

# مسئله‌ی همبندی پویا

سید ناصر رضوی [n.razavi@tabrizu.ac.ir](mailto:n.razavi@tabrizu.ac.ir)

۱۳۹۵

# مراحل توسعه‌ی یک الگوریتم

۲

□ مراحل توسعه‌ی یک الگوریتم.

□ مدل‌سازی مسئله

□ یافتن یک الگوریتم برای حل آن

□ تحلیل زمان و حافظه

■ آیا الگوریتم به اندازه‌ی کافی سریع است؟

■ آیا حافظه به اندازه‌ی کافی موجود است؟

□ اگر این گونه نیست، علت را بررسی کن

□ روشی برای حل مشکل پیدا کن

□ مراحل فوق را تا رسیدن به یک الگوریتم سریع و کم مصرف تکرار کن

□ روش علمی.

□ تحلیل ریاضی.

# مسئله‌ی همبندی پویا

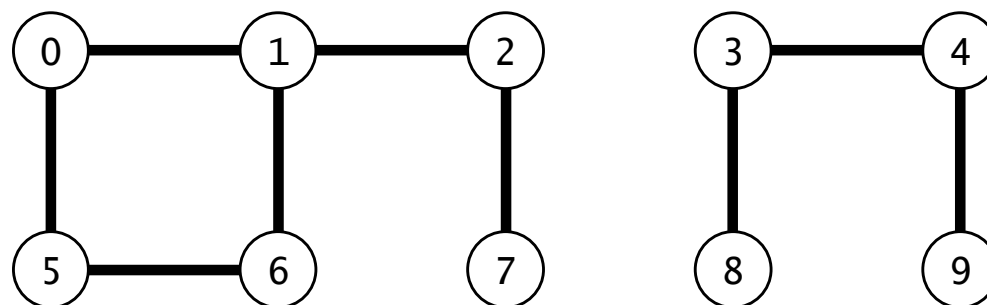
۳

□ مجموعه‌ای از  $N$  شی داده شده است.

□ دستور اجتماع: دو شی را به هم وصل کن.

□ بررسی متصل بودن (همبندی): آیا مسیری وجود دارد که دو شی را به هم وصل کند؟

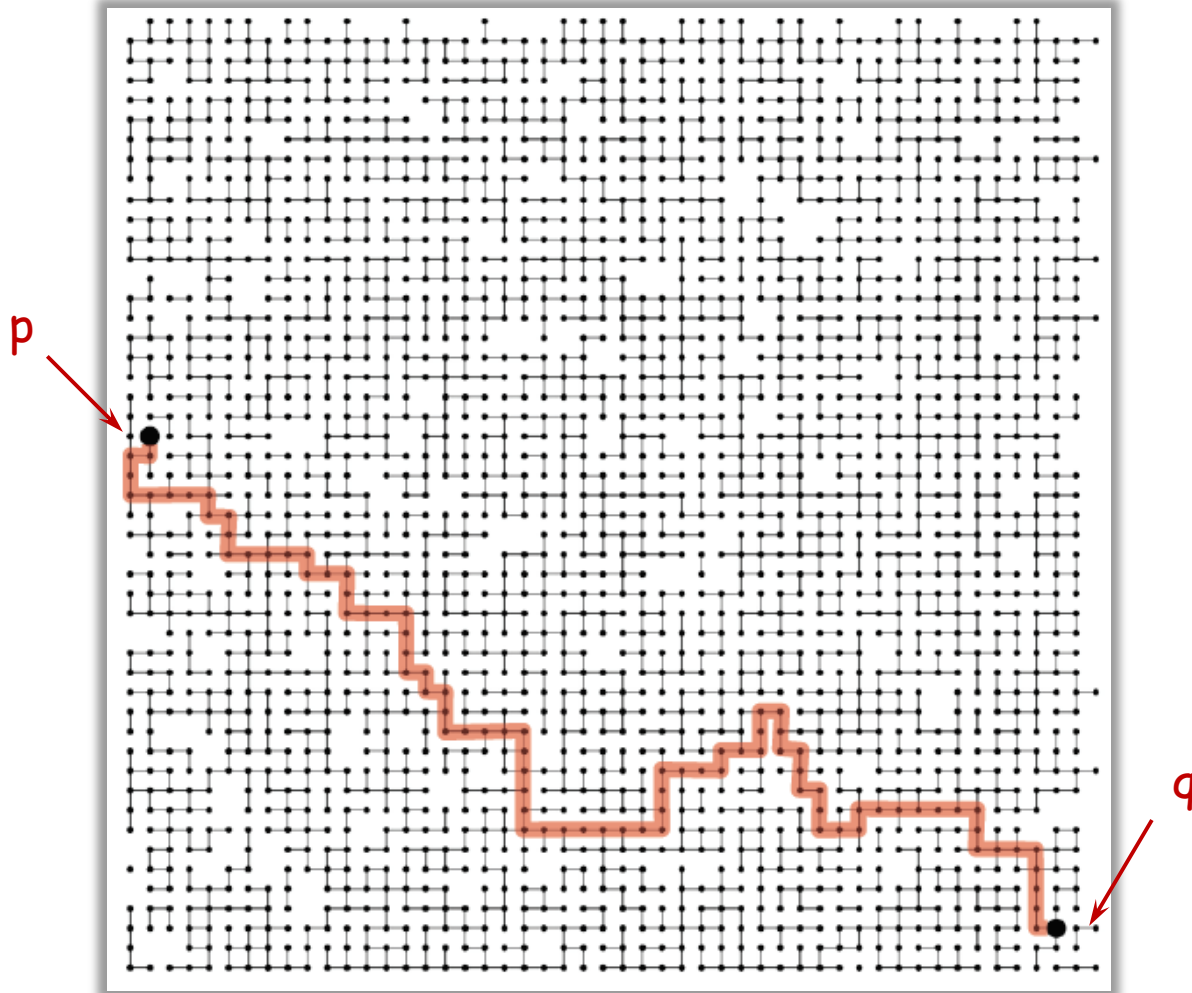
```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
connected(0, 7) ✗
connected(8, 9) ✓
union(5, 0)
union(7, 2)
union(6, 1)
connected(0, 7) ✓
union(1, 0)
```



# مثال همبندی

۴

□ س. آیا بین  $p$  و  $q$  مسیری وجود دارد؟



# مدل سازی اشیا

□ کاربردها شامل کار با انواع مختلفی از اشیا است.

□ پیکسل‌ها در یک تصویر دیجیتال

□ کامپیوترها در یک شبکه

□ دوستان در یک شبکه‌ی اجتماعی

□ ترانزیستورها در یک تراشه‌ی کامپیوتری

□ عناصر در یک مجموعه‌ی ریاضی

□ نام متغیرها در یک برنامه‌ی فرترن

□ در برنامه‌نویسی، اشیا را با شماره‌های صفر تا  $N - 1$  نامگذاری می‌کنیم.

□ استفاده از اعداد صحیح برای دسترسی به خانه‌های یک آرایه.

□ حذف جزییاتی که به این مسئله مربوط نیستند.

# مدل سازی اتصال‌ها

۶

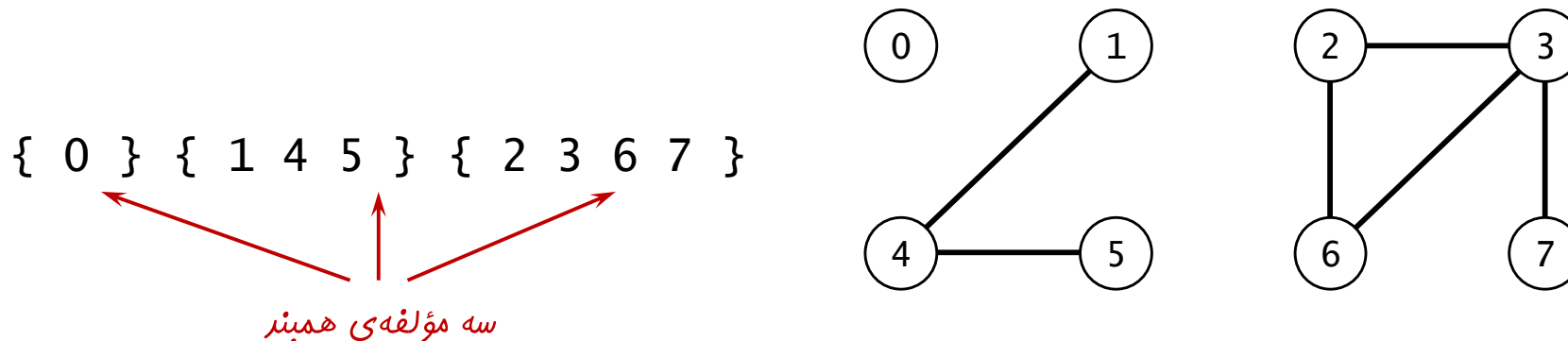
□ رابطه‌ی «متصل بودن» یک رابطه‌ی هم‌ارزی است:

□ بازتابی:  $p$  به  $p$  متصل است.

□ تقارنی: اگر  $p$  به  $q$  متصل است، آنگاه  $q$  به  $p$  متصل است.

□ تراگذری: اگر  $p$  به  $q$  و  $q$  به  $r$  متصل است، آنگاه  $p$  به  $r$  متصل است.

□ مؤلفه‌های متصل (همبند). یک مجموعه‌ی بیشینه از اشیا که دو به دو متصل هستند.

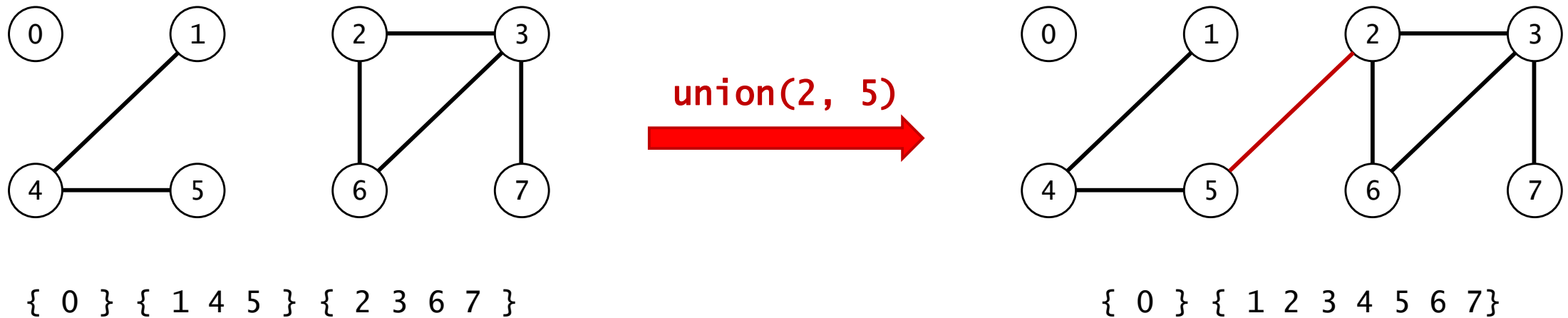


# پیاده‌سازی عملیات

۷

□ بررسی همبندی. آیا دو شی داده شده در یک مؤلفه قرار دارند؟

□ عمل اجتماع. مؤلفه‌های شامل دو شی داده شده را با اجتماع آنها جایگزین کن.



# ساختمان داده‌ی زیرمجموعه‌های مجزا

۸

□ هدف. طراحی یک ساختمان داده‌ی کارا برای زیرمجموعه‌های مجزا.

```
public class UF
```

```
    UF(int N)
```

ایجاد N زیرمجموعه‌ی مجزا با اشیای 0 تا 1 - N

```
    void union(int p, int q)
```

اجتماع زیرمجموعه‌های شامل p و q

```
    boolean connected(int p, int q)
```

آیا p و q در یک زیرمجموعه قرار دارند؟

```
    int find(int p)
```

یافتن زیرمجموعه‌ی شامل p

```
    int count()
```

برگرداندن تعداد زیرمجموعه‌ها



# آزمایش درستی پیاده‌سازی: برنامه‌ی مشتری

۹

□ تعداد اشیا را از ورودی بخوان.

□ مراحل زیر را تکرار کن:

□ از ورودی دو عدد صحیح بخوان.

□ اگر این دو هنوز به هم وصل نیستند، آنها را متصل و در خروجی چاپ کن.

```
public static void main(String[] args)
{
    int N = StdIn.readInt();
    UF uf = new UF(N);
    while (!StdIn.isEmpty())
    {
        int p = StdIn.readInt();
        int q = StdIn.readInt();
        if (!uf.connected(p, q))
        {
            uf.union(p, q);
            StdOut.println(p + " " + q);
        }
    }
}
```

```
% more 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
```

پیاده سازی: quick-find

# ساختمان داده برای quick-find

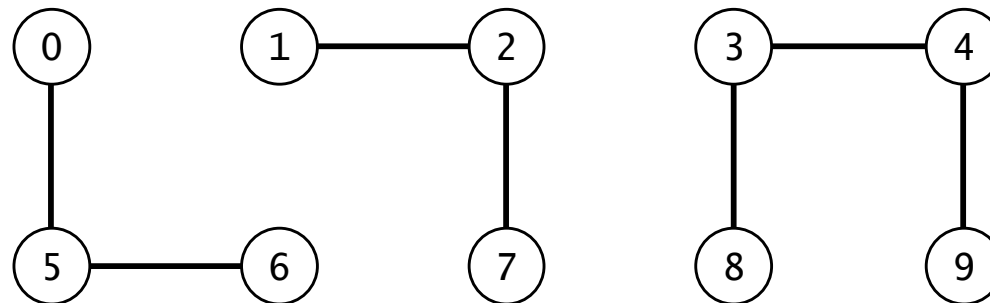
۱۱

□ ساختمان داده.

□ آرایه‌ی  $id[]$  شامل  $N$  عدد صحیح.

□ تفسیر:  $p$  و  $q$  متصل هستند اگر و فقط اگر دارای  $id$  یکسانی باشند.

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



# ساختمان داده برای quick-find

۱۲

□ ساختمان داده.

□ آرایه‌ی `id[]` شامل  $N$  عدد صحیح.

□ تفسیر:  $p$  و  $q$  متصل هستند اگر و فقط اگر دارای `id` یکسانی باشند.

	0	1	2	3	4	5	6	7	8	9
<code>id[]</code>	0	1	1	8	8	0	0	1	8	8

□ `find`. بررسی این که  $p$  و  $q$  دارای `id` یکسانی هستند.

`id[6] = 0;`    `id[1] = 1`

بنابراین ۶ و ۱ متصل نیستند

# ساختمان داده برای quick-find

۱۳

□ ساختمان داده.

□ آرایه‌ی  $id[]$  شامل  $N$  عدد صحیح.

□ تفسیر:  $p$  و  $q$  متصل هستند اگر و فقط اگر دارای  $id$  یکسانی باشند.

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

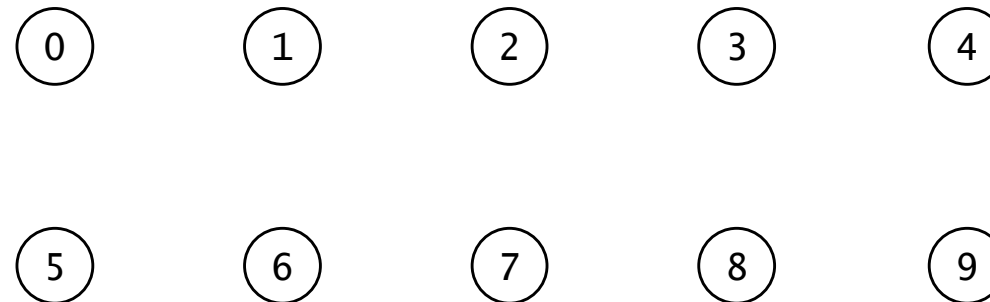
□ **union**. برای ادغام مولفه‌های شامل  $p$  و  $q$  تمام عناصری را که  $id$  آنها برابر با  $id[p]$  است، به  $id[q]$  تغییر بده.

