

数据结构基础——堆栈、队列、堆、并查集

邓丝雨



STL

- 标准库STL，就是C++自身携带的，国际通用的，没有BUG的，老少皆宜的，令一众算法竞赛选手喜闻乐见的C++的附加产品。
- 主要分两大类：算法类（又称为操作类）和容器类。



算法类

- 最大值最小值（别看简单，C语言自身却没有这个）
 - `template <class _Tp>`
 - `_Tp min(_Tp a, _Tp b)`
 - `_Tp max(_Tp a, _Tp b)`
- 快速排序 `sort(a, a + 10);`
- 数据交换 `swap(a, b);`
- 求下一个排列 `next_permutation(a, a+n)`



容器类

- 字符串string
- 不限制长度的数组, vector
- 队列queue, 堆栈stack, 双端队列deque
- 优先队列 (堆) priority_queue
- 简单红黑树 (一种较高效率的平衡二叉树) set
- 通过键值来建立的平衡二叉树map
- 允许多个相同值的上述两种结构multiset和multimap
- 位集bitset
- <http://www.cplusplus.com/>



VECTOR

- 一般翻译成向量——借用了数学中 n 维空间下的向量的定义，由于不能确定 n 到底是多少，所以：
- VECTOR是一个不限制数组长度的数组！（最简单的理解方式）
- 如果学习过链表，那么你也可以理解成一个链表（稍复杂的理解）
- 实际上，这是一个看上去很像链表的顺序容器。
- 出于实战需要，还是理解成一个数组比较好（常数比数组大）



VECTOR实现了什么

- 首先是最基础的数据访问功能

`operator[], top(), back()`

- 其次是简单的数据操作:

删除`erase(int index, int size)`

插入`insert(int index, int size)`

添加到数据后面`push_back(int value)`

弹出最后一个数据`pop_back()`

- 还有一些自身属性操作: 数据大小和重新整理内存单元`size(), resize(int size)`
- 最后是整体的操作: 清空`clear()`, 是否为空`empty()`



迭代器Iterator

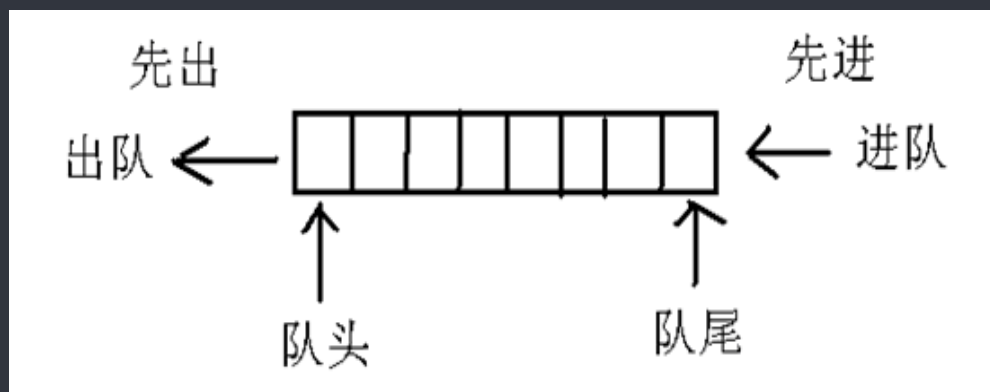
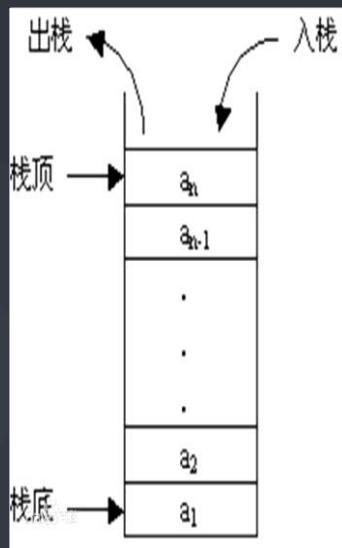
- 定位：既能像指针一样访问数据，又能保证整体工程的稳定性
 - 实际作用：用于访问一个容器内的数据的指针。
 - 具体实现了什么？
 - 返回第一个元素的迭代器begin()
 - 返回容器末尾的迭代器end()（同样遵循左开右闭原则）
-
- `vector<int> a;`
 - `vector<int>::iterator it`
 - `for (it =a.begin(); it != a.end(); it++)`

