【DS】Day1

Tags	
Date	@May 19, 2022
Summary	

【Week1】Union-Find

Steps to developing a usable algorithm:

- Model the problem
- Find an algorithm to solve it
- Fast enough? Fit in memory?
- If not, figure out why.
- Find a way to address the problem
- Iterate until satisfied

1.1 Dynamic Connectivity

Given a set of N objects.

- Union command: Connect two objects
- Find/connected query: Is there a path connecting the two objects?

```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
connected(0, 7)  ×
connected(8, 9)  ✔
```

Modelling the Objects

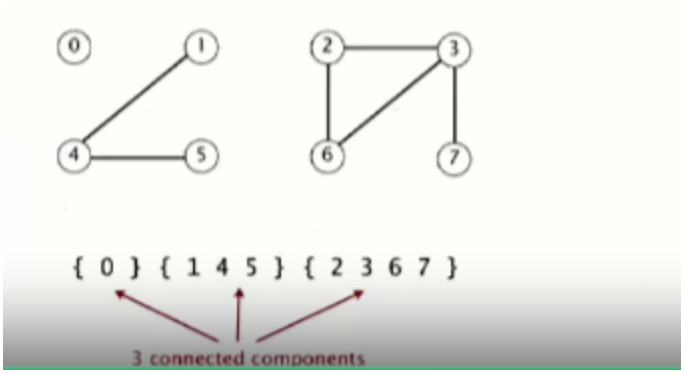
- Computers in the network
- Friends
- Pixels on a photo

Modelling the connections

We assume “is connected to” is an equivalence relation:

- Reflexive: p is connected to p
- Symmetric: If p is connected to q, then q is connected to p
- Transitive: If p is connected to q and q is connected to r, then p is connected to r.

Connected Components: Maximal set of objects that are mutually connected



Implementing the Operations

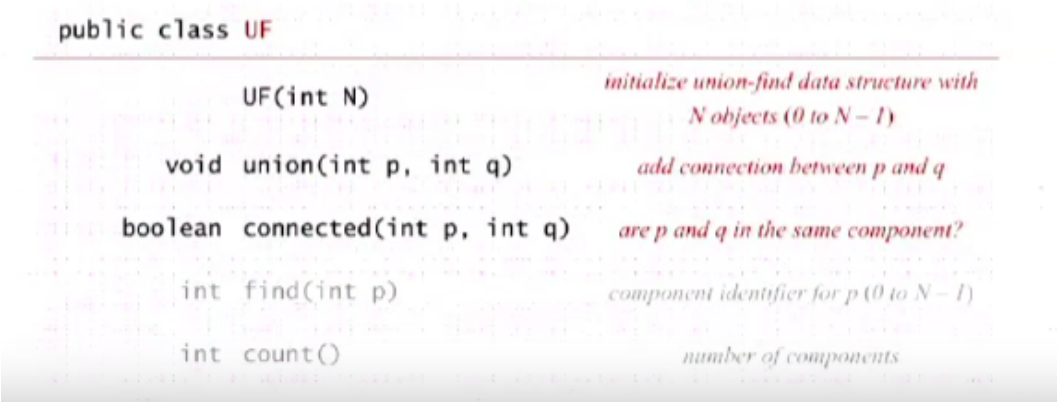
Find query: Check if two objects are in the same component

Union command: Replace components containing two objects with their union.

Union-find Data Type(API)

Goal: Design efficient data structure for union-find

- Number of objects N can be huge
- Number of operations M can be huge
- Find queries and union commands may be intermixed



Test client: test if each operation does what we expect

```
public static void main(String[] args) {
  int N = StdIn.readInt();
  UF uf = new UF(N);

  // 当StdIn输入不为空时，进行判断
  while (!StdIn.isEmpty()) {
    int p = StdIn.readInt();
    int q = StdIn.readInt();

    // 检查数组p和q在uf中是否相连
    // 若已经相连，不做操作。若未相连，将他们连起来
    if (!uf.connected(p, q)) {
      uf.union(p, q);
      StdOut.println(p + " " + q);
```

```
    }
  }
]
```