L13：并查集

吉林大学计算机学院
谷方明
fmgu2002@sina.com

学习目标

- 了解等价类问题；
- 掌握并查集的定义、操作及实现
- 掌握按秩合并 和 路径压缩
- 了解并查集的效率分析

例题：亲戚

- 若某个家族人员过于庞大，要判断两个是否是亲戚，确实还很不容易。现在给出某个亲戚关系图，求任意给出的两个人是否具有亲戚关系。

- 规定：x和y是亲戚，y和z是亲戚，那么x和z也是亲戚。即如果x,y是亲戚，那么x的亲戚都是y的亲戚，y的亲戚也都是x的亲戚。
- 亲戚关系是一种等价关系

数据输入：

- 第一行：三个整数n,m,p，（n<=5000,m<=5000,p<=5000），分别表示有n个人，m个亲戚关系，询问p对亲戚关系。
- 以下m行：每行两个数Mi，Mj，1<=Mi，Mj<=N，表示Mi和Mj具有亲戚关系。
- 接下来p行：每行两个数Pi，Pj，询问Pi和Pj是否具有亲戚关系。

数据输出

- p行，每行一个’Yes’或’No’。表示第i个询问的答案为“具有”或“不具有”亲戚关系。

样例

- input.txt

6 4 3
1 2
1 3
5 4
5 3

1 4
2 3
5 6

- output.txt

Yes
Yes
No

等价类方法

- 亲戚关系是一个等价关系；把人的集合划分成若干个等价类；

- 初始时，每个等价类都是一个人
- 每次遇到两个人有亲戚关系，就把两个人所在的等价类合并
- 查询两个人是否有亲戚关系时，就是查询两个人是否属于同一等价类

并查集

- 并查集用于维护一些不相交集合，
  $S = \{ S_1, S_2 , \dots , S_r \}$。

- 主要操作
  - ✓ UNION(x,y)：两个集合合并；
  - ✓ FIND (x) ：查询某个元素所在的集合；

并查集的操作

- 集合代表元：每个集合S_i都有一个特殊元素rep[S_i];
- **MAKE_SET(x):** 初始化x为单元素集
- **UNION(x, y):** 把x和y所在的两个不同集合合并。相当于从S中删除S_x和S_y并加入 $S_x \cup S_y$
- **FIND(x):** 返回x所在集合S_x的代表rep[S_x]

并查集的实现——顺序存储

- 每个集合用一个长度为**n**的数组表示
  - ✓ 空间：O(n^2)
- 操作的实现及时间效率分析
  - ✓ 查找：O(n)
  - ✓ 合并：O(n)
- 标识数组？

并查集的实现——链式存储

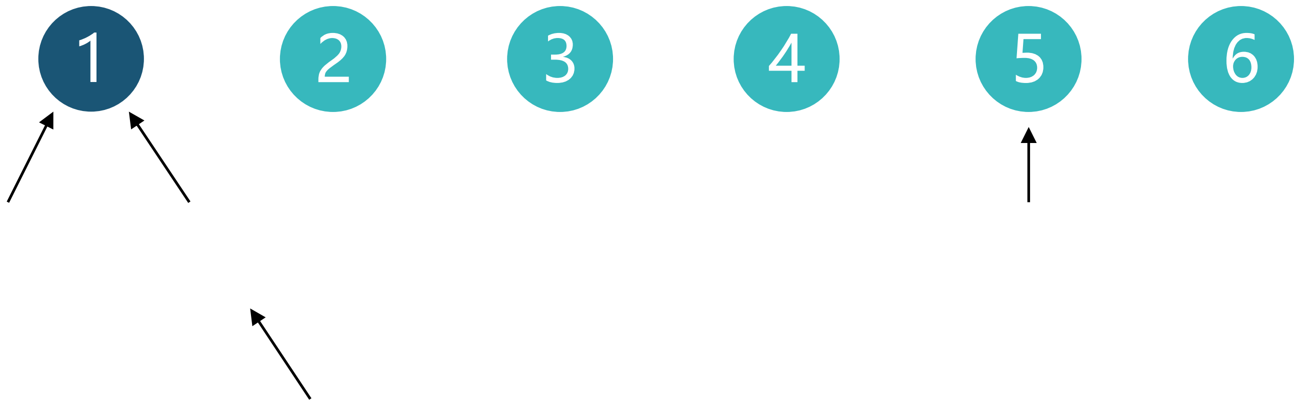
- 每个集合用一个链表表示
  - 空间：O(n)