栈和队列



Stack 和 Queue——栈和队列

- 栈的定义：栈是限定仅在表头进行插入和删除操作的线性表（先进后出）
- 队列的定义：队列是一种特殊的线性表，特殊之处在于它只允许在表的前端（front）进行删除操作，而在表的后端（rear）进行插入操作，和栈一样，队列是一种操作受限制的线性表。进行插入操作的端称为队尾，进行删除操作的端称为队头。（先进先出）





- **stack常用函数:**

- push() ——向栈顶压入元素
- pop()——弹出栈顶元素
- top()——访问栈顶元素

- **queue常用函数:**

- front()——访问队首元素
- back()——访问队尾元素
- push——向队尾插入元素
- pop——弹出队首元素



例1:

- 有 n 个人，按照 $1, 2, 3, 4, \dots, n$ 的顺序依次进栈，判断能否以题目所给序列出栈。
($n \leq 100000$)
- Eg:
- 4 3 2 1
- 1 2 3 4
- 1 3 2 4
- 1 4 2 3

例2：NC212914牛牛与后缀表达式

- 给定牛牛一个后缀表达式s，计算它的结果，例如， $1+1$ 对应的后缀表达式为 $1\#1\#+$ ，‘#’作为操作数的结束符号。
- 其中，表达式中只含有 ‘+’ 、 ‘-’ 、 ‘*’ 三种运算，不包含除法。
- 本题保证表达式一定合法，且计算过程和计算结果的绝对值一定不会超过 10^{18}



例3： NC21874 好串

- 牛牛喜欢跟字符串玩耍，他刚刚学会了一个新操作，将一个字符串x插入另一个字符串y中（包括放在开头和结尾）
- 牛牛认为如果一个串是好的当这个串能按照如下方法被构造出来：
- z一开始，有一个空串，然后执行0次或者若干次操作，每次操作将ab插入当前的字符串
- 根据上面的定义，ab, aabb, aababb都是好串，aab,ba,abbb并不是好串
- 现在给你一个字符串s，判断s是否是好串




例4: NC 16430 蚯蚓

- 蛐蛐国最近蚯蚓成灾了！隔壁跳蚤国的跳蚤也拿蚯蚓们没办法，蛐蛐国王只好去请神刀手来帮他们消灭蚯蚓。蛐蛐国里现在共有 n 只蚯蚓 (n 为正整数)。每只蚯蚓拥有长度，我们设第 i 只蚯蚓的长度为 a_i ($i=1,2,\dots,n$)，并保证所有的长度都是非负整数 (即：可能存在长度为 0 的蚯蚓)。
- 每一秒，神刀手会在所有的蚯蚓中，准确地找到最长的那一只 (如有多个则任选一个) 将其切成两半。神刀手切开蚯蚓的位置由常数 p (是满足 $0 < p < 1$ 的有理数) 决定，设这只蚯蚓长度为 x ，神刀手会将其切成两只长度分别为 $[px]$ 和 $x-[px]$ 的蚯蚓。特殊地，如果这两个数的其中一个等于 0，则这个长度为 0 的蚯蚓也会被保留。此外，除了刚刚产生的两只新蚯蚓，其余蚯蚓的长度都会增加 q (是一个非负整常数)。



- 蛐蛐国王知道这样不是长久之计，因为蚯蚓不仅会越来越多，还会越来越长。蛐蛐国王决定求助于一位有着洪荒之力的神秘人物，但是救兵还需要 m 秒才能到来 (m 为非负整数) 蛐蛐国王希望知道这 m 秒内的战况。具体来说，他希望知道：
- m 秒内，每一秒被切断的蚯蚓被切断前的长度 (有 m 个数) ；
- m 秒后，所有蚯蚓的长度 (有 $n + m$ 个数) 。
- $1 \leq n \leq 10^5, 0 \leq m \leq 7 \times 10^6, 1 \leq t \leq 71, 0 \leq a_i \leq 10^8, 1 \leq v \leq 10^9, 0 \leq q \leq 200$

- 
- 用三个队列直接维护有序的序列：一个按从大到小存没有被切的蚯蚓，一个存切出来的第一段，一个存切出来的另一段，这样他们各自始终是有顺序的，每次只需要把队首分别拿出来看看就好了。而每次需要给蚯蚓增加的长度，我们可以在放进去的时候把长度统一到某一时刻（比如0时刻），然后拿出来时统一加上长时的长度。