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

پس از اجتماع ۶ و ۱

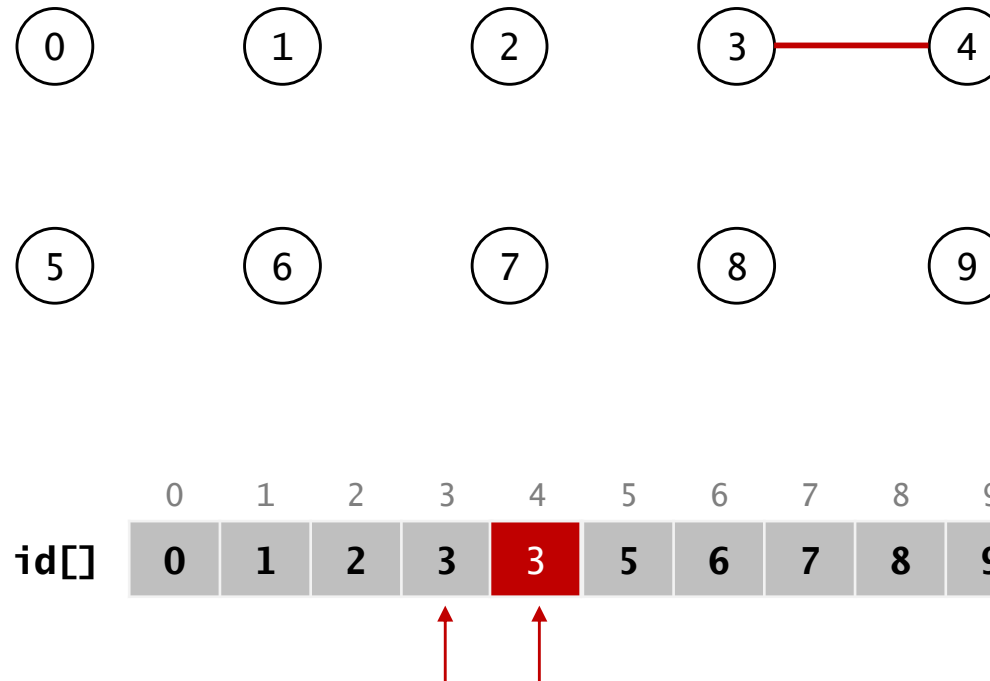
# اجرای نمایشی

۱۴

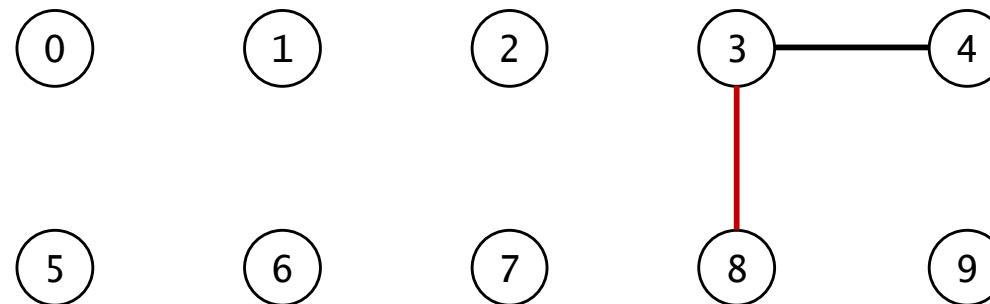


	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	3	4	5	6	7	8	9

`union(4, 3)`



`union(3, 8)`

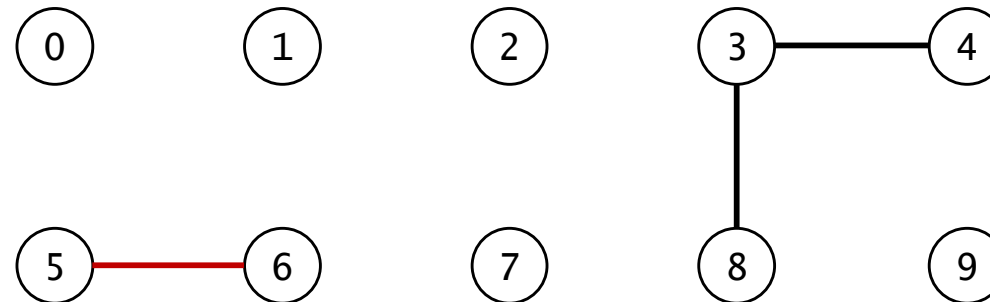


	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	8	8	5	6	7	8	9

Two red arrows point upwards to the cells containing '8' at index 3 and index 8 in the id[] array.



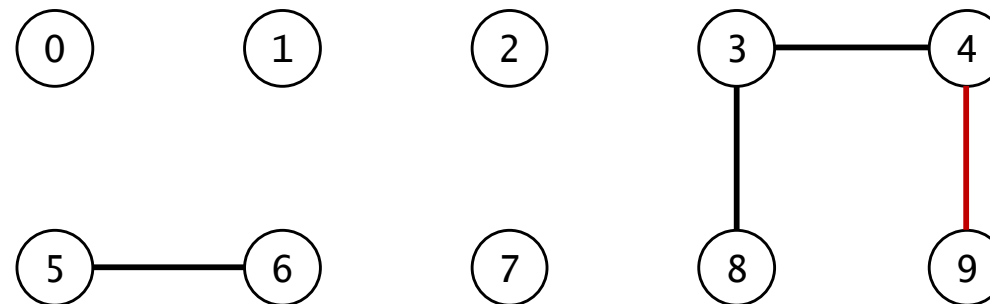
`union(6, 5)`



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

Two red arrows point to the values 5 in the id[] array at indices 5 and 6.

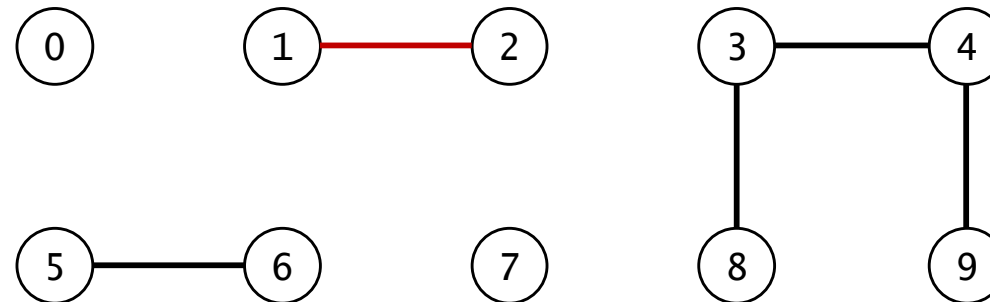
`union(9, 4)`



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

Two red arrows point upwards to the values 8 at index 4 and index 9 in the id[] array.

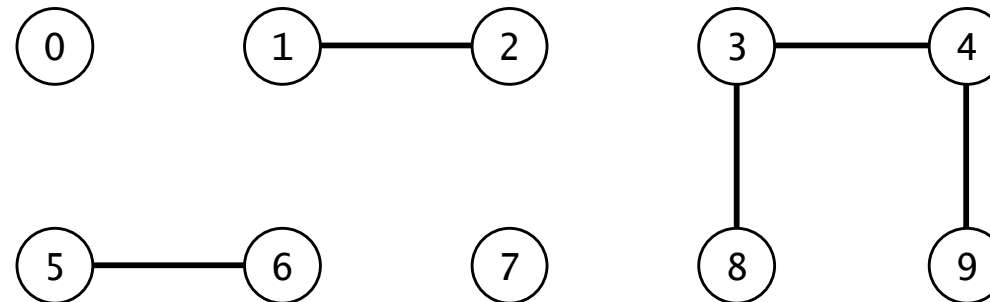
`union(2, 1)`



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

Two red arrows point upwards to the values 1 in the id[] array at indices 1 and 2.

`union(8, 9)`



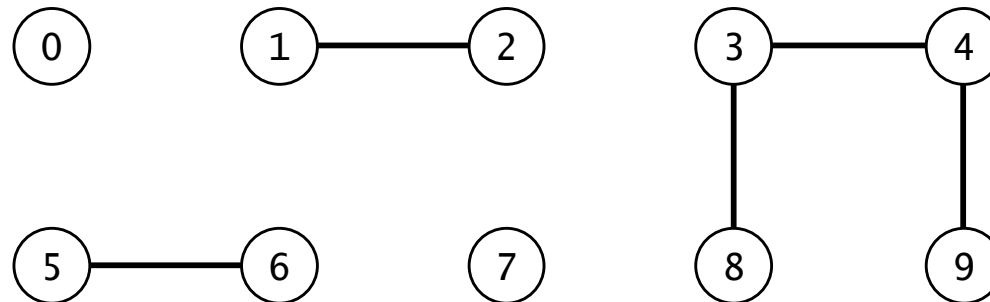
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	1	8	8	5	5	7	8	8

↑ ↑  
متصل هستند

# اجرای نمایشی

21

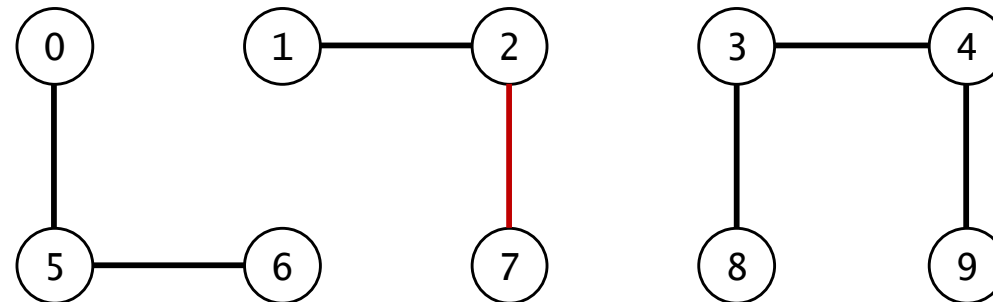
**connected(5, 0)** فیر



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

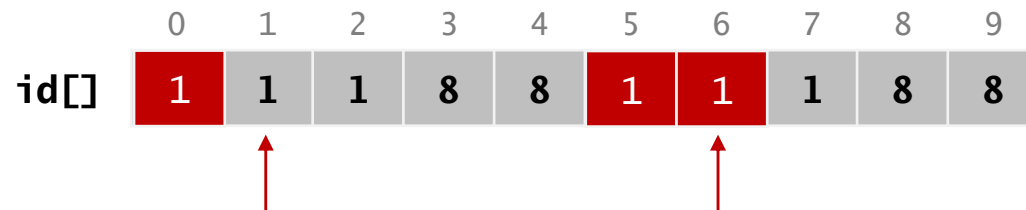
تحليل و طراحی الگوریتم‌ها - سید ناصر رضوی - ۱۳۹۵

`union(7, 2)`

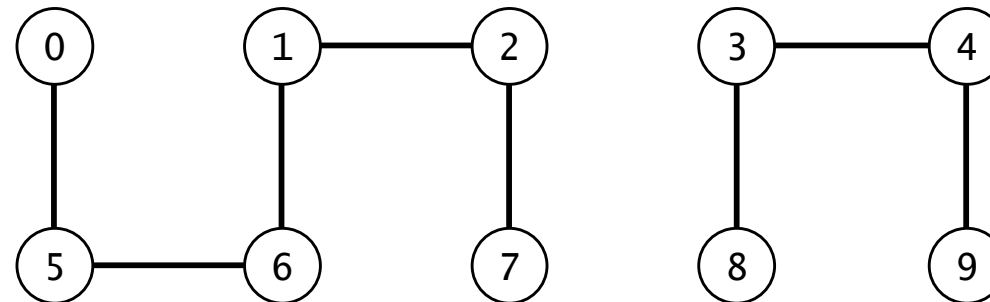


	0	1	2	3	4	5	6	7	8	9
id[]	0	1	1	8	8	0	0	1	8	8

Red arrows point to the values 1 at index 2 and index 7 in the id[] array.







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

# پیاده‌سازی جاوا: Quick-Find

۲۶

```
public class QuickFindUF {  
    private int[] id;  
    public QuickFindUF(int N) {  
        id = new int[N];  
        for (int i = 0; i < N; i++)  
            id[i] = i;  
    }  
    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];  
        if (pid == qid) return;  
        for (int i = 0; i < id.length; i++)  
            if (id[i] == pid) id[i] = qid;  
    }  
}
```

id هر شی را برابر با خودش قرار می‌دهد  
(N دستیابی به آرایه)

بررسی این که p و q در یک مولفه قرار دارند  
(۲ دستیابی به آرایه)

تمام عناصر با id[p] را به id[q] تغییر بده  
(حد اکثر  $2N+2$  دستیابی به آرایه)

# پیاده‌سازی quick-find کند است

۲۷

□ مدل هزینه.

یافتن	اجتماع	مقداردهی اولیه	الگوریتم
1	N	N	quick-find

□ ایراد quick-find. عمل اجتماع خیلی کند است.

□ مثال. برای انجام دنباله‌ای از N عمل اجتماع بر روی N شی، به  $N^2$  دسترسی به آرایه نیاز است.

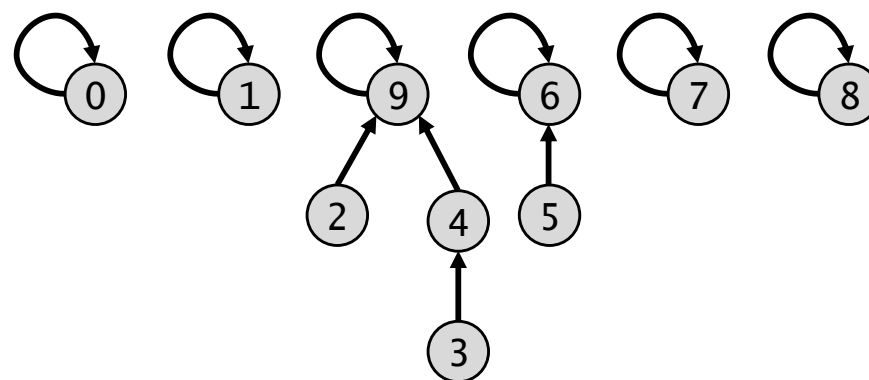
پیادهسازی quick-union

□ ساختمان داده.

□ آرایه‌ی  $id[]$  با اندازه‌ی  $N$

□ تفسیر:  $id[i]$  بیانگر پدر شی  $i$  است.

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



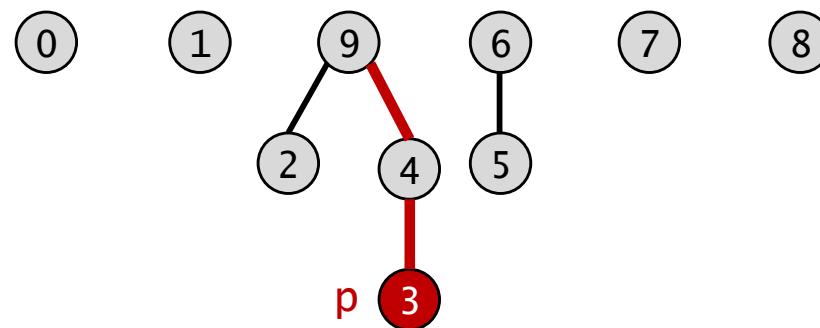
□ عمل  $\text{find}(p)$ .

□ برگرداندن ریشه‌ی درختی که شی  $p$  در آن قرار دارد.

$\text{find}(3)$

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

```
public int find(int p)
{
    while (p != id[p])
        p = id[p];
    return p;
}
```



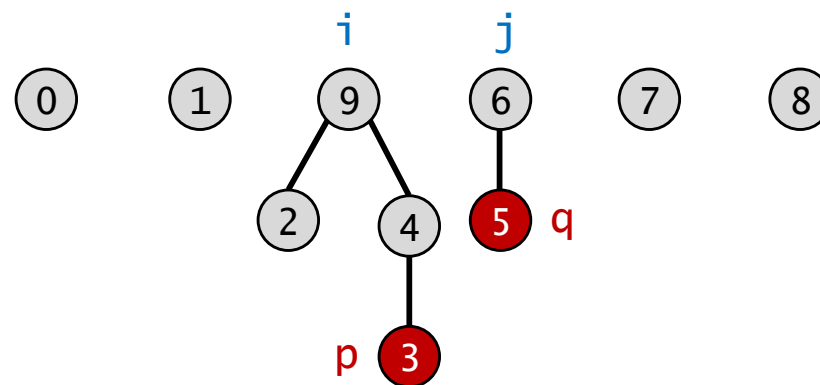
□ عمل  $\text{union}(p, q)$ .

□ اجتماع زیرمجموعه‌های شامل اشیای  $p$  و  $q$

$\text{union}(3, 5)$

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

```
public void union(int p, int q)
{
    int i = find(p);
    int j = find(q);
    if (i == j) return;
    id[i] = j;
}
```



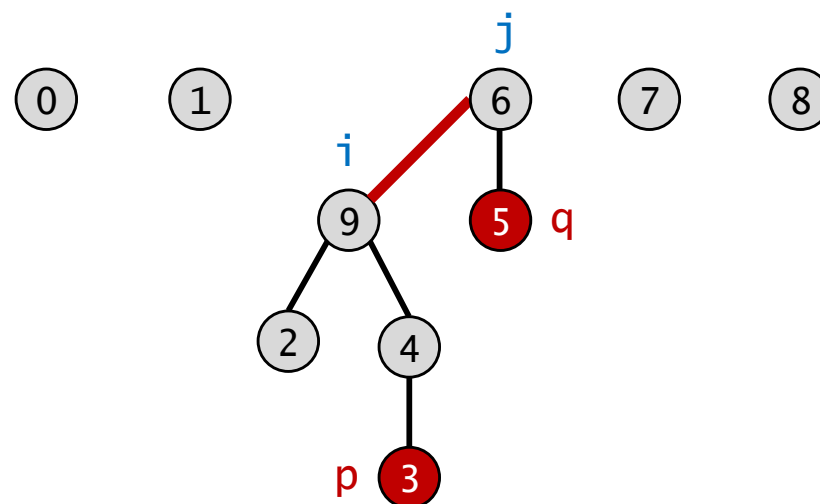
□ عمل  $\text{union}(p, q)$ .

