Disjoint-set Union

Eklavya Sharma

Contents

1 Problem **2**

2 Forest algorithm **2**

2.1 Performance with no optimizations . . . 3
2.2 `rank` upper-bounds height . . . 3
2.3 Performance when `union_by` is not `None` . . . 4
2.4 Lower bound on time when `compress_path` is `False` . . . 5
2.5 Both union-by-rank and path-compression . . . 5
2.5.1 Alt-Ackermann function . . . 5
2.5.2 level and iter . . . 6
2.5.3 Potential function . . . 7

All (pseudo-)code in this document is based on the python programming language.

1 Problem

In the Disjoint-set Union (DSU) problem, we are given a set S of n singleton sets, i.e. $S = \{\{i\} : 0 \leq i < n\}$.

We have to perform m operations on S. Each operation can modify S while maintaining these 2 invariants:

1. All elements of S are sets.
2. Every integer from 0 to n-1 lies in exactly one set in S.

Also, for every set $X \in S$, one of the elements of X will be known as the 'representative of X', denoted as $\text{repr}(X)$.

Types of operations allowed

1. `find(x)`: If $x \in X$, return $\text{repr}(X)$.
2. `union(x, y)`: Let $x \in X$ and $y \in Y$. Then remove X and Y from S and add $X \cup Y$ to S.

`union(x, y)` is the only operation which can modify S. It is easy to see that `union(x, y)` maintains the 2 invariants.

2 Forest algorithm

The 'forest algorithm' for DSU maintains a forest $F = (V, E)$ of rooted trees where $V = \{i \in \mathbb{Z} : 0 \leq i < n\}$. Each tree in F corresponds to a set in S. The representative of a set is the root of the corresponding tree.

The forest is stored by keeping track of the parent of each vertex in an array `parent` of size n. If a vertex x has no parent, then `parent[x] = x`. The algorithm (optionally) maintains 2 additional arrays `rank` and `size`. `rank[i]` is an upper-bound on the height of vertex i and `size[i]` is the size of the subtree rooted at vertex i. Initially `parent[i] = i`, `rank[i] = 0` and `size[i] = 1` for all $0 \leq i < n$.

This algorithm offers 2 hyperparameters. These are optional optimizations for speeding up DSU.

1. `union_by`: can be `None`, `rank` or `size`.
2. `compress_path`: can be `False` or `True`.

This is how `find` and `union` are implemented:

```
def find(x):
    if parent[x] == x:
        return x
    else:
        r = find(parent[x])
        if compress_path:
            parent[x] = r
        return r

def link(x, y):
    parent[y] = x
    size[x] += size[y]
    rank[x] = max(rank[x], rank[y] + 1)

def union(x, y):
    x = find(x)
    y = find(y)
    if union_by is not None and union_by[x] < union_by[y]:
        x, y = y, x

    link(x, y)
    return x != y
```

2.1 Performance with no optimizations

Consider the following operations:

```
for i in range(1, n):
    union(i, i-1)
for i in range(1, m - n):
    find(0)
```

When `union_by` is `None`, `union(x, y)` makes the tree of y a subtree of x. Therefore, after all the `union` operations, the forest will be a single chain from 0 to $n-1$. If `compress_path` is `False`, each `find(0)` operation will take $\Theta(n)$ time. Each `union` operation takes $\Theta(1)$ time. Therefore, total time taken is $\Theta((m-n)n)$.

2.2 `rank` upper-bounds height

For a tree T, let $h(T)$ denote its height, $n(T)$ denote the number of nodes in it and $r(T) = \text{rank}(\text{repr}(T))$.

Theorem 1. *$h(T) \le r(T)$ throughout the algorithm.*

Proof. Initially, $h(T) = r(T) = 0$ for every tree T.

In a `find` operation, the height of a tree can only reduce (it can reduce if `compress_path` is `True`, otherwise it doesn't change).

