【DS】Day1(2)

Tags	
Date	@May 19, 2022
Summary	Quick-Find and Quick-Union

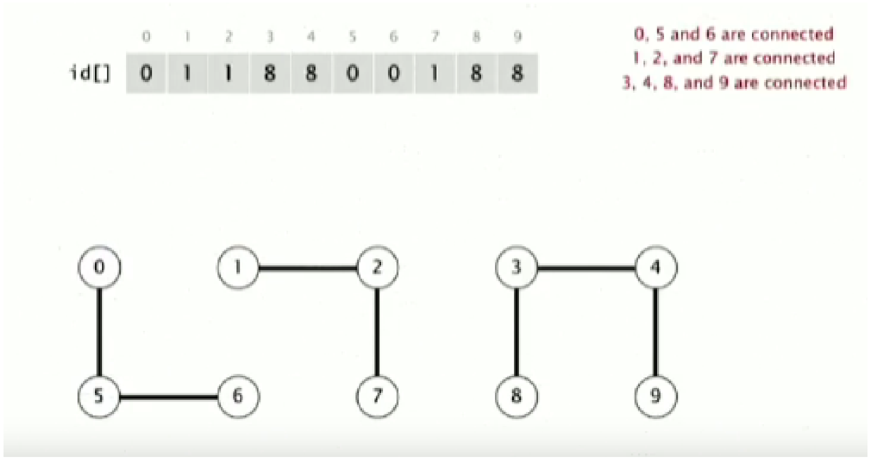
【Week1】Union-find

1.2 Quick Find

Data Structure

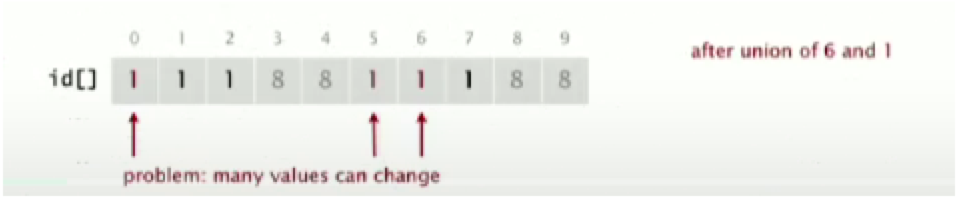
Integer array: `id[]` of size N.

Interpretation: p and q are connected iff they have the same id.



Find: Check if q and p have the same id.

Union: To merge components containing p and q, change all entries whose id equals id[p] to id[q].



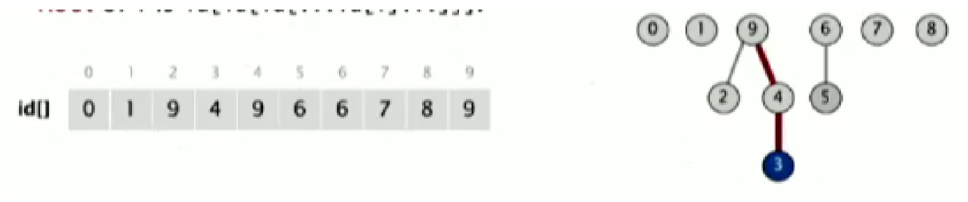
Java Implementation

```
public class QuickFindUF {
  private int[] id;

  // N为object的个数
  public QuickFindUF(int N) {
    id = new int[N];
    for(int i = 0; i < N; ++i)
      id[i] = i;
  }

  // Check whether p and q are in the same component
  public boolean connected(int p,  int q) {
    return id[p] == id[q];
  }

  public void union(int p, int q) {
    int pid = id[p];
    int qid = id[q];

    // Change all entries with id[p] to id[q]
    for(int i = 0; i < id.length; ++i) {
      if(id[i] == pid)
        id[i] = qid;
    }
  }
}
```

1.3 Quick Union

Data Structure

- Integer array: id[] of size N
- Interpretation: id[i] is parent of i
- Root of i: Is id[id[id[…id[i]…]]].



