北京大学暑期课《ICPC竞赛训练》

课程网页：http://acm.pku.edu.cn/summerschool/pku_acm_train.htm

郭 炜

微博：http://weibo.com/guoweiofpku

微信公众号

学会程序和算法，走遍天下都不怕!

讲义照片均为郭炜拍摄

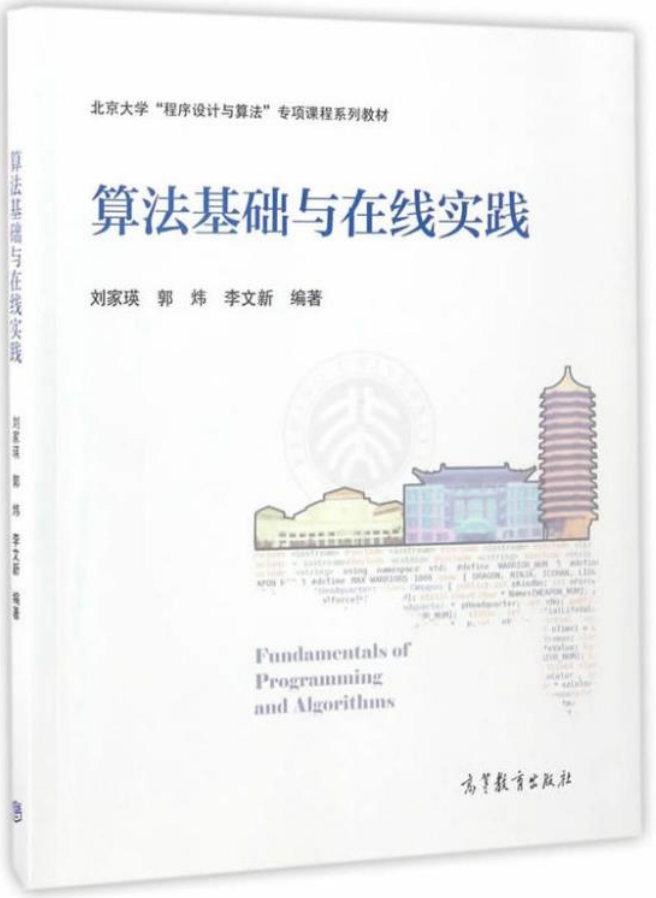
配套教材：

高等教育出版社

《算法基础与在线实践》

刘家瑛 郭炜 李文新 编著

本讲义中所有例题，根据题目名称在
http://openjudge.cn
“百练”组进行搜索即可提交



并查集

并查集的概念

Disjoint-Set 并查集

N 个不同的元素分布在若干个互不相交集合中，需要多次进行以下3个操作：

1. 合并a,b两个元素所在的集合 Merge(a,b)
2. 查询一个元素在哪个集合
3. 查询两个元素是否属于同一集合 Query(a,b)

并查集操作示例

Operation	**Disjoint sets**					
初始状态	{a}	{b}	{c}	{d}	{e}	{f}
Merge(a,b)	{a,b}		{c}	{d}	{e}	{f}
Query(a,c)	False					
Query(a,b)	True					
Merge(b,e)	{a,b,e}		{c}	{d}		{f}
Merge(c,f)	{a,b,e}		{c,f}	{d}		
Query(a,e)	True					
Query(c,b)	False					
Merge(b,f)	{a,b,c,e,f}			{d}		
Query(a,e)	True					
Query(d,e)	False					

简单算法

- 给集合编号

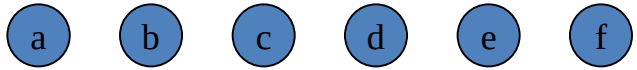
Op \ *Element*	**{a}**	**{b}**	**{c}**	**{d}**	**{e}**	**{f}**
	1	2	3	4	5	6
Merge(a,b)	1	1	3	4	5	6
Merge(b,e)	1	1	3	4	1	6
Merge(c,f)	1	1	3	4	1	3
Merge(b,f)	1	1	1	4	1	1

Query(a,e)

