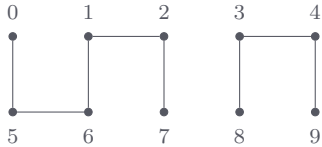


Union Find

Outline

- 1 The Dynamic Connectivity Problem
- 2 Union Find (UF)
- 3 Quick Find UF
- 4 Quick Union UF
- 5 Weighted Quick Union UF

The Dynamic Connectivity Problem



Union Find (UF)

☰ *UF*

`int find(int p)`

returns the canonical site of the component containing site p

`int count()`

returns the number of components

`boolean connected(int p, int q)`

returns `true` if sites p and q belong to the same component, and `false` otherwise

`void union(int p, int q)`

connects sites p and q

Union Find (UF)

Program: `Components.java`

- ↪ Standard input: n (int) and a sequence of pairs of integers representing sites
- ↪ Standard output: number of components left after merging the sites that are in different components

```
>_ ~/workspace/dsa/programs
```

```
$ cat ../data/tinyUF.txt
```

```
10
```

```
4 3
```

```
3 8
```

```
6 5
```

```
9 4
```

```
2 1
```

```
8 9
```

```
5 0
```

```
7 2
```

```
6 1
```

```
1 0
```

```
6 7
```

```
$ java Components < ../data/tinyUF.txt
```

```
2 components
```

```
$ _
```

Union Find (UF)

Components.java

```
1 import dsa.WeightedQuickUnionUF;
2 import stdlib.StdIn;
3 import stdlib.StdOut;
4
5 public class Components {
6     public static void main(String[] args) {
7         int n = StdIn.readInt();
8         WeightedQuickUnionUF uf = new WeightedQuickUnionUF(n);
9         while (!StdIn.isEmpty()) {
10             int p = StdIn.readInt();
11             int q = StdIn.readInt();
12             uf.union(p, q);
13         }
14         StdOut.println(uf.count() + " components");
15     }
16 }
```

Quick Find UF

Instance variables:

~→ An array of component identifiers: `int[] id`

~→ Number of components: `int count`

Quick Find UF

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

Quick Find UF

QuickFindUF.java

```
1 package dsa;
2
3 import stdlib.StdIn;
4 import stdlib.StdOut;
5
6 public class QuickFindUF implements UF {
7     private int[] id;
8     private int count;
9
10    public QuickFindUF(int n) {
11        id = new int[n];
12        for (int i = 0; i < n; i++) {
13            id[i] = i;
14        }
15        count = n;
16    }
17
18    public int find(int p) {
19        return id[p];
20    }
21
22    public int count() {
23        return count;
24    }
25
26    public boolean connected(int p, int q) {
27        return find(p) == find(q);
28    }
29
30    public void union(int p, int q) {
31        int pID = find(p);
32        int qID = find(q);
33        if (pID == qID) {
34            return;
35        }
```

Quick Find UF

QuickFindUF.java

```
36     for (int i = 0; i < id.length; i++) {
37         if (id[i] == pID) {
38             id[i] = qID;
39         }
40     }
41     count--;
42 }
43
44 public static void main(String[] args) {
45     int n = StdIn.readInt();
46     QuickFindUF uf = new QuickFindUF(n);
47     while (!StdIn.isEmpty()) {
48         int p = StdIn.readInt();
49         int q = StdIn.readInt();
50         if (uf.connected(p, q)) {
51             continue;
52         }
53         uf.union(p, q);
54         StdOut.println(p + " " + q);
55     }
56     StdOut.println(uf.count() + " components");
57 }
58 }
```

Quick Find UF

Operation	$T(n)$
QuickFindUF(int n)	n
int find(int p)	1
int count()	1
boolean connected(int p, int q)	1
void union(int p, int q)	n

Quick Union UF

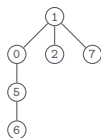
Instance variables:

~> An array of parent identifiers: `int[] parent`

~> Number of components: `int count`

Quick Union UF

		parent[]									
p	q	0	1	2	3	4	5	6	7	8	9
6	7	1	1	1	8	3	0	5	1	8	8



Quick Union UF

QuickUnionUF.java

```
1 package dsa;
2
3 import stdlib.StdIn;
4 import stdlib.StdOut;
5
6 public class QuickUnionUF implements UF {
7     private int[] parent;
8     private int count;
9
10    public QuickUnionUF(int n) {
11        parent = new int[n];
12        for (int i = 0; i < n; i++) {
13            parent[i] = i;
14        }
15        count = n;
16    }
17
18    public int find(int p) {
19        while (p != parent[p]) {
20            p = parent[p];
21        }
22        return p;
23    }
24
25    public int count() {
26        return count;
27    }
28
29    public boolean connected(int p, int q) {
30        return find(p) == find(q);
31    }
32
33    public void union(int p, int q) {
34        int rootP = find(p);
35        int rootQ = find(q);
```

Quick Union UF

QuickUnionUF.java

```
36     for (int i = 0; i < id.length; i++) {
37         if (id[i] == pID) {
38             id[i] = qID;
39         }
40     }
41     count--;
42 }
43
44 public static void main(String[] args) {
45     int n = StdIn.readInt();
46     QuickFindUF uf = new QuickFindUF(n);
47     while (!StdIn.isEmpty()) {
48         int p = StdIn.readInt();
49         int q = StdIn.readInt();
50         if (uf.connected(p, q)) {
51             continue;
52         }
53         uf.union(p, q);
54         StdOut.println(p + " " + q);
55     }
56     StdOut.println(uf.count() + " components");
57 }
58 }
```

Quick Union UF

Operation	$T(n)$
<code>QuickUnionUF(int n)</code>	n
<code>int find(int p)</code>	tree height
<code>int count()</code>	1
<code>boolean connected(int p, int q)</code>	tree height
<code>void union(int p, int q)</code>	tree height

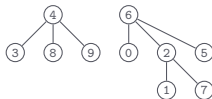
Weighted Quick Union UF

Instance variables:

- ↪ An array of parent identifiers: `int[] parent`
- ↪ An array of component sizes: `int[] size`
- ↪ Number of components: `int count`

Weighted Quick Union UF

p	q	parent[], size[]									
		0	1	2	3	4	5	6	7	8	9
6	7	6	2	6	4	4	6	6	2	4	4
		1	1	3	1	4	1	6	1	1	1



Weighted Quick Union UF

WeightedQuickUnionUF.java

```
1 package dsa;
2
3 import stdlib.StdIn;
4 import stdlib.StdOut;
5
6 public class WeightedQuickUnionUF implements UF {
7     private int[] parent;
8     private int[] size;
9     private int count;
10
11     public WeightedQuickUnionUF(int n) {
12         parent = new int[n];
13         size = new int[n];
14         for (int i = 0; i < n; i++) {
15             parent[i] = i;
16             size[i] = 1;
17         }
18         count = n;
19     }
20
21     public int find(int p) {
22         while (p != parent[p]) {
23             p = parent[p];
24         }
25         return p;
26     }
27
28     public int count() {
29         return count;
30     }
31
32     public boolean connected(int p, int q) {
33         return find(p) == find(q);
34     }
35 }
```

Weighted Quick Union UF

WeightedQuickUnionUF.java

```
36 public void union(int p, int q) {
37     int rootP = find(p);
38     int rootQ = find(q);
39     if (rootP == rootQ) {
40         return;
41     }
42     if (size[rootP] < size[rootQ]) {
43         parent[rootP] = rootQ;
44         size[rootQ] += size[rootP];
45     } else {
46         parent[rootQ] = rootP;
47         size[rootP] += size[rootQ];
48     }
49     count--;
50 }
51
52 public static void main(String[] args) {
53     int n = StdIn.readInt();
54     WeightedQuickUnionUF uf = new WeightedQuickUnionUF(n);
55     while (!StdIn.isEmpty()) {
56         int p = StdIn.readInt();
57         int q = StdIn.readInt();
58         if (uf.connected(p, q)) {
59             continue;
60         }
61         uf.union(p, q);
62         StdOut.println(p + " " + q);
63     }
64     StdOut.println(uf.count() + " components");
65 }
66 }
```

Weighted Quick Union UF

Operation	$T(n)$
<code>WeightedQuickUnionUF(int n)</code>	n
<code>int find(int p)</code>	$\log n$
<code>int count()</code>	1
<code>boolean connected(int p, int q)</code>	$\log n$
<code>void union(int p, int q)</code>	$\log n$