□ اجتماع زیرمجموعه‌های شامل اشیای  $p$  و  $q$

$\text{union}(3, 5)$

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

```
public void union(int p, int q)
{
    int i = find(p);
    int j = find(q);
    if (i == j) return;
    id[i] = j;
}
```





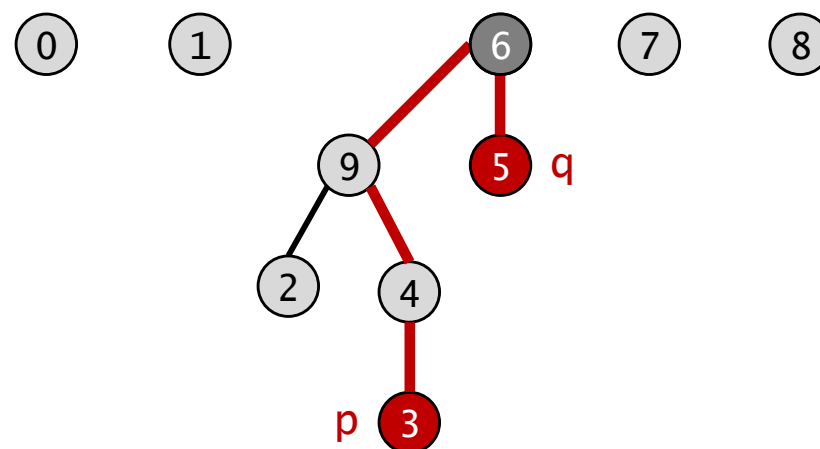
□ عمل  $\text{connected}(p, q)$ .

□ آیا اشیای  $p$  و  $q$  در یک زیرمجموعه قرار دارند؟

$\text{connected}(3, 5)$

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

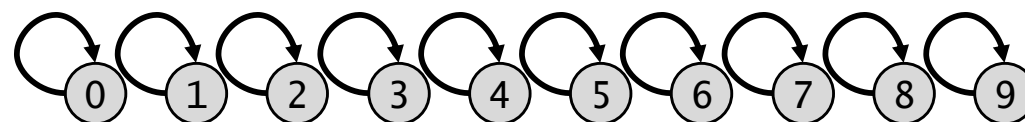
```
public boolean connected(int p, int q)
{
    return find(p) == find(q);
}
```



□ مقداردهی اولیه

□ ایجاد N زیرمجموعه‌ی مجزا.

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



{0} {1} {2} {3} {4} {5} {6} {7} {8} {9}

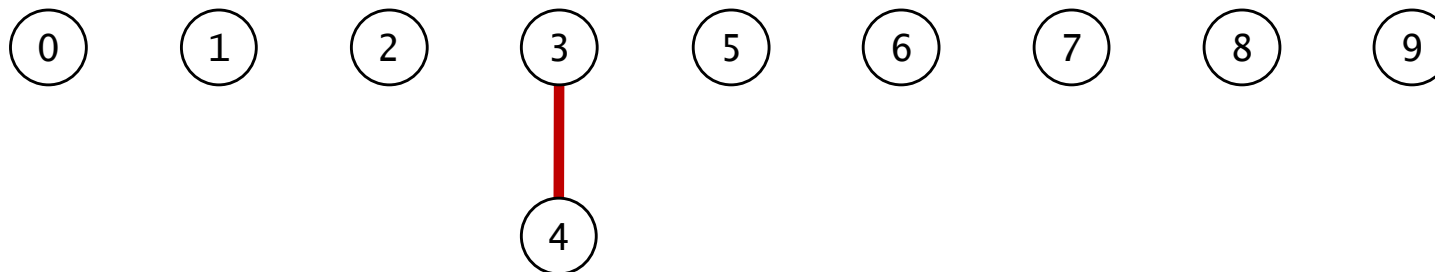
```
public UF(int N)
{
    id = new int[N];
    for (int i = 0; i < N; i++)
        id[i] = i;
}
```

**union(4, 3)**



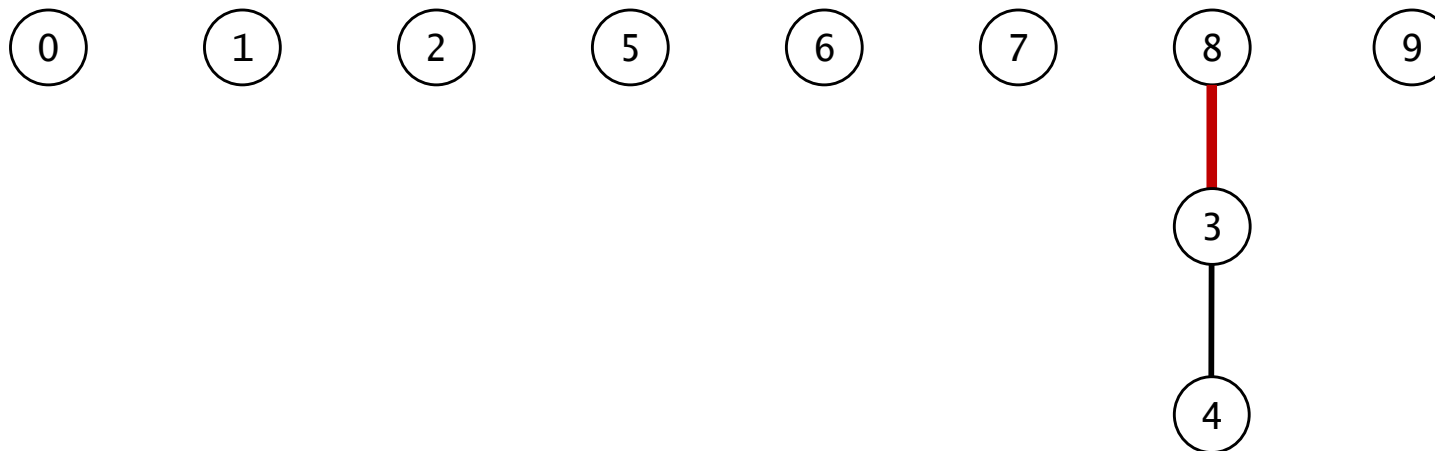
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	3	4	5	6	7	8	9

**union(3, 8)**



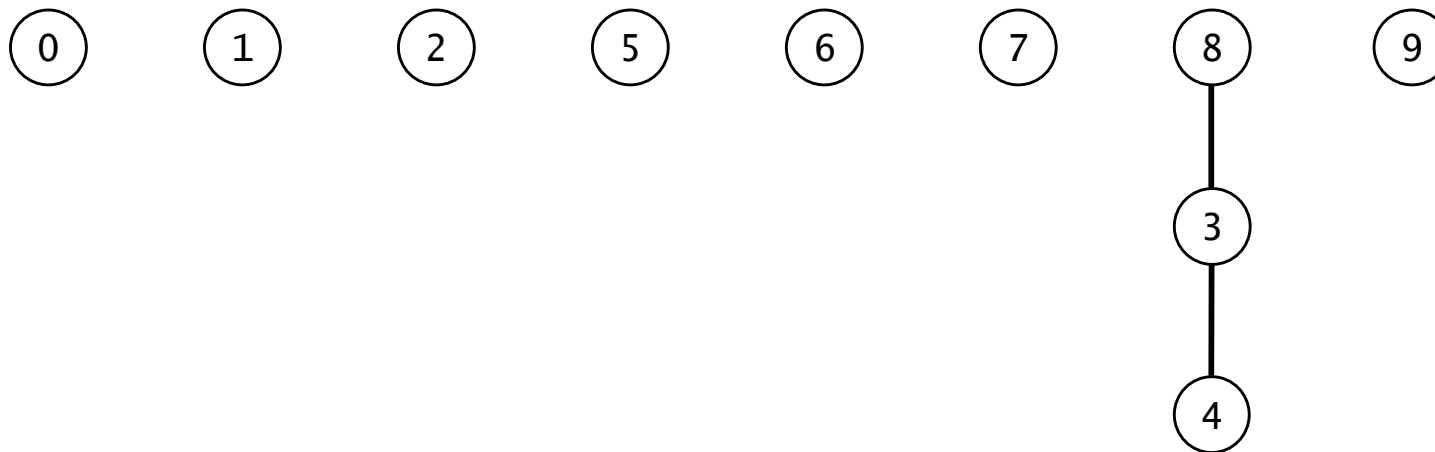
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	3	3	5	6	7	8	9

**union(3, 8)**



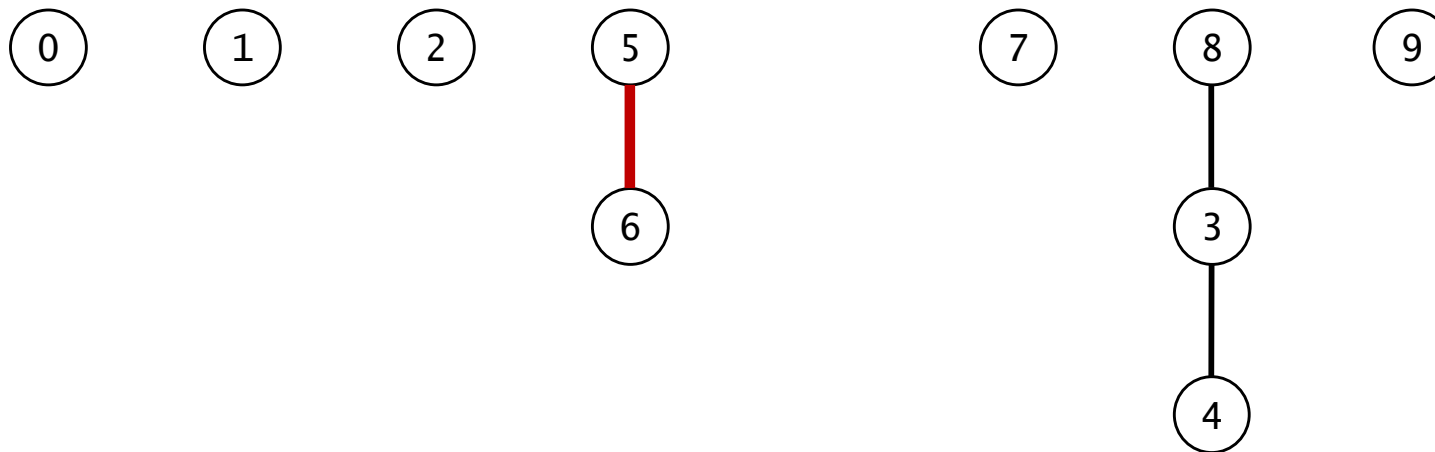
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	8	3	5	6	7	8	9

**union(6, 5)**



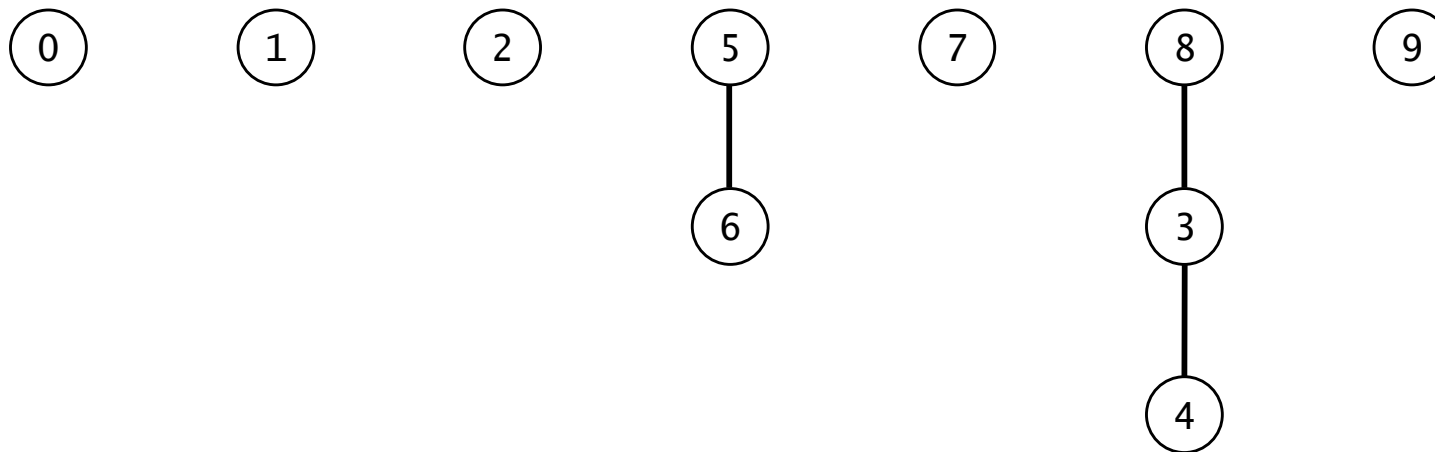
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	8	3	5	6	7	8	9

**union(6, 5)**



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

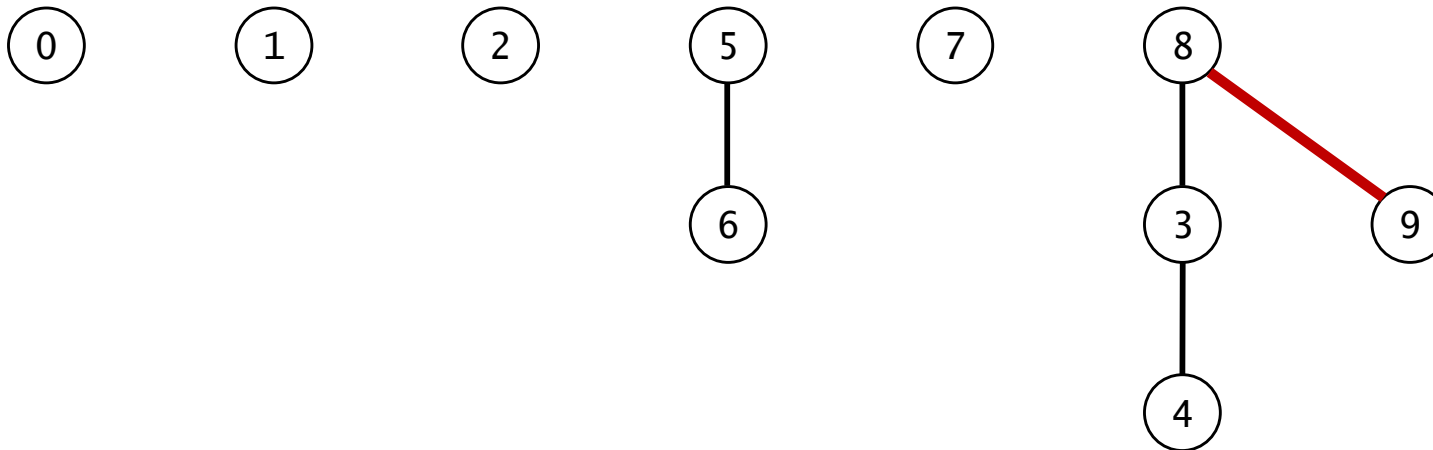
**union(9, 4)**



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

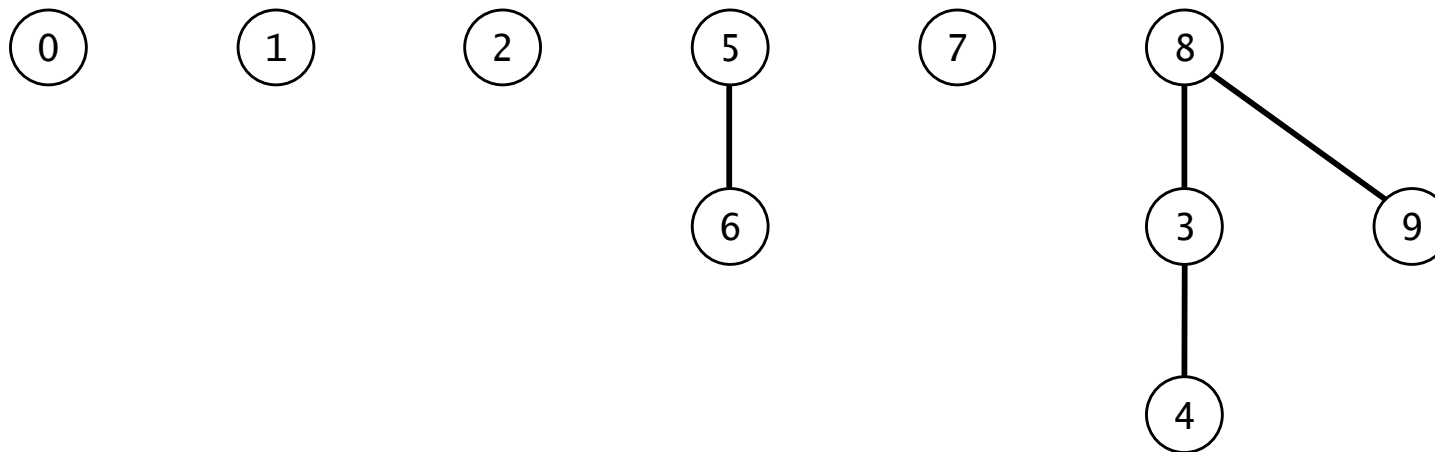


**union(9, 4)**



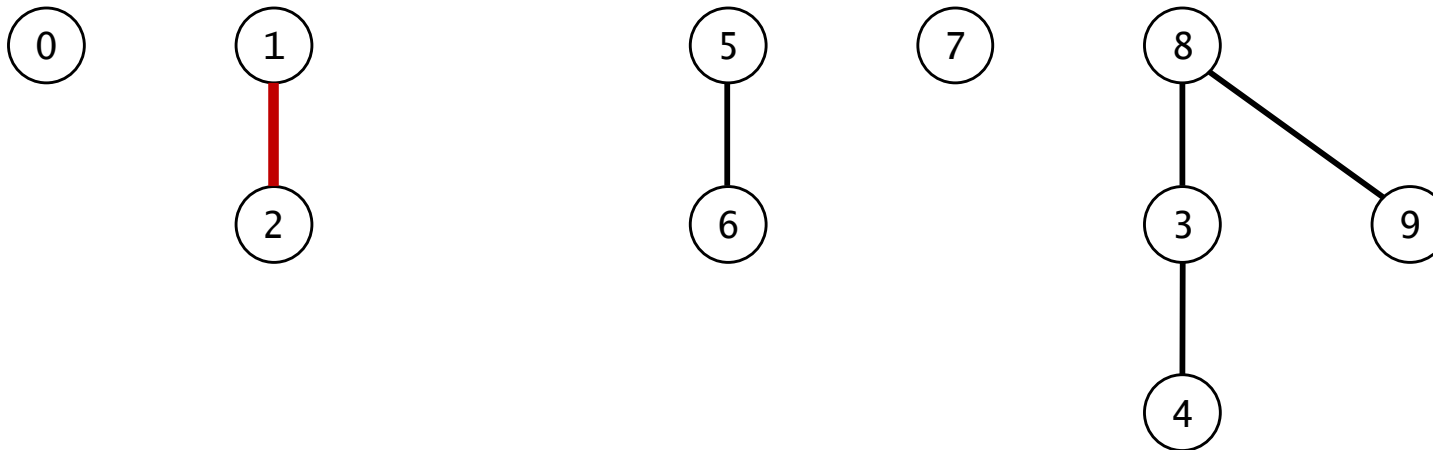
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	8	3	5	5	7	8	8