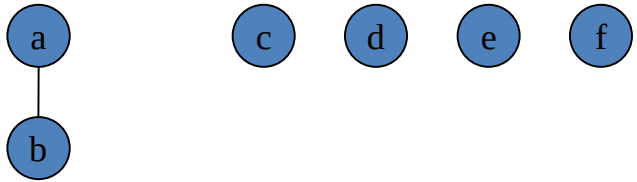
- Query – O(1); Merge – O(N)

用树表示集合的算法

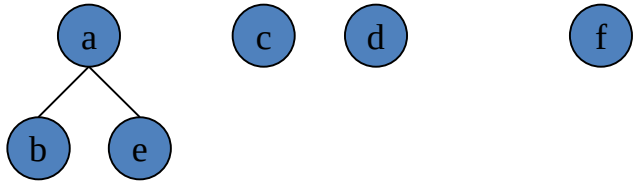
- 开始:



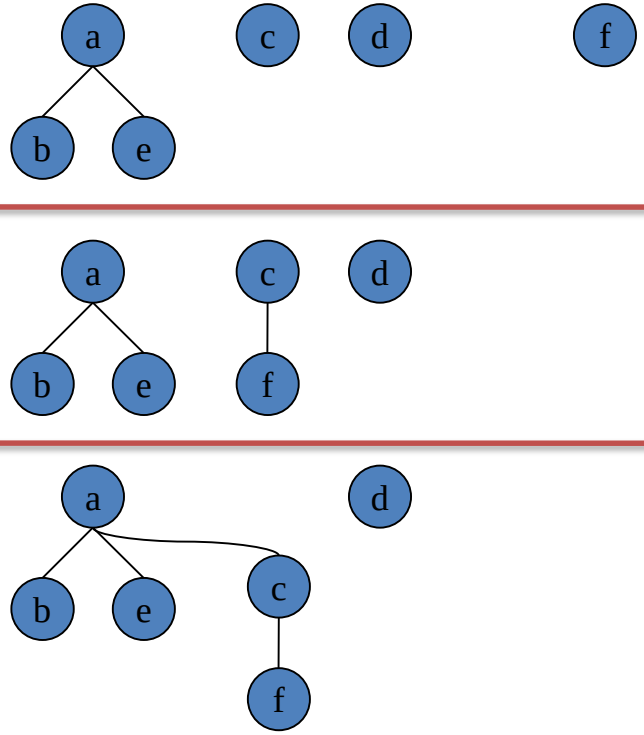
- *Merge(a,b)*



- *Merge(b,e)*



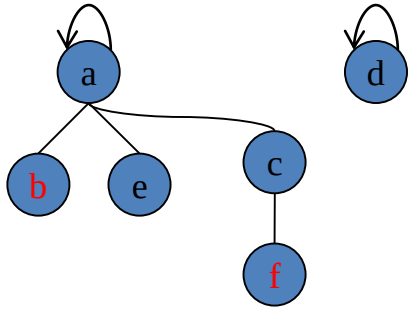
用树表示集合的算法



- *Merge(c,f)*
- *Merge(b,f)*

用树表示集合的算法

- 设置数组 parent, parent[i] = j 表示元素 i 的父亲是 j
- parent[i] = i 表示 i 是一棵树的根节点
- 若开始每个元素自成一个集合，则 对任何 i， 都有 parent[i] = i



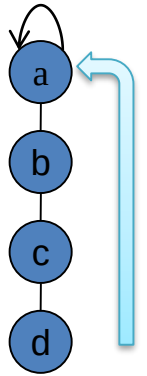
用树表示集合的算法

- 基本操作 GetRoot (a) 求a的树根

```
int GetRoot(int a)
{
      if( parent[a] == a)
            return a;
      return GetRoot(parent[a]);
}
```

用树表示集合的算法

- *Query(a,b)*
  - 比较b和f所在树的根节点是否相同
- *Merge(a,b)*
  - 将b的树根的父亲，设置为a的树根
- *缺点:*
  - 树的深度失控则查树根可能太慢!



O(N)!

