parent;
    }
    return findSetRepresentative(sets[index].parent);
}
```

The index of any set bundle is given as the parameter. We look at the given index within our array of sets. If the parent of that specific set bundle is itself, then it is the set representative.

If its parent is another node, then we run the method recursively on the parent node. Once the parent is equal to the node itself, the method ends.

merging two disjoint sets (union)



```java
public void union(int index1, int index2) {
    int rep1 = findSetRepresentative(index1);
    int rep2 = findSetRepresentative(index2);

    if (rep1 == rep2) {
        return;
    }

    if (sets[rep1].depth > sets[rep2].depth) {
        sets[rep2].parent = rep1;
    } else if (sets[rep1].depth < sets[rep2].depth) {
        sets[rep1].parent = rep2;
    } else {
        sets[rep2].parent = rep1;
        sets[rep1].depth++;
    }
}
```

The index of two any set nodes are given in the parameters. First, we find the set representatives for each set.

If their representatives are equal, it means they already belong to the same set.

We check the depth of each representative. The root of the less deep set is linked to the root of the deeper set.

If the depths are the same, link **rep2** to **rep1**. Increment the depth of the new root, **rep1**.