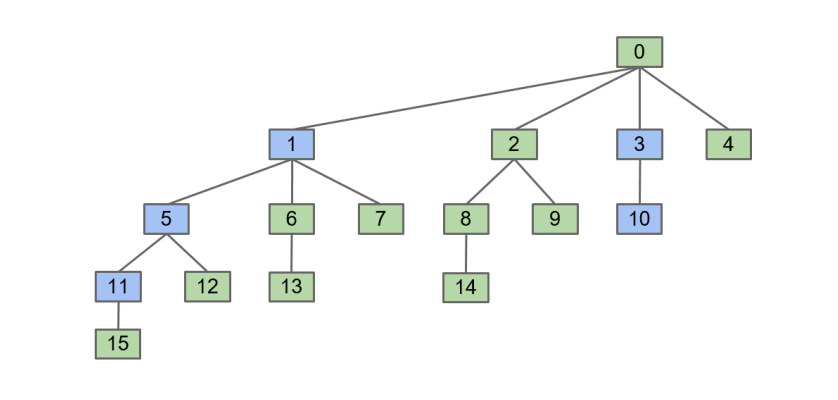
**Part A: Union Find with Path Compression**

We have constructed a Union Find data structure called Weighted Quick Union, and achieved O(log N) time for union, find and isSameGroup operations. It turns out that we can do even better, to get almost constant time for both operations! Weighted Quick Union data structure with Path Compression.

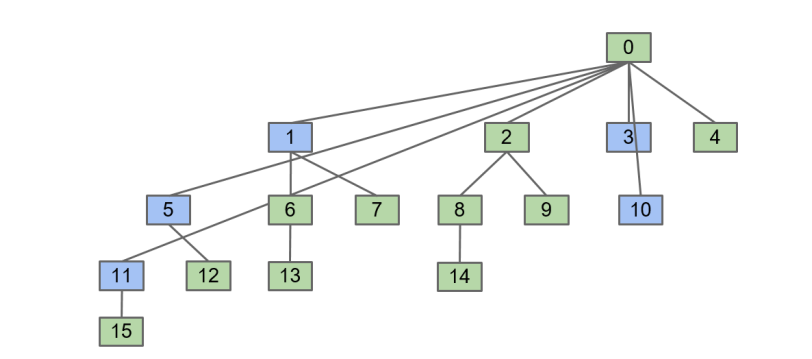
**Idea of Path Compression**

Consider the connectivity tree below. It is one of the possible valid trees of 16 items in a Weighted Quick Union data structure, resulting from some series of operations. Now imagine you call isSameGroup(10, 11). That will involve finding the root of 10 and 11, and will be preceded by finding the parent of the elements in blue.



The key idea is this: since we have found the root of the elements in blue while climbing the tree, whose root is 0, we want to change and set the parents of those blue elements directly to the root.

The result is the following tree:



Notice that this changes nothing about which group each element belongs to. They are still in the tree where 0 is the root. The additional cost of this path compression operation to isSameGroup is still in the same order of growth, but now the future operations that require finding the root will be faster! We are going to use the same path compression idea on the other operations as well. Note that this path compression results in amortized running time on N operations of union/isSameGroup/size of O(α(N)), where α is the inverse Ackermann function, of which the value, for practical purposes, is always less than 5.

**Part A: Weighted Quick Union with Path Compression**

In part A, your task is to complete an implementation of the Union Find data structure using Weighted Quick Union of Path Compression.

You will have to implement the following methods to complete the data structure:

**1. public UnionFind(int N) creates a Union Find data structure with N elements: 0 through N-1. Initially, each element is in its own group.**

**2. public void validate(int p) checks whether p is a valid element. It throws an IllegalArgumentException if p is not a valid index.**

**3. public int size(int p) returns the size of the group the element p belongs to.**

**4. public int find(int p) returns the group identity number which is the root of the tree element p belongs to. Assume p is a valid element. The path compression operation is applied in this method to reduce the finding root's running time. Note that now, the given method public boolean isSameGroup(int p, int q) is then implemented by simply calling validate on p and q, and then checking whether find(p) is the same as find(q).**

**5. public void union(int p, int q) connects two elements p and q together, by combining the groups containing them, connecting the root of the smaller size tree to the root of the larger size tree. If the sizes of the trees are equal, break the tie by connecting p's root to q's root. It throws an IllegalArgumentException if p or q is not a valid index.**

**Advice**

The following advice may be found useful in implementing Part A:

1. Use the same Automated Regression Unit Testing and Integration Testing strategy that you have been using in Weighted Quick Union. Note that with the use of the Path Compression strategy, the output may be different from the result in Weighted Quick Union.

2. Add more test cases, and create a good suite of test cases and practice the Partitioning/Boundary, Black-box/White-box, and Coverage testing.

3. Debug with the help of Java Visualizer plugin in IntelliJ IDEA.

4. You may define your own private helper methods. Include them in each of your submissions.

5. Do not define your own instance variables. They are not going to be used in the hidden test cases and may cause unpredictable errors in the grading system.