Weekplan: Union Find

The 02105+02326 DTU Algorithms Team

Reading

Introduction to Algorithms, Cormen, Rivest, Leisersons and Stein (CLRS): Chapter 21.1-21.4 and *Algorithms*, 4ed., Sedgewick and Wayne: Chapter 1.5.

Exercises

1 Run Union Find by Hand Look at the following sequence of operations: $\textsc{Init}(7)$, $\textsc{Union}(3,4)$, $\textsc{Union}(5,0)$, $\textsc{Union}(4,5)$, $\textsc{Union}(4,3)$, $\textsc{Union}(0,1)$, $\textsc{Union}(2,6)$, $\textsc{Union}(0,4)$ and $\textsc{Union}(6,0)$.

- **1.1** [w] Run the sequence of operations using quick find by hand. Show the contents of the *id* array after every step. Assume the $\textsc{Union}(i,j)$ operation always updates *id* for the set given by i.
- **1.2** [w] Run the sequence using quick union by hand. Show the trees after every step. Assume $\textsc{Union}(i,j)$ always sets the root of the tree given by i to be a child of the root of the tree given by j.
- **1.3** Run the sequence using weighted quick union by hand. Show the trees after very step. Assume $\textsc{Union}(i,j)$ sets the root of the tree given by i to be a child of the root of the tree given by j when the sizes of two trees are equal.
- **1.4** Show the result of path compression after a $\textsc{Find}(x)$ operation, where x is respectively a leaf, an internal node of depth 1, and an internal node of height 1, in one of the trees from the above exercises.
- **1.5** Give a sequence of operations that results in a tree of maximal depth using quick union.
- **1.6** Give a sequence of operations that results in a tree of maximal depth using weighted quick union.
- **1.7** Write pseudo code for a algorithm to do path compression. *Hint:* traverse the path twice.

2 Alternative to the Quick Find Algorithm One of your fellow students suggests the following intuitive variant of quick find $\textsc{Union}$. Does it work?

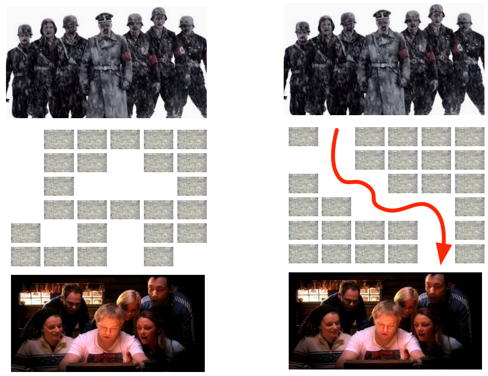
```
UNION(i, j)
if FIND(i) ≠ FIND(j) then
  for k = 0 to n − 1 do
    if id[k] == id[i] then
      id[k] = id[j]
    end if
  end for
end if
```

3 Dynamic Connected Components and Graph Search Using graph search (DFS or BFS) we can find the connected components of a graph. Give a simple solution for dynamic connected component using graph search and compare the complexity with the solutions based on union find.

4 Implementation of Union Find We want to implement data structures for union find that supports $\textsc{Init}$, $\textsc{Union}$, and $\textsc{Find}$.

- **4.1** [BEng†] Implement quick find.
- **4.2** [†] Implement quick union.
- **4.3** [†] Extend the solution with weighted quick union.
- **4.4** [†] Extend the solution with path compression.

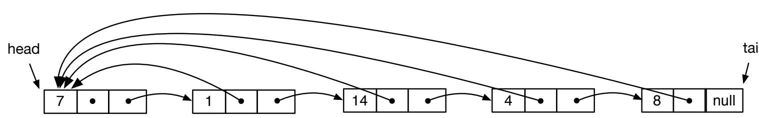
5 [$*$] **Zombie Invasion** In the post apocalyptic zombie world you and a small group of survivors have barricaded yourself in a small building. The only thing keeping the brutal zombies from eating you is a strong fortification. The fortification consists of a $k \times k$ grid of walls. Here illustrated by a 6×6 grid of walls.[1]



In the top of the grid the zombies are waiting to come in, and you and your group is located in the bottom. Unfortunately, the walls are weak and collapse regularly. If a path of walls between the top and the bottom of the grid is collapsed the zombies can get in and eat you. In order to start evacuation you want to monitor if there currently is a path through the fortification (from top to bottom). Give a data structure that can efficiently keep track of this while the walls are collapsing one by one.

6 [$*$] **Recursive Path Compression** Write pseudo code for a *recursive* algorithm for path compression. *Hint:* it can be done with only few lines of code.

7 **Union Find using Linked Lists and Weights** We want to implement a variant of quick find using linked lists in the following way. Each set is represented by a singly linked list. The representative for a set is the first element in the list and each element in the list has a pointer to the representative. Furthermore we maintain a pointer to the tail of the list. For instance, the data structure for the set $\{1, 4, 7, 8, 14\}$ with representative 7 could look like this:



7.1 Using the representation, show how to implement INIT(n) in $O(n)$ time, FIND(i) in $O(1)$ time and UNION(i, j) in $O(|S(i)|)$ time, where $S(i)$ is the set containing i.

7.2 Show how to extend the solution such that INIT and FIND runs in the same time, but the time for UNION(i, j) is $O(\min(|S(i)|, |S(j)|))$. *Hint:* maintain a little extra information.

7.3 [$*$] Show that for p FIND and m UNION operations on n elements the above solution gives the running time $O(p + m \log n)$.

[1]Pictures from "Død snø", 2009.