例5: NC51001 滑动窗口

- 给定一个长度为 n 的数列, 求长度为 k 的定长连续子区间 $\{a_1, a_2, a_3, a_4, \dots, a_{k-1}, a_k\}$
 $\{a_2, a_3, \dots, a_k, a_{k+1}\}$中每个区间的最大值和最小值。

```
15     int l = 0;
16     int r = 1;
17     du[0] = 1;
18     if (m == 1) printf("%d ", a[1]);
19     for (int i = 2; i <= n; i++)
20     {
21         if (i - du[l] >= m && (l < r)) l++;
22
23         while (r > l && a[du[r - 1]] >= a[i]) r--;
24         du[r++] = i;
25
26         if (i >= m) printf("%d ", a[du[l]]);
27     }
```

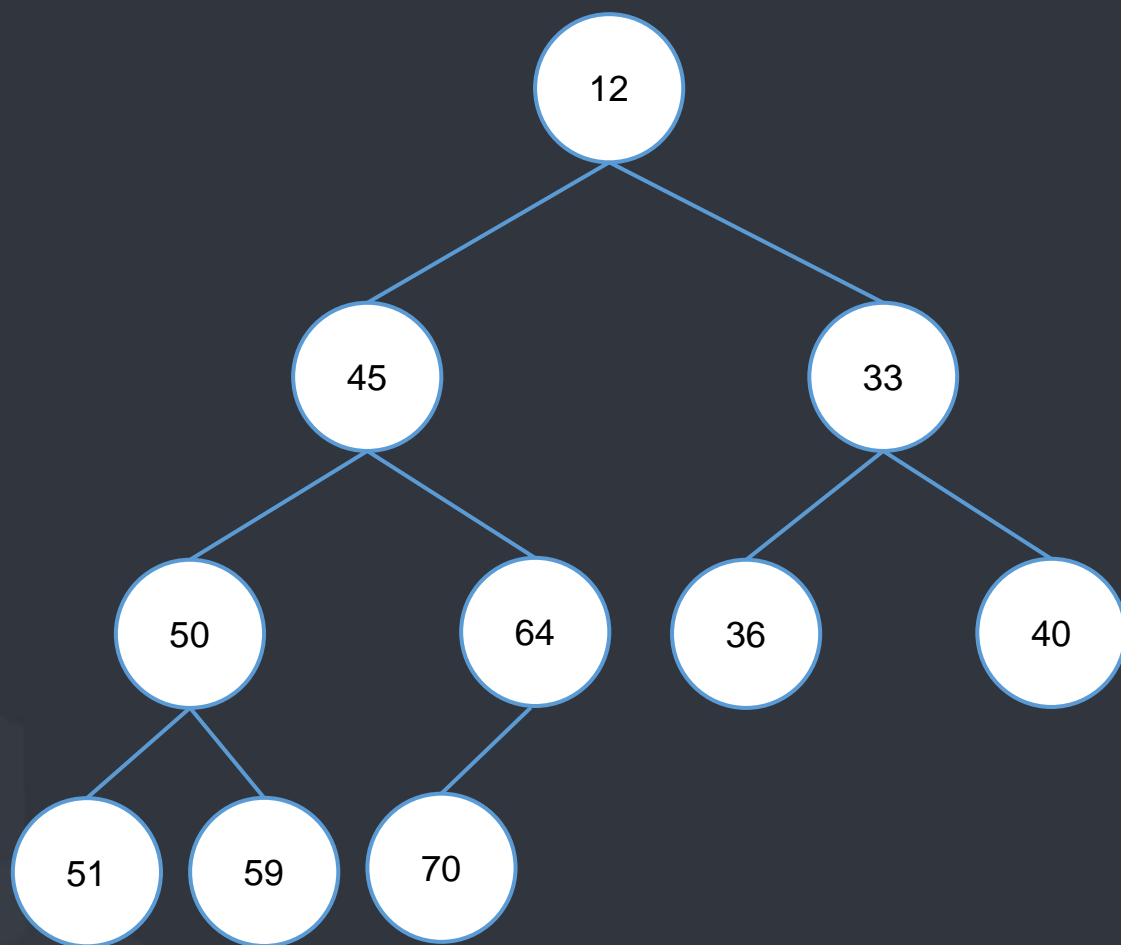
优先队列



优先队列/堆

- 优先队列又名二叉堆，是特殊的二叉树。二叉堆有两种：最大堆和最小堆。
- 最大堆（大根堆）：父结点的键值总是大于或等于任何一个子节点的键值。
- 最小堆（小根堆）：父结点的键值总是小于或等于任何一个子节点的键值。