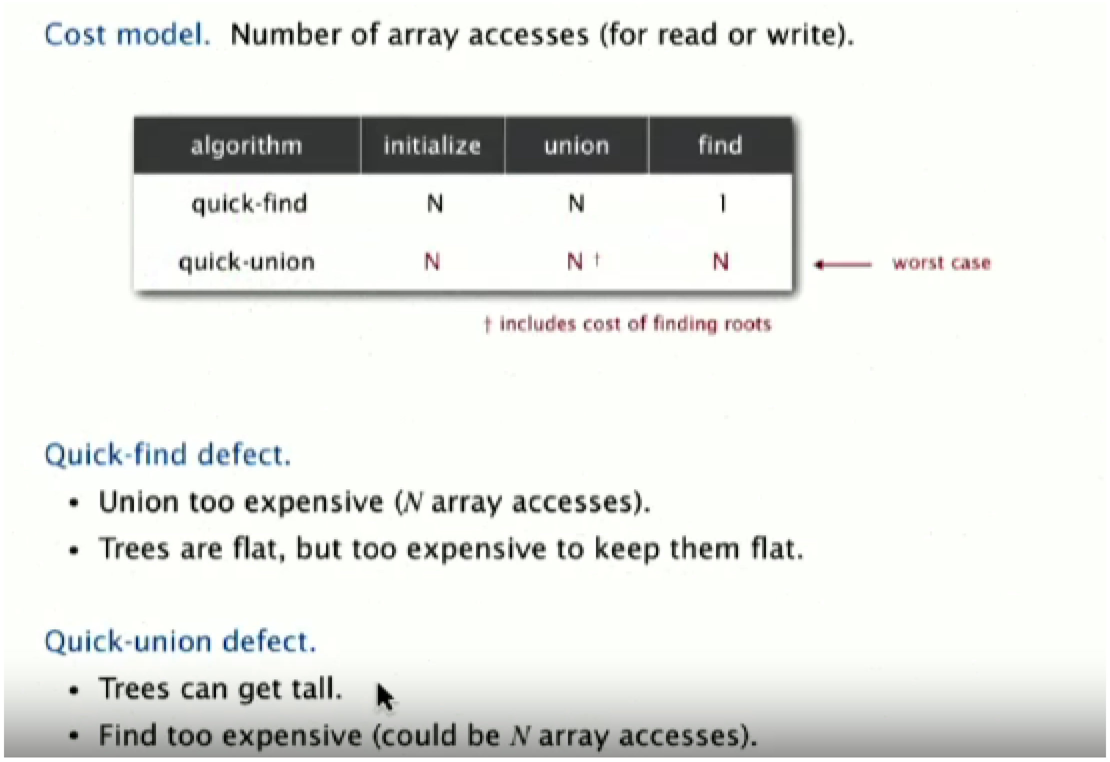
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	9	4	9	6	6	7	8	9

Find: Check if p and q have the same root.

Union: To merge components containing p and q, set the id of p's root to the id of q's root.

Java Implementation

```java
public class QuickUnionUF {
  private int[] id;

  public QuickUnionUF(int N) {
    id = new int[N];
    for(int i = 0; i < N; ++i) {
      id[i] = i;
  }

  private int root(int i) {
    while(i != id[i])
      i = id[i];
    return i;
  }

  // Check if p and q have the same roots
  public boolean connected(int p, int q) {
    return root(p) == root(q);
  }

  // Change root of p to point to root of q
  public void union(int p, int q) {
    id[root(p)] = root(q);
  }
}
```



Cost model. Number of array accesses (for read or write).

algorithm	initialize	union	find
quick-find	N	N	1
quick-union	N	N †	N

← worst case

† includes cost of finding roots

Quick-find defect.

- Union too expensive (N array accesses).
- Trees are flat, but too expensive to keep them flat.

Quick-union defect.

- Trees can get tall.
- Find too expensive (could be N array accesses).