Set and UFDS

Set

Set is an unordered collection of items with no duplicates. It is the same as in Math.

The main operations are:

- Find – Find an item in a set if it exists
- Insert – Add an item into a set
- Remove – Remove an item from a set
- Union – Union two sets together to form a new set with no duplicates
- Intersect – Find the items that are common in both sets

Set Implementation #1: HashTable

If we are not required to union or intersect any sets, then a HashTable is a possible implementation of a set. An existing Java API for it is the HashSet.

- Find(item) – O(1) on average as we just need to do O(1) hashing and finding
- Insert(item) – O(1) on average as we just do Insert(item, item) into the table
- Remove(item) – O(1) on average as we just do O(1) removal
- Union(set) – Not efficient
- Intersect(set) – Not efficient

Set Implementation #2: Union-Find Disjoint Sets

UFDS is a collection of disjoint sets. The key ideas are:

- Each set is modelled as a tree
    - Thus a collection of disjoint sets form a forest of trees
- Each set is represented by a **representative item**
    - Which is the root of the corresponding tree of that set

UFDS Implementation #1: Arrays for Parents and Rank

The simplest way to represent a UFDS is to use an array p, where p[i] = parent of item i. If p[i] = i, then i is a root, and is also the representative item of the set that contains i.

A second array, rank, is also used to support Union-by-Rank heuristic.

For example, we can have p = {2,3,3,3,3,6,6,6,8}, with indices 0,1,2,3,4,5,6,7,8. This means that the parent of item 0 is 2, item 1 is 3, and 3 is a representative item as it's the "parent of itself".

- findSet – O(α(N)) (to be elaborated on later)
    - Recursively visit p[i] until p[i] = i, then compress the path
    - **Path Compression**: when exiting, set p[i] to be representative item

```java
public int findSet(int i) {
  if (p.get(i) == i) return i;
  else {
    int ret = findSet(p.get(i));
    p.set(i, ret);
    return ret;
  }
}
```

- isSameSet(i,j) - O(α(N))
  - Check if findSet(i) == findSet(j)
- unionSet(i, j) - O(α(N))
  - If the two items i and j are of different disjoint sets, we can union the two sets by setting the representative item of the taller tree to be the representative item of the new combined set
  - Union-by-Rank heuristic
    - Makes the resulting tree shorter
    - If both trees are equally tall then it does not matter
    - We use another integer array **rank,** where rank[i] stores the upper bound of the height of (sub)tree rooted at i
    - This is just an upper bound as path compressions can make (sub)trees shorter than its upper bound and we do not want to waste effort maintaining the correctness of rank[i]

```java
public void unionSet(int i, int j) {
  if (!isSameSet(i, j)) {
    int x = findSet(i), y = findSet(j);
    // rank is used to keep the tree short
    if (rank.get(x) > rank.get(y))
      p.set(y, x);
    else {
      p.set(x, y);
      if (rank.get(x) == rank.get(y)) // rank increases
        rank.set(y, rank.get(y)+1); // only if both trees } // initially have the
same rank
    }
  }
}
```

Inverse Ackermann Function - α(N)
If both Union-by-Rank and Path Compression heuristics are used, then UFDS operations run in O(α(N)) time. This function grows very slowly, and we can assume it as constant O(1) for practical values of N (<1M).

Static DS: UFDS is a static data structure as we cannot add new items to the sets after it has been created.

Constructor for UFDS:

```
class UnionFind { // OOP style private ArrayList<Integer> p, rank;
  public UnionFind(int N) {
  p = new ArrayList<Integer>(N);
  rank = new ArrayList<Integer>(N);
  for (int i = 0; i < N; i++) {
  p.add(i);
        rank.add(0);
      }
  }
  // ... other methods in the previous slides
}
```

Learning Points from Tutorial

Given *n* disjoint sets initially in a UFDS, is it possible to call unionSet(i, j) and/or findSet(i) operations to get a single tree with actual height *h* that represents a certain set? Both 'path-compression' and 'union-by-rank' heuristics are used.

- Possible if $h \leq \log_2 n$