改进方法一

- 合并两棵树时，把深度浅的树直接挂在另一棵树的根节点下面

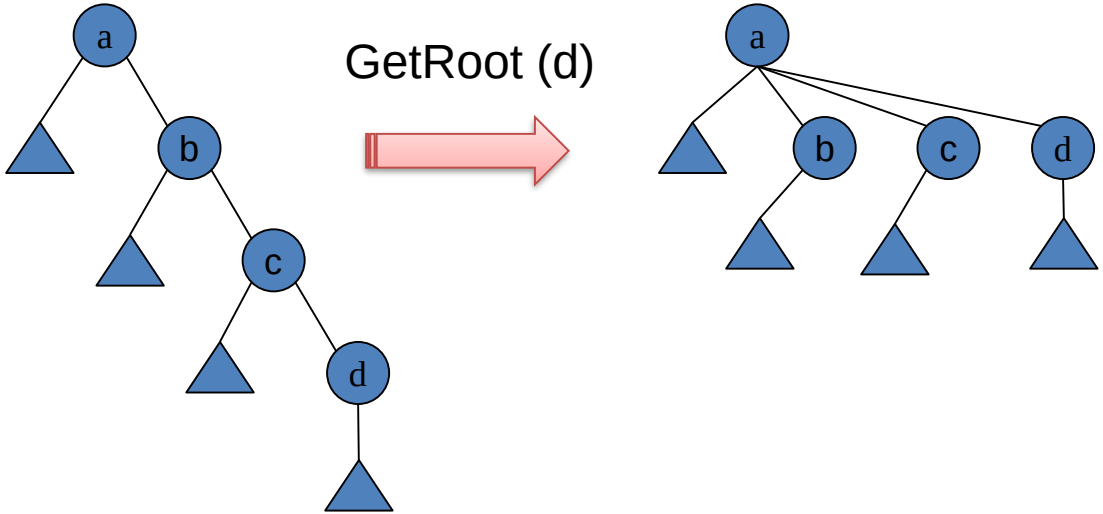
- 合并两棵同深度的树时，必然导致深度增加，不够理想

改进方法二：路径压缩

- 查询一个节点的根时，直接将该节点到根路径上的每个节点，都直接挂在根下面。

- 经过多次查询，很可能所有树的深度都是 <= 2

改进方法二：路径压缩



改进方法二：路径压缩

- 基本操作 GetRoot (a) 求a的树根

```
int GetRoot(int a)
{
    if( parent[a] != a)
        parent[a] = GetRoot(parent[a]);
    return parent[a];
}
```

```
int GetRoot(int a) { //old one
    if( parent[a] == a)
        return a;
    return GetRoot(parent[a]);
}
```

改进方法二：路径压缩

```
int parent[N];
int Merge(int a,int b)
{  //把b树根挂到a树根下
      parent[GetRoot(b)] = GetRoot(a) ;
}
bool Query(int a,int b)
{
      return GetRoot(a) == GetRoot(b) ;
}
```

并查集的空间复杂度O(N)，时间复杂度就是GetRoot的复杂度

并查集的时间复杂度

GetRoot的平摊时间复杂度，在N不是很大的时候，是常数，且不超过4

并查集的时间复杂度

GetRoot的平摊时间复杂度，在N不是很大的时候，是常数，且不超过4

“不是很大”，指的是不比宇宙中原子的数目更多

例题 1
The Suspects

例题1： POJ1611 The Suspects

- 有n个学生，编号0到 n-1, 以及m个团体，(0 < n <= 30000 , 0 <= m <= 500) 一个学生可以属于多个团体，也可以不属于任何团体。一个学生疑似疑似患病，则它所属的整个团体都疑似患病。

- 已知0号学生疑似患病，以及每个团体都由哪些学生构成，求一共多少个学生疑似患病。

例题1： POJ1611 The Suspects

Sample Input

```
100 4     //100人,4团体
2 1 2     //本团体有2人，编号 1,2
5 10 13 11 12 14
2 0 1
2 99 2
200 2   //200人,2团体
1 5     //本团体有1人，编号 5
5 1 2 3 4 5
1 0     //1人,0团体
0 0     //结束标记
```