**union(2, 1)**



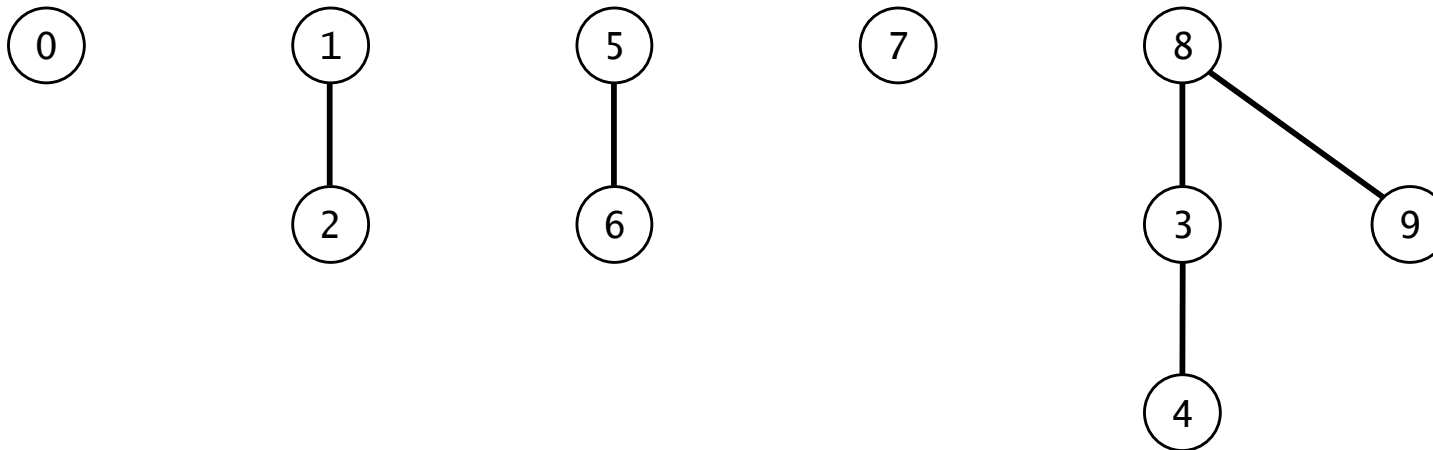
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	8	3	5	5	7	8	8

**union(2, 1)**



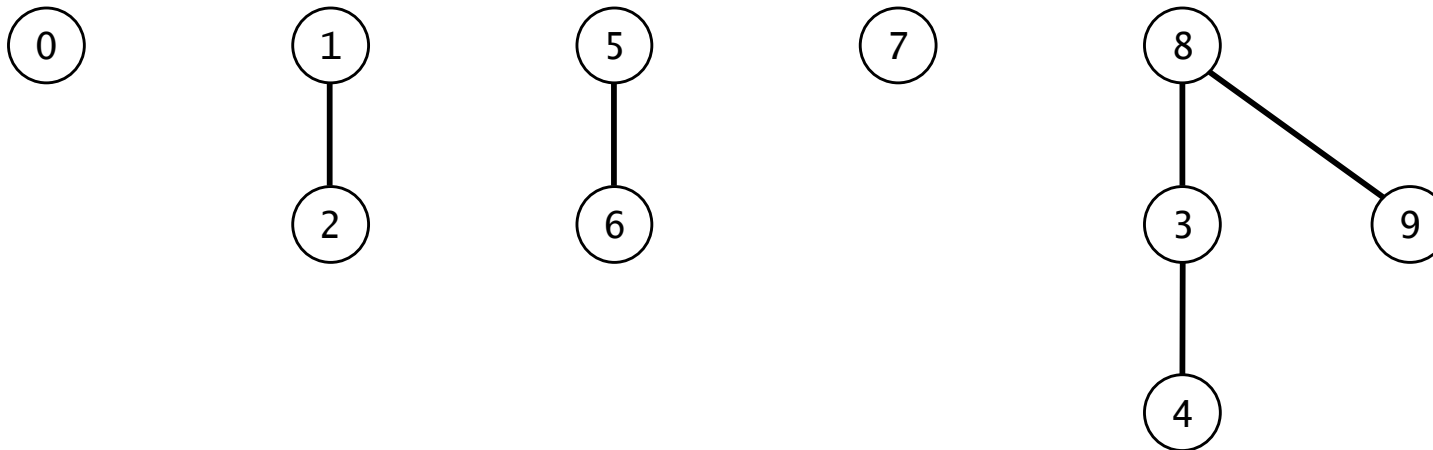
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	1	8	3	5	5	7	8	8

connected(8, 9) ✓



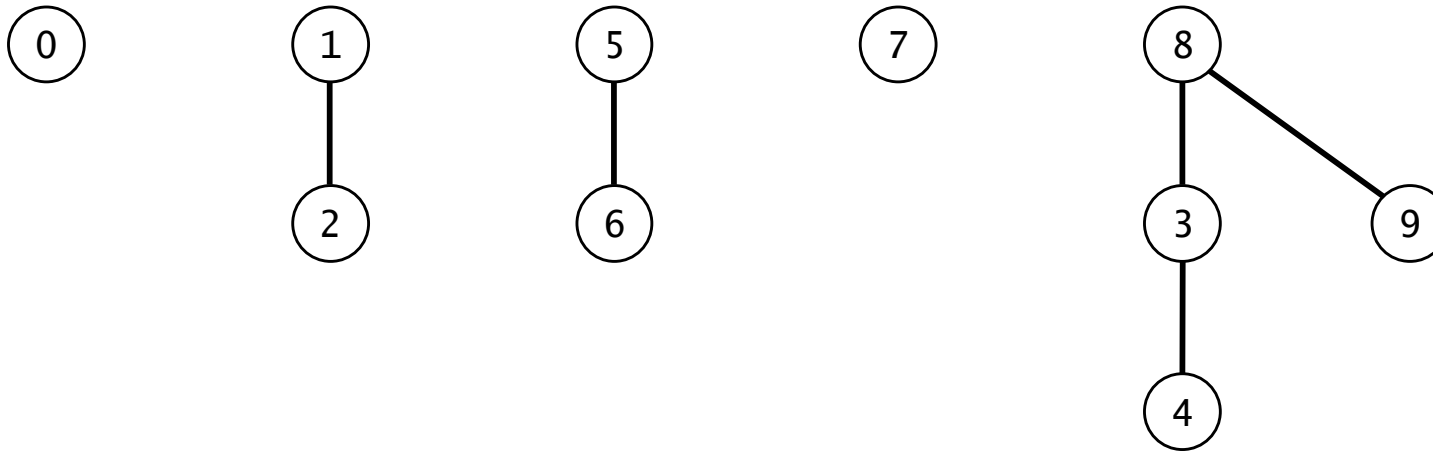
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	1	8	3	5	5	7	8	8

connected(5, 0) ❌



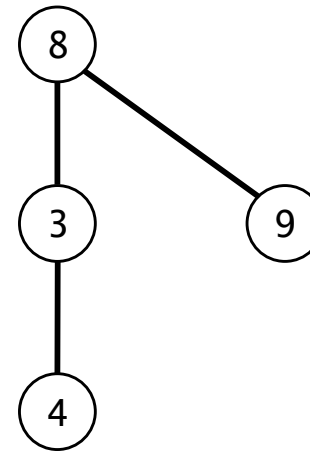
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	1	8	3	5	5	7	8	8

**union(5, 0)**



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

**union(5, 0)**



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

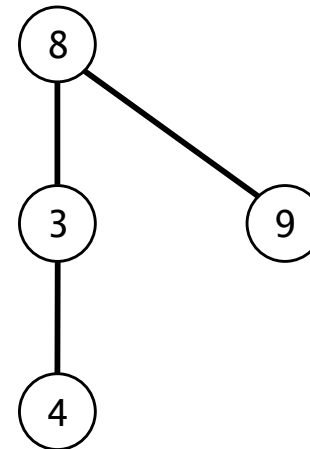
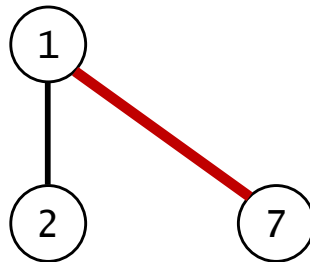
**union(7, 2)**



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

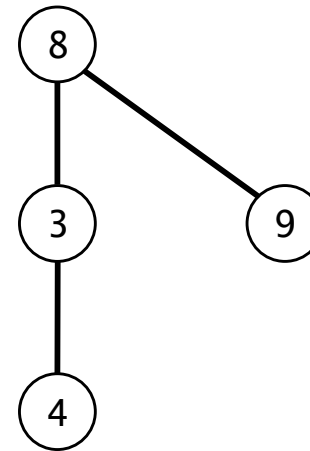
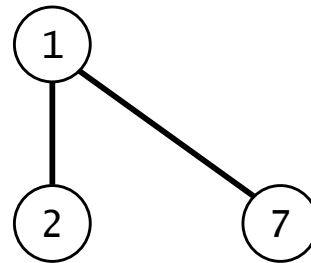


**union(7, 2)**



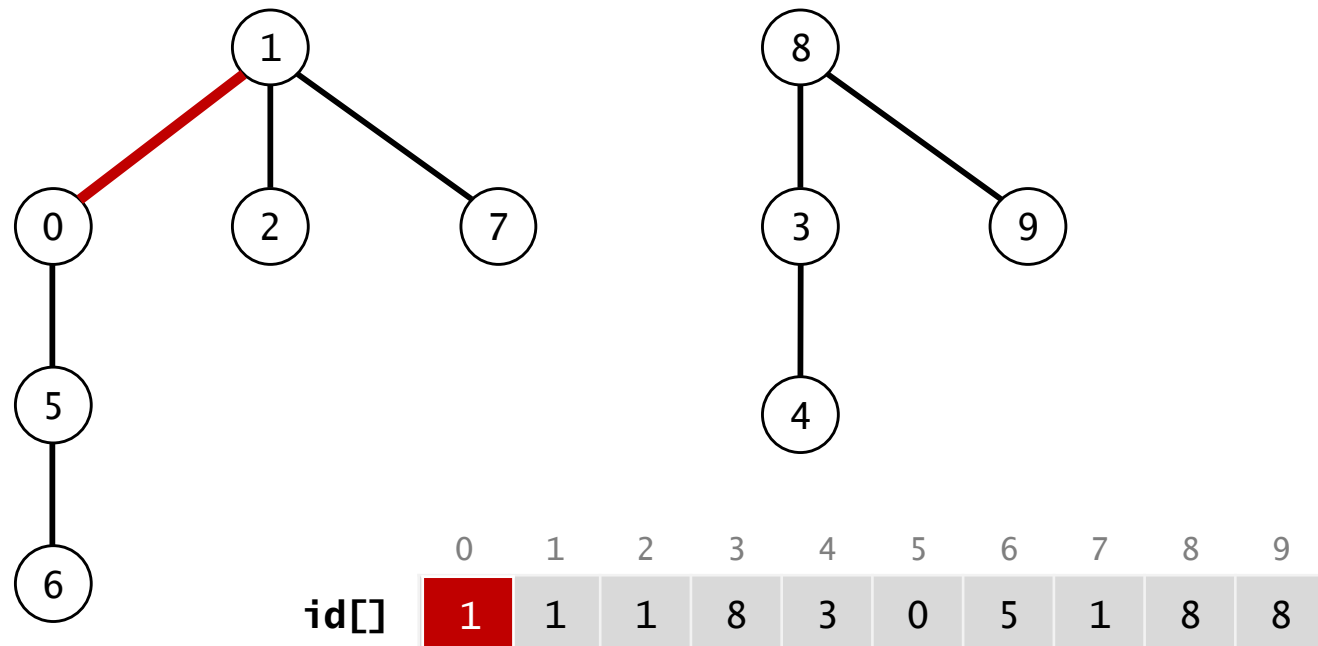
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	1	8	3	0	5	1	8	8

