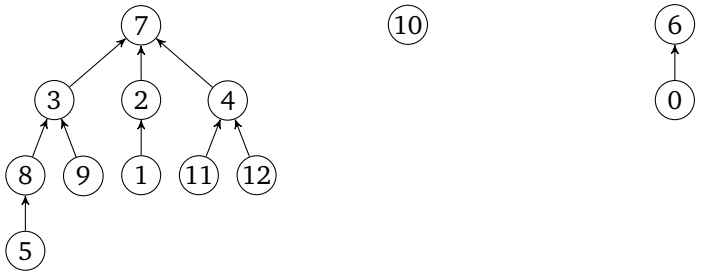
Section 08: Disjoint Sets, Topological Sort, and Tries

1. Disjoint sets

(a) Consider the following trees, which are a part of a disjoint set data-structure:

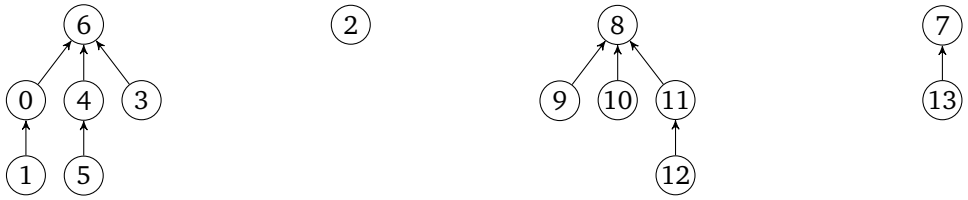


For these problems, use both the **weighted quick union by size** and **path compression** optimizations.

(i) Draw the resulting tree(s) after calling `find(5)` on the above. What value does the method return?

(ii) Draw the final result of calling `union(2,6)` on the result of part a.

(b) Consider the disjoint-set shown below



What would be the result of the following calls on union if we add the **weighted quick union by size** and **path compression** optimizations.

(i) `union(2, 13)`

(ii) `union(4, 12)`

(iii) `union(2, 8)`

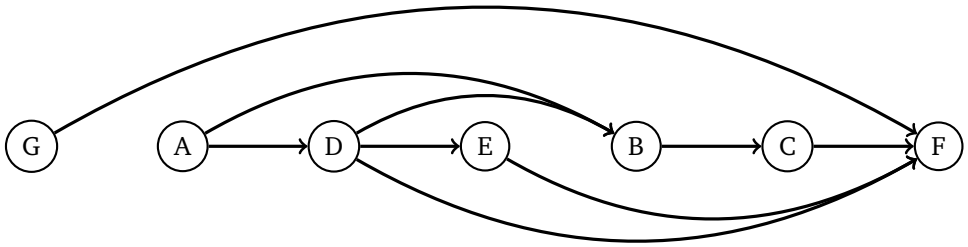
(c) Consider the disjoint-set shown below

What would be the result of the following calls on union if we add the **weighted quick union by size** and **path compression** optimizations.

(i) `union(10, 0)`

2. Topological sort

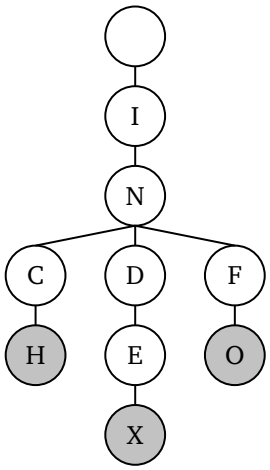
(a) Give a valid topological sort of the graph below. For your reference, some orderings of the graph are provided below the graph.




```
DFS preorder: ABCFDE (G)
DFS postorder: FCBEDA (G)
BFS: ABDCEF (G)
```

3. Tries

(a) Consider the trie shown below:



(i) What strings are stored in the trie?

(ii) Insert the strings *indent*, *inches*, and *trie* into the trie.

(b) How could you modify a trie so that you can efficiently determine the number of words with a specific prefix in the trie?