- 操作的实现及时间效率分析
  - 查找：O(n)
  - 合并：O(n)
  - 启发式合并：O(nlogn)

并查集的实现——集合树

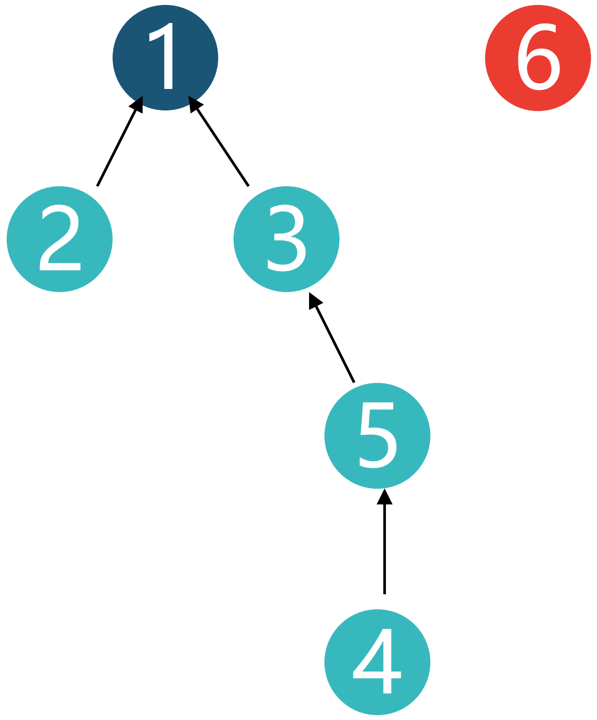
- 每个集合用一棵树表示
- 树的根节点为集合的代表元素

合并示例



- 合并1和2
- 合并1和3
- 合并5和4
- 合并5和3

查找示例



rep[1]=1

rep[2]=1

rep[3]=1

rep[4]=1

rep[5]=1

rep[6]=6

集合树的Father数组实现

- 存储结构：节点之间的关系用**father**数组维护

  int father[MAXN];

- **MAKE_SET**：初始化时 **father[v]=v**

/* 根据实际情况，father[v] 可以为 0或特殊值 */

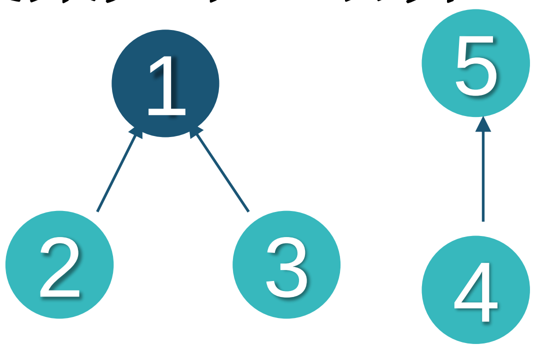
查找操作

1. 从结点v开始，沿father链向上，一直根结点。
2. 可用循环实现（课后练习）;也可用递归实现

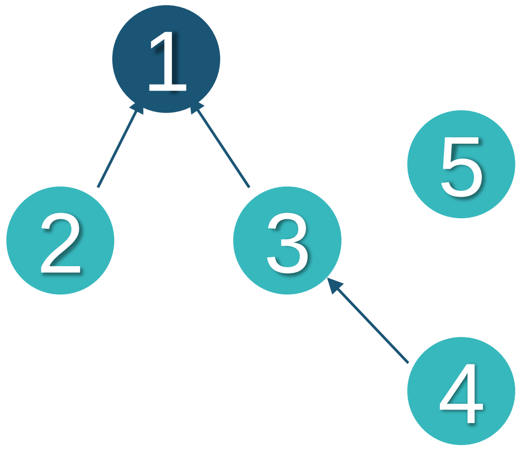
```
int FIND ( int v )
{
    if( father[v]==v ) return v;
    return FIND( father[v] );
}
```

