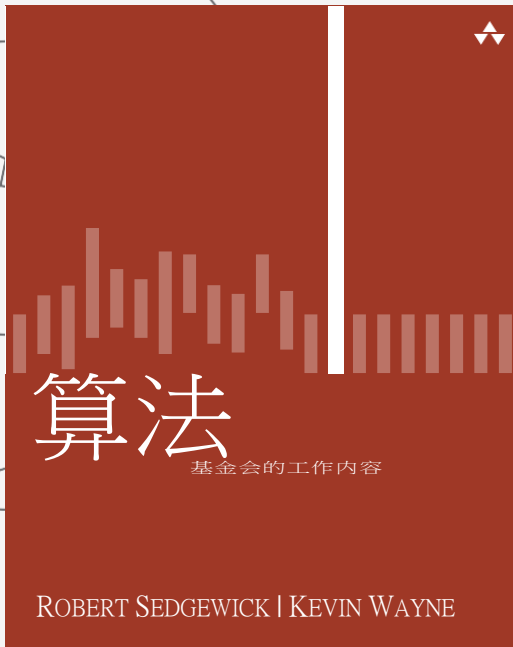


算法

罗伯特·塞奇威克 | 凯文·韦恩



[http:// algs 4.cs.princeton.edu](http://algs4.cs.princeton.edu)

## 1.5 联盟-寻找

- ▶ 动态连接
- ▶ 快速找到
- ▶ 快速联合
- ▶ 改进
- ▶ 应用

## 今天讲座的潜台词（以及这个课程）

---

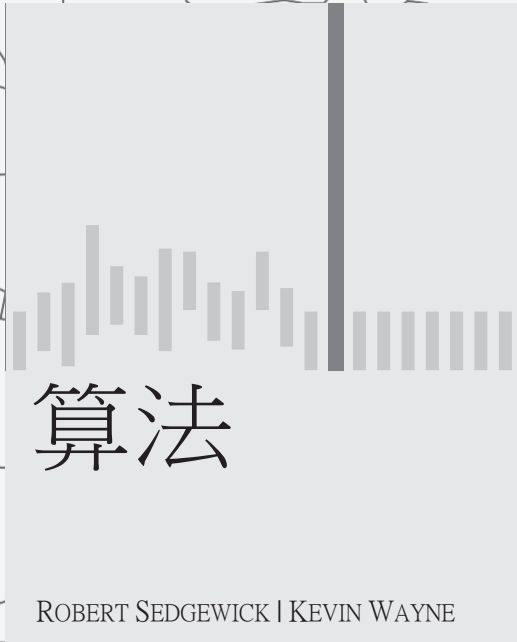
开发一个可用的算法的步骤。

- 建立问题的模型。
- 找到一种算法来解决它。
- 够快吗？能否装入内存？

如果没有，请找出原因。

- 找到一个解决问题的方法。
- 迭代直到满意为止。

科学方法。数学分析。



# 算法

ROBERT SEDGEWICK | KEVIN WAYNE

[http:// algs4.cs.princeton.edu](http://algs4.cs.princeton.edu)

## 1.5 联盟-寻找

---

▶ 动态连接

▶ 快速找到

▶ 快速联合

▶ 改进

▶ 应用

# 动态连接

给出一个有N个对象的集合。

- **联接**命令。 连接两个对象。
- **找到/连接的**查询。 是否有一条路径连接这两个物体？

联盟(4, 3)

union(3, 8)

联盟(6, 5)

union(9, 4)

union(2, 1)

连接(0, 7) ❌

连接(8, 9) ✔️

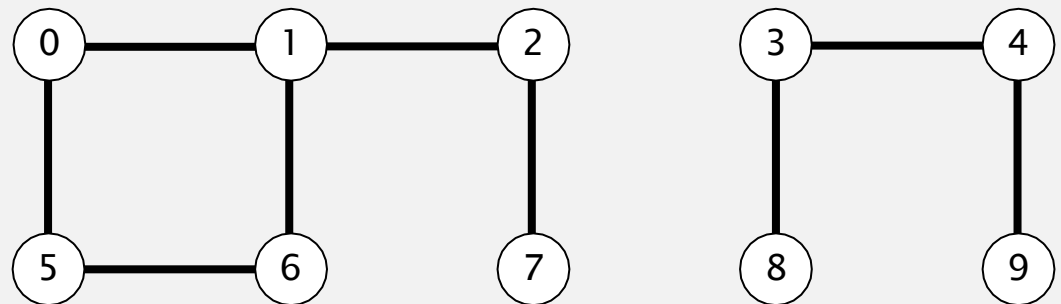
union(5, 0)

union(7, 2)

union(6, 1)

union(1, 0)

连接(0, 7) ✔️

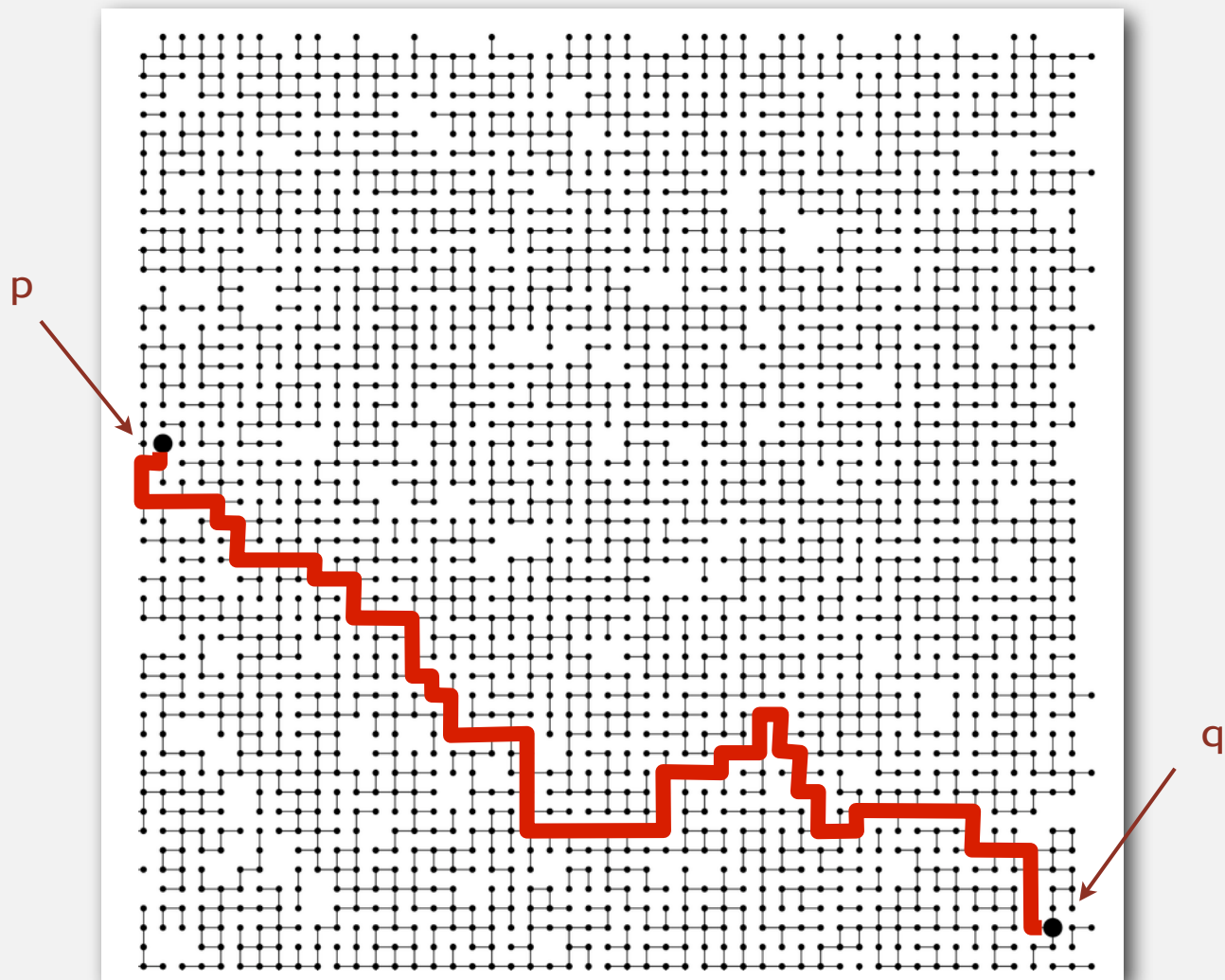




## 连接实例

---

Q. 是否有一条连接 $p$ 和 $q$ 的路径？



A. 是的。



# 为物体建模

---

应用程序涉及操纵所有类型的对象。

- 数字照片中的像素。
- 网络中的计算机。
- 社交网络中的朋友。
- 计算机芯片中的晶体管。
- \*一个数学集合中的元素。
- \*Fortran程序中的变量名称。
- 复合系统中的金属点。

在编程时，方便命名对象0到N-1。

- 使用整数作为数组索引。

抑制与联合搜索无关的细节。

可以使用符号表将网站名称翻译成整数：敬请关注（第三章）。





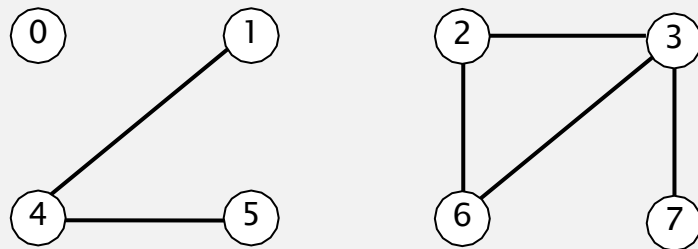
## 建立连接的模型

我们假设 "与之相连" 是一种等价关系。

- 反射性。  $p$  与  $p$  相连。
- 对称性：如果  $p$  与  $q$  相连，则  $q$  与  $p$  相连。
- 传递性：如果  $p$  与  $q$  相连， $q$  与  $r$  相连，则  $p$  与  $r$  相连。

连接的组件。

相互连接的对象的最大集合。



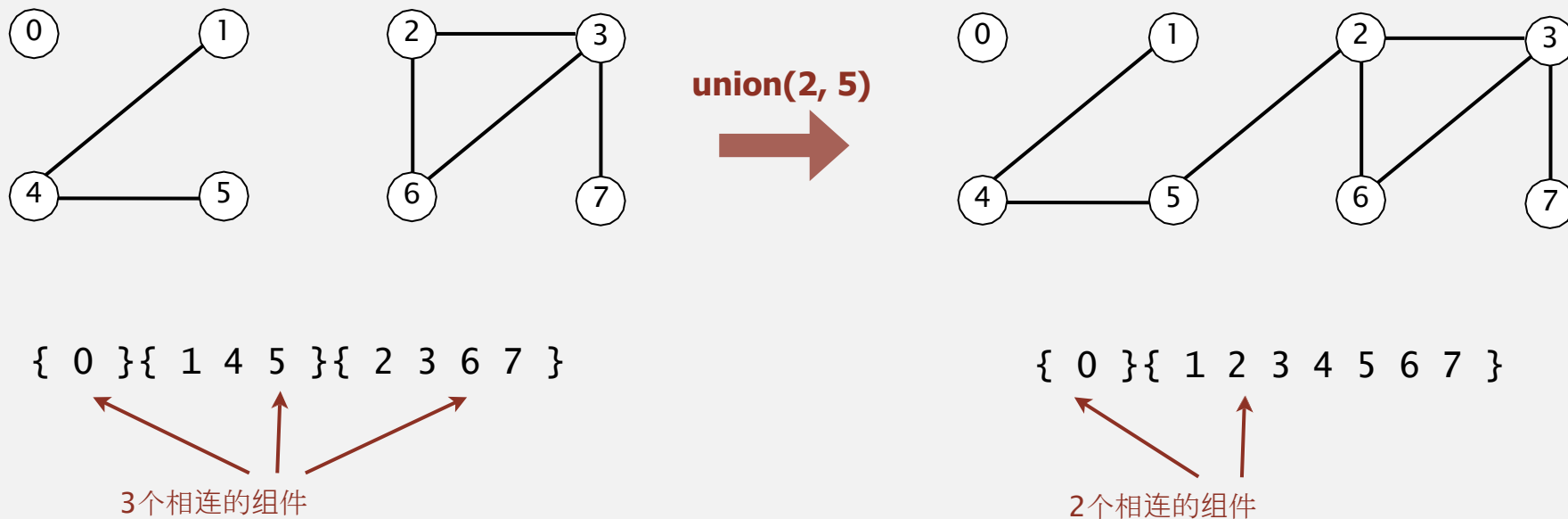
{ 0 } { 1 4 5 } { 2 3 6 7 }

3个相连的组件

## 执行业务

**查找查询。** 检查两个对象是否在同一个组件中。

**联合命令。** 用它们的联合体替换包含两个对象的组件。



## 联合查找数据类型（API）。

---

**目标。** 为联合查找设计有效的数据结构。

\*对象的**数量** $N$ 可以是巨大的。

操作**数量** $M$ 可以是巨大的。

\***查找**查询和联合命令可以混用。

公开课 **UF**

**UF(int N)**

用以下方式初始化union-find数据结构

空白的**union(int p, int q)**

$N$ 个对象（0至 $N-1$ ）。  
在 $p$ 和 $q$ 之间添加连接

。布尔型 **连接(int p, 黠黠 q)**

$p$ 和 $q$ 是在同一个组件中吗？

黠黠 **find(int p)**

$p$ 的组件标识符（0至 $N-1$ ）。

黠黠 **计数()**

部件数



## 动态连接的客户端

---

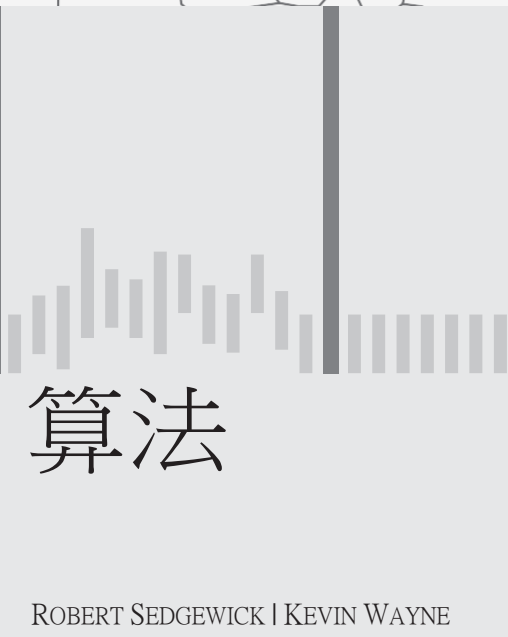
从标准输入中读入对象的数量 $N$ 。

- 重复。
  - 从标准输入中读入一对整数
  - 如果他们还没有连接，请连接他们并打印出一对

```
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)。
        }
    }
}
```

```
% 更多 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
```