删除堆顶





例1：合并果子

- 在一个果园里，多多已经将所有果子打了下来，而且按果子的不同种类分成了不同的堆。多多决定把所有的果子合成一堆。每一次合并，多多可以把两堆果子合并到一起，消耗的体力等于两堆果子的重量之和。可以看出，所有的果子经过 $n-1$ 次合并之后，就只剩下一堆了。多多在合并果子时总共消耗的体力等于每次合并所耗体力之和。假定每个果子重量都为1，并且已知果子的种类数和每种果子的数目，你的任务是设计出合并的次序方案，使多多耗费的体力最少，并输出这个最小的体力耗费值。

例2: NC50940 Running Median

- N个数按顺序加入数组，每次加入的时候就输出其中位数

例3： NC20185 [JSOI2010]缓存交换

- 在计算机中，CPU只能和高速缓存Cache直接交换数据。当所需的内存单元不在Cache中时，则需要从主存里把数据调入Cache。此时，如果Cache容量已满，则必须先从中删除一个。
- 例如，当前Cache容量为3，且已经有编号为10和20的主存单元。此时，CPU访问编号为10的主存单元，Cache命中。接着，CPU访问编号为21的主存单元，那么只需将该主存单元移入Cache中，造成一次缺失（Cache Miss）。接着，CPU访问编号为31的主存单元，则必须从Cache中换出一块，才能将编号为31的主存单元移入Cache，假设我们移出了编号为10的主存单元。接着，CPU再次访问编号为10的主存单元，则又引起了一次缺失。



- 我们看到，如果在上一次删除时，删除其他的单元，则可以避免本次访问的缺失。在现代计算机中，往往采用LRU(最近最少使用)的算法来进行Cache调度——可是，从上一个例子就能看出，这并不是最优的算法。
- 对于一个固定容量的空Cache和连续的若干主存访问请求，聪聪想知道如何在每次Cache缺失时换出正确的主存单元，以达到最少的Cache缺失次数。
- $1 \leq M \leq N \leq 100,000$

例4: NC 50439 tokitsukaze and Soldier

- 在一个游戏中, tokitsukaze需要在 n 个士兵中选出一些士兵组成一个团去打副本。第 i 个士兵的战力为 $v[i]$, 团的战力是团内所有士兵的战力之和。但是这些士兵有特殊的要求: 如果选了第 i 个士兵, 这个士兵希望团的人数不超过 $s[i]$ 。(如果不选第 i 个士兵, 就没有这个限制。)
tokitsukaze想知道, 团的战力最大为多少。

例5：NC20154 [JSOI2007]建筑抢修

- 小刚在玩JSOI提供的一个称之为“建筑抢修”的电脑游戏：经过了一场激烈的战斗，T部落消灭了所有Z部落的入侵者。但是T部落的基地里已经有N个建筑设施受到了严重的损伤，如果不尽快修复的话，这些建筑设施将会完全 毁坏。
- 现在的情况是：T部落基地里只有一个修理工人，虽然他能瞬间到达任何一个建筑，但是修复每个建筑都需要一定的时间。同时，修理工人修理完一个建筑才能修理下一个建筑，不能同时修理多个建筑。
- 如果某个建筑在一段时间之内没有完全修理完毕，这个建筑就报废了。你的任务是帮小刚合理的制订一个修理顺序，以抢修尽可能多的建筑。

其他常用stl——map/set



例1:

- 给你 n 个敌人的坐标，再给你 m 个炸弹和爆炸方向，每个炸弹可以炸横排或竖排的敌人，问你每个炸弹能炸死多少个人。

```

7  typedef map<int,multiset<int> > line;
8  map<int,multiset<int> >mx;
9  map<int,multiset<int> >my;
10 int n, m;
11 int bomb(line &x, line &y, int pos)
12 {
13     int ans = x[pos].size();
14     multiset<int>::iterator it;
15     for(it = x[pos].begin(); it != x[pos].end(); it++)
16         y[*it].erase(pos);
17     x[pos].clear();
18     return ans;
19 }