合并操作的关键

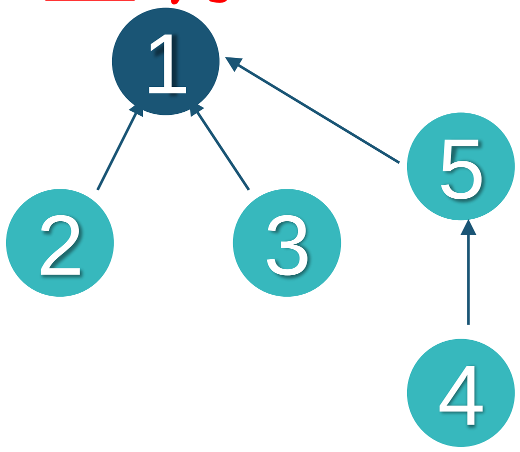
- **结点y(或x)所在树的根结点的父亲指向结点x(或y)所在树的根结点。**
- 例：两个不相交集如下，合并3和4



错误！



正确！

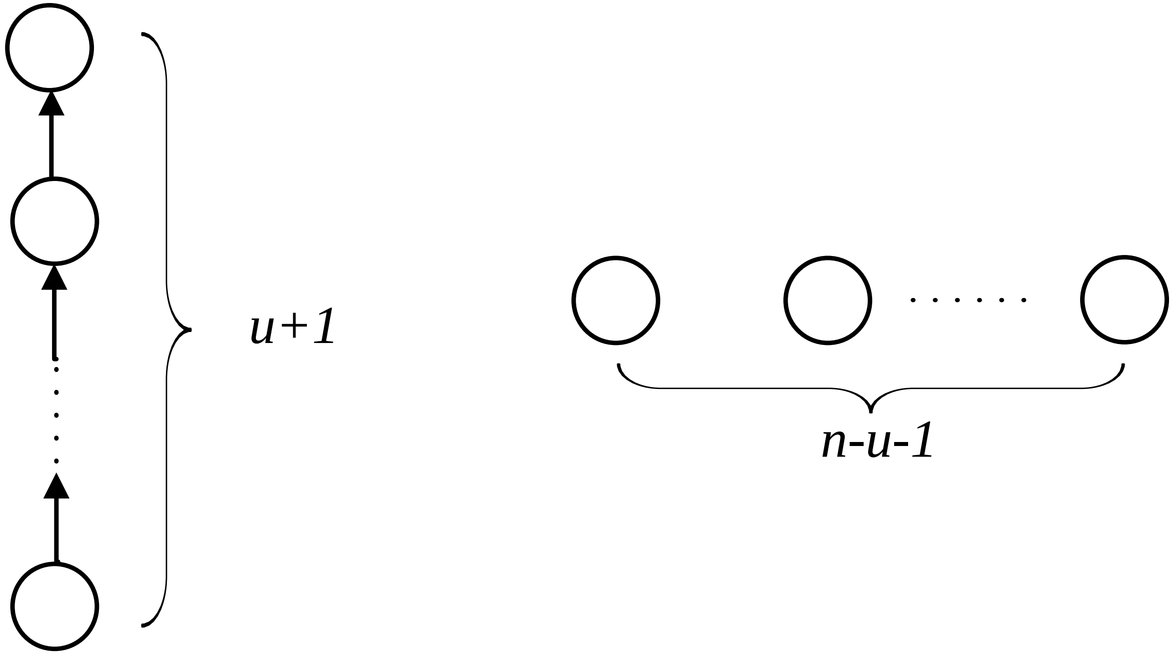


合并操作

```
void UNION( int x , int y) //安全
{
        int fx = FIND (x);
        int fy = FIND (y);
        if( fx != fy ) father[fy] = fx;
 }

void UNION( int x , int y) //f[v]=v
{
      father[FIND (y)] = FIND(x);
}
```

分析



- 查找的时间复杂度O(n)
- 合并O(n)

按秩合并

- 为了避免产生退化树，使用启发式的合并规则

- 按秩合并规则：对于每个结点，维护一个秩(rank)，表示以该结点为根的子树高度的一个上界。按秩合并策略让具有较小秩的根指向具有较大秩的根。
- 最直接的方法是选择以某结点为根的子树的高度作为该结点的秩。当然，也可以使用其它量，如以某结点为根的子树的结点个数(size).