# 算法

ROBERT SEDGEWICK | KEVIN WAYNE

[http:// algs4.cs.princeton.edu](http://algs4.cs.princeton.edu)

## 1.5 联盟-寻找

---

▶ 动态连接

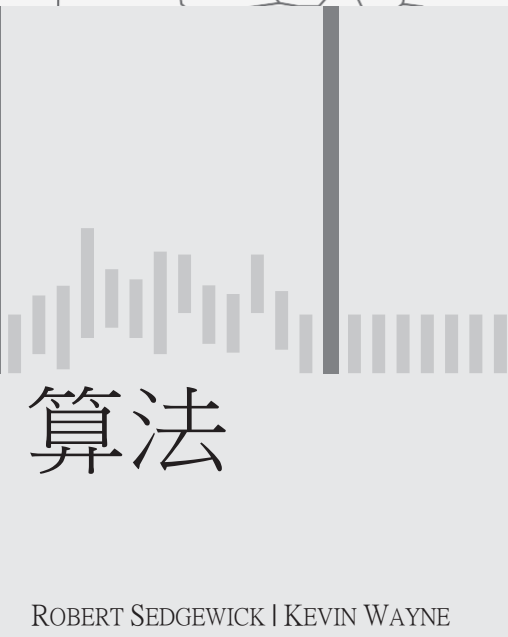
▶ 快速找到

▶ 快速联合

▶ 改进

▶ 应用





# 算法

ROBERT SEDGEWICK | KEVIN WAYNE

[http:// algs4.cs.princeton.edu](http://algs4.cs.princeton.edu)

## 1.5 联盟-寻找

---

▶ 动态连接

▶ 快速找到

▶ 快速联合

▶ 改进

▶ 应用

## 快速查找[急切的方法]

数据结构。

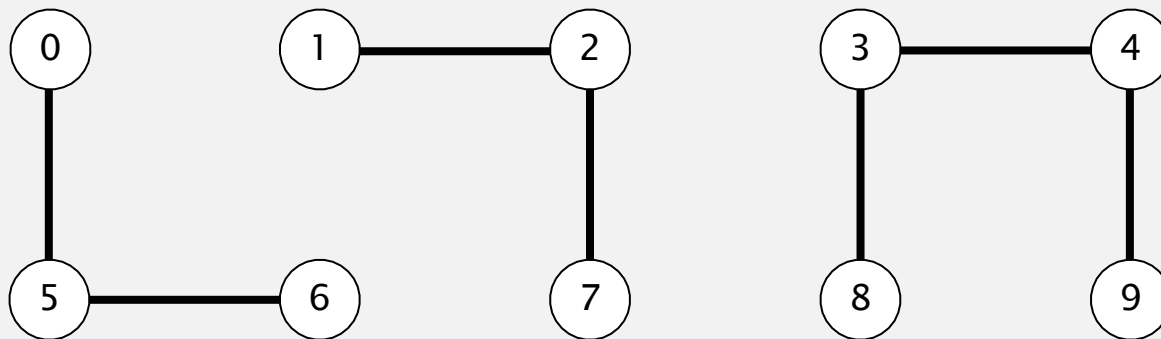
- 长度为N的整数数组id[]。

当且仅当

- 解释。 如果p和q有相同的id，则它们是相连的。

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

0、5和6是相连的  
1、2、7相连  
3、4、8和9是相连的



## 快速找到[急切的方法]

数据结构。

- 长度为N的整数数组id[]。
- 解释。 如果p和q有相同的id，则它们是相连的。

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

查找。 检查p和q是否有相同的id。

id[6] = 0; id[1] = 1  
6和1没有连接

联合。 要合并包含p和q的组件，将所有id等于id[p]的条目改为id[q]。

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

在6和1的结合之后

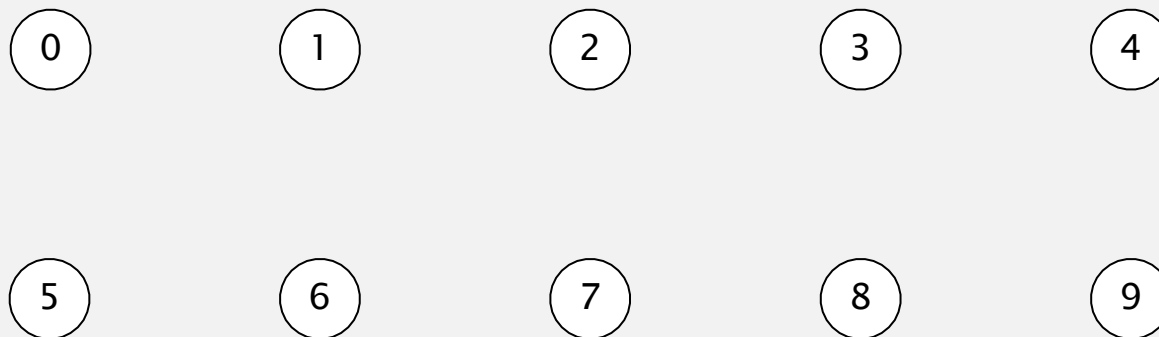


问题：许多数值都会改变



# 快速查找演示

---

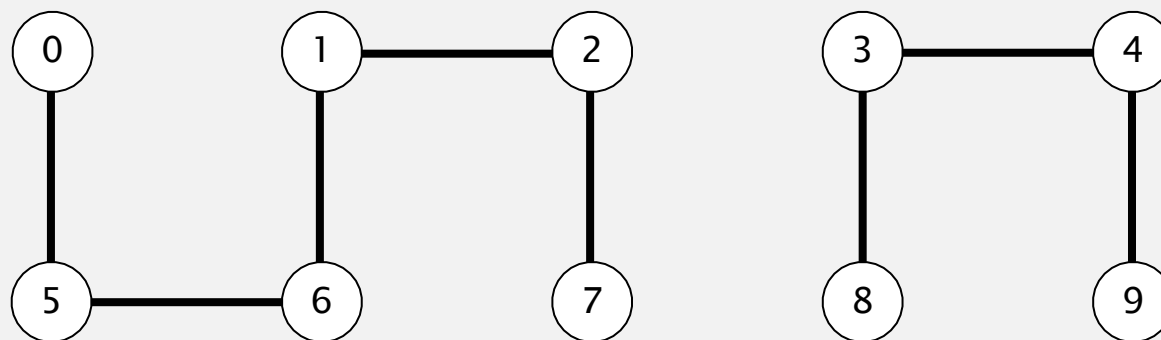


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	3	4	5	6	7	8	9



## 快速查找演示

---



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

## 快速查找。 Java实现

---

```
public class QuickFindUF
{
```

```
    私人int[] id。
```

```
    public QuickFindUF(int N)
    {
```

```
        id = new int[N];
        for (int i = 0; i < N; i++)
            id[i] = i。
```

← 将每个对象的id设为自己（N个数组访问）。

```
    }
```

```
    public boolean connected(int p, int q)
    { 返回 id[p] == id[q]; }
```

← 检查p和q是否是在同一个组件中（2个数组访问）。

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

```
        int pid = id[p];
        int qid = id[q];
        for (int i = 0; i < id.length; i++)
            if (id[i] == pid) id[i] = qid;
```

← 将所有id[p]的条目改为id[q]（最多2N+2个数组访问）。

```
    }
```

```
}
```





## 快速查找的速度太慢

---

成本模式。 阵列访问的数量（用于读或写）。

算法	初始化	联盟	发现
快速查找	N	N	1

阵列访问次数的增长顺序

联合的成本太高。

$N$  个对象上的  $N$  个联合命令。

二次方  
它需要  $N^2$  次数组访问来处理一连串的

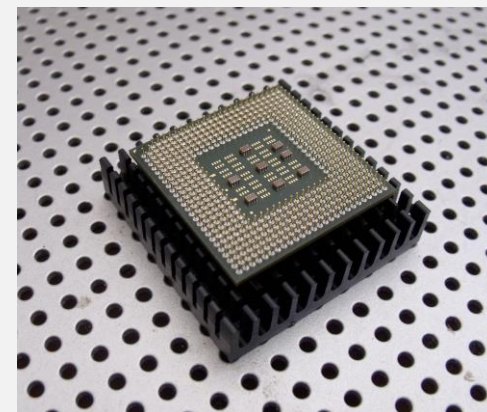


## 二次方算法没有规模

粗略的标准（目前）。

：每秒 $10^9$ 个操作。  
： $10^9$ 个字的主存储器。  
触摸所有的字，大约在1秒内。

自1950年以来，一个不争的事实（大致如此）！



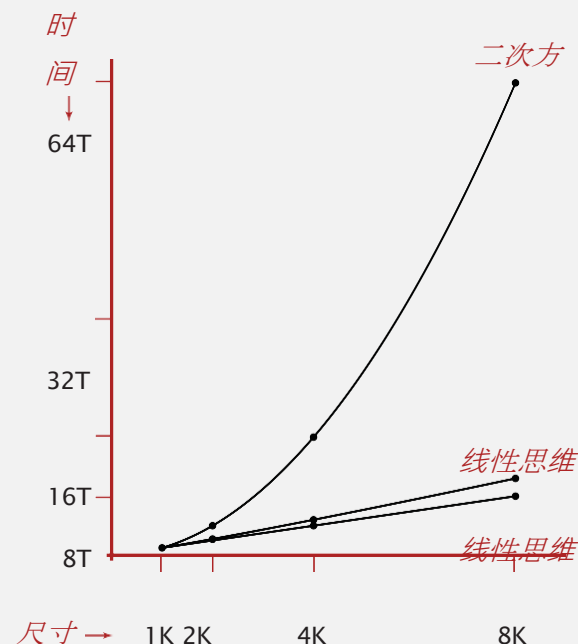
前。快速查找的 巨大问题。

- $10^9$ 个对象的联合命令。
- 快速查找需要 $10^{18}$ 次以上的操作。
- 30年以上的计算机时间！

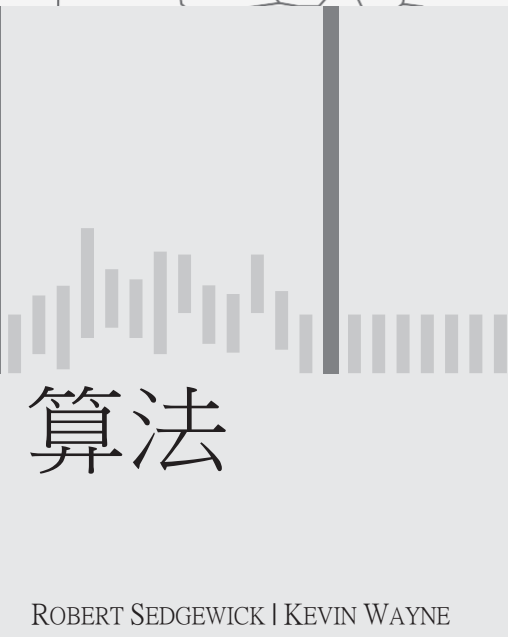
二次方算法不会随着技术的发展而扩展。

• 新的计算机可能有10倍的速度。  
但是，有10倍的内存  
想解决一个10倍大的问题。

遇到这种情况时，要花10倍的时间。







# 算法

ROBERT SEDGEWICK | KEVIN WAYNE

[http:// algs4.cs.princeton.edu](http://algs4.cs.princeton.edu)

## 1.5 联盟-寻找

---

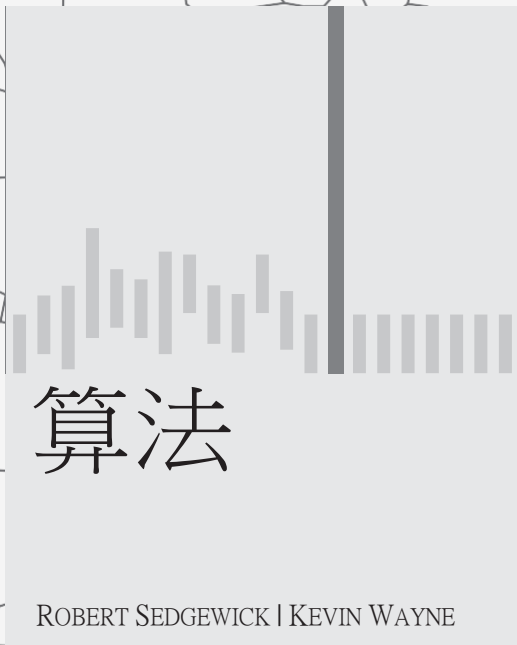
▶ 动态连接

▶ 快速找到

▶ 快速联合

▶ 改进

▶ 应用



# 算法

ROBERT SEDGEWICK | KEVIN WAYNE

[http:// algs4.cs.princeton.edu](http://algs4.cs.princeton.edu)