Sample Output

```
4
1
1
```

三组数据，分别是:
100个人，4个团体
200个人,2个团体
1个人,0个团体

例题1： POJ1611 The Suspects

关键:

互相感染的人，应该属于同一个集合。
最终问0所在的集合有几个元素

例题1： POJ1611 The Suspects

```
#include <iostream>
using namespace std;
const int MAX = 30000;
int n,m,k;
int parent[MAX+10];
int total[MAX+10]; //total[GetRoot(a)]是a所在的group的人数

int GetRoot(int a)
{//获取a的根，并把a的父节点改为根
        if( parent[a]!= a)
                parent[a] = GetRoot(parent[a]);
        return parent[a];
}
```

例题1： POJ1611 The Suspects

```
void Merge(int a,int b)
{
        int p1 = GetRoot(a);
        int p2 = GetRoot(b);
        if( p1 == p2 )
                return;
        total[p1] += total[p2];
        parent[p2] = p1;
}
```

例题1： POJ1611 The Suspects

```
int main()  {
        while(true) {
                scanf("%d%d",&n,&m);
                if( n == 0 && m == 0)  break;
                for(int i = 0;i < n; ++i) {
                        parent[i] = i;
                        total[i] = 1;
                }
                for(int i = 0;i < m; ++i) {
                        int h,s;
                        scanf("%d",&k);    scanf("%d",&h);
                        for( int j = 1; j < k; ++j) {
                            scanf("%d",&s);  Merge(h,s);
                        }
                }
                printf("%d\n",total[GetRoot(0)]); //此处写parent[0]可否?
        }  return 0;
}
```

例题1： POJ1611 The Suspects

```
int main()  {
        while(true) {
                scanf("%d%d",&n,&m);
                if( n == 0 && m == 0)  break;
                for(int i = 0;i < n; ++i) {
                        parent[i] = i;
                        total[i] = 1;
                }
                for(int i = 0;i < m; ++i) {
                        int h,s;
                        scanf("%d",&k);    scanf("%d",&h);
                        for( int j = 1; j < k; ++j) {
                            scanf("%d",&s);   Merge(h,s);
                        }
                }
                printf("%d\n",total[GetRoot(0)]); //此处写parent[0]可否?
        }  return 0;
}
```

不行，因为可能还没进行过路径压缩，parent[0]的值不正确

并查集解题的套路

在GetRoot 和 Merge中维护必要的信息

注意：两棵树合并以后，parent[a]未必就是a的根，因为从a到根的路径可能还没经过压缩。GetRoot(a)后parent[a]才是a的根

例题 2
Cube stacking

例题2： POJ 1988 Cube stacking

- 有N(N<=30,000)堆方块，开始每堆都是一个方块。方块编号1 – N. 有两种操作:
- M x y： 表示把方块x所在的堆，拿起来叠放到y所在的堆上。
- C x : 问方块x下面有多少个方块。
- 操作最多有 P (P<=100,000)次。对每次C操作，输出结果。

Sample Input

```
6
M 1 6
C 1
M 2 4
M 2 6
C 3
C 4
```

Sample Output

```
1
0
2
```

例题1： POJ1611 The Suspects

```
int main()  {
        while(true) {
                scanf("%d%d",&n,&m);
                if( n == 0 && m == 0)  break;
                for(int i = 0;i < n; ++i) {
                        parent[i] = i;
                        total[i] = 1;
                }
                for(int i = 0;i < m; ++i) {
                        int h,s;
                        scanf("%d",&k);    scanf("%d",&h);
                        for( int j = 1; j < k; ++j) {
                            scanf("%d",&s);   Merge(h,s);
                        }
                }
                printf("%d\n",total[GetRoot(0)]); //此处写parent[0]可否?
        }  return 0;
}
```

不行，因为可能还没进行过路径压缩，parent[0]的值不正确

POJ 1988 Cube stacking

除了parent数组，还要维护哪些信息?

POJ 1988 Cube stacking

除了parent数组，还要维护哪些信息?

- sum数组：记录每堆一共有多少方块。
  若parent[a] = a, 则sum[a]表示a所在的堆的方块数目。
  何时更新?