**union(6, 1)**

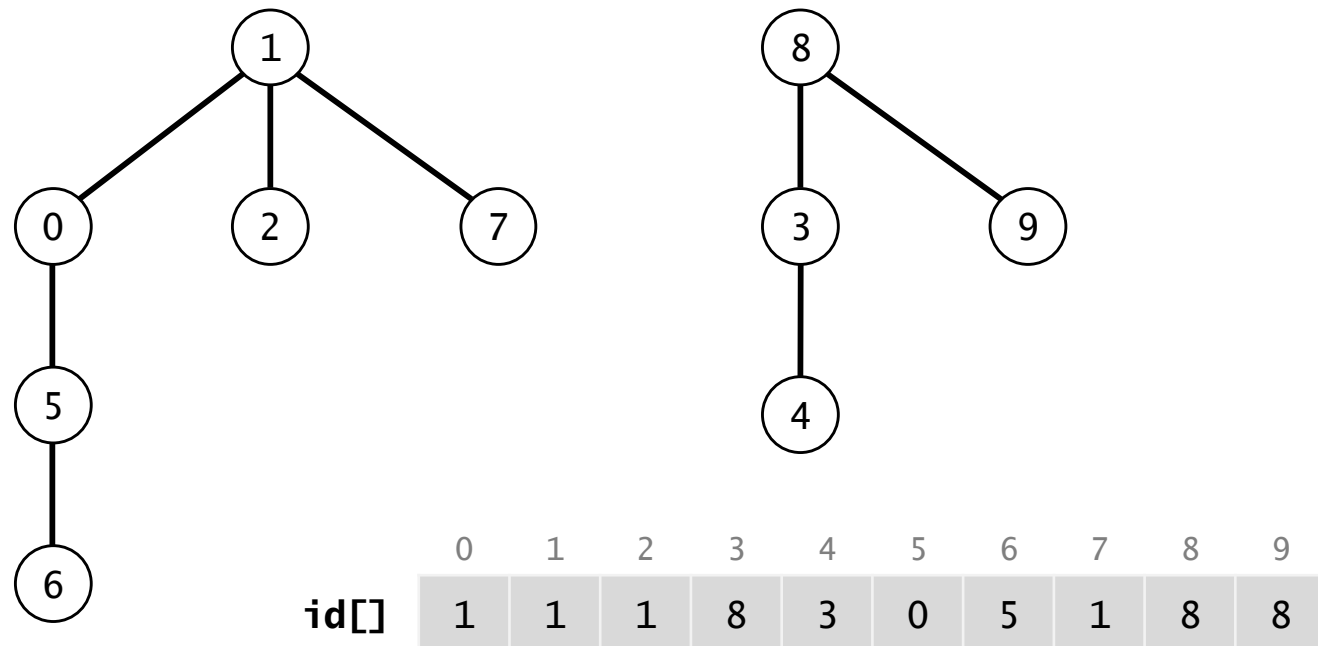


	0	1	2	3	4	5	6	7	8	9
id[]	0	1	1	8	3	0	5	1	8	8

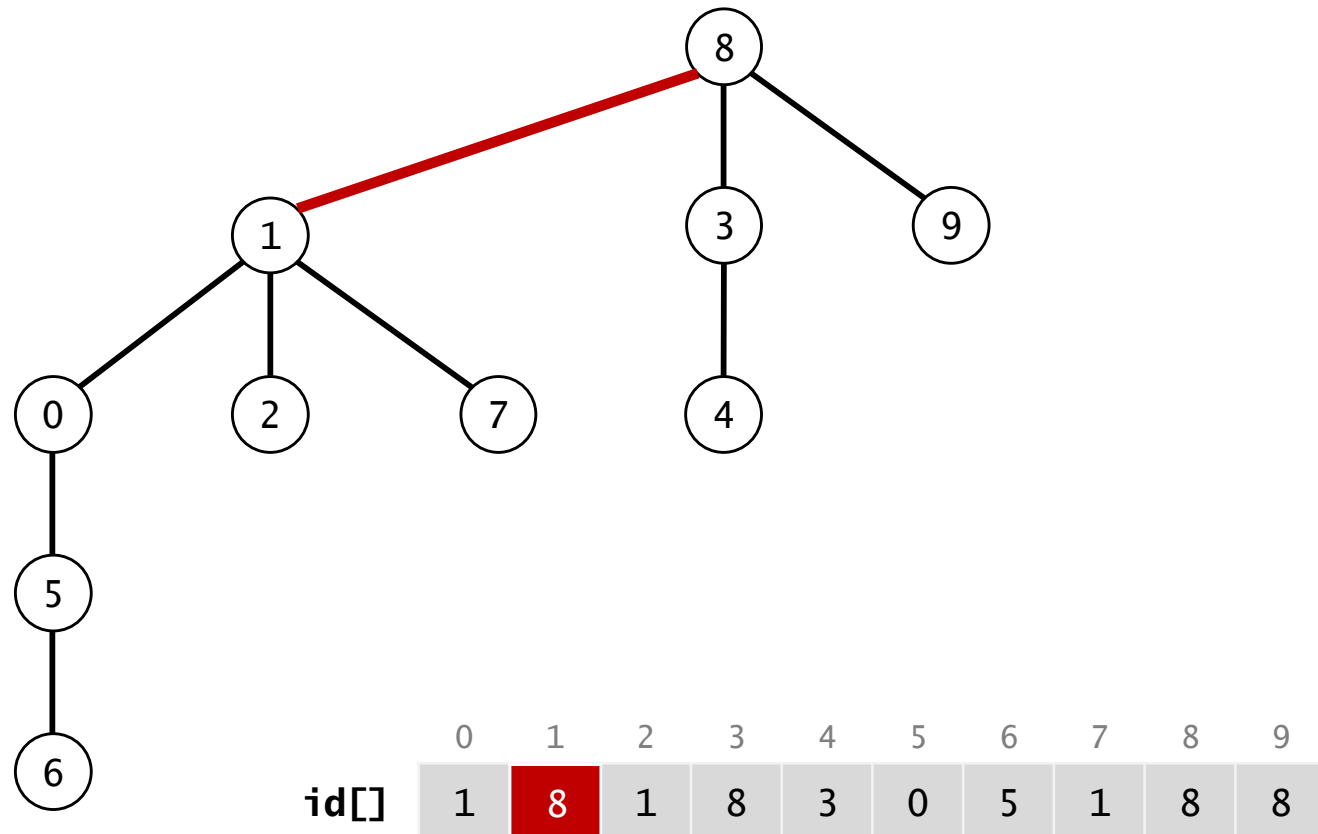
**union(6, 1)**

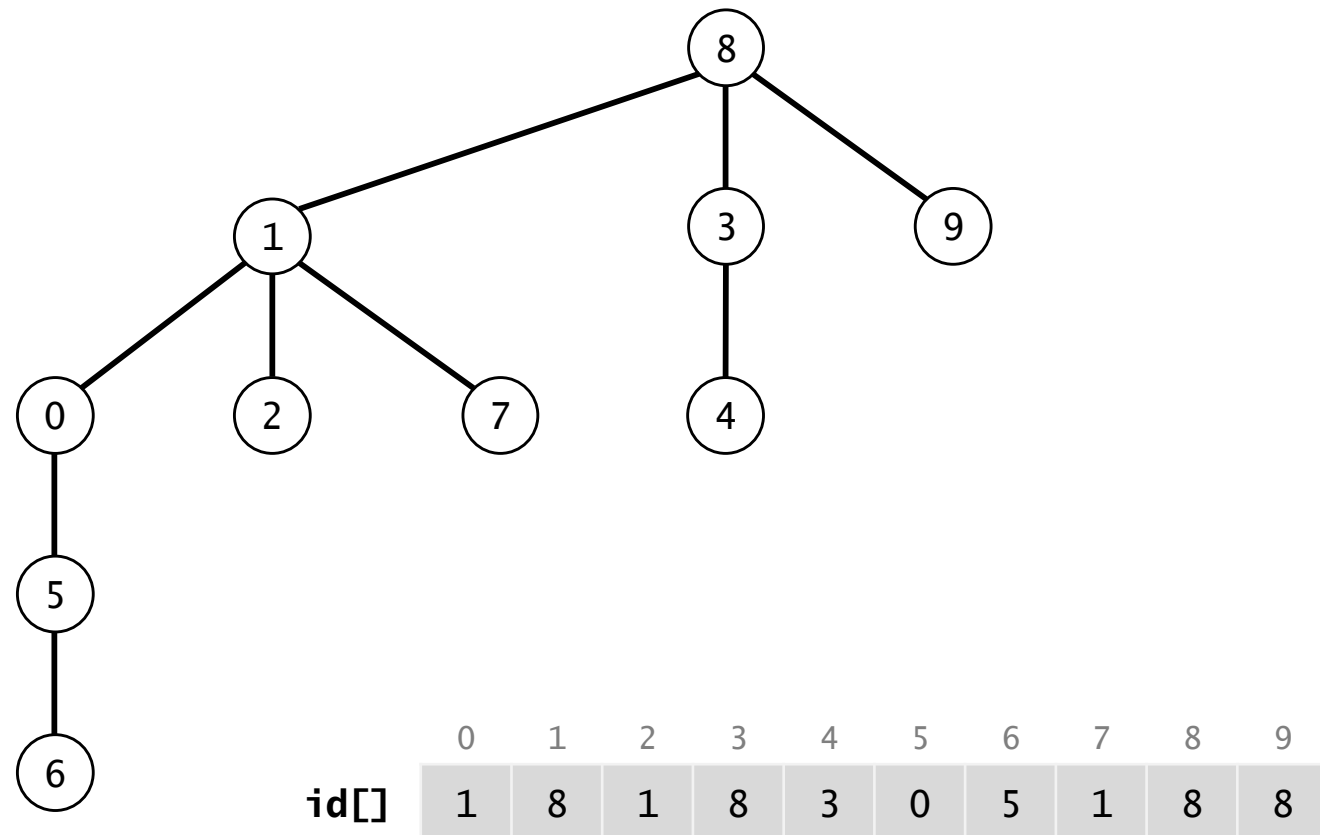


**union(7, 3)**



**union(7, 3)**





# این پیاده‌سازی کند است

۵۵

□ مدل هزینه.

□ تعداد دستیابی‌ها به آرایه (برای خواندن و نوشتن)

مقداردهی اولیه	اجتماع	یافتن	بررسی اتصال
N	N	N	N

□ ایراد پیاده‌سازی.

□ ارتفاع درخت‌ها می‌تواند زیاد باشد!

□ در نتیجه عمل `find` کند انجام می‌شود. [حداکثر  $N$  دسترسی به آرایه]

بهبود quick-union



# بهبود ۱: وزن دهی

۵۷

□ وزن دهی.

□ اجتناب از درخت‌های با ارتفاع زیاد.

□ ذخیره کردن اندازه‌ی هر درخت (تعداد گره‌ها).

□ ایجاد توازن با متصل کردن ریشه‌ی درخت کوچک‌تر به ریشه‌ی درخت بزرگ‌تر.

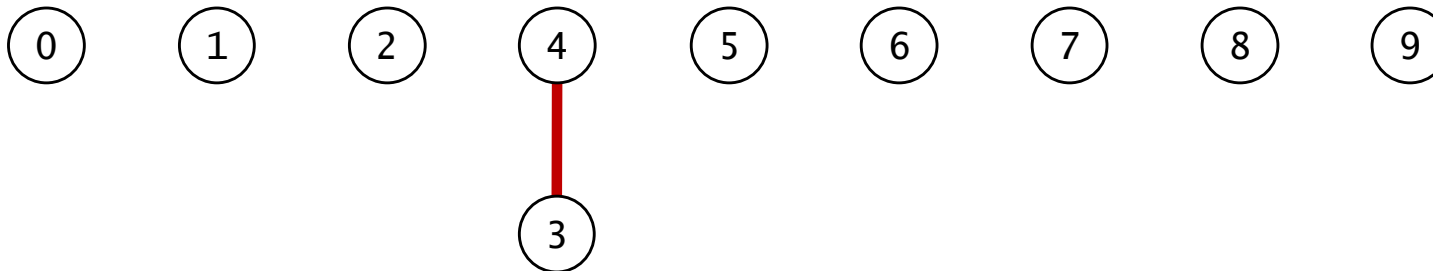


**union(4, 3)**



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

**union(3, 8)**

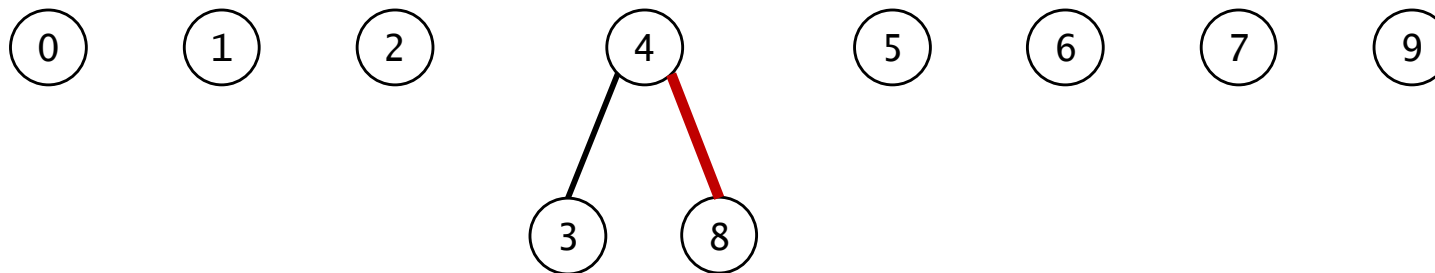


	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	4	4	5	6	7	8	9

# اجرای نمایشی

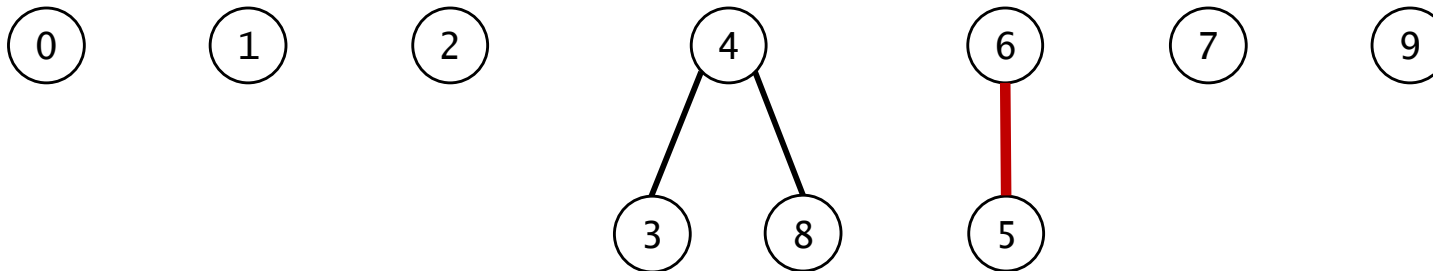
۶۰

**union(6, 5)**



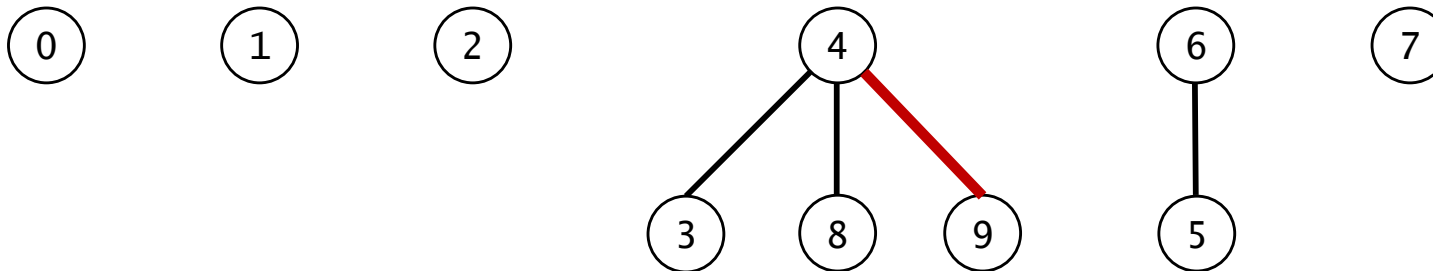
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	4	4	5	6	7	4	9

**union(9, 4)**



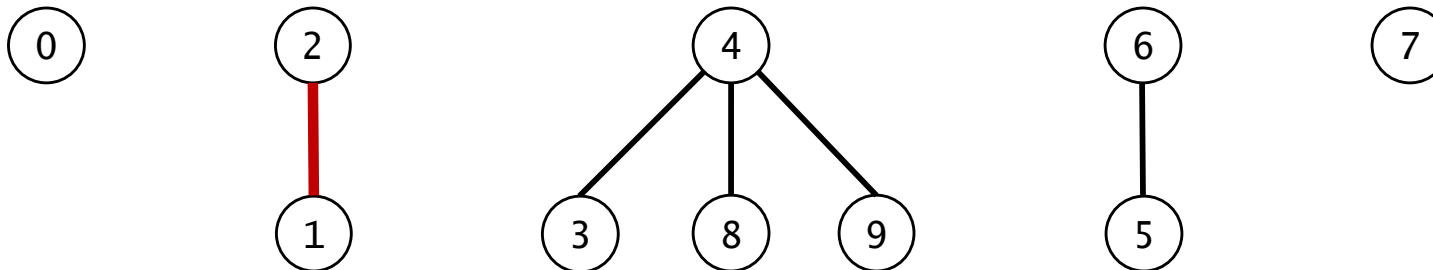
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	4	4	6	6	7	4	9

**union(2, 1)**



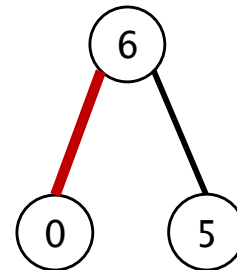
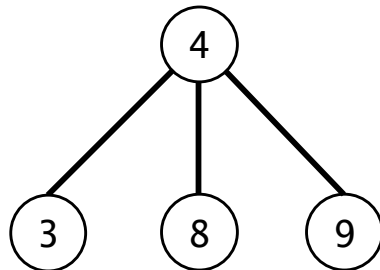
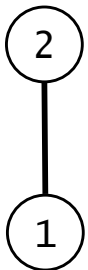
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	4	4	6	6	7	4	4

**union(5, 0)**



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

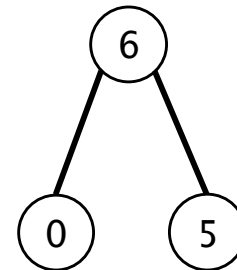
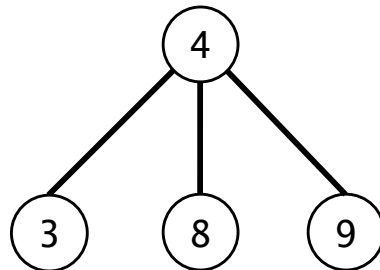
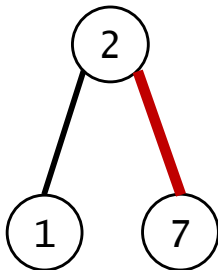
**union(7, 2)**