## 1.5 联盟-寻找

---

▶ 动态连接

▶ 快速找到

▶ 快速联合

▶ 改进

▶ 应用

## 快速结合[懒惰的方法]

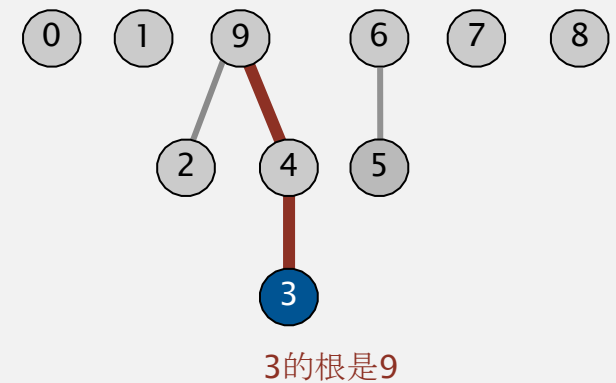
数据结构。

- 长度为N的整数数组id[]。
- 解释。 id[i]是i的父母。

一直到它没有变化为止（算法确保没有循环）。

i的根是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







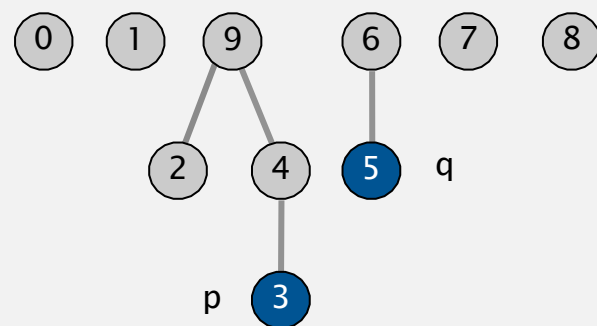
## 快速结合[懒惰的方法]

数据结构。

- 长度为N的整数数组id[]。
- 解释。  $\text{id}[i]$  是 i 的父母。

i 的根是  $\text{id}[\text{id}[\text{id}[\dots \text{id}[i] \dots]]]$ 。

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

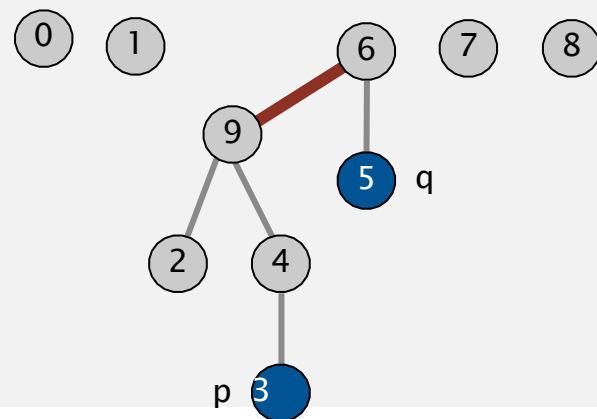


找到。 检查p和q是否有相同的根。

合并。 要合并包含p和q的组件，将p的根 id 设为 q 的根 id。

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

↑  
只有一个值改变





## 快速接头演示

---



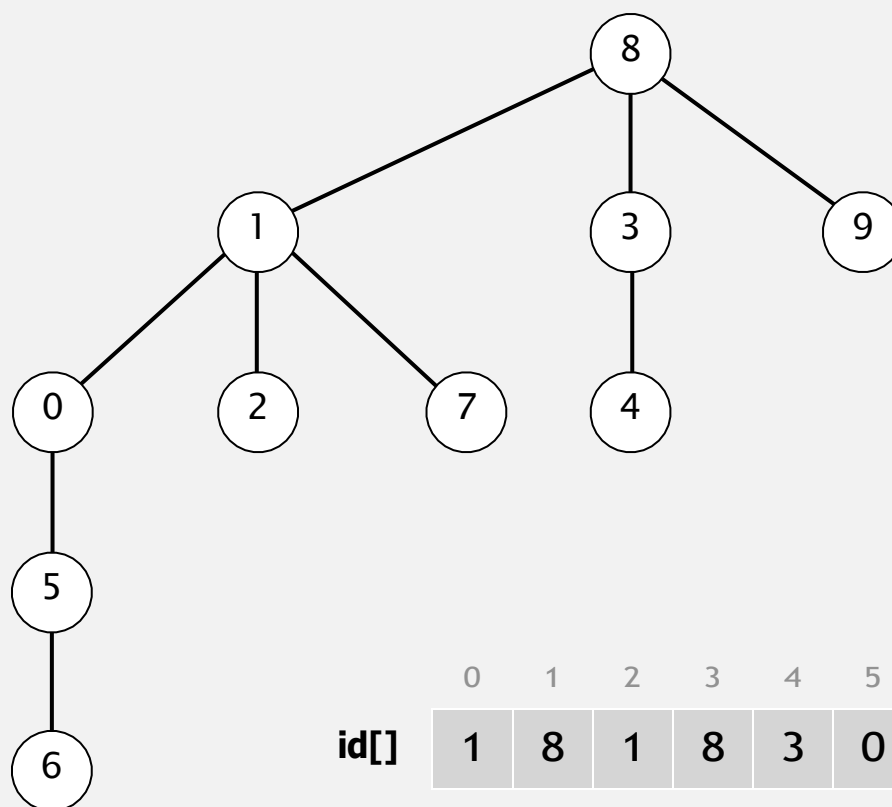
0 1 2 3 4 5 6 7 8 9

**id[]**



## 快速接头演示

---



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

## Quick-union。 Java实现

```
public class QuickUnionUF
{
```

```
    私人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];
```

```
        返回i。
```

```
    }
```

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

```
    {
```

```
        返回 root(p) == root(q)。
```

```
    }
```

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

```
    {
```

```
        int i = root(p);
```

```
        int j = root(q);
```

```
        id[i] = j;
```

```
    }
```

```
}
```

将每个对象的id设为自己（N个数组访问）。

追逐父指针，直到到达根（i数组访问的深度）。

检查p和q是否有相同的根（p和q数组访问的深度）。

将p的根指向q的根（p和q数组访问的深度）。

## 快速结合也太慢了

成本模式。 阵列访问的数量（用于读或写）。

算法	初始化	联盟	发现
快速查找	N	N	1
快速接头	N	N †	N

← 最坏的情况

†包括寻找根的费用

快速查找缺陷。

导致了在使用过程中出现的一些问题，例如，在使用过程中出现了一些问题，例如，在使用过程中出现了一些问题。

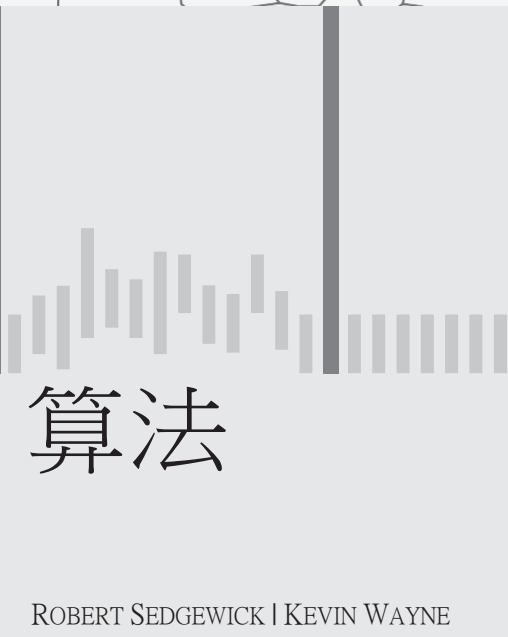
- 树木是平的，但要保持其平坦，成本太高。

快速结合的缺陷。

- 树木可以变得很高。

找到太贵了（可能是 $N$ 个数组访问）。





# 算法

ROBERT SEDGEWICK | KEVIN WAYNE

[http:// algs4.cs.princeton.edu](http://algs4.cs.princeton.edu)

## 1.5 联盟-寻找

---

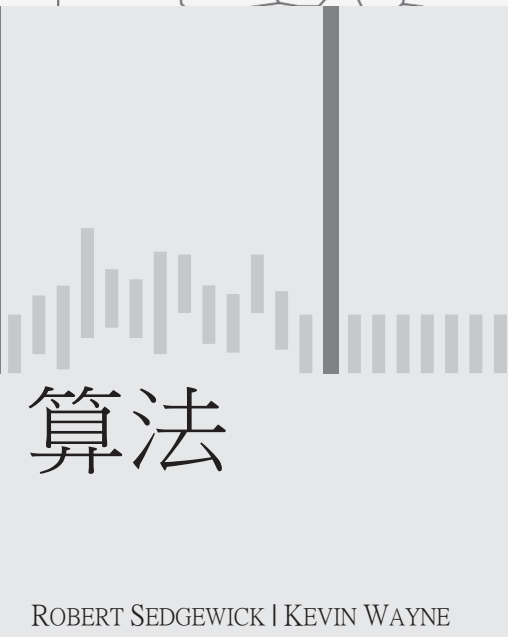
▶ 动态连接

▶ 快速找到

▶ 快速联合

▶ 改进

▶ 应用



# 算法

ROBERT SEDGEWICK | KEVIN WAYNE

[http:// algs4.cs.princeton.edu](http://algs4.cs.princeton.edu)

## 1.5 联盟-寻找

---

▶ 动态连接

▶ 快速找到

▶ 快速联合

▶ 改进

▶ 应用

## 改进1：加权

加权的快速接头。

- 修改快速接头，以避免高大的树木。
- 跟踪每棵树的大小（对象的数量）。
- 通过将小树的根部与大树根部连接起来进行平衡。

快速接头



合理的选择：按身高或 "等级" 联合

加权



## 加权快速接头演示

---



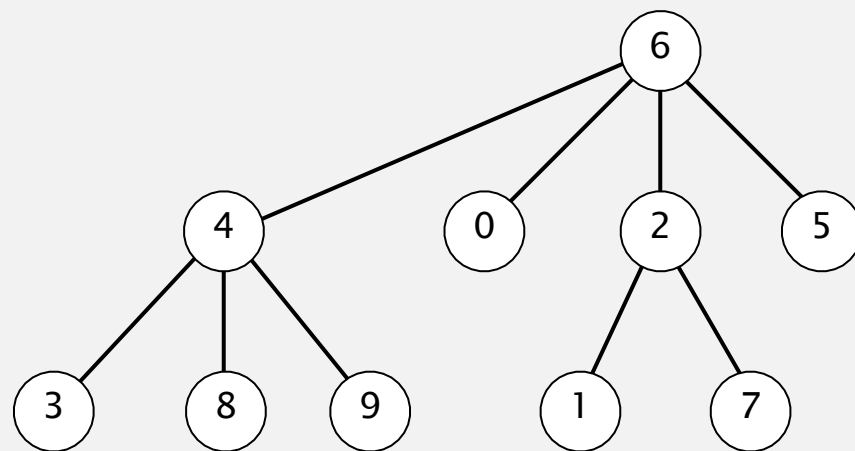
0 1 2 3 4 5 6 7 8 9

**id[]**

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

## 加权快速接头演示

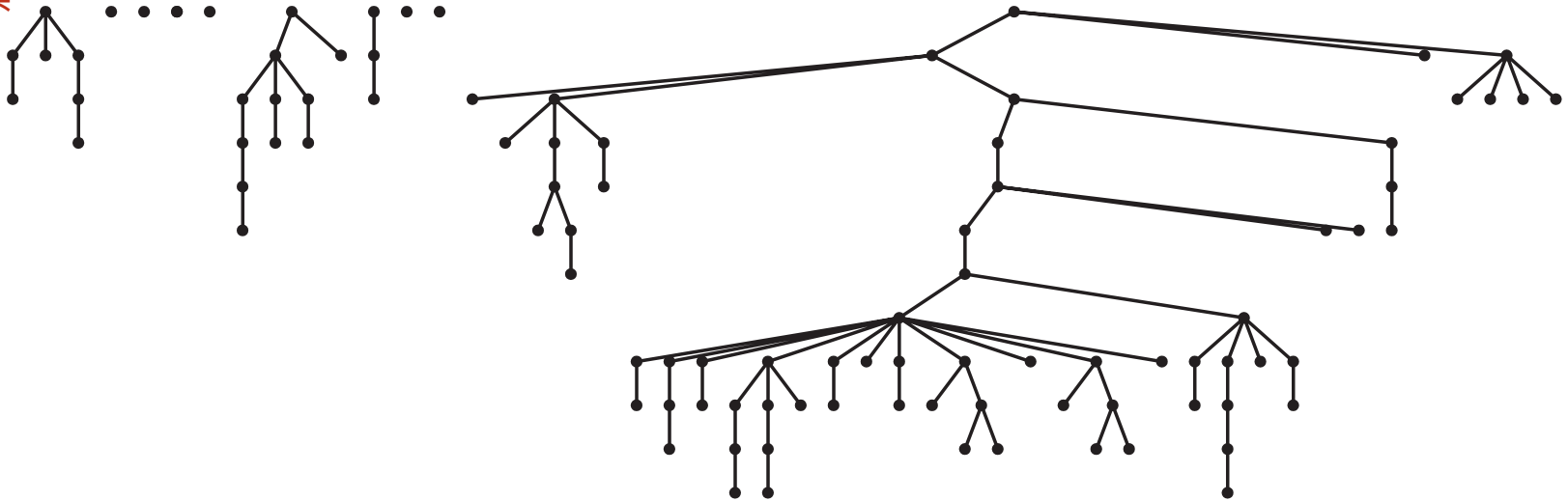
---



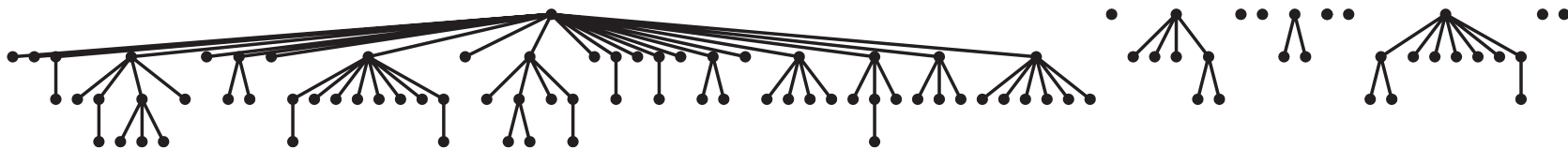
	0	1	2	3	4	5	6	7	8	9
id[]	6	2	6	4	6	6	6	2	4	4