按秩合并的实现

- **MAKE_SET时，每个结点的*rank*初始为0.**
- **UNION(*x*,*y*)操作时，设x和y所在树的根分别为*fx*和*fy*，**
  - ✓ 如果***rank*(*fx*) = *rank*(*fy*)**，那么让***fy***指向***fx***，***rank*(*fx*)增1**；
  - ✓ 如果***rank*(*fx*)≠*rank*(*fy*)**，那么让***rank***较小的根指向***rank***较大的根，秩不变。
- 技巧：利用***father***域保存结点的***rank***。
  - ✓ 如果x是根结点，Father[x]保存结点x的rank的相反数；
  - ✓ 如果x不是根结点，Father[x]保存结点x的父亲的地址。

按秩合并规则的实现

```
void MAKE_SET(x){

        father[x] = 0;

}

void UNION(x,y){
    int fx = FIND(x), fy = FIND(y);
    if( fx == fy ) return;
    if( father[fx] < father[fy]) father[fy]=fx;
    else{
        if(father[fx]==father[fy]) fahter[fy]--;
        father[fx]=fy;
    }
}
```

分析约定

- 设*n*表示MAKE_SET操作的次数，亦即并查集的元素总数；*u*表示UNION操作的次数；*f*表示FIND操作的次数；*m*表示MAKE_SET、UNION和FIND操作的总次数
  - ✓ $m = n + u + f$.
  - ✓ $u \leq n$-1
  - ✓ $f \geq u$

定理5.4

- 设F是从初始并查集经过*u*次UNION操作形成的森林，UNION操作使用了按秩合并规则，则F中任一结点的秩最多为$\lfloor \log_2 (u+1) \rfloor$.

证明

- u = 1时，定理成立。
- 假设对于所有的k < u 都成立。当 k = u 时，

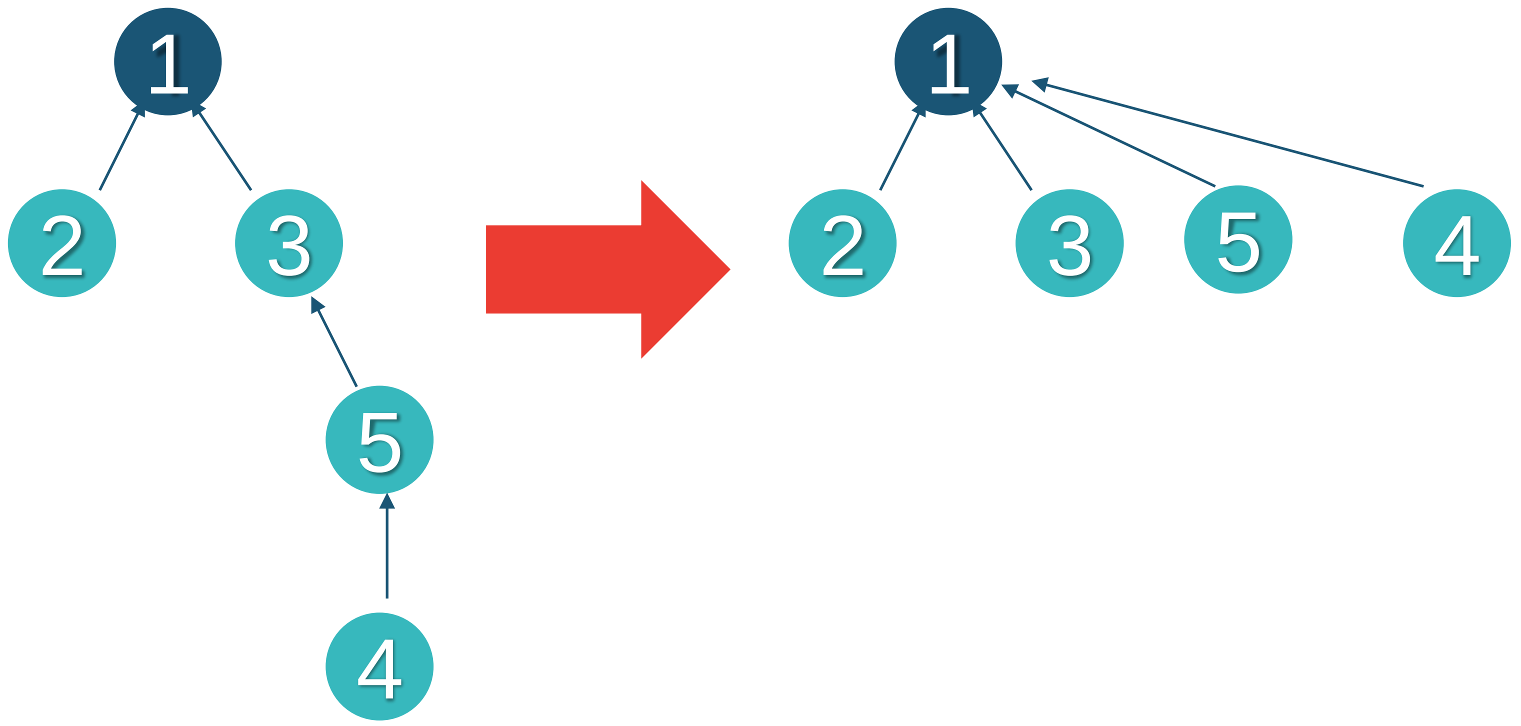
考虑最后一次调用UNION操作的情况。设最后一次调用为UNION(x,y)，x所在树由p次UNION操作形成，根为fx，y所在树由q次UNION操作形成，根为fy. 显然，$p + q \leq u - 1$.