POJ 1988 Cube stacking

除了parent数组，还要维护哪些信息?

- sum数组：记录每堆一共有多少方块。

  若parent[a] = a, 则sum[a]表示a所在的堆的方块数目。

  何时更新？ 在堆合并的时候要更新

POJ 1988 Cube stacking

除了parent数组，还要维护哪些信息?

- sum数组：记录每堆一共有多少方块。
  若parent[a] = a, 则sum[a]表示a所在的堆的方块数目。
  何时更新？ 在堆合并的时候要更新
- under数组，under[i]表示第i个方块下面有多少个方块。
  何时更新?

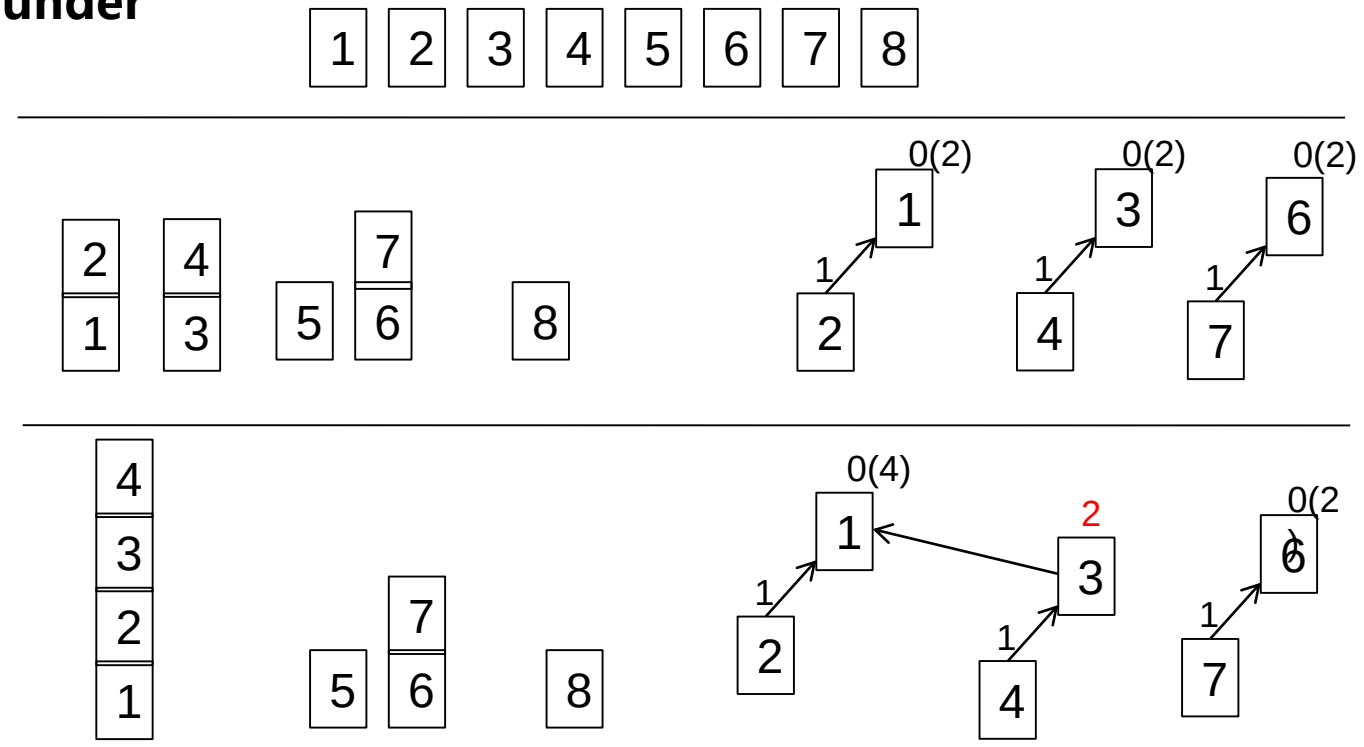
POJ 1988 Cube stacking

除了parent数组，还要维护哪些信息?