## 快速连接和加权快速连接的例子

快速接头



加权



到根的平均距离。 5.11

到根的平均距离。 1.52

快速联合和加权快速联合（100个站点，88个联合（）操作）。





## 加权的快速联盟。 Java实现

---

**数据结构。** 与快速组合相同，但保持额外的数组`sz[i]`。  
来计算以`i`为根的树中的对象的数量。

**发现。** 与快速接头相同。

返回 `root(p) == root(q)`。

**联盟。** 将快速联盟修改为：

\*将小树的根与大树根相连。

- 更新`sz[]`数组。

```
int i = root(p);
```

```
int j = root(q);
```

如果 (`i == j`) 返回。

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

```
else{ id[j] = i; sz[i] += sz[j]; }
```



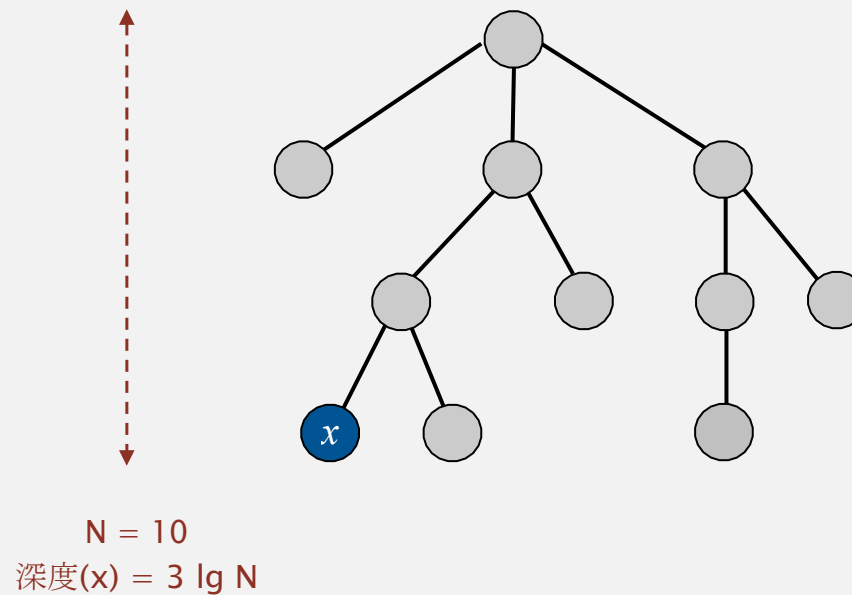
# 加权的快速结合分析

运行时间。

找到：需要的时间与 $p$ 和 $q$ 的深度成正比。  
\*联合：需要恒定的时间，给定根数。

命题。 任何节点 $x$ 的深度最多只有 $\lg N$ 。

$\lg$  = 以2为基数的对数





## 加权的快速结合分析

---

运行时间。

**找到**：需要的时间与 $p$ 和 $q$ 的深度成正比。

**\*联合**：需要恒定的时间，给定根数。

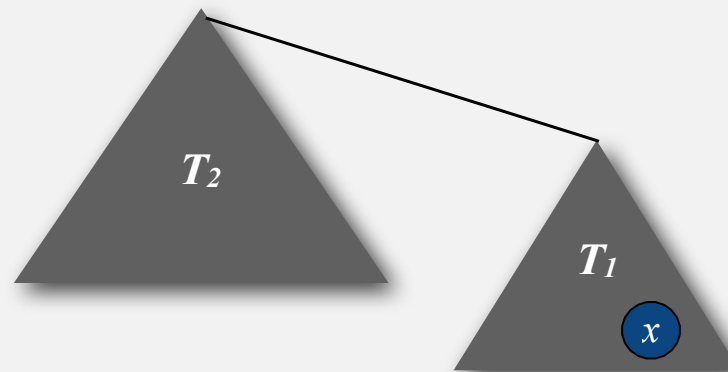
**命题。** 任何节点 $x$ 的深度最多为 $\lg N$ , **Pf.**  $x$

的深度何时增加？

当包含 $x$ 的树 $T_1$  被合并到另一棵树 $T_2$  时，增加1。

- 包含 $x$ 的树的大小至少增加一倍，因为 $T_2 \geq T_1$ 。

**\*包含 $x$ 的树的大小**最多可以翻倍 $\lg N$ 次。为什么？





## 加权的快速结合分析

---

运行时间。

**找到**：需要的时间与 $p$ 和 $q$ 的深度成正比。

**\*联合**：需要恒定的时间，给定根数。

**命题。** 任何节点 $x$ 的深度最多只有 $\lg N$ 。

算法	初始化	联盟	连接的
快速查找	$N$	$N$	<b>1</b>
快速接头	$N$	$N +$	$N$
加权的QU	$N$	$\lg N +$	$\lg N$