Suppose `link(x, y)` is called and $x \in X$ and $y \in Y$. Then Y is made a subtree of X. Let the resulting tree be Z. Suppose $h(X) \leq r(X)$ and $h(Y) \leq r(Y)$.

$$h(Z) = \max(h(X), h(Y) + 1) \leq \max(r(X), r(Y) + 1) = r(Z)$$

Since $h(T) \leq r(T)$ is initially true and remains true across `find` and `union` operations, $h(T) \leq r(T)$ is true for all trees across the entire DSU algorithm. $\square$

2.3 Performance when `union_by` is not `None`

Theorem 2. `union_by` $\neq$ `None` $\implies \forall T, r(T) \leq \lg n(T)$.

Proof. Initially, $\forall T, r(T) = 0 = \lg 1 = \lg n(T)$.

`find` operations affect neither r nor n.

Suppose `link(x, y)` is called and $x \in X$ and $y \in Y$. Then Y is made a subtree of X. Let the resulting tree be Z. Suppose $r(X) \leq \lg n(X)$ and $r(Y) \leq \lg n(Y)$. $r(Z) = \max(r(X), 1 + r(Y))$ and $n(Z) = n(X) + n(Y)$.

Case 1: `union_by = size`
`union_by = size` $\implies n(Y) \leq n(X)$.

$$\begin{aligned} r(Z) &= \max(r(X), r(Y) + 1) \\ &\leq \max(\lg n(X), \lg n(Y) + 1) \\ &\leq \max(\lg n(X), \lg(2n(Y))) \\ &\leq \lg \max(n(X), 2n(Y)) \end{aligned}$$

$n(X) \leq n(X) + n(Y)$ and $n(Y) \leq n(X) \Rightarrow 2n(Y) \leq n(X) + n(Y)$.

$$\implies r(Z) \leq \lg \max(n(X), 2n(Y)) \leq \lg(n(X) + n(Y)) = \lg n(Z)$$

Case 2: `union_by = rank`
`union_by = rank` $\implies r(Y) \leq r(X)$.

Case 2a: $r(Y) < r(X)$

$$\begin{aligned} & r(Z) = \max(r(X), 1 + r(Y)) = h(X) \\ & \leq \lg n(X) \leq \lg n(X) + n(Y) \leq \lg n(Z) \end{aligned}$$

Case 2b: $r(Y) = r(X)$

$$\begin{aligned} & r(Z) = \max(r(X), 1 + r(Y)) = 1 + r(Y) = 1 + r(X) \\ & \Rightarrow r(Z) \leq 1 + \lg n(Y) \wedge r(Z) \leq 1 + \lg n(X) \\ & \Rightarrow r(Z) \leq 1 + \min(\lg n(Y), \lg n(X)) \\ & \Rightarrow r(Z) \leq \lg(2 \min(n(X), n(Y))) \\ & \Rightarrow r(Z) \leq \lg(n(X) + n(Y)) = \lg n(Z) \end{aligned}$$

For both cases 1 and 2, $r(Z) \leq \lg n(Z)$. Therefore, `union` preserves the invariant $\forall T, r(T) \leq \lg n(T)$. $\square$

This means that any tree can have height at most $\lg n$. Therefore, `find` and `union` have a worst-case time complexity of $O(\lg n)$ and `link` has a worst-case time complexity of $O(1)$.

2.4 Lower bound on time when `compress_path` is `False`

When there is no path compression, we can lower bound the worst-case time complexity of `find`.

Consider these union operations:

```
for i in range(int(log2(n))):
    for j in range(0, n, 1 << (i+1)):
        union(j, j + (1 << i))
```

The body of the outer loop is called a round. There are $\lfloor \lg n \rfloor$ rounds.

Number of union operations:

$$\sum_{i=1}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor \le n \sum_{i=1}^{\lfloor \lg n \rfloor} \frac{1}{2^i} \le n\left(1 - 2^{\lfloor \lg n \rfloor}\right) \le n - 1$$

Theorem 3. *After i rounds, there are $\left\lfloor \frac{n}{2^i} \right\rfloor$ trees with height i and size 2^i.*

Proof by induction. Initially there are n trees of height 0 and size 1, so this is true for $i = 0$.