```

```

22 while(scanf("%d%d", &n, &m) != EOF)
23 {
24     if(n == 0 && m == 0)break;
25     mx.clear();
26     my.clear();
27     for(int i = 0; i < n; i++)
28     {
29         int x, y;
30         scanf("%d%d", &x, &y);
31         mx[x].insert(y);
32         my[y].insert(x);
33     }
34     for(int i = 0; i < m; i++)
35     {
36         int x, y;
37         scanf("%d%d", &x, &y);
38         if(x == 0) ans = deal(mx, my, y);
39         else ans = deal(my, mx, x);
40         printf("%d\n", ans);
41     }
42     printf("\n");

```

例2: NC 116634 uva11020 Efficient Solutions

- 有 n 个人, 每个人有两个属性 x, y , 如果对于一个人 $P(x, y)$, 不存在另外一个人 (a, b) , 使得 $a < x, b \leq y$ 或者 $a \leq x, b < y$, 则这个人是有优势的。
- 动态插入每个人, 要求统计当前已经插入的人中, 有优势的人的个数。

```

23 scanf("%d",&n);
24 st.clear();
25 for (int i = 1; i <= n; i++)
26 {
27     int x ,y;
28     scanf("%d%d", &x, &y);
29     ty p;
30     p.x = x;
31     p.y = y;
32     multiset <ty>::iterator it = st.lower_bound(p);
33     if(it == st.begin() || (--it) -> y > y)
34     {
35         st.insert(p);
36         it = st.upper_bound(p);
37         while(it != st.end() && it -> y >= y) st.erase(it++);
38     }
39     printf("%d\n", st.size());
40 }

```

```

6 struct ty
7 {
8     int x,y;
9     bool operator < (const ty & a) const
10     {
11         return x < a.x || (x == a.x && y < a.y);
12     }
13 };

```


例4: NC124063 UVALive7146 Defeat the Enemy

- 我方和敌方的每一个军队都有一个攻击力属性和防御力属性，当一方军队的攻击力大于等于对方的防御力，就可以摧毁敌军（有可能双方同归于尽）。对战只允许单挑，且一个队伍只能上场一次，现给出我方和敌方所有军队的攻击力和防御力，问要摧毁敌方所有军队，最多能保留多少的存活的军队。



- 是的这是一个贪心。
- 按照敌方军队的防御从大到小排序（先打对方防御厉害的，避免剩下的人打不死对方了）。
- 这个时候肯定希望选一个能打死对方且方能存活能存活的军队去对阵，而且选能存活的里面防御最小的，防御更大的留给后面说不定会更好。（如果选一个当前不能存活的X，我方存活军队数目必然最多；选一个当前能存活的，也许X换对手之后不能存活了，存活军队数目依然最多减一；所有换一个不能存活的肯定的不会更好。）
- 如果我方不管上哪一个都不能存活呢？那肯定是选一个防御最小的，把最容易死的用掉，存活的机会留给后面。