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

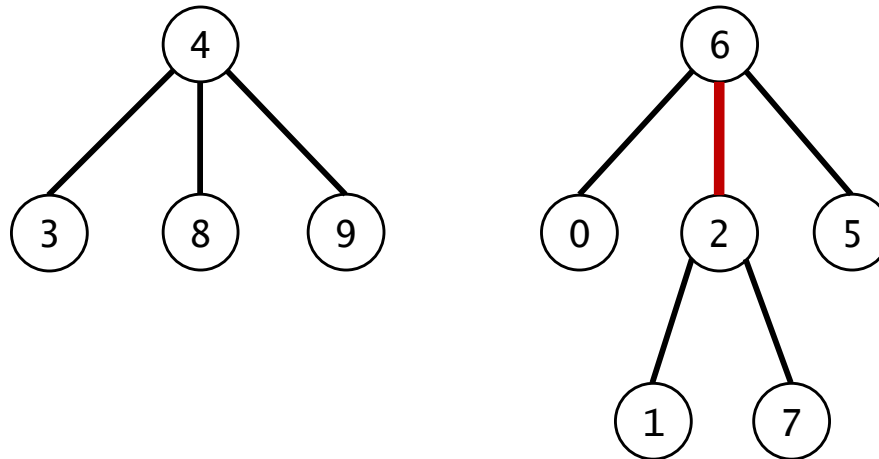


**union(6, 1)**

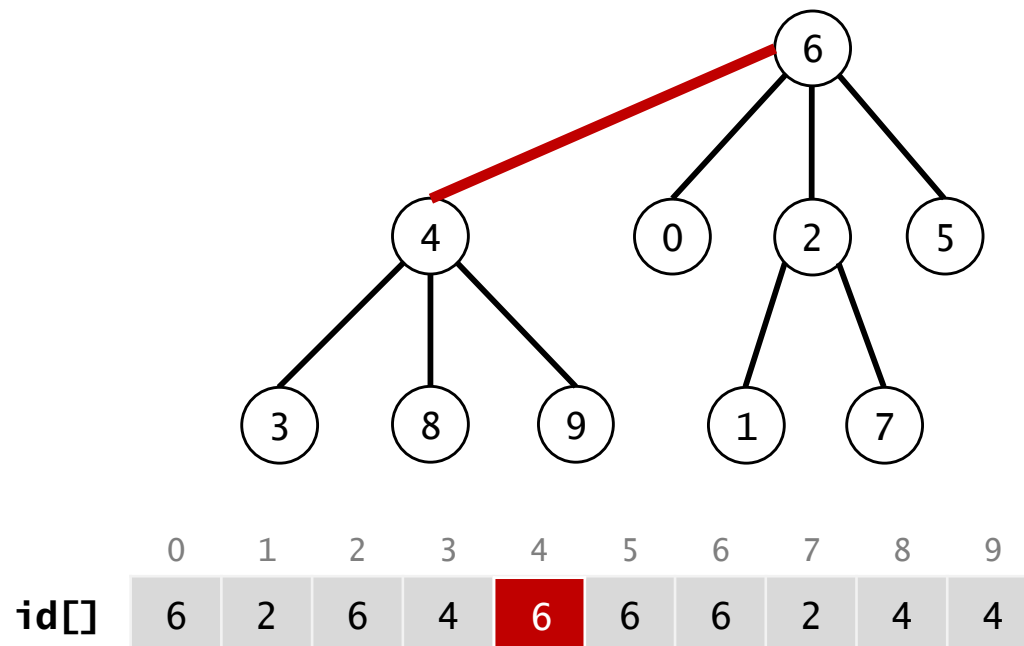


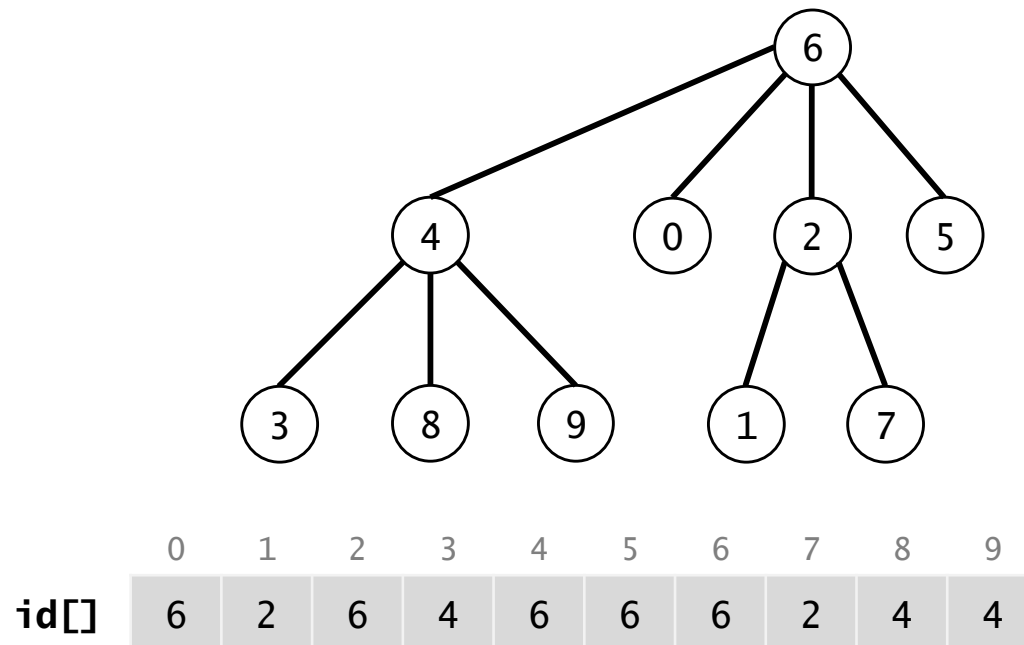
	0	1	2	3	4	5	6	7	8	9
id[]	6	2	2	4	4	6	6	2	4	4

**union(7, 3)**



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

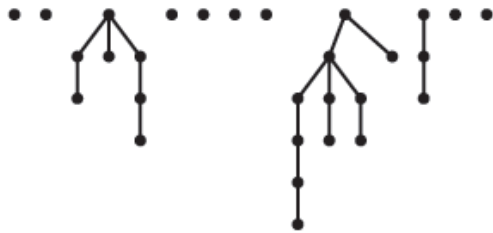




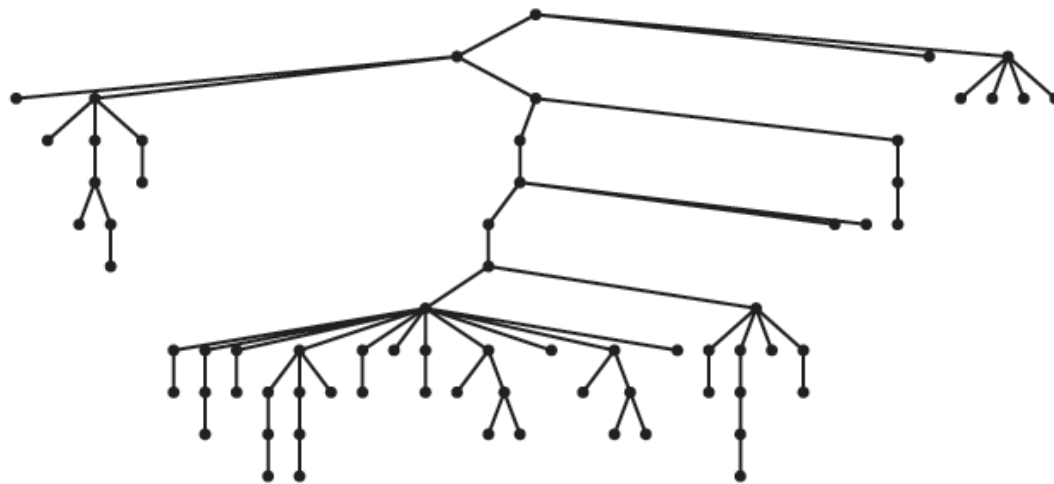
# بهبود ۱: مقایسه

۶۹

quick-union



متوسط ارتفاع تا ریشه: ۵/۱۱



weighted



متوسط ارتفاع تا ریشه: ۱/۵۲



ساختمان داده‌ی زیرمجموعه‌های مجزا شامل ۱۰۰ عنصر (پس از ۸۸ عمل اجتماع)

# بهبود ۱: پیاده‌سازی

۷۰

□ عمل اجتماع.

□ ریشه‌ی درخت کوچک‌تر را فرزند ریشه‌ی درخت بزرگ‌تر قرار بده.

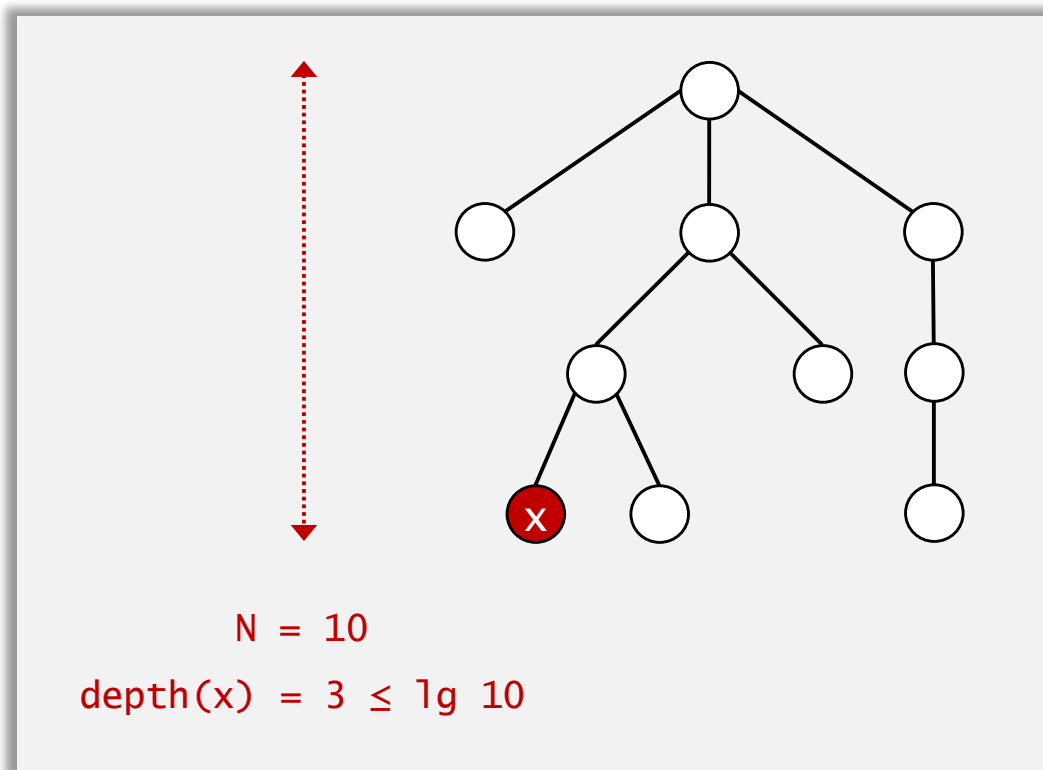
□ اندازه‌ی درخت بزرگ‌تر را به روز رسانی کن.

```
public void union(int p, int q)
{
    int i = find(p);
    int j = find(q);
    if (i == j) return;

    if (sz[i] < sz[j]) { id[i] = j; sz[j] += sz[i]; }
    else                { id[j] = i; sz[i] += sz[j]; }
}
```

# تحلیل ساختمان داده‌ی بهبود یافته

۷۱



□ زمان اجرا.

□ عمل find: زمانی متناسب با عمق عنصر در درخت.

□ عمل union: زمان ثابت، با داشتن ریشه‌ی دو درخت.

□ گزاره. عمق هر گره مانند  $x$  حداکثر برابر است با  $\lg N$ .

# تحلیل ساختمان داده‌ی بهبود یافته

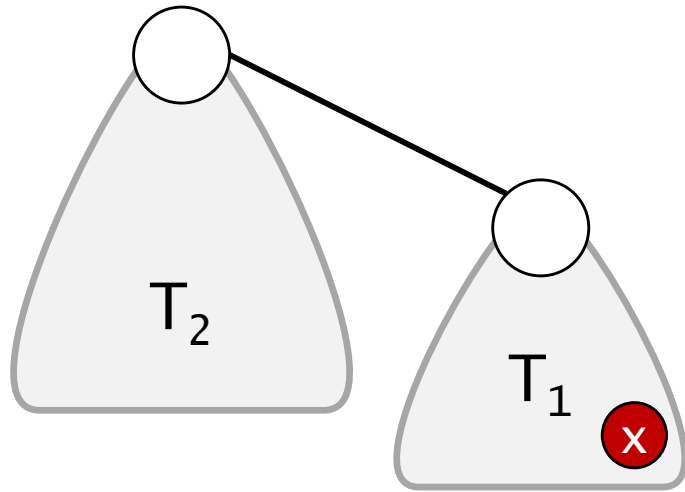
۷۲

□ زمان اجرا.

□ عمل **find**: زمانی متناسب با عمق عنصر در درخت.

□ عمل **union**: زمان ثابت، با داشتن ریشه‌ی دو درخت.

□ گزاره. عمق هر گره مانند  $x$  حداکثر برابر است با  $\lg N$ .



□ اثبات. عمق گره  $x$  چه زمانی افزایش می‌یابد؟

زمانی که درخت  $T_1$  شامل گره  $x$  با درخت بزرگ‌تر  $T_2$  ادغام می‌شود، عمق  $x$  یک واحد افزایش می‌یابد.

□ اندازه‌ی درخت شامل گره  $x$  حداقل دو برابر می‌شود زیرا  $|T_2| \geq |T_1|$ .

□ اندازه‌ی درخت شامل  $x$  حداکثر  $\lg N$  بار می‌تواند افزایش یابد. چرا؟



# تحلیل ساختمان داده‌ی بهبود یافته

۷۳

□ زمان اجرا.

□ عمل `find`: زمانی متناسب با عمق عنصر در درخت.

□ عمل `union`: زمان ثابت، با داشتن ریشه‌ی دو درخت.

□ گزاره. عمق هر گره مانند  $x$  حداکثر برابر است با  $\lg N$ .

الگوریتم	مقداردهی اولیه	اجتماع	بررسی اتصال
quick-find	$N$	$N$	1
quick-union	$N$	$N$	$N$
weighted QU	$N$	$\lg N$	$\lg N$

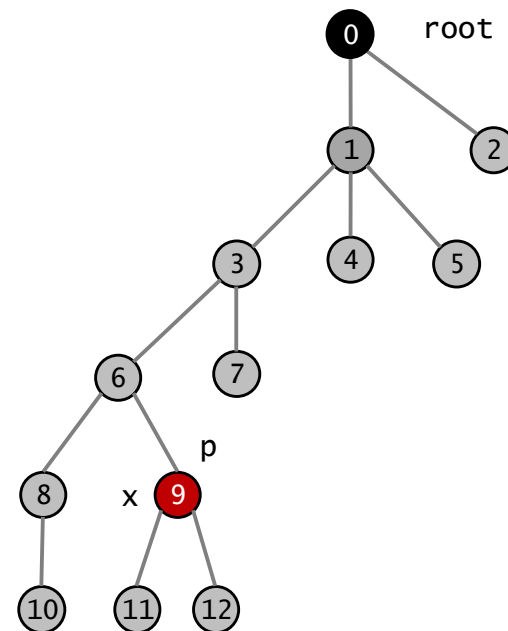
□ س. آیا پس از یافتن یک الگوریتم با کارایی قابل قبول باید توقف کنیم؟

□ ج. خیر، در صورت امکان باز هم کارایی را بهبود می‌دهیم.

# بهبود ۲: فشردسازی مسیر

۷۴

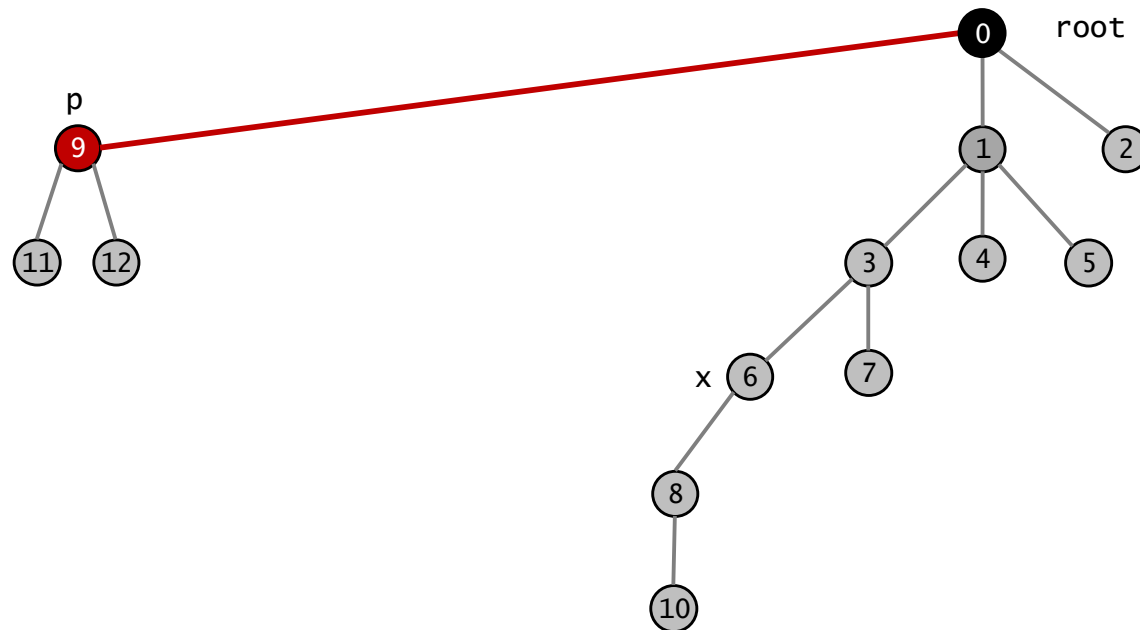
□ فشردسازی مسیر. درست پس از یافتن ریشه‌ی درخت شامل  $p$ ، پدر تمام گره‌های بررسی شده را برابر با ریشه قرار می‌دهیم.