如果rank(fx)≠rank(fy)，那么fx和fy的秩都不变；

如果rank(fx) = rank(fy)，秩要增1.

路径压缩

- 在**FIND**操作中，找到元素 **x** 所在树的根 **fx** 之后，将 **x** 到根 **fx** 路径上的所有结点的父亲都改成 **fx .** 这种策略称为路径压缩。
- 例如：**Find(4)**



带路径压缩的FIND操作

```
int FIND(int v)
{
   if( father[v]<=0 ) return v;
   return   father[v]= FIND (father[v]);
}
```

分析

- 路径压缩增加了一次**FIND**操作的时间，但可能导致树的高度变小，从而提高后续操作的效率
- 一组m个**MAKE_SET**、**UNION**和**FIND**操作的序列，其中n个是**MAKE_SET**操作，只使用路径压缩，最坏时间复杂度为$O(n+f(1+\log_{2+f/n} n))$.

- 一组*m*个MAKE_SET、UNION和FIND操作的序列，其中*n*个是MAKE_SET操作，在不相交集合森林上使用按秩合并与路径压缩，最坏时间时间复杂度为O(*m* α(*n*)).
- α(·)是Ackerman函数的反函数，增长得非常缓慢。只有对于非常大的n值，才会有α(*n*) > 4. 对于实际的应用，都有α(*n*)≤4.

更一般集合的表示

- 线性表
  - 查
  - 并
  - 交
  - ……
- 标志数组（集合中的元素范围较少）
- **bitset （最大32位)**

Union/Find算法分析

- **Union/Find算法分析是第一批最坏情形下的 实现简单分析复杂 的经典例子**
- **一个稍弱的结果：任意顺序的M(=Ω(N))次Union/Find操作花费总的时间为O(M log*N).**
  - ✓ log*N： N变为≤1时取对数的次数，例如：log*65536 = 4，log*(2^{65536}) = 5
  - ✓ Union按秩合并，Find路径压缩。为简化处理，假定Union指令只根据结点的秩进行指针调整，仅花费常数时间。

引理1

- 引理1：不考虑路径压缩时，当执行一系列Union指令时，一个秩为r的结点必然至少有2^r个后裔（包含自己）.

- 证明：对r用数学归纳法
  - ✓ 基础情形 $r = 0$ 时引理显然成立
  - ✓ 令T是秩为r的具有最少后裔数的树，并令X是T的根。设涉及X的最后一次Union是在T1 和 T2之间进行的，不妨设T1的根是X。
  - ✓ T1的秩不能是r，否则T1就是秩为r且比T后裔数更少，矛盾。故 T1的秩 $\leq r-1$.
  - ✓ T2的秩≤ T1的秩； T的秩为r只能因为T2增加，故T2的秩为 $r-1$, T1的秩也只能为 $r-1$。根据归纳假设，T1、T2都至少有 2^{r-1} 个后裔，从而T至少有 2^r 个后裔。

引理2

- 秩为 r 的结点的个数最多是 $N/2^r$.

- 证明：
  - 若无路径压缩，每个秩为r的结点都是至少有2^r个结点的子树的根，且在该子树中没有其它秩为r的结点，因此秩为r的那些结点的所有子树是不相交的。因此，至多存在$N/2^r$个不相交的子树，从而最多有存在$N/2^r$个秩为r的结点。
  - 有路径压缩时，路径压缩不改变结点的秩；同，时路径上改变父亲的结点，不再直接影响Union，即不再影响其它结点的秩的变化，因此不影响秩的计数。