$+$ 包括寻找根的费用

**Q.** 停在保证可接受的性能上？

**A.** 没有，容易进一步改进。

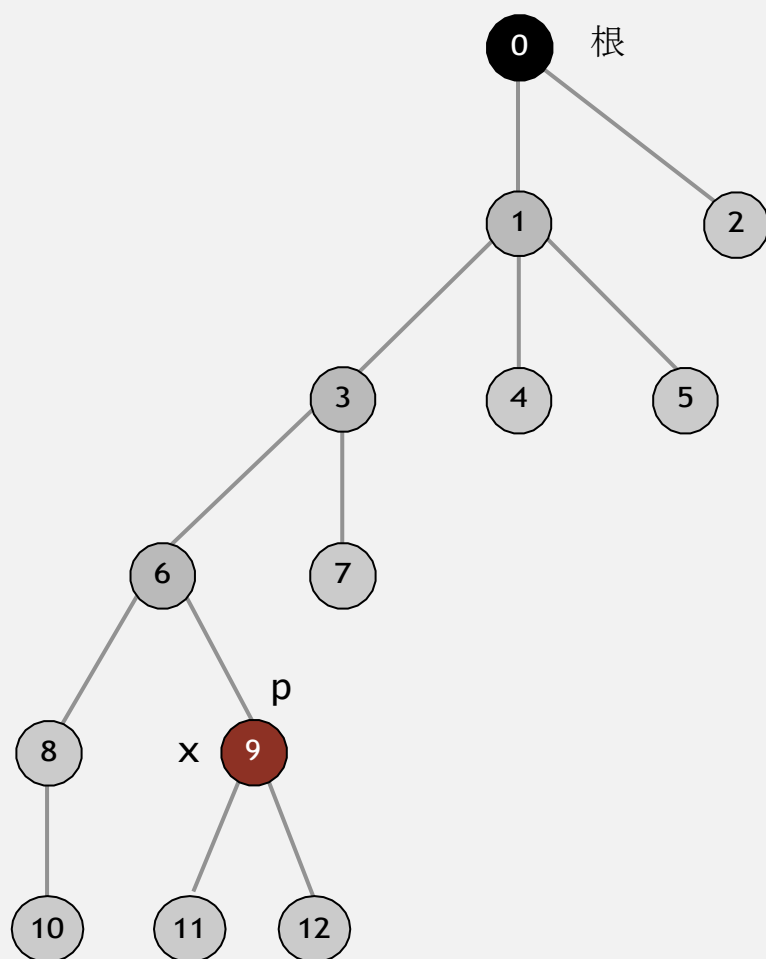




## 改进2：路径压缩

带路径压缩的快速联合。  
节点的id设置为指向该根。

在计算完 $p$ 的根之后，将每个被检查的

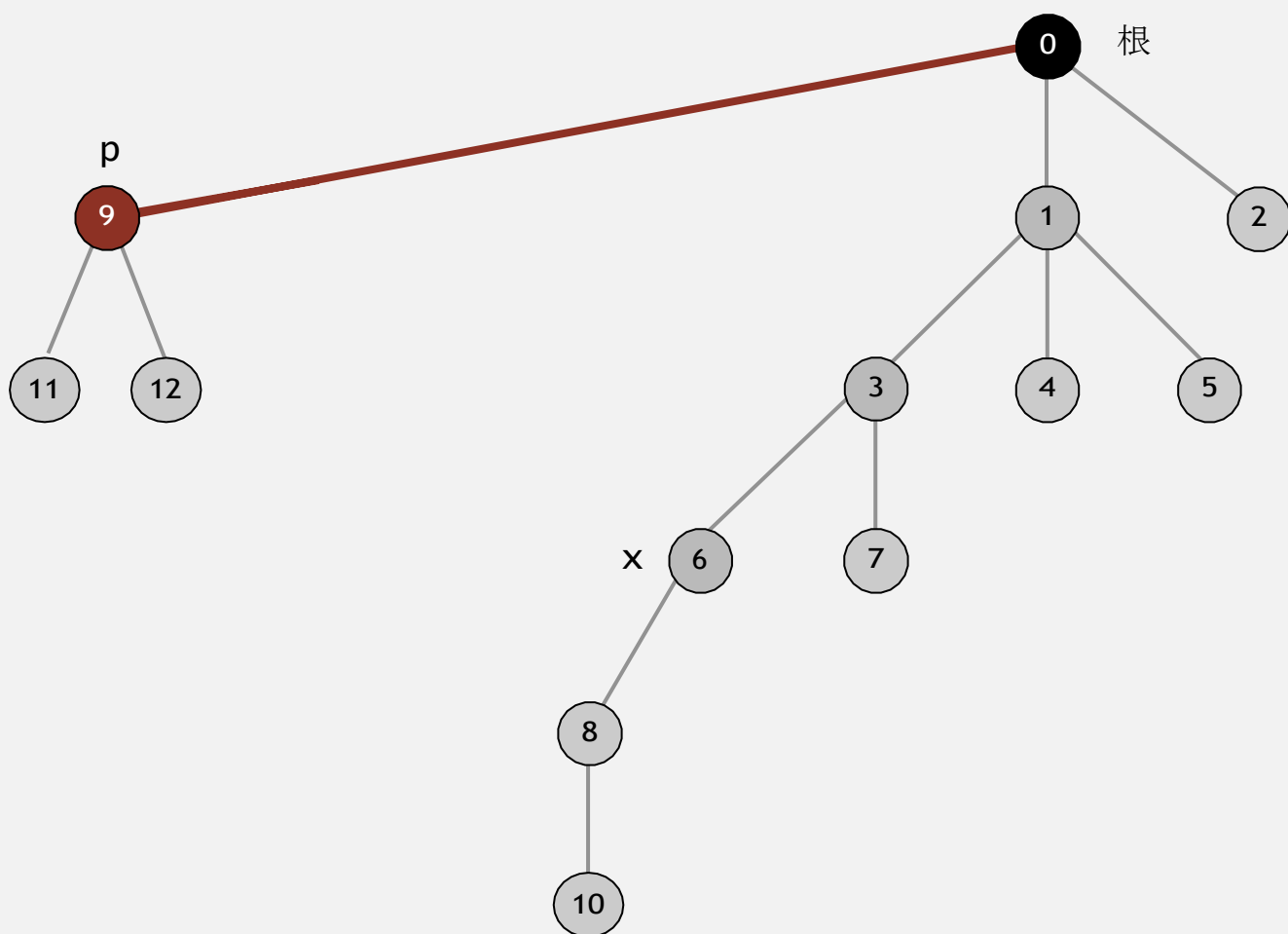




## 改进2：路径压缩

带路径压缩的快速联合。  
节点的id设置为指向该根。

在计算完 $p$ 的根之后，将每个被检查的



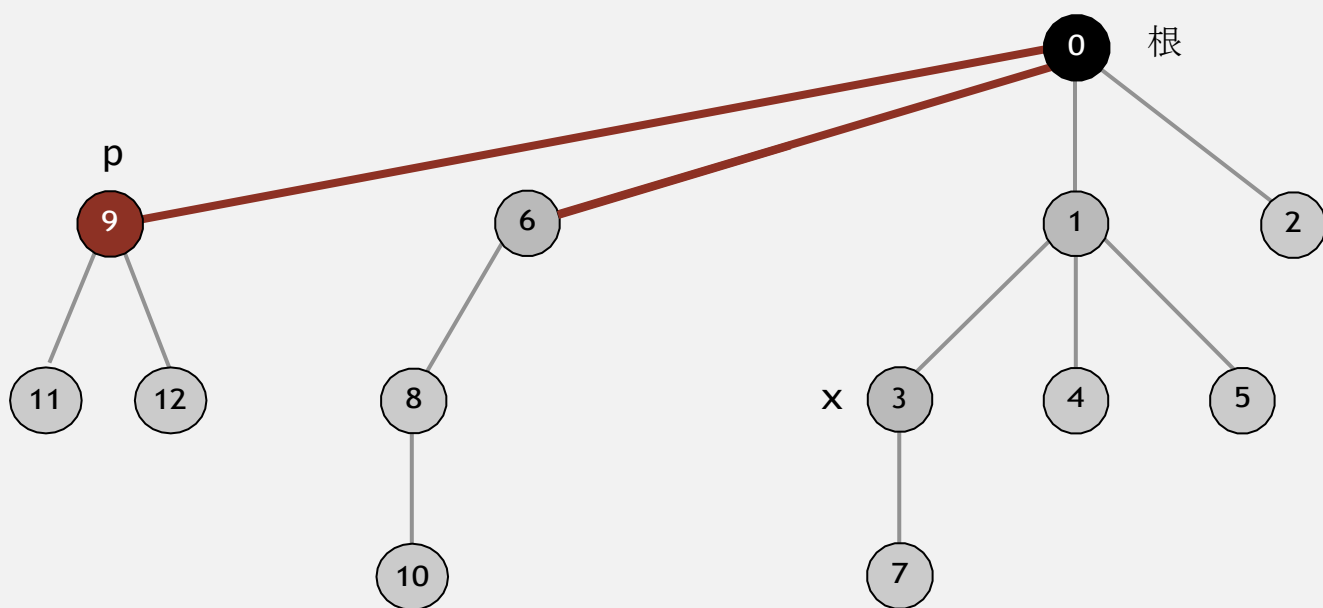


## 改进2：路径压缩

带路径压缩的快速联合。

节点的id设置为指向该根。

在计算完 $p$ 的根之后，将每个被检查的



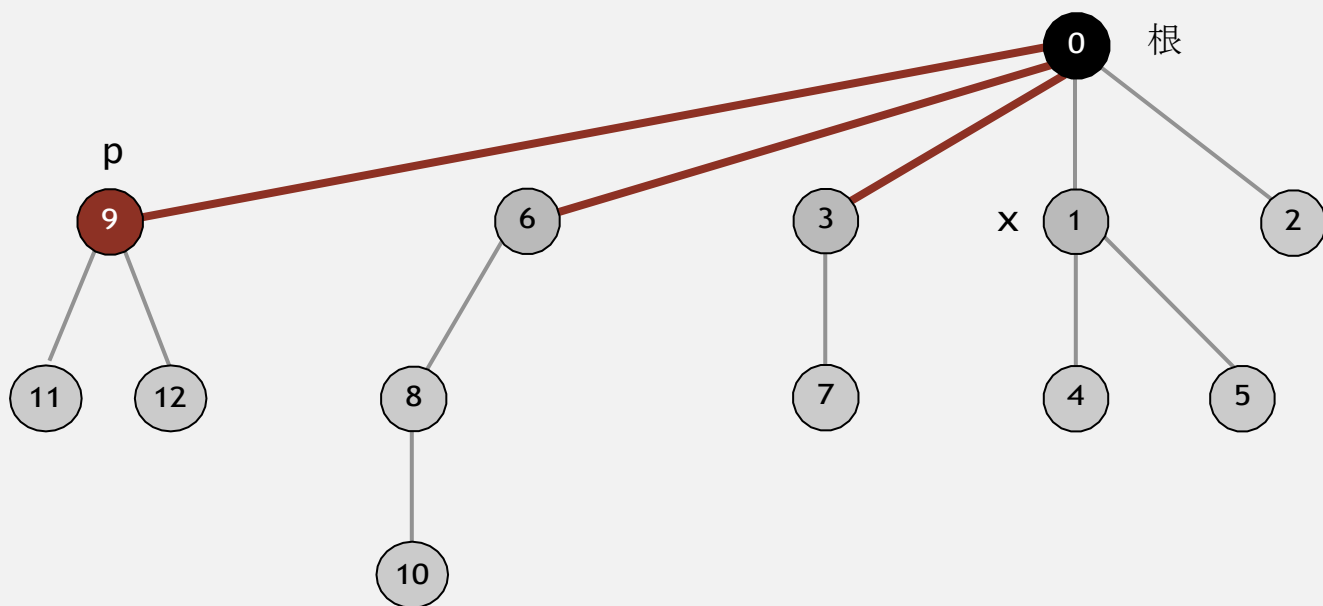


## 改进2：路径压缩

带路径压缩的快速联合。

节点的id设置为指向该根。

在计算完 $p$ 的根之后，将每个被检查的





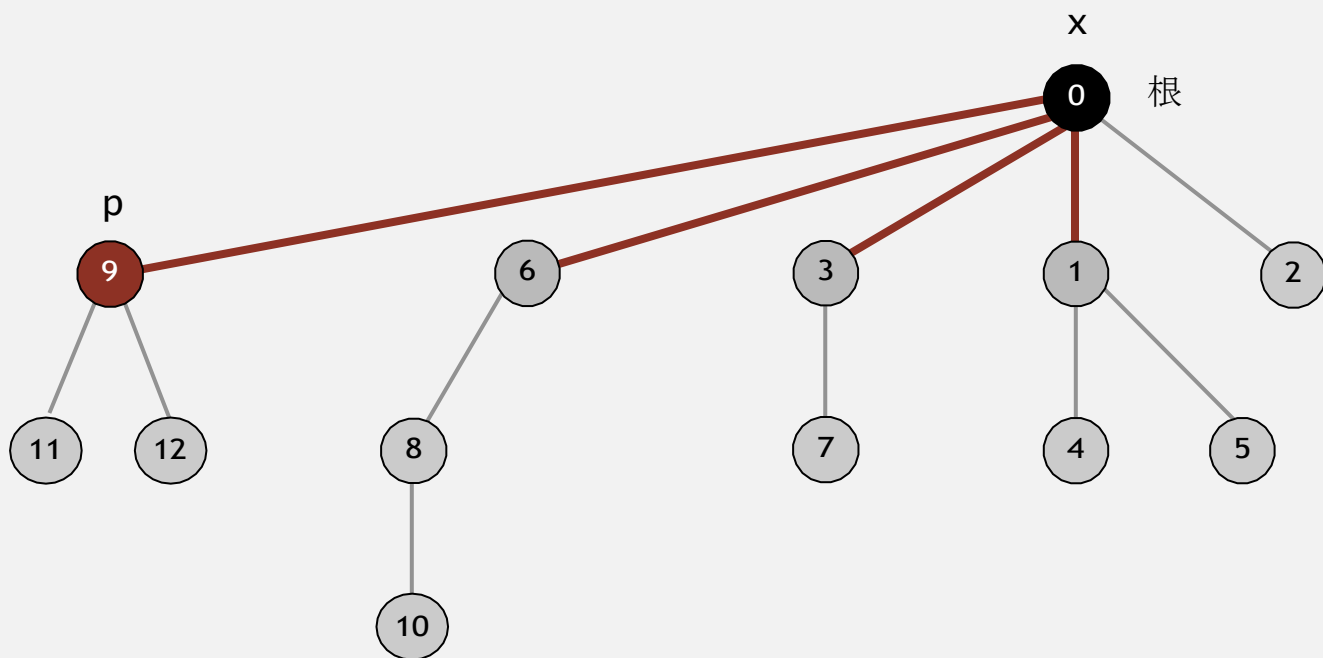


## 改进2：路径压缩

带路径压缩的快速联合。

节点的id设置为指向该根。

在计算完 $p$ 的根之后，将每个被检查的





## 路径压缩。Java实现

---

双通道的实现。

在root()中加入第二个循环，以设置id[]

的每个被检查的节点到根。

更简单的单程变体：

使路径中的每个其他节点都指向其祖先（从而

使路径长度减半）。

```
private int root(int i)
{
    while (i != id[i])
    {
        id[i] = id[id[i]].
        i = id[i];
    }
    返回i。
}
```

只多了一行代码!

在实践中。

没有理由不这样做!使树几乎完全平坦。



## 带路径压缩的加权快速联合：摊销分析

**命题。** [Hopcroft-Ulman, Tarjan] 从一个空的数据结构开始，任何在 $N$ 个对象上的 $M$ 个联合查找操作序列都会产生 $\leq c(N + M \lg^* N)$ 的数组访问。

- **分析**可以改进为 $N+M$  ( $M, N$ )。
- **简单的**算法，迷人的数学。

N	$\lg^* N$
1	0
2	1
4	2
16	3
65536	4
265536	5

迭代对数函数

$N$ 个对象上的 $M$ 个联合查找操作的线性时间算法？

- 在读入数据的恒定系数内的**成本**。
- **在**理论上，WQUPC不完全是线性的。

在实践中，WQUPC是线性的。



令人惊讶的事实。[Fredman-Saks] 不在线性时间算法。

在 "细胞-探针 "的计算模型中

## 摘要

---

一句话。 加权快速联合（有路径压缩）使解决其他方式无法解决的问题成为可能。

算法	最坏情况下的时间
快速查找	$M N$
快速接头	$M N$
加权的QU	$N + M \log N$
QU + 路径压缩	$N + M \log N$
加权QU+路径压缩	$N + M \lg^* N$

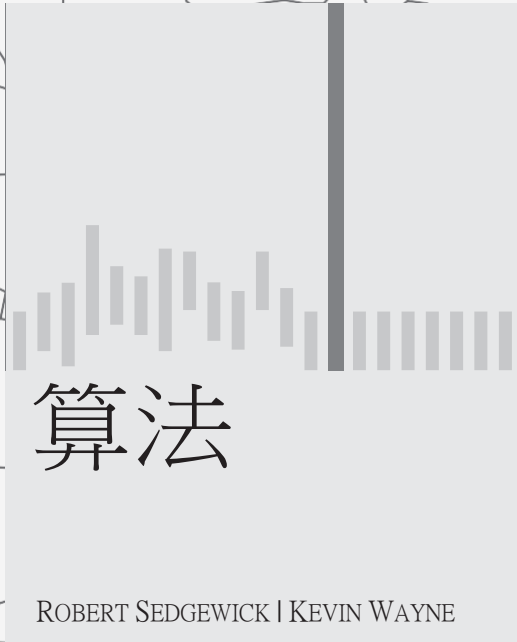
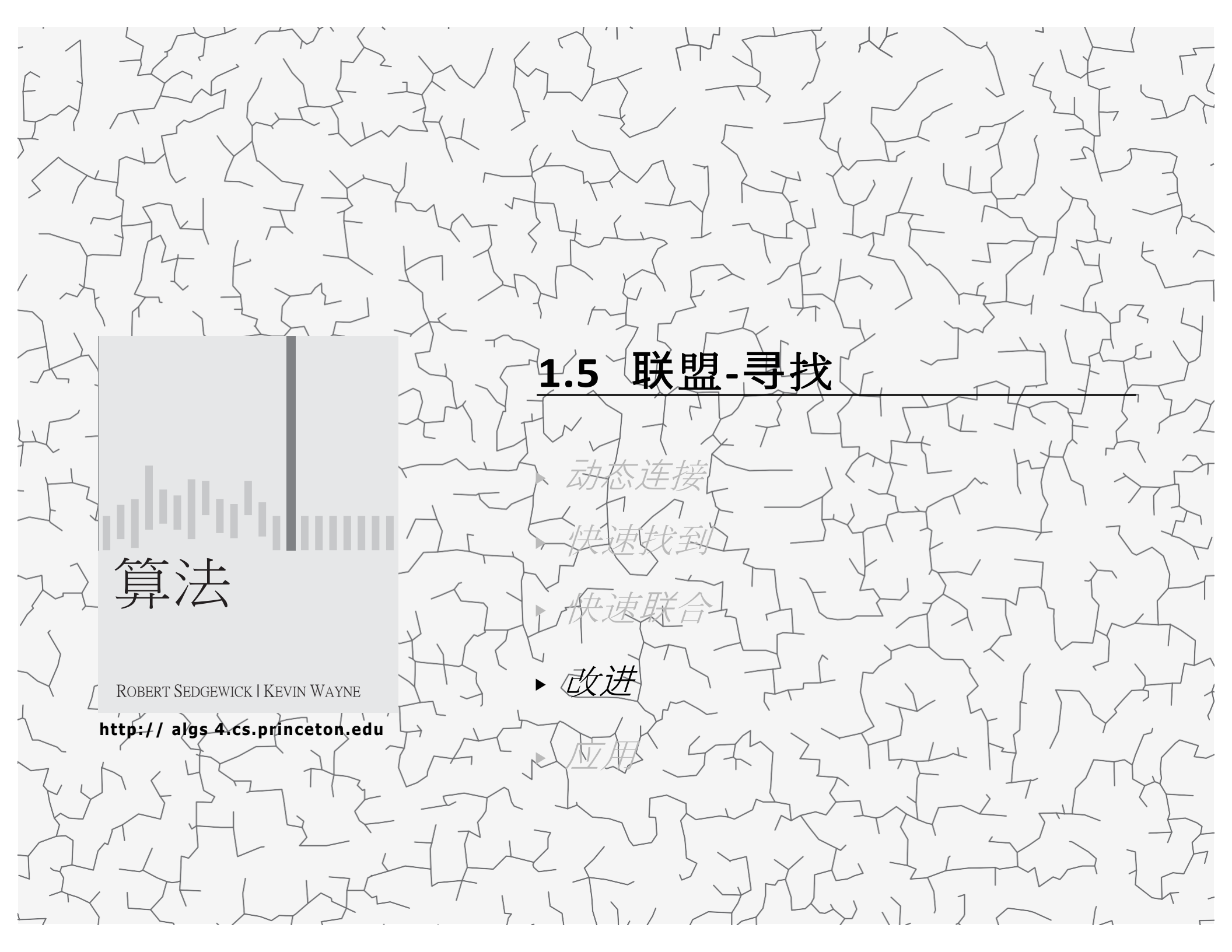
对**N**个对象的集合进行**M**个联合查找操作

前。 [10<sup>9</sup>个联盟和10<sup>9</sup>个对象的发现]

- WQUPC将时间从30年减少到6秒。
- 超级计算机不会有什么帮助；好的算法可以解决问题。







# 算法

ROBERT SEDGEWICK | KEVIN WAYNE

[http:// algs4.cs.princeton.edu](http://algs4.cs.princeton.edu)