# بهبود ۲: فشرده سازی مسیر

۷۵

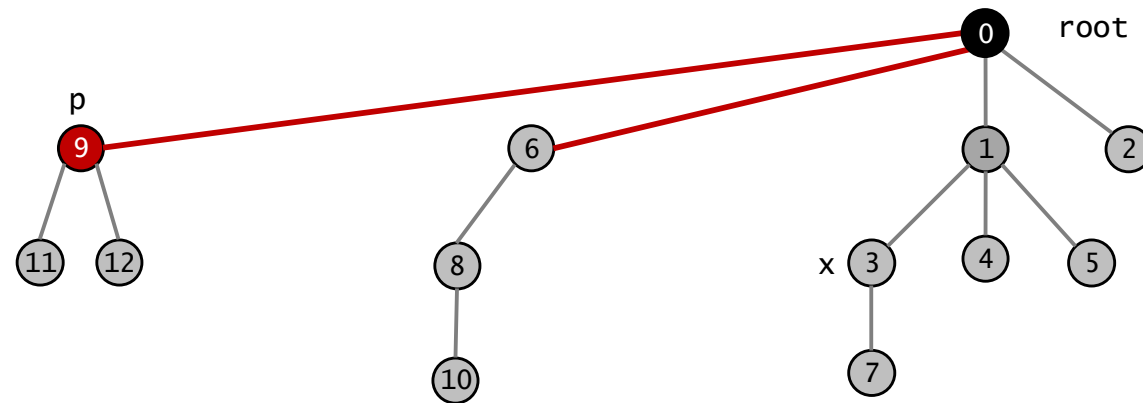
□ فشرده سازی مسیر. درست پس از یافتن ریشه ی درخت شامل  $p$ ، پدر تمام گره های بررسی شده را برابر با ریشه قرار می دهیم.



## بهبود ۲: فشردسازی مسیر

۷۶

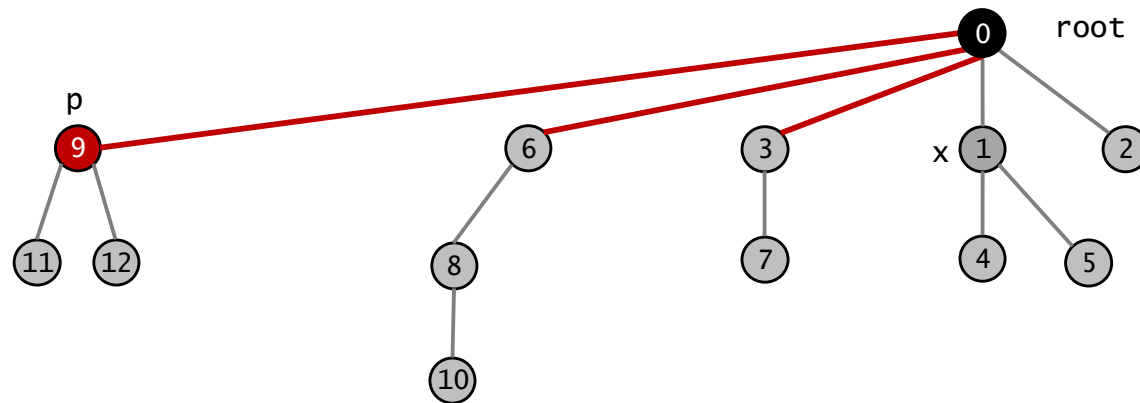
□ فشردسازی مسیر. درست پس از یافتن ریشه‌ی درخت شامل  $p$ ، پدر تمام گره‌های بررسی شده را برابر با ریشه قرار می‌دهیم.



## بهبود ۲: فشردگی سازی مسیر

۷۷

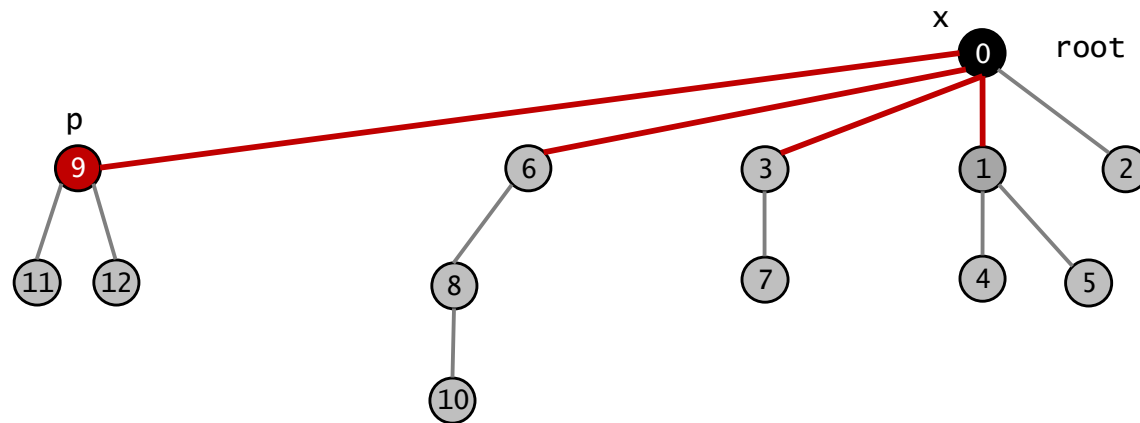
□ فشردگی سازی مسیر. درست پس از یافتن ریشه‌ی درخت شامل  $p$ ، پدر تمام گره‌های بررسی شده را برابر با ریشه قرار می‌دهیم.



## بهبود ۲: فشردگی‌سازی مسیر

۷۸

□ فشردگی‌سازی مسیر. درست پس از یافتن ریشه‌ی درخت شامل  $p$ ، پدر تمام گره‌های بررسی شده را برابر با ریشه قرار می‌دهیم.



# فشرده‌سازی مسیر: پیاده‌سازی جاوا

۷۹

□ پیاده‌سازی با یک گذر. کاری کن که هر گره دیگر در مسیر به پدربزرگش اشاره کند. (در نتیجه طول مسیر نصف می‌شود)

```
public int find(int i)
{
    while (i != id[i])
    {
        id[i] = id[id[i]];
        i = id[i];
    }
    return i;
}
```

□ سخن آخر. وزن‌دهی (به همراه فشردن‌سازی مسیر) باعث می‌شود بتوانیم مسایلی را حل کنیم که در غیر این صورت حل آنها ممکن نبود.

N	$\lg^* N$	الگوریتم	زمان اجرای بدترین حالت
1	0	quick-find	$M N$
2	1	quick-union	$M N$
4	2	weighted QU	$N + M \lg N$
65536	4	QU + path compression	$N + M \lg N$
$2^{65536}$	5	weighted QU + path compression	$N + M \lg^* N$

M عمل بر روی مجموعه ای از N شی

□ مثال.  $[10^9 \text{ عمل بر روی } 10^9 \text{ شی}]$

□ کاهش زمان اجرا از ۳۰ سال به ۶ ثانیه با استفاده از WQUPC

□ در این مورد ابررایانه‌ها کمک چندانی نمی‌کنند؛ راه‌حل استفاده از یک **الگوریتم خوب** است.



کاربرد: مسئله‌ی تراوش

# مسئله تراوش

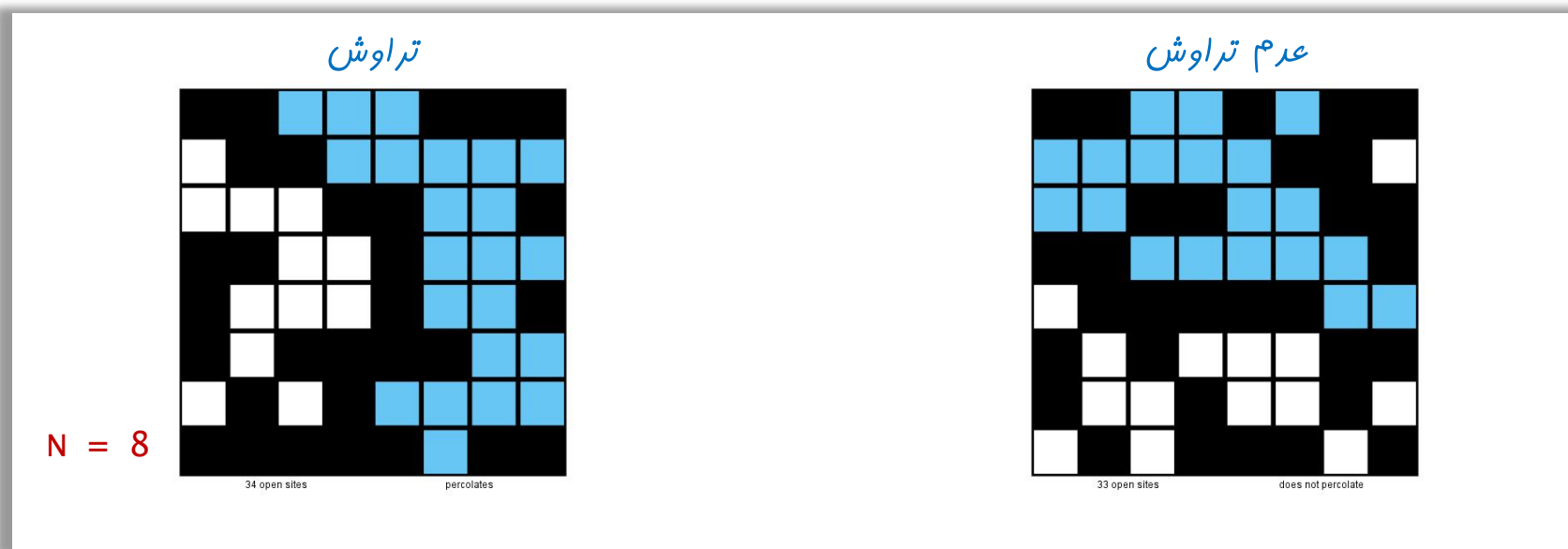
۸۲

□ یک مدل برای بسیاری از سیستم‌های فیزیکی:

□ یک جدول  $N$  در  $N$  از خانه‌ها

□ هر خانه‌ی جدول با احتمال  $p$  باز است.

□ اگر سطر ابتدا و انتهای جدول از طریق خانه‌های باز به هم وصل باشند، **تراوش** وجود دارد.



# مسئله تراوش

۸۳

□ یک مدل برای بسیاری از سیستم‌های فیزیکی:

□ یک جدول  $N$  در  $N$  از خانه ها

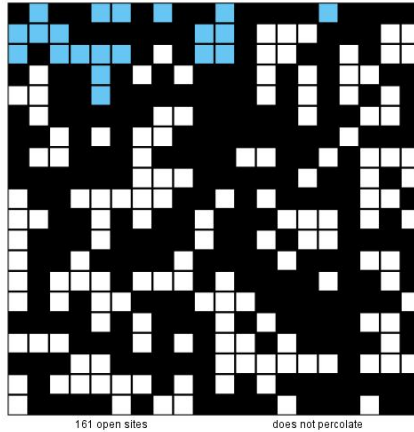
□ هر خانه‌ی جدول با احتمال  $p$  باز است.

□ اگر سطر ابتدا و انتهای جدول از طریق خانه‌های باز به هم وصل باشند، **تراوش** وجود دارد.

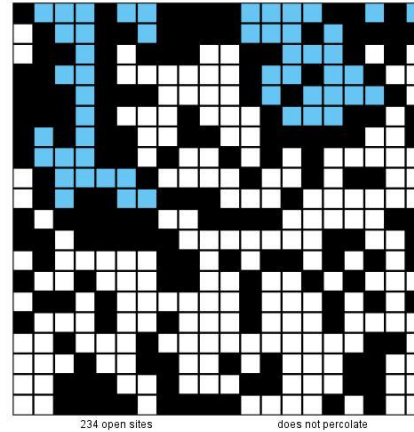
مدل	سیستم	خانه‌های خالی	خانه‌های پر	تراوش
الکتریسته	ماده	رسانا	عایق	رسانایی
جریان سیال	ماده	خالی	بسته	تخلخل
تعاملات اجتماعی	جمعیت	فرد	خالی	ارتباط

# احتمال تراوش

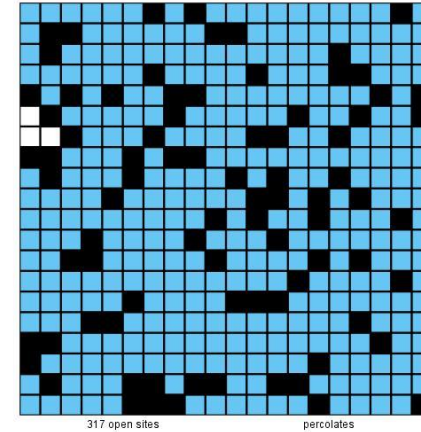
۸۴



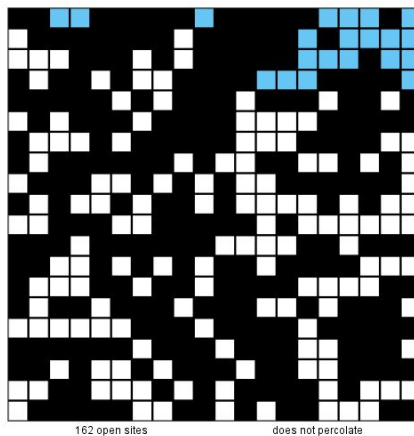
$p = 0.4$



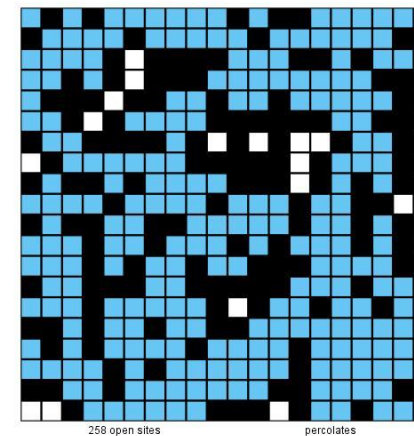
$p = 0.6$



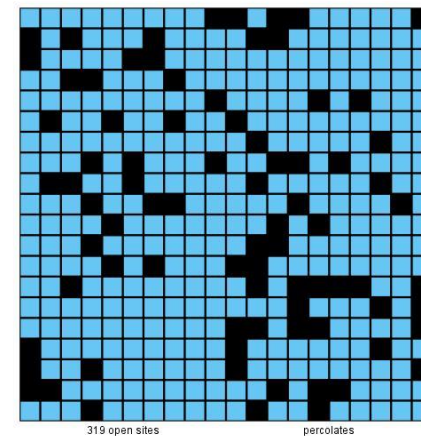
$p = 0.8$



عدم تراوش



تراوش؟



تراوش

# تغییر فاز تراوش

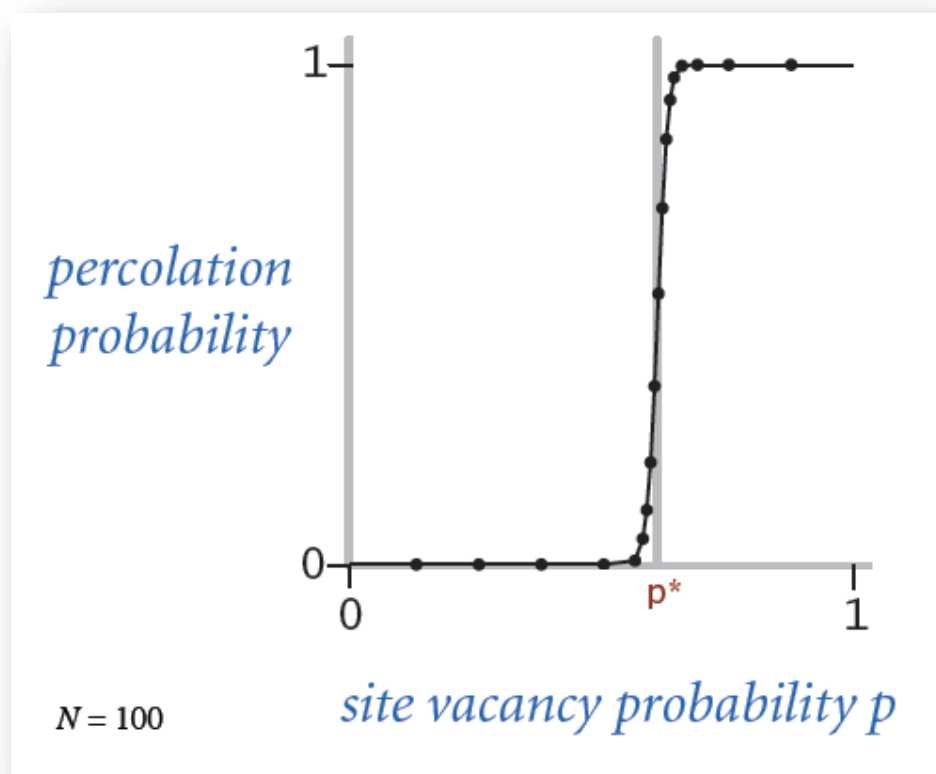
۸۵

□ به لحاظ نظری برای  $N$  های بزرگ، یک آستانه مانند  $p^*$  وجود دارد:

□  $p < p^*$ : به احتمال نزدیک به یقین عدم تراوش

□  $p > p^*$ : به احتمال نزدیک به یقین تراوش

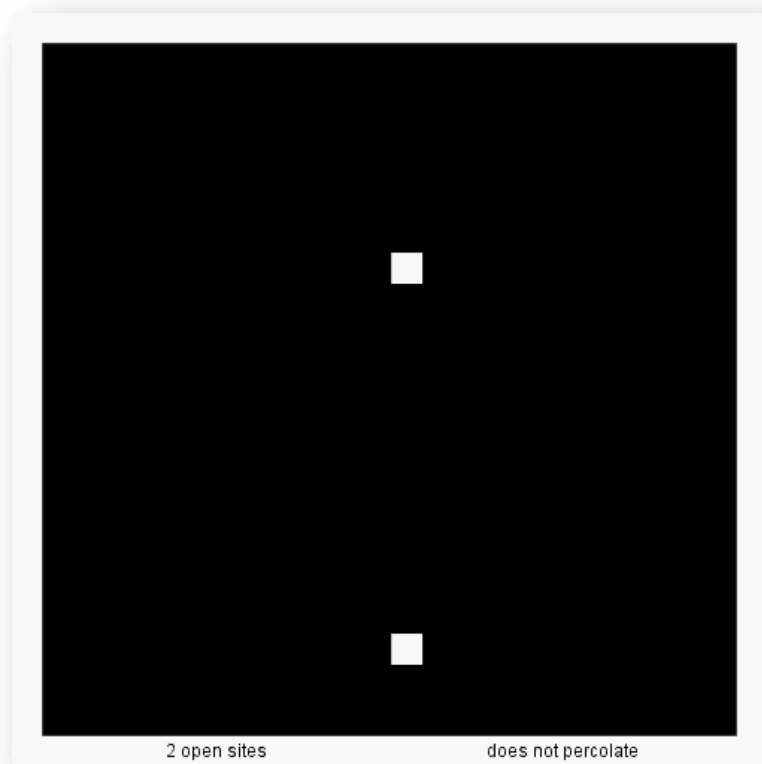
□ س. مقدار  $p^*$  چند است؟






# شبیه‌سازی مونت-کارلو

۸۶

- در ابتدا تمام خانه‌های جدول  $N$  در  $N$  بسته هستند.
- هر بار یک خانه‌ی تصادفی را باز کن تا زمانی که سطر اول و آخر به هم متصل شوند.
- درصد خانه‌های خالی تخمینی از مقدار  $p^*$  است.



