Algorithm

YIXIANG QIU

February 2025

Contents

1 Asymptotic Analysis

Asymptotic Analysis is a method used to quantify the **time** and **space** cost in an algorithm.

1.1 Symbols

First we can define the order of growth of Θ. Instead of saying a function has order of growth W, we say that the function belongs to $\Theta(W)$. In other words it belongs to the family of functions that have the same order of growth.

Theorem 1

For some function $R(N)$ with order of growth $f(N)$ we write that $R(N) \in \Theta(f(N))$.

Example 2

Suppose a function is defined to be $R(N^3 + 3N^4)$ then the order of growth is N^4. Then we can write as $R(N^3 + 3N^4) \in \Theta(N^4)$.

The Difference between $O(N)$ and $\Theta(N)$:

- The Θ means that the same order of growth, which also means the tight-bound (both upper-bound and lower-bound).
- The O can be thought as less then or equal to some order of growth. Which is equivalent to the upper bound.

Example 3

Suppose $f(N) = 2N$, all of these statements are true.

$$f(N) \in \Theta(N) \quad \text{and} \quad f(N) \in O(N) \quad \text{and} \quad f(N) \in O(N^2)$$

2 Searching

3 Sorting

Sorting Algorithm is an important strategy in Computer. In this section will introduce some useful sorting algorithm. And analysis their Time and Space Complexity.

3.1 Selection Sort

Selection sort is perhaps the easiest sorting algorithm. A formal selection sort consists of three steps. Suppose an array has N elements.

1. Find the smallest element.
2. Move it to the front.
3. Selection sort the remaining $N - 1$ elements.

4 Disjoint Set

Two sets are named disjoint sets if they have no elements in common. A Disjoint-Sets data structure keeps track of a fixed number of elements partitioned into a number of disjoints sets. The data structure has two main operations:

1. Connect : Connect x, y as a set.
2. Is Connect : Judge whether two sets are connected.

Here is the overview of different implementation

- Quick Find : Set the value stored at index as the ID of the element.
- Quick Union: Set the value stored at index as the parent and -1 represent the root itself.
- Weighted Quick Union: Set the value stored at index as parent or the negative of the size of the tree.
- Weighted Quick Union With Path Compression : Same as Weighted Quick Union with optimization that every time call `find_root` will set its parent to the root.

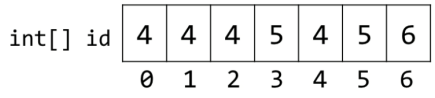
Implementation	Constructor	connect	Is Connect	Find Root
Quick Find	$\Theta(N)$	$O(N)$	$O(1)$	N/A
Quick Union	$\Theta(N)$	$O(N)$	$O(N)$	$O(N)$
Weighted Quick Union	$\Theta(N)$	$O(\log N)$	$O(\log N)$	$O(logN)$
Weighed Quick Union With Compression	$\Theta(N)$	$O(\alpha(N))*$	$O(\alpha(N))*$	$O(\alpha(N))*$

Table 1: Overview Runtime Analysis

4.1 Quick Find

Using Quick Find to implement our Disjoint Set using a simple strategy

- The indices of the array represent the elements of our set.
- The value at the index is the set number (set ID) it belongs to.



The array indices $(0, 6)$ are elements, the value at id[i] is the set it belongs to. We call this implementation Quick Find because finding if elements are connected takes constant time.

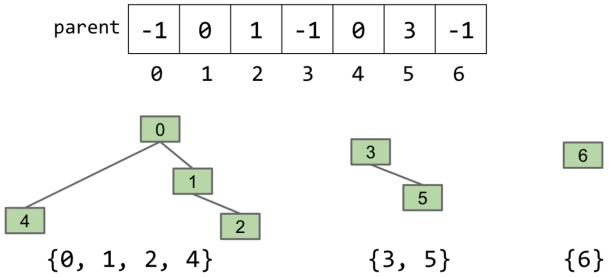
Implementation	Constructor	connect	Is Connect
Quick Find	$\Theta(N)$	$O(N)$	$O(1)$

Table 2: Runtime Analysis Of Quick Find

4.2 Quick Union

Suppose we prioritize making the `connect` operation fast. We will still represent our sets with an array. Instead of an id, we assign each item the index of its parent. If an item has no parent, then it is a root and we assign it a negative value.

This approach allows us to imagine each of our sets as a tree.



For Quick Union we define a helper function `find_root(int item)` which returns the root of the tree `item` is in.

In the best case, if `x` and `y` are both roots of their trees, then `connect(x, y)` just makes `x` point to `y`, a $\Theta(1)$ operation. That's why we call it Quick Union.

