Weekplan: Union-Find

Philip Bille

Reading


Exercises

1 Run Union-Find by Hand  Look at the following sequence of operations: INIT(7), UNION(3, 4), UNION(5, 0), UNION(4, 5), UNION(4, 3), UNION(0, 1), UNION(2, 6), UNION(0, 4) and UNION(6, 0).

1.1 Run the sequence of operations using fast find by hand. Show the contents of the id array after every step. Assume the UNION(i, j) operation always updates id for the set given by i.

1.2 Run the sequence using fast union by hand. Show the trees after every step. Assume 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 union by hand. Show the trees after every step. Assume 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 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 fast union.

1.6 Give a sequence of operations that results in a tree of maximal depth using weighted union.

1.7 Write pseudo code for a algorithm to do path compression. *Hint:* traverse the path twice.

2 Alternative to the Fast Find Algorithm  One of your fellow students suggests the following intuitive variant of UNION fast find. Does it work?

```plaintext
UNION(i, j)
if FIND(i) \neq 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 for union and find.

4 Implementation of Union-Find  We will like to implement data structures for union and find that supports INIT, UNION, and FIND.

4.1 [BEng†] Implement fast find.

4.2 [†] Implement fast union.

4.3 [†] Extend the solution with weighted 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 $6 \times 6$ 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 is currently 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 fast 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 $O(p + m \log n)$.

M Mandatory Exercise: Floor Plan (exercises from the exam 2015)  A floor plan consists of a set of $R$ rooms $r_0, \ldots, r_{R-1}$ and $D$ doors $d_0, \ldots, d_{D-1}$ that each connects exactly two rooms. Each room is a geometric figure and a door between two rooms are indicated by three small bold lines between the rooms. For instance the following floor plan $P$ consists of 11 rooms and 12 doors.

---

¹Pictures from "Død snø", 2009.
M.1 Describe how to model a floor plan as a graph.

M.2 Draw the graph corresponding to the floor plan $P$ in the above example.

M.3 We are interested in examining if it is possible to evacuate the rooms in case of fire. The entrance is a special room on the floor plan. A fire door is a door that will automatically close in case of fire. A room can be evacuated to the entrance if there is a connection from the room to the entrance that does not use any fire doors. Give an algorithm that given a floor plan, an entrance $e$ and a set $B$ of $k$ fire doors determines if all rooms can be evacuated to $e$. Analyze the running time of your algorithm as a function of $R$, $D$, and $k$.

M.4 We are now interested in displaying art in the form of either a sculpture of a painting in all rooms in a nice fashion. A route on the floor plan is a sequence of rooms $r_0, \ldots, r_{z-1}$ such that room $r_i$ and $r_{i-1}$ are connected by a door, and $r_0 = r_{z-1}$. A room can be visited multiple times on a route. A route is beautiful if the rooms on the route alternates between having a sculpture and a painting. Look at the example floor plan $P$ from before. Is it possible to display either a sculpture or a painting in each room such that all routes that starts and ends in the room furthest to the left are beautiful?

M.5 Give an algorithm that given a floor plan and an entrance $e$ determines if all routes that starts and ends in $e$ are beautiful. We assume that there is a path from $e$ to all rooms. Analyze the running time of your algorithm as a function of $R$ and $D$. 