## 1.5 联盟-寻找

---

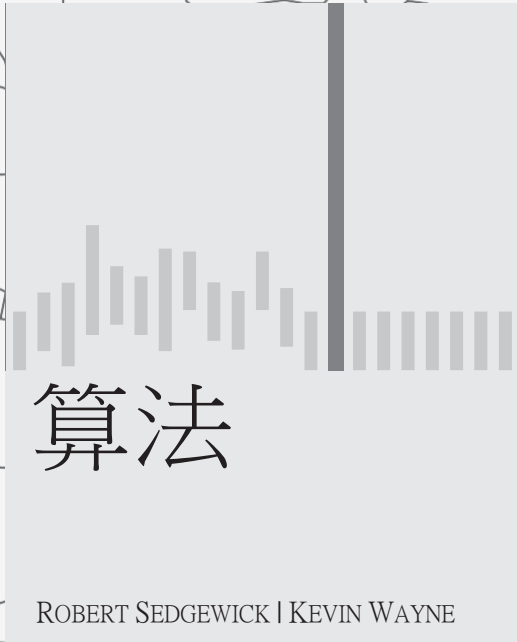
▶ 动态连接

▶ 快速找到

▶ 快速联合

▶ 改进

▶ 应用



# 算法

ROBERT SEDGEWICK | KEVIN WAYNE

[http:// algs4.cs.princeton.edu](http://algs4.cs.princeton.edu)

## 1.5 联盟-寻找

---

▶ 动态连接

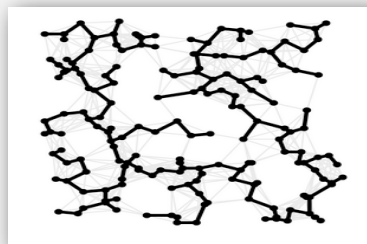
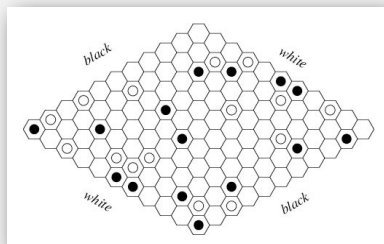
▶ 快速找到

▶ 快速联合

▶ 改进

▶ 应用

- 颠簸。
- 游戏（围棋、六角）。
- ✓ 动态连接。
- 最小的共同祖先。
- \*有限状态自动机的等价性。
- 物理学中的Hoshen-Kopelman算法。
- Hinley-Milner多态类型推断。
- 克鲁斯卡的最小生成树算法。
- \*在Fortran中编译等效语句。
- 形态属性的开放和关闭。
- \*Matlab的bwlabel()函数在图像处理中的应用。

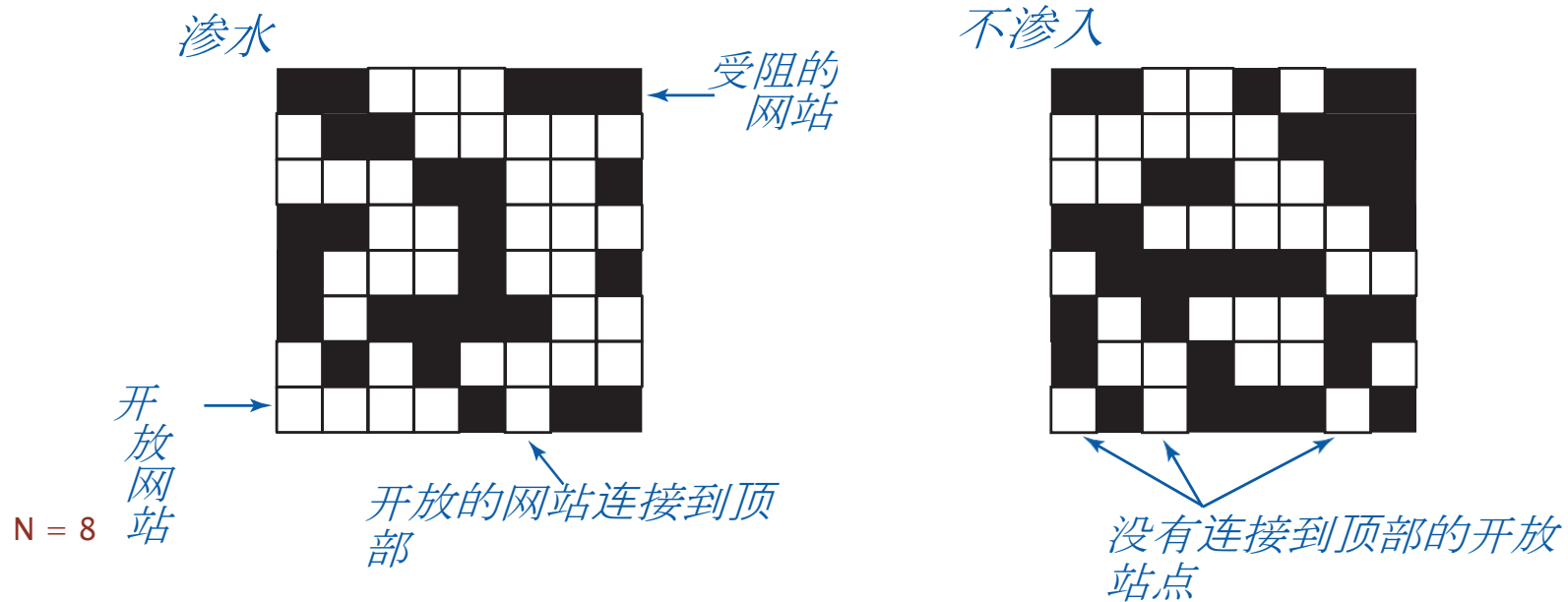


# 渗滤

许多物理系统的模型。

- $N$ 乘 $N$ 的站点网格。
- 每个站点以概率 $p$ 开放（或以概率 $1-p$ 封锁）。

\*如果顶部和底部由开放站点连接，系统就会出现渗滤。





# 渗滤

---

许多物理系统的模型。

- $N$ 乘 $N$ 的站点网格。
- 每个站点以概率 $p$ 开放（或以概率 $1-p$ 封锁）。

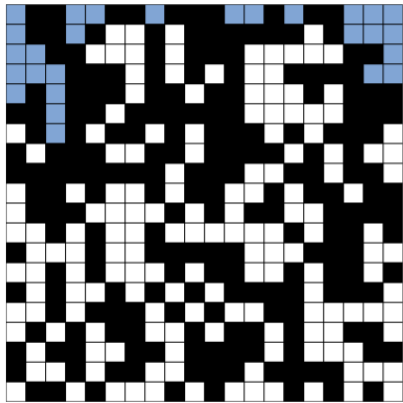
\*如果顶部和底部由开放站点连接，系统就会出现渗滤。

模型	系统	空置场地	占用场地	渗水
电力	材料	指挥者	绝缘的	进行
流体流动	材料	空的	受阻的	多孔的
社会互动	人口	人	空的	沟通

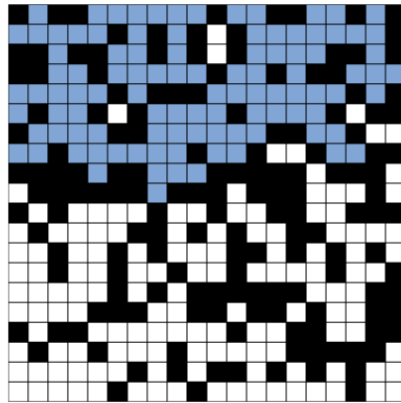


## 渗入的可能性

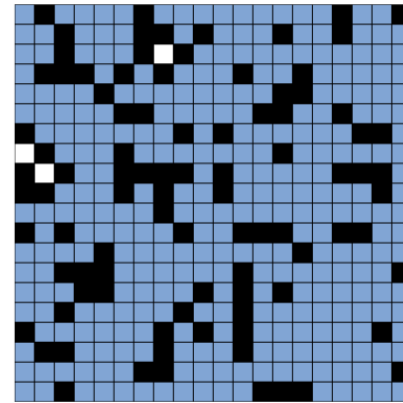
取决于现场的空缺概率 $P$ 。



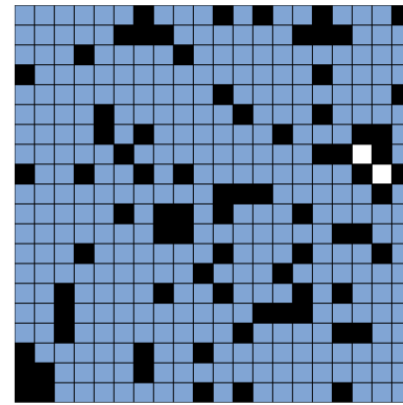
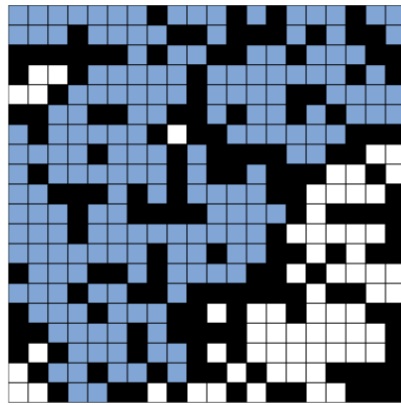
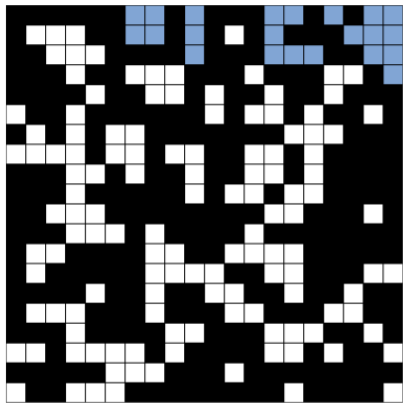
p低 (0.4)  
不渗入



p 中等 (0.6)  
渗出？



p高(0.8)  
渗水



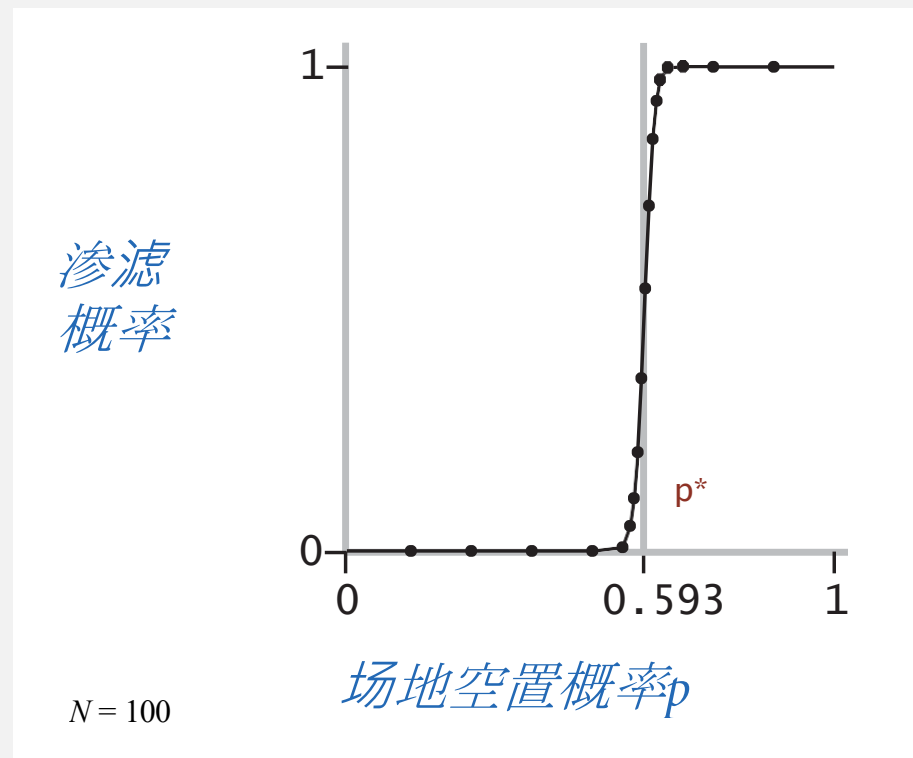


## 渗滤相变

当 $N$ 很大时，理论上保证了一个尖锐的阈值 $P^*$ 。

- $p < p^*$ ：几乎可以肯定是渗入的。
- $p > p^*$ ：几乎可以肯定不会渗漏。

Q.  $p^*$ 的值是多少？

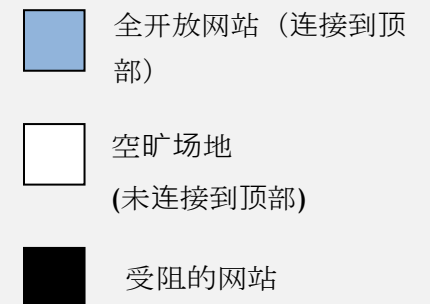
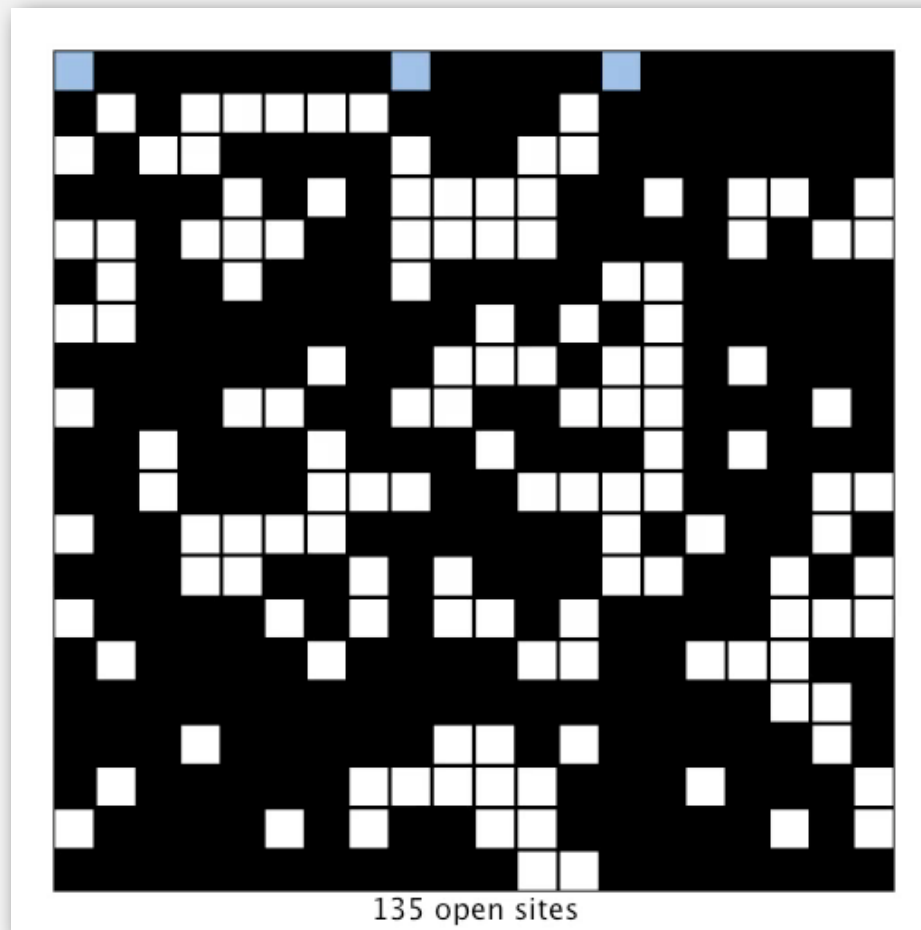