$N = 20$

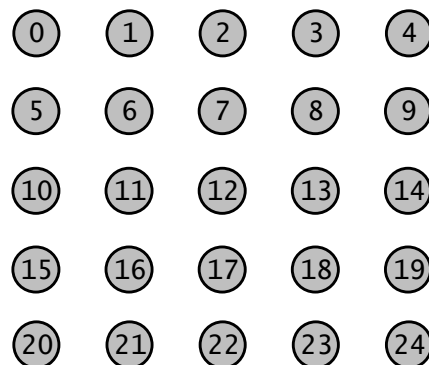
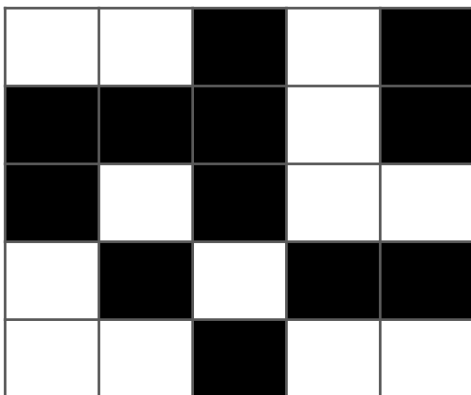
فانه‌های پر   
فانه‌های خالی   
فانه‌های بسته 

# همبندی پویا برای حل مسئله تراوش

۸۷

- س. چگونه می‌توان فهمید یک سیستم  $N$  در  $N$  تراوش می‌کند؟
- به ازای هر خانه در جدول، یک شی ایجاد کنید و آنها را از صفر تا  $N^2 - 1$  نام‌گذاری کنید.

$N = 5$



# همبندی پویا برای حل مسئله تراوش

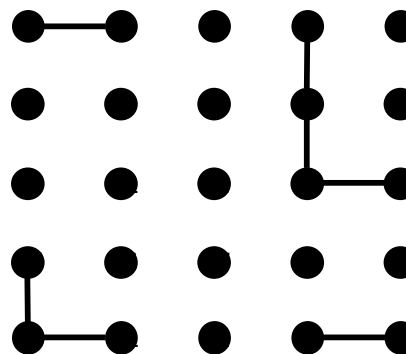
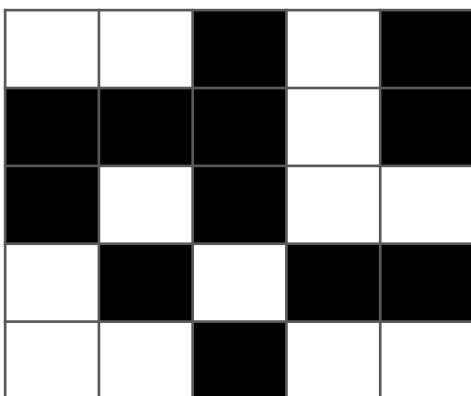
۸۸

□ س. چگونه می‌توان فهمید یک سیستم  $N$  در  $N$  تراوش می‌کند؟

□ به ازای هر خانه در جدول، یک شی ایجاد کنید و آنها را از صفر تا  $N^2 - 1$  نام‌گذاری کنید.

□ خانه‌هایی که از طریق خانه‌های باز به هم وصل هستند، در یک مؤلفه قرار دارند.

$N = 5$





# همبندی پویا برای حل مسئله تراوش

۸۹

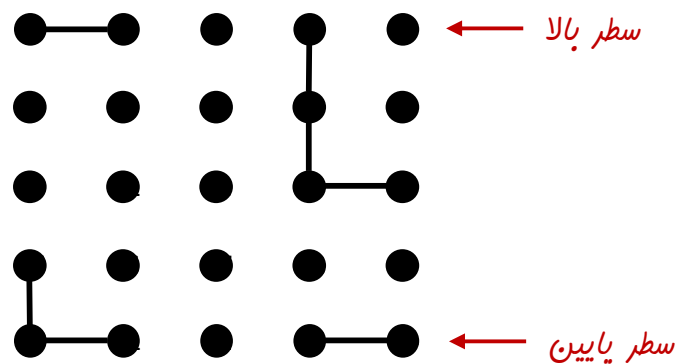
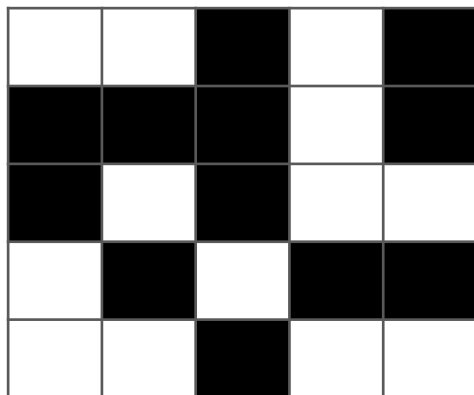
□ س. چگونه می‌توان فهمید یک سیستم  $N$  در  $N$  تراوش می‌کند؟

□ به ازای هر خانه در جدول، یک شی ایجاد کنید و آنها را از صفر تا  $N^2 - 1$  نام‌گذاری کنید.

□ خانه‌هایی که از طریق خانه‌های باز به هم وصل هستند، در یک مؤلفه قرار دارند.

□ تراوش: اگر حداقل یکی از خانه‌های سطر پایین به یکی از خانه‌های سطر بالا وصل باشد.

$N = 5$



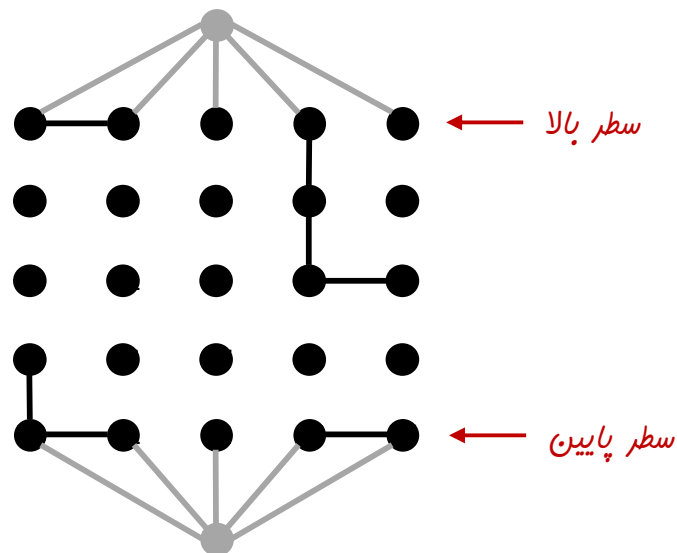
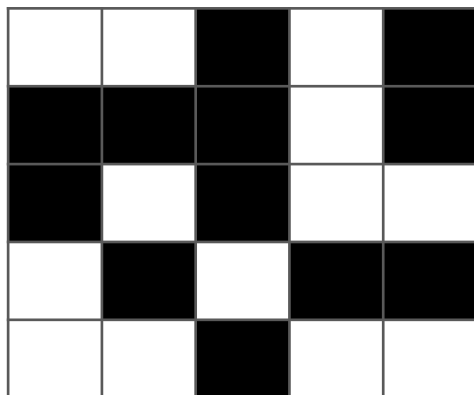
# همبندی پویا برای حل مسئلهی تراوش

۹۰

□ یک ترند هوشمندانه. دوشی مجازی ایجاد کنید (به همراه اتصالات لازم).

□ تراوش: اگرشی مجازی بالایی و پایینی به یکدیگر وصل باشند.

$N = 5$



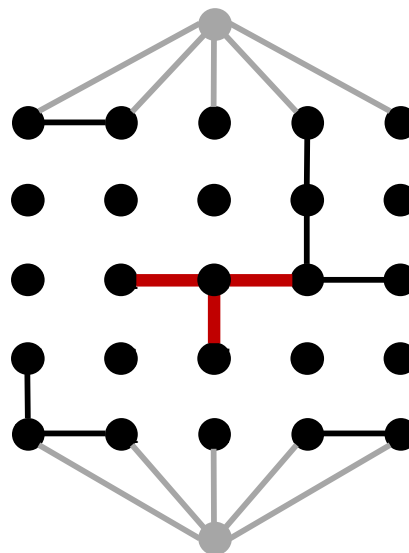
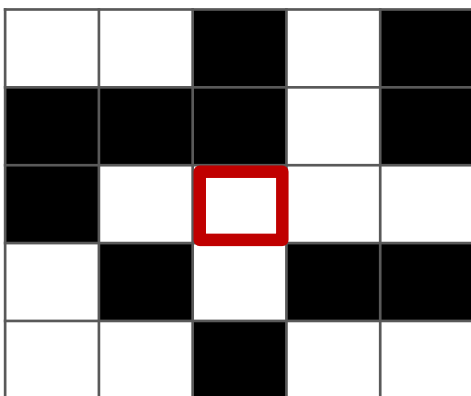
# همبندی پویا برای حل مسئلهی تراوش

۹۱

□ س. چگونه باز کردن خانه‌ها را مدل‌سازی کنیم؟

□ ج. خانه‌ی باز شده را به تمام خانه‌های همسایه که باز هستند، وصل کنید.

$N = 5$



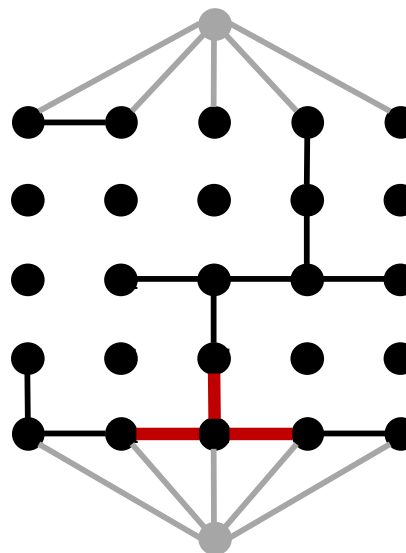
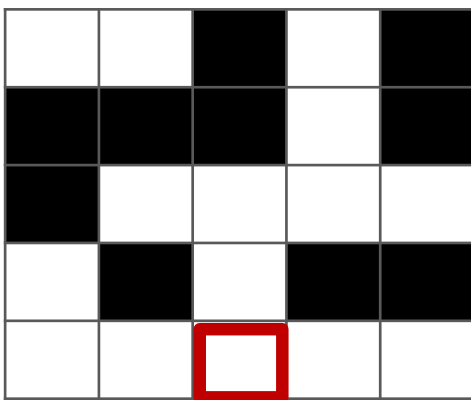
# همبندی پویا برای حل مسئلهی تراوش

۹۲

□ س. چگونه باز کردن خانه‌ها را مدل‌سازی کنیم؟

□ ج. خانه‌ی باز شده را به تمام خانه‌های همسایه که باز هستند، وصل کنید.

$N = 5$



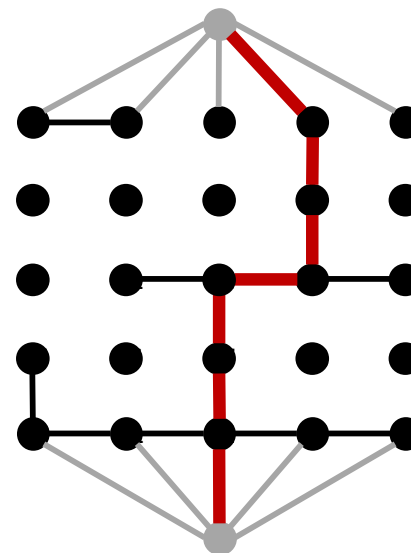
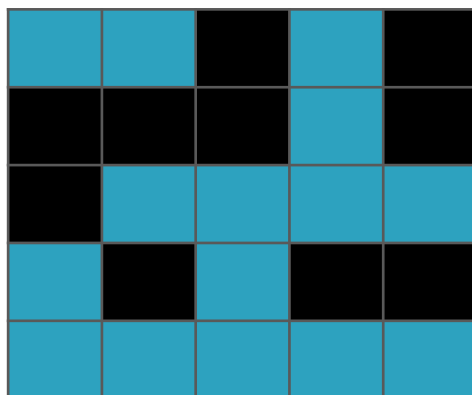
# همبندی پویا برای حل مسئله تراوش

۹۳

□ تشخیص تراوش.

□ تراوش: اگر شی مجازی بالایی و پایینی به یکدیگر وصل باشند.

$N = 5$

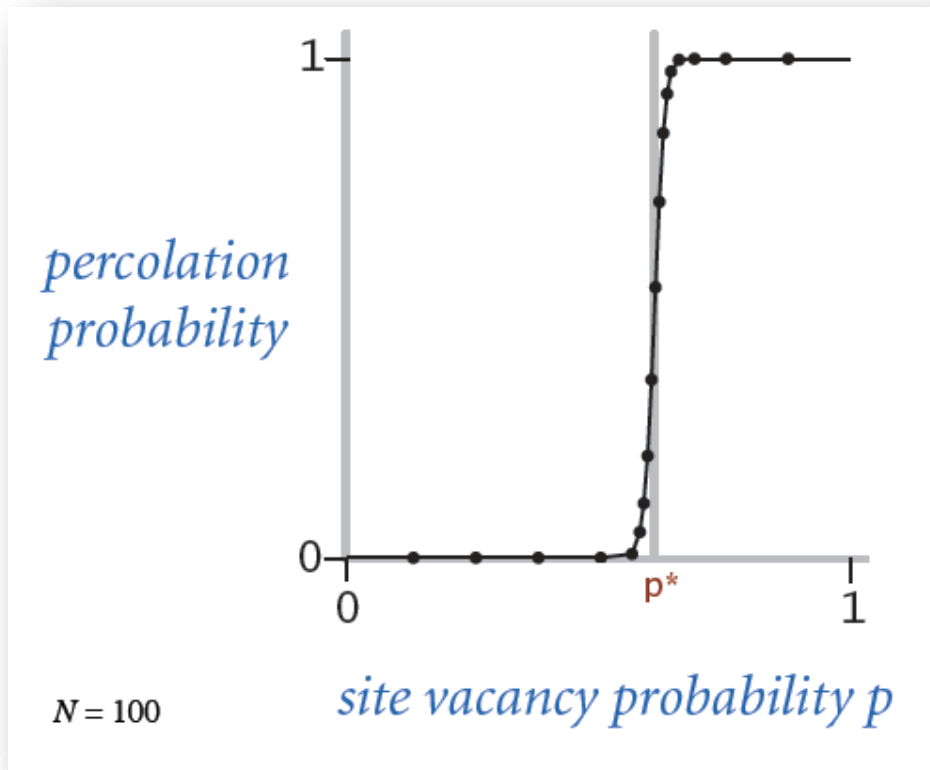


# آستانه‌ی تراوش

۹۴

□ س. مقدار  $p^*$  چند است؟

□ ج. تقریباً برابر با  $۰/۵۹۲۷۴۶$  [برای  $N$  های بزرگ]



□ به کمک الگوریتم‌های سریع می‌توان پاسخ دقیقی برای مسایل علمی فراهم کرد.

# خلاصه: مراحل توسعه یک الگوریتم

۹۵

□ مراحل توسعه یک الگوریتم.

□ مدل سازی مسئله

□ یافتن یک الگوریتم برای حل آن

□ تحلیل زمان و حافظه

■ آیا الگوریتم به اندازه‌ی کافی سریع است؟

■ آیا حافظه به اندازه‌ی کافی موجود است؟

□ اگر این گونه نیست، علت را بررسی کن

□ روشی برای حل مشکل پیدا کن

□ مراحل فوق را تا رسیدن به یک الگوریتم سریع و کم مصرف تکرار کن.

□ روش علمی.

□ تحلیل ریاضی.