引理3

- 在Union/Find算法的任一时刻，从叶到根的路径上结点的秩单调递增。

- 证明：
  - 若无路径压缩，显然成立。
  - 有路径压缩时，某个结点v是w的后裔，那么只考虑Union时显然v还是w的一个后裔。因此v的秩少于w的秩。

把秩划分成组

- 秩r被分到组G(r)，而G根据需要确定。
- 例：G(r) = cell(√r)

- 任何秩组g中最大的秩为F(g)，其中$F=G^{-1}$。
- 因此,任何秩组g>0中秩的个数为F(g)-F(g-1)。
- 保证：秩0组只包含秩为0的元素。

组	秩
0	0
1	1
2	2,3,4
3	5,…,9
4	10,…,16
i	$(i-1)^2+1,\ldots,i^2$

记账思想

- 思想：每个**Find(i)**花费的时间正比于从代表**i**的顶点到根的路径上的顶点个数。因此，对于路径上的每一个顶点都存入一个硬币。算法结束时，统计所有硬币的个数，就是总的花费

记账规则

- 利用路径压缩，记账规则为：对从代表i的顶点到根的路径上的每一个顶点v，我们在两个账户之一存入一个硬币

1. 如果v是根，或者v的父亲是根，或者v的父亲在与v不同的秩组中，那么收一个单位的费用，将一个金币存入到公共账户中。
2. 否则，将一个银币存入到该结点中。

引理4

- 对于任意的**Find(v)**，不论存入公共账户还是存入顶点，所存硬币的总数恰好等于从**v**到根的路径上的结点数。

- 证明：显然

引理5

- 引理5：进行M次Find，经过整个算法，在规则1下金币总的存入量总计最多为M(G(N)+2).

- 证明：
  - 对于任意Find，由于有根和它的儿子，因此存入两个金币。
  - 由引理3，沿路径向上分布的结点按秩单调递增，又由于最多有G(N)组，因此对任意特定的Find，在路径上最多只有G(N)个其它结点能按照规则1存入金币。
  - 于是，任意一次Find，最多又有G(N) + 2个金币存放到公共账户中。
  - M次Find最多可以存入M(G(N)+2)个金币

规则2下所有银币存入量的估算

- 按照顶点把存入的硬币加起来
- 如果一枚硬币在规则2下存入顶点v，那么v将通过路径压缩被移动并 得到比它原来的父亲更高的秩 的 新的父亲。
- 于是，秩组 g >0 中的结点v 在它的父亲被推离秩组g之前最多可以移动F(g) – F(g-1)次，因为这是该组秩的大小。在这以后，对v的所有未来的收费均按照规则1进行。

引理6

- 秩组 $g > 0$ 中的顶点个数 $V(g)$ 至多为 $N / 2^{F(g-1)}$.

- 证明：
- ✓ 由引理2，至多存在$N/2^r$个秩为 r 的结点，对组g中的秩求和，得到

$$V(G) \le \sum_{r=F(g-1)+1}^{F(g)} N/2^r$$

$$\le \sum_{r=F(g-1)+1}^{\infty} N/2^r$$

$$\le \frac{N}{2^{F(g-1)+1}} \sum_{s=0}^{\infty} 1/2^s$$

$$\le N / 2^{F(g-1)}$$

引理7

- 引理7：存入秩组g的所有顶点的银币的最大个数至多是 $NF(g) / 2^{F(g-1)}$.

- 证明：
  - 该秩组的每一个顶点当它的父结点同在该秩组时最多可以接受F(g)-F(g-1) ≤ F(g)个银币。
  - 由引理6，可得引理7.

引理8

- 在规则2下总的存入硬币量至多为 $\mathrm{N}\sum_{g=1}^{G(N)} F(g)/2^{F(g-1)}$ 个银币

- 证明：
- ✓ 因为秩组0只含有秩为0 的元素，，这样的元素在该秩组中不可能有父结点，不能按照规则2接收硬币。
- ✓ 通过对其它秩组求和可得引理8.

总硬币量的估算

- 根据规则1和规则2存入的硬币数的上界为：

  $$\mathrm{M(G(N) + 2) + N}\sum_{g=1}^{G(N)} F(g)/2^{F(g-1)}$$

- 需要指定G(N) 或其逆F(N).
- 一个明显的理想选择是： $F(i) = 2^{F(i-1)}$.
- 于是G(N) = 1 + floor(log*N)
- 代入估计式，可得 O(Mlog*N) + O(N log*N)
- 根据约定： M =Ω(N)，从而 O(Mlog*N)

定理

- **M次Union 和 Find的运行时间为O(M log*N)**