## 蒙特卡洛模拟

\*初始化 $N$ 乘 $N$ 的整个网格，以进行封锁。

- 宣布随机站点开放，直到顶部连接到底部。
- 空缺率估计  $p^*$ 。



$N = 20$

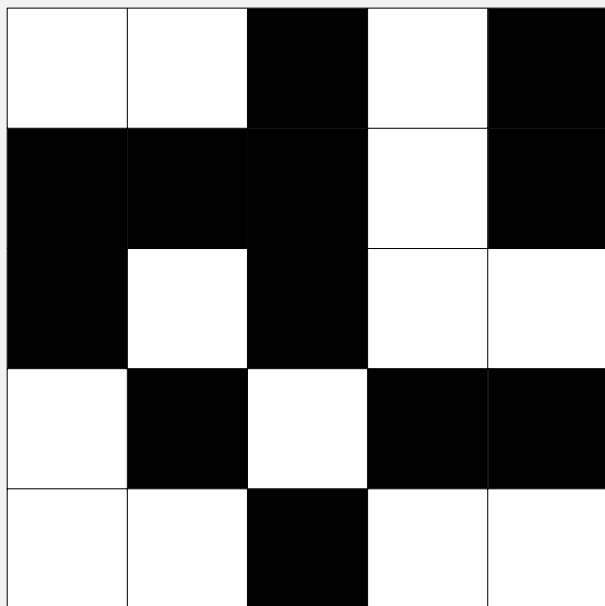


## 估计渗滤阈值的动态连接方案

---

Q. 如何检查一个 $N$ 乘 $N$ 的系统是否有渗漏？

$N = 5$



开放网站



受阻的网站

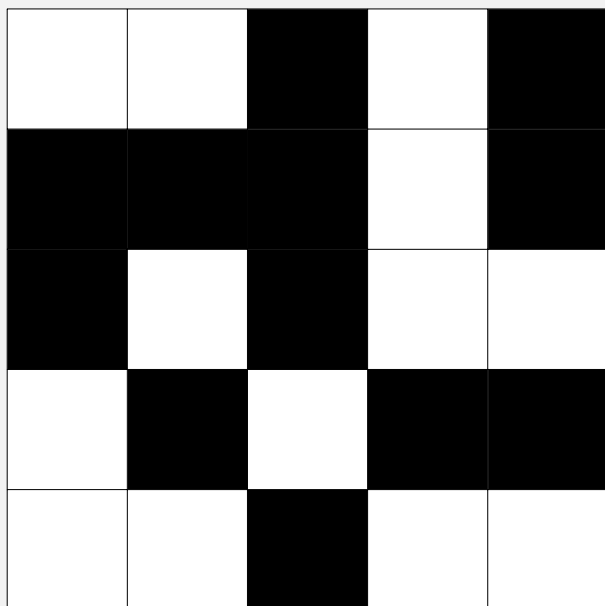


## 估计渗滤阈值的动态连接方案

Q. 如何检查一个 $N$ 乘 $N$ 的系统是否有渗漏？

- 为每个站点**创建**一个对象，并将其命名为0至 $N^2 - 1$ 。

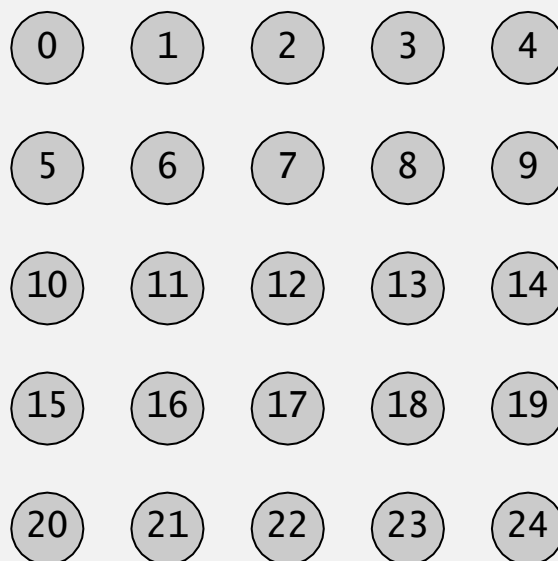
$N = 5$



开放网站



受阻的网站







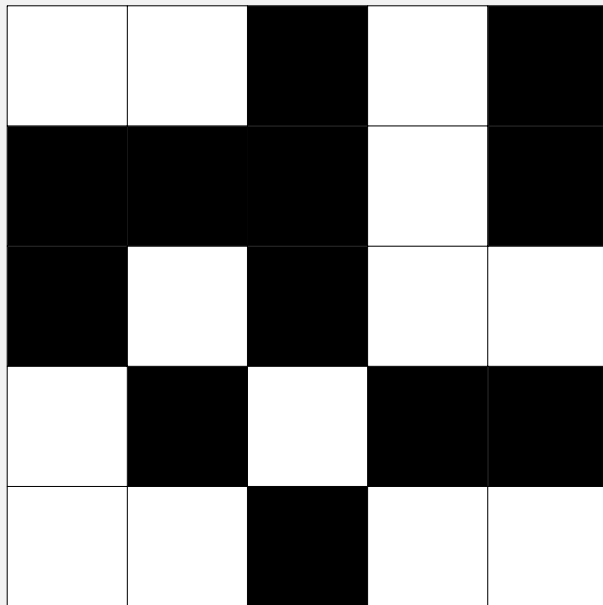
## 估计渗滤阈值的动态连接方案

Q. 如何检查一个 $N$ 乘 $N$ 的系统是否有渗漏？

- 为每个站点**创建**一个对象，并将其命名为0至 $N^2 - 1$ 。

如果通过开放的站点连接，则**站点**属于同一组件。

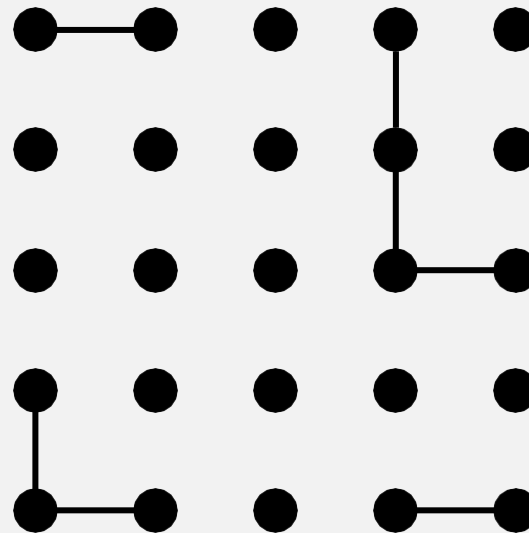
$N = 5$



开放网站



受阻的网站





## 估计渗滤阈值的动态连接方案

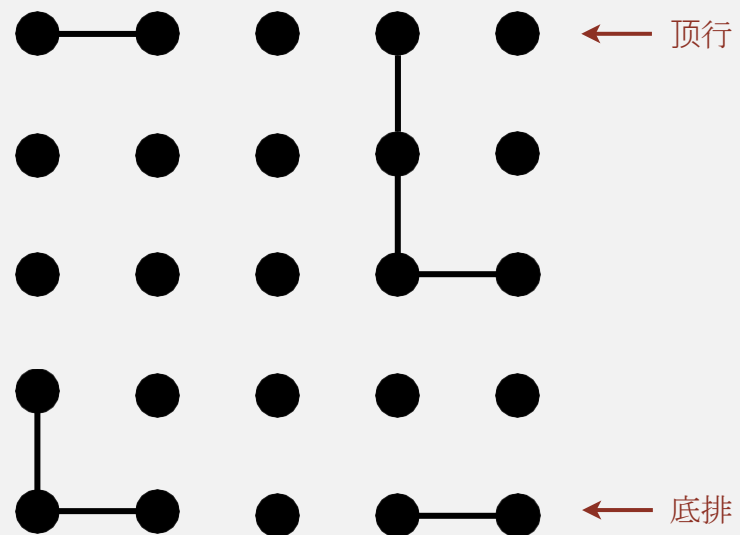
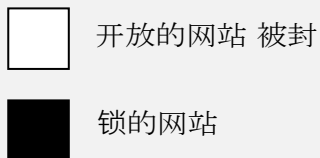
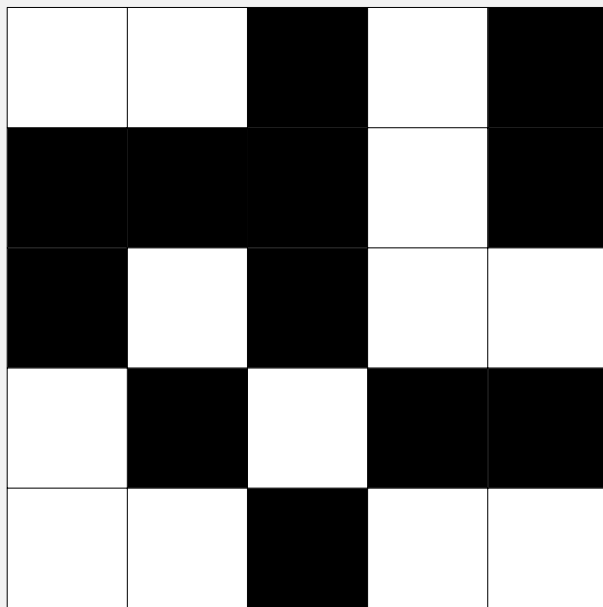
Q. 如何检查一个 $N$ 乘 $N$ 的系统是否有渗漏？