Assume the theorem is true for some i (induction hypothesis). Just before the $(i+1)^{\text{th}}$ round, there are $\left\lfloor \frac{n}{2^i} \right\rfloor$ trees of height i and size 2^i. We can pair them up (if there are odd number of trees, leave the last one unpaired). When we union them, we get $\left\lfloor \frac{n}{2^{i+1}} \right\rfloor$ trees with height $i + 1$ and size 2^{i+1} (this doesn't depend on the value of `union_by`).

Therefore, the theorem is true by mathematical induction. □

Theorem 4.

$$\left\lfloor \frac{n}{2^{\lfloor \lg n \rfloor}} \right\rfloor = 1$$

Therefore, after $\lfloor \lg n \rfloor$ rounds, there is one tree of height $\lfloor \lg n \rfloor$. Therefore, worst-case time complexity of `find` is $\Omega(\lg n)$.

2.5 Both union-by-rank and path-compression

2.5.1 Alt-Ackermann function

Definition 1. *For $j \ge 0$ and $k \ge 0$,*

$$A_k(j) = \begin{cases} j + 1 & k = 0 \\ A_{k-1}^{(j+1)}(j) & k \ge 1 \end{cases}$$

Here $A_k^{(0)}(j) = j$ and $A_k^{(i)}(j) = A_k(A_k^{(i-1)}(j))$.

Theorem 5. $A_k(0) = 1$

Theorem 6. $A_1(j) = 2j + 1$

Theorem 7. $A_2(j) = 2^{j+1}(j + 1) - 1$

Theorem 8. $A_3(1) = 2047$

Theorem 9. *$A_k(j)$ is a non-decreasing function of k and j.*

Theorem 10. *$A_4(1)$ is way too large.*

Proof.

$$
\begin{aligned}
& A_4(1) \\
& = A_3(A_3(1)) \\
& = A_3(2047) \\
& = A_2^{(2048)}(2047) \\
& \geq A_2^{(2)}(2047) \\
& = A_2(A_2(2047)) \\
& = A_2(2^{2048} \times 2048 - 1) \\
& = 2^{\left(2^{2059}-1\right)}\left(2^{2059}\right) - 1 \\
& > 2^{2^{2059}} \\
& > 16^{16^{514}}
\end{aligned}
$$

$\square$

Definition 2. $\alpha(n) = \min(\{k : A_k(1) \geq n\})$

Theorem 11. $p < \alpha(n) \leq q \iff A_p(1) < n \leq A_q(1)$

2.5.2 level and iter

Let F be a DSU forest with n nodes. For a node x, let $x.p$ be its parent and $x.rank$ be its rank.

Theorem 12. $x \neq x.p \implies x.rank < x.p.rank$

Theorem 13. $x.rank \leq \lfloor \lg n \rfloor \leq n - 1$

We can partition the set of nodes into 3 parts:

- root nodes: $\{x : x = x.p\}$.
- leaf nodes: $\{x : x.rank = 0\}$.
- internal nodes: non-root and non-leaf nodes.

level and iter are functions which map an internal node x to an integer.

Definition 3. $\text{level}(x) = \max(\{k : A_k(x.rank) \leq x.p.rank\})$

Theorem 14. $k \leq \text{level}(x) \iff A_k(x.rank) \leq x.p.rank$

Theorem 15. $0 \leq \text{level}(x) < \alpha(\lfloor \lg n \rfloor + 1) \leq \alpha(n)$

Definition 4. $\text{iter}(x) = \max(\{i : A^{(i)}_{\text{level}(x)}(x.rank) \leq x.p.rank\})$

Theorem 16. $i \leq \text{iter}(x) \iff A^{(i)}_{\text{level}(x)}(x.rank) \leq x.p.rank\})$

Theorem 17. $1 \leq \text{iter}(x) \leq x.rank$

2.5.3 Potential function

Definition 5. *For a node x, the potential function $\phi(x)$ is given by*

$$\phi(x) = \begin{cases} \alpha(n) \cdot x.rank & x \text{ is a root or leaf node} \\ (\alpha(n) - \text{level}(x)) \cdot x.rank - \text{iter}(x) & \text{otherwise} \end{cases}$$

Theorem 18. *x is an internal node $\implies 0 \leq \phi(x) < \alpha(n) \cdot x.rank$.*

To be continued ...