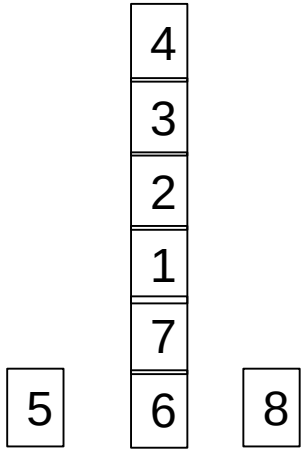
- sum数组：记录每堆一共有多少方块。
  若parent[a] = a, 则sum[a]表示a所在的堆的方块数目。
  何时更新？ 在堆合并的时候要更新
- under数组，under[i]表示第i个方块下面有多少个方块。
  何时更新？在堆合并和路径压缩的时候都要更新。

POJ 1988 Cube stacking

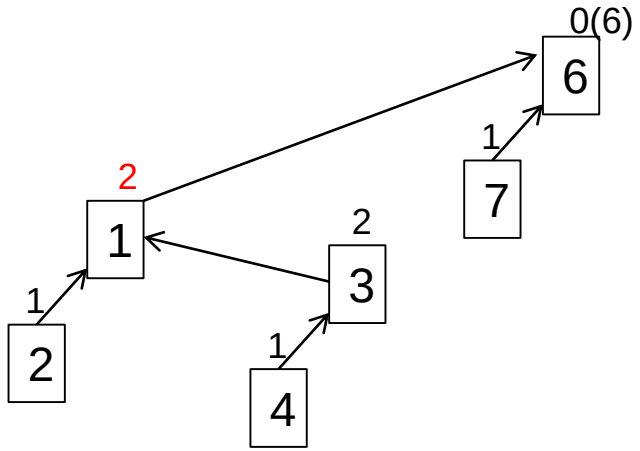
括号内数字是sum
括号外是 under



POJ 1988 Cube stacking

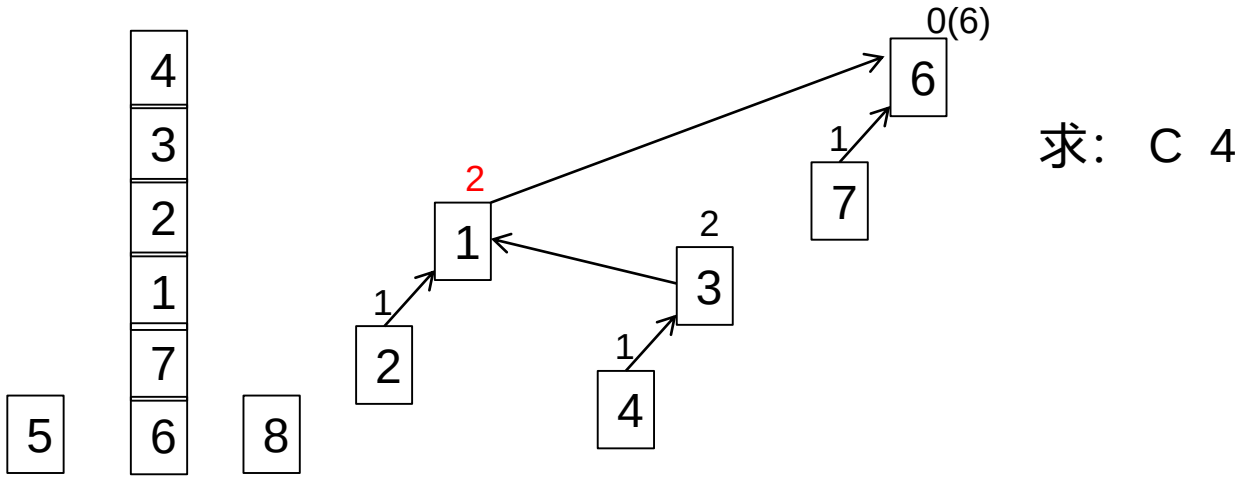


M 3 7



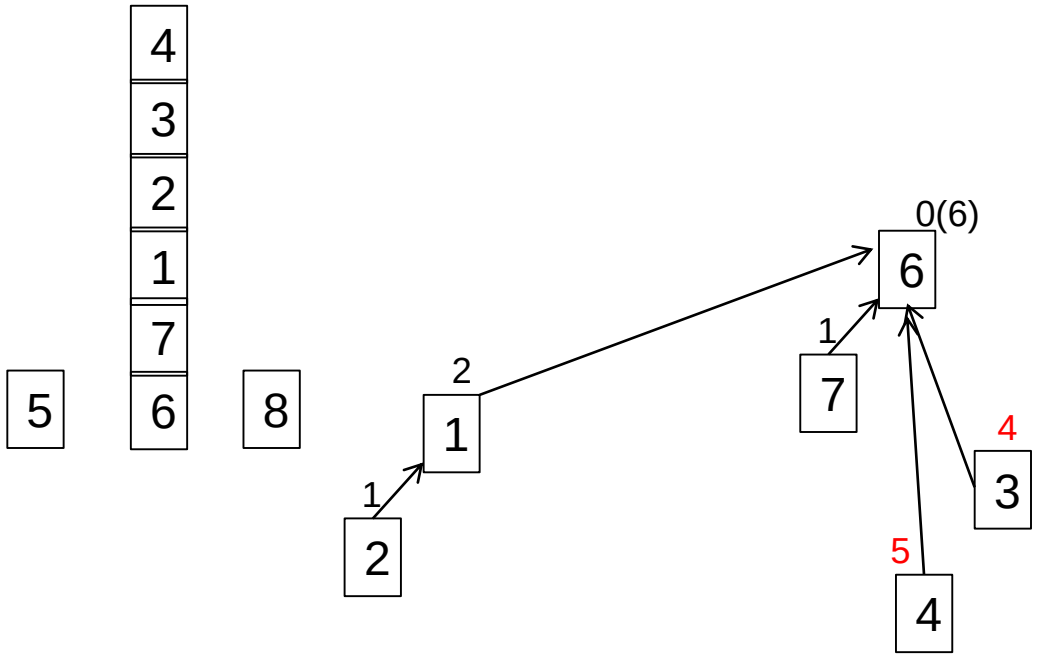
求： C 4

POJ 1988 Cube stacking



- 调用GetRoot(4)进行路径压缩的同时，更新under[4]
- under[4]等于从4到根路径上所有节点的under值之和
- 路径压缩的同时，该路径上所有节点的under值也都更新

POJ 1988 Cube stacking



POJ 1988 Cube stacking

```
#include <iostream>
#include <cstdio>
using namespace std;
const int MAX = 31000;
int parent[MAX];
int sum[MAX]; // 若parent[i]=i,sum[i]表示砖块i所在堆的砖块数目
int under[MAX]; // under[i]表示砖块i下面有多少砖块
int GetRoot(int a)
{//路径压缩
        if( parent[a] == a)
                return a;
        int t = GetRoot(parent[a]);
        under[a] += under[parent[a]];
        parent[a] = t;
        return parent[a];
}
```

POJ 1988 Cube stacking

```
void Merge(int a,int b)
//把b所在的堆，叠放到a所在的堆。
{       int n;
        int pa = GetRoot(a);
        int pb = GetRoot(b);
        if( pa == pb )
                return ;
        parent[pb] = pa;
        under[pb] = sum[pa]; //under[pb] 赋值前一定是0，因为
parent[pb] = pb,pb一定是原b所在堆最底下的
        sum[pa] += sum[pb];
}
```

POJ 1988 Cube stacking

```
int main()
{
        int p;
        for(int i = 0; i < MAX; ++ i) {
                sum[i] = 1;
                under[i] = 0;
                parent[i] = i;
        }
        scanf("%d",&p);
```

POJ 1988 Cube stacking

```
        for( int i = 0;i < p; ++ i) {
                char s[20];
                int a,b;
                scanf("%s",s);
                if( s[0] == 'M') {
                        scanf("%d%d",&a,&b);
                        Merge(b,a);
                }
                else {
                        scanf("%d",&a);
                        GetRoot(a);
                        printf("%d\n",under[a]);
                }
        }
        return 0;
}
```