- 为每个站点**创建**一个对象，并将其命名为0至 $N^2 - 1$ 。

如果通过开放的站点连接，则**站点**属于同一组件。

如果下行的任何一个站点与上行的站点相连，则该站点将被**折叠**。  
暴力算法。对**connected()**的 $N$ 次调用

$N = 5$



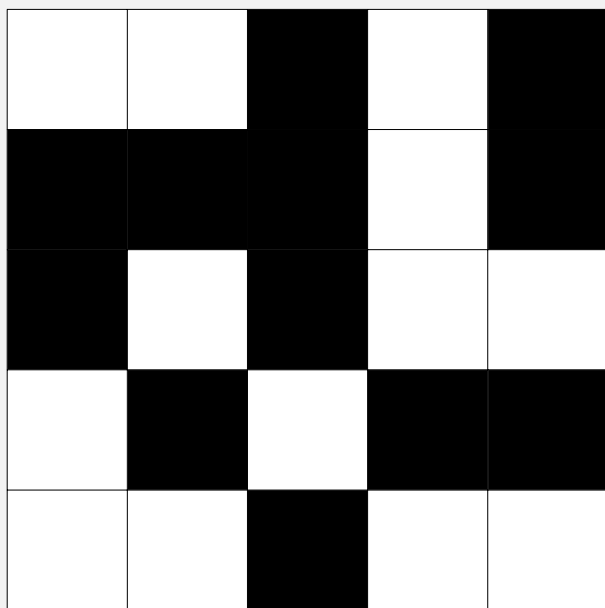
## 估计渗滤阈值的动态连接方案

巧妙的技巧。 引入2个虚拟站点（以及与顶部和底部的连接）。

- 如果虚拟顶点与虚拟底点相连，就会出现折叠。

高效的算法：只需调用`connected()`即可。

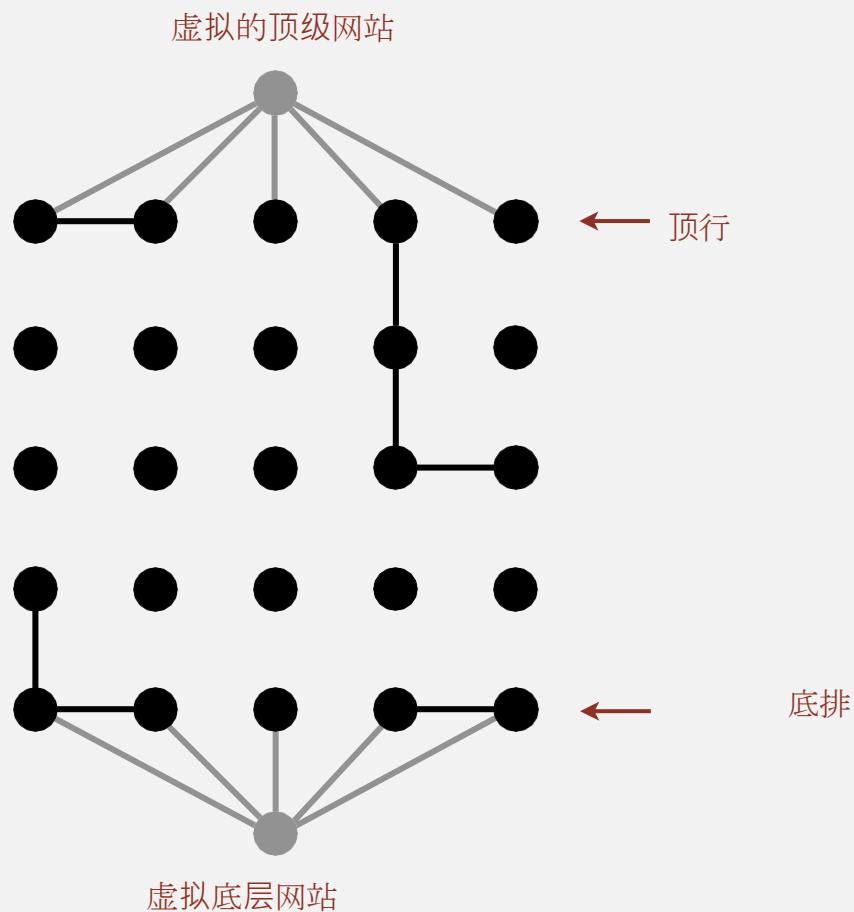
$N = 5$



开放网站



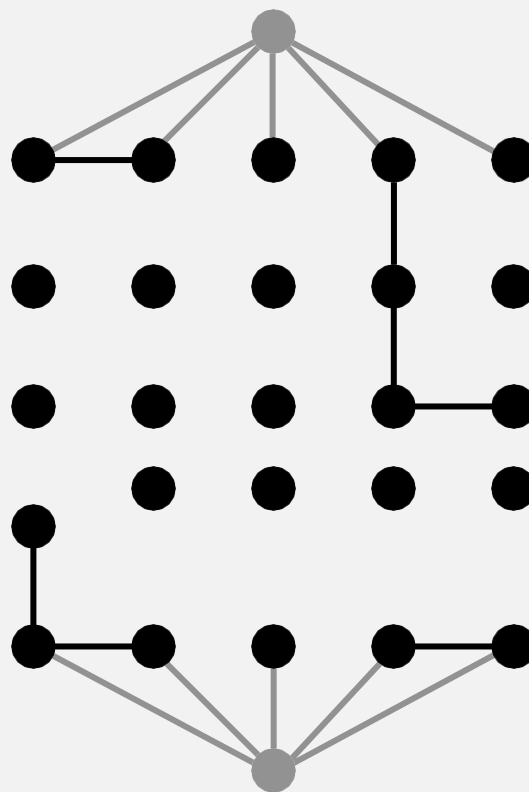
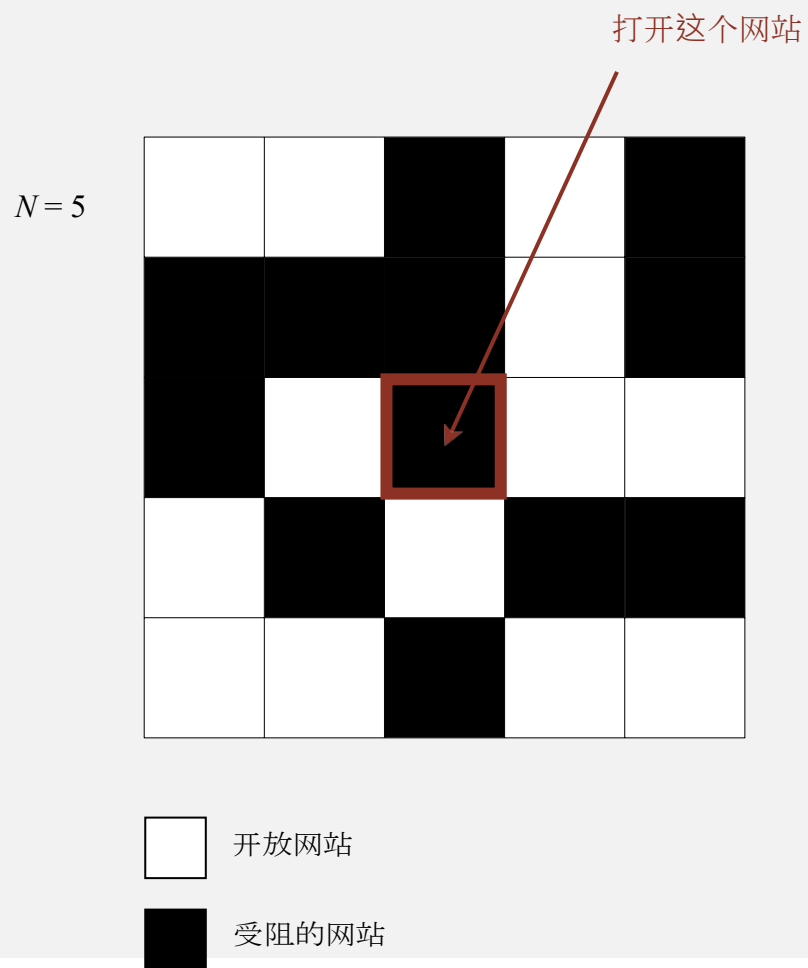
受阻的网站





# 估计渗透阈值的动态连接方案

Q. 如何建立一个新网站的模型？





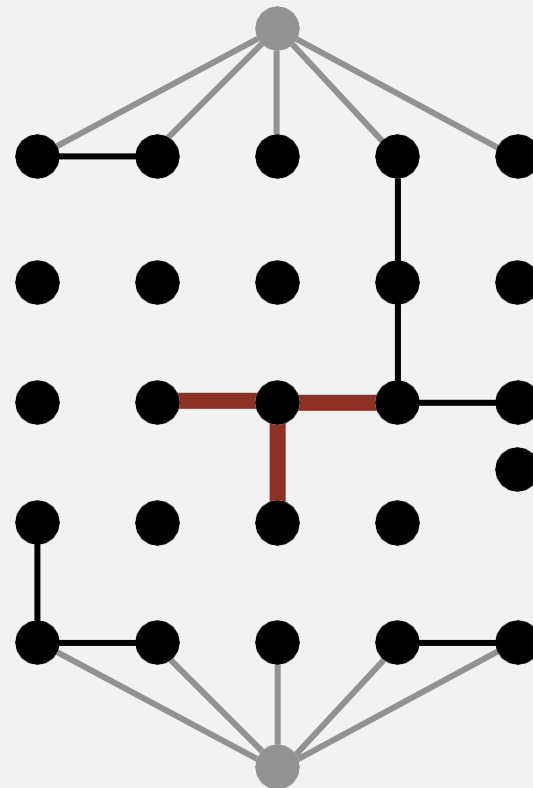
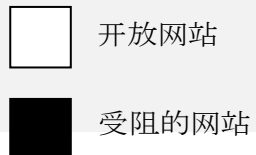
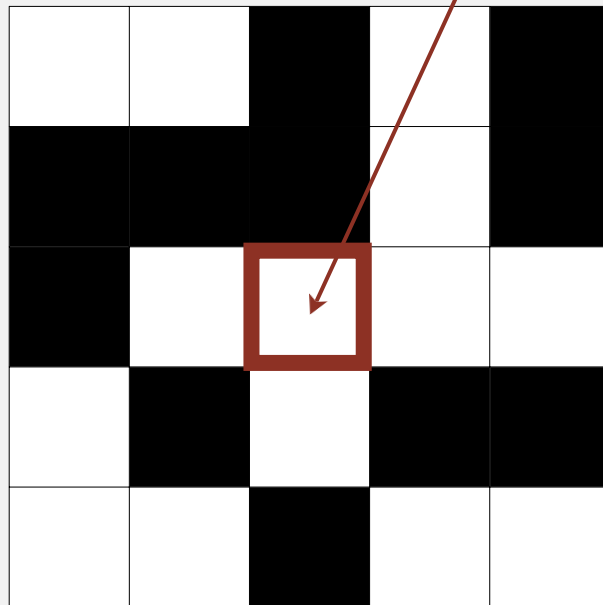
## 估计渗滤阈值的动态连接方案

Q. 如何建立一个新网站的模型？

A. 将新场地标记为开放场地；将其与所有相邻的开放场地相连。

最多调用4次union()。

$N = 5$





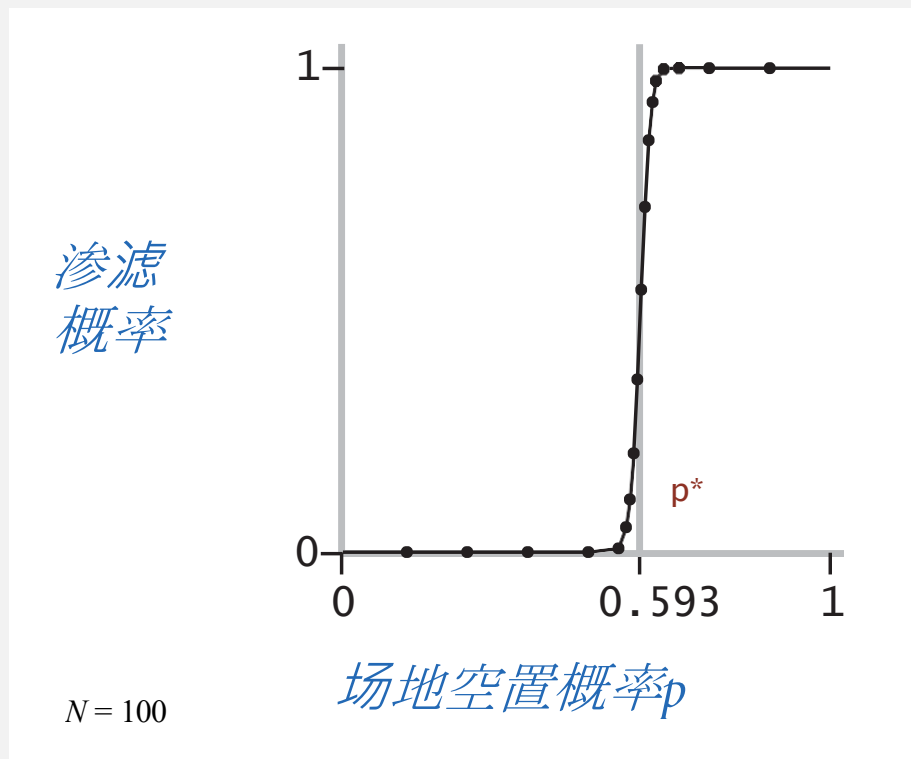


## 渗滤阈值

Q. 什么是渗滤阈值 $p^*$ ?

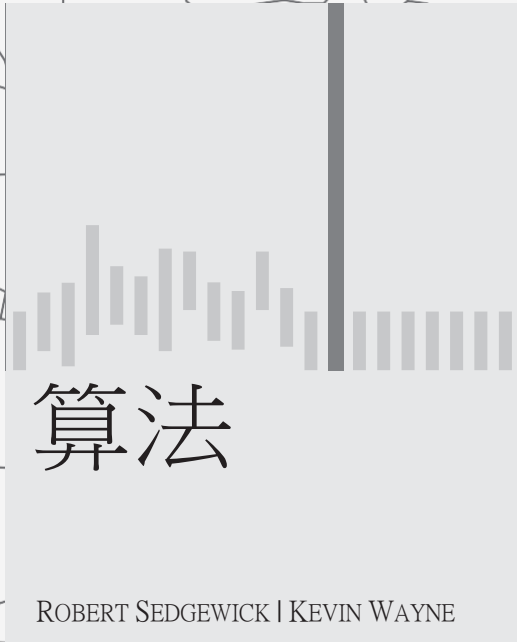
A. 对于大的正方形格子，大约是0.592746。

仅通过模拟知道的常数



快速的算法能够准确回答科学问题。





# 算法

ROBERT SEDGEWICK | KEVIN WAYNE

[http:// algs4.cs.princeton.edu](http://algs4.cs.princeton.edu)

## 1.5 联盟-寻找

---

- ▶ 动态连接
- ▶ 快速找到
- ▶ 快速联合
- ▶ 改进
- ▶ 应用

## 今天讲座的潜台词（以及这个课程）

---

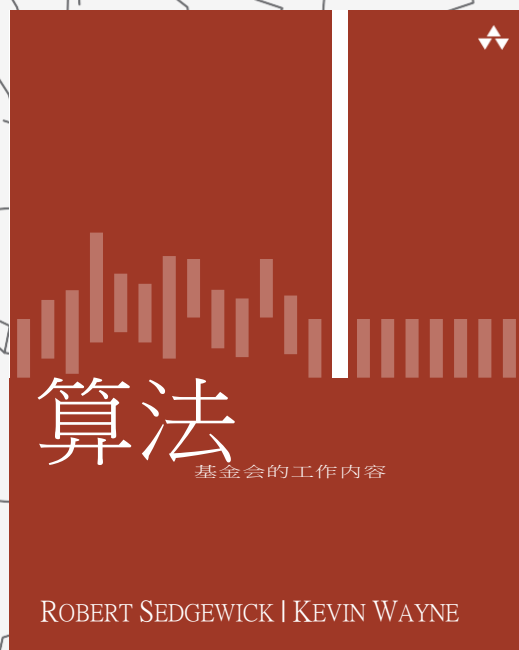
开发一个可用的算法的步骤。

- 建立问题的模型。
- 找到一种算法来解决它。
- 够快吗？能否装入内存？
- 如果没有，请找出原因。
- 找到一个解决问题的方法。
- 迭代直到满意为止。

科学方法。数学分析。

# 算法

罗伯特-塞奇威克 | 凯文-韦恩



<http://algs4.cs.princeton.edu>

## 1.5 联盟-寻找

---

- ▶ 动态连接
- ▶ 快速找到
- ▶ 快速联合
- ▶ 改进
- ▶ 应用