There is a potential performance issue with Quick Union: the tree can become very long. In this case, finding the root of an item can be very expensive. In the worst case, we have to traverse all the items to get to the root which is $\Theta(N)$.

Implementation	Constructor	connect	Is Connect	Find Root
Quick Union	$\Theta(N)$	$O(N)$	$O(N)$	$O(N)$

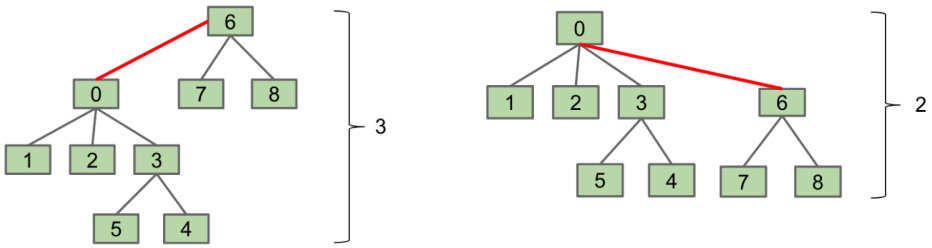
Table 3: Runtime Analysis Of Quick Union

4.3 Weighted Quick Union

Improving on Quick Union relies on a key insight: whenever we call `find`, we have to climb to the root of a tree. Thus, the shorter the tree the faster it takes!

Now we can do some optimization on `connect` : whenever we call `connect`, we always link the root of the smaller tree to the larger tree. With this Optimization we can get a tree with maximum height $logN$.

We can change our structure of data. We determine smaller or larger tree by the number of items of the tree. Thus we connecting two trees by storing the size of the tree by replacing the -1 with $-$(size of tree).



We will connect the smaller tree to the larger tree. Namely the option 2 which the final height of the tree will be 2.

Implementation	Constructor	connect	Is Connect	Find Root
Quick Union	$\Theta(N)$	$O(\log N)$	$O(\log N)$	$O(logN)$

Table 4: Runtime Analysis Of Weighted Quick Union

Theorem 4

Suppose a WQU has height H. And N is the number of elements in the WQU. So $H \leq log_2 N$. Namely $N \geq 2^H$.

4.4 WQU With Path Compression

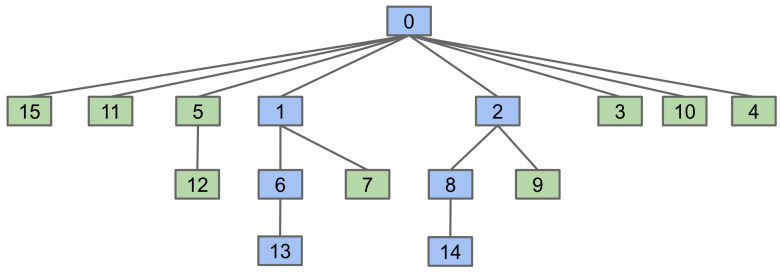
Quick Union With weighted tree we can optimize the runtime to $\log N$. With even more optimization we can get a better result which near constant runtime $O(N)$.

Optimization Strategy

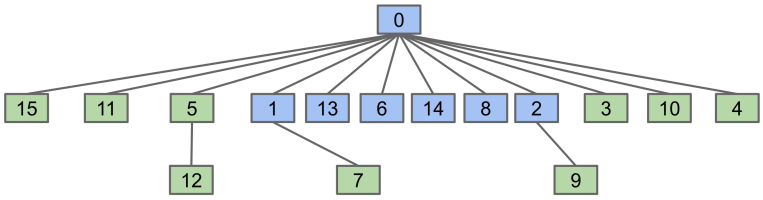
Whenever the `find_root` is called we have traverse the path from x to root. So we can connect all the items we visit to their root at no asymptotic cost. Namely we can connect all the items along the wat to root.

By extension, the average runtime of `connect` and `is_connect` becomes almost constant in long term. This is called amortized runtime. Which will cost $O(\lg * N)$ and can be written as $O(\alpha(N))$.

Below is the graph of the tree before we call `connect(13, 14)`.



And here is after we called `connect(13, 14)`, we can see from the figure that every parent node from 13 and 14 has been reset index to their root.



And with this optimization on the path the runtime will behave like constant in the long term.

Implementation	Constructor	connect	Is Connect	Find Root
Quick Union	$\Theta(N)$	$O(\alpha(N))*$	$O(\alpha(N))*$	$O(\alpha(N))*$

Table 5: Runtime Analysis Of WQU with Path Compression