并查集

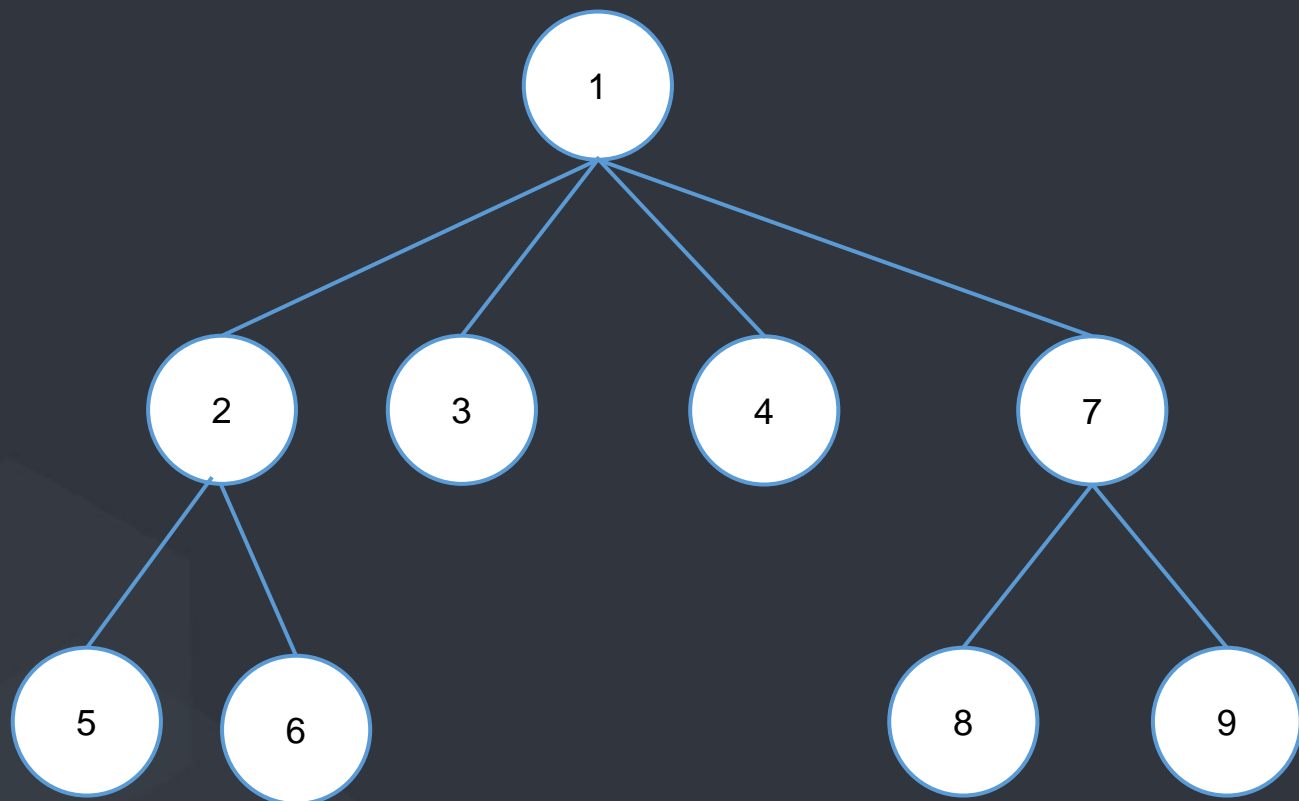


并查集

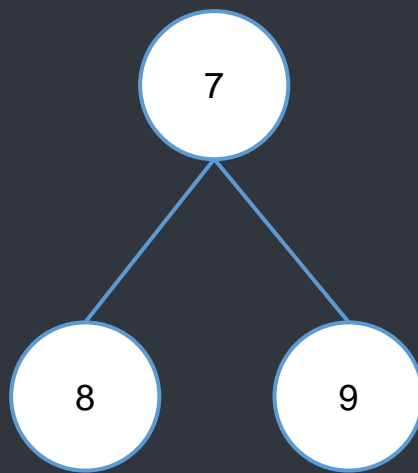
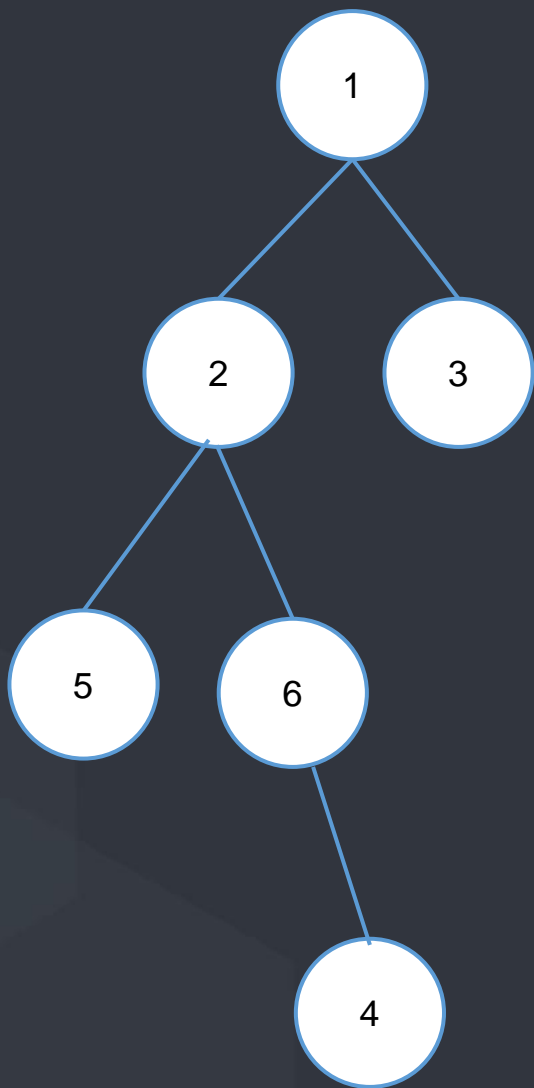
- 实现集合的合并与查找
- 用树来存储一个集合
- 如果两个点有共同的根，他们就在一个集合里
- 合并两个点所在集合只需要把一个点的根接到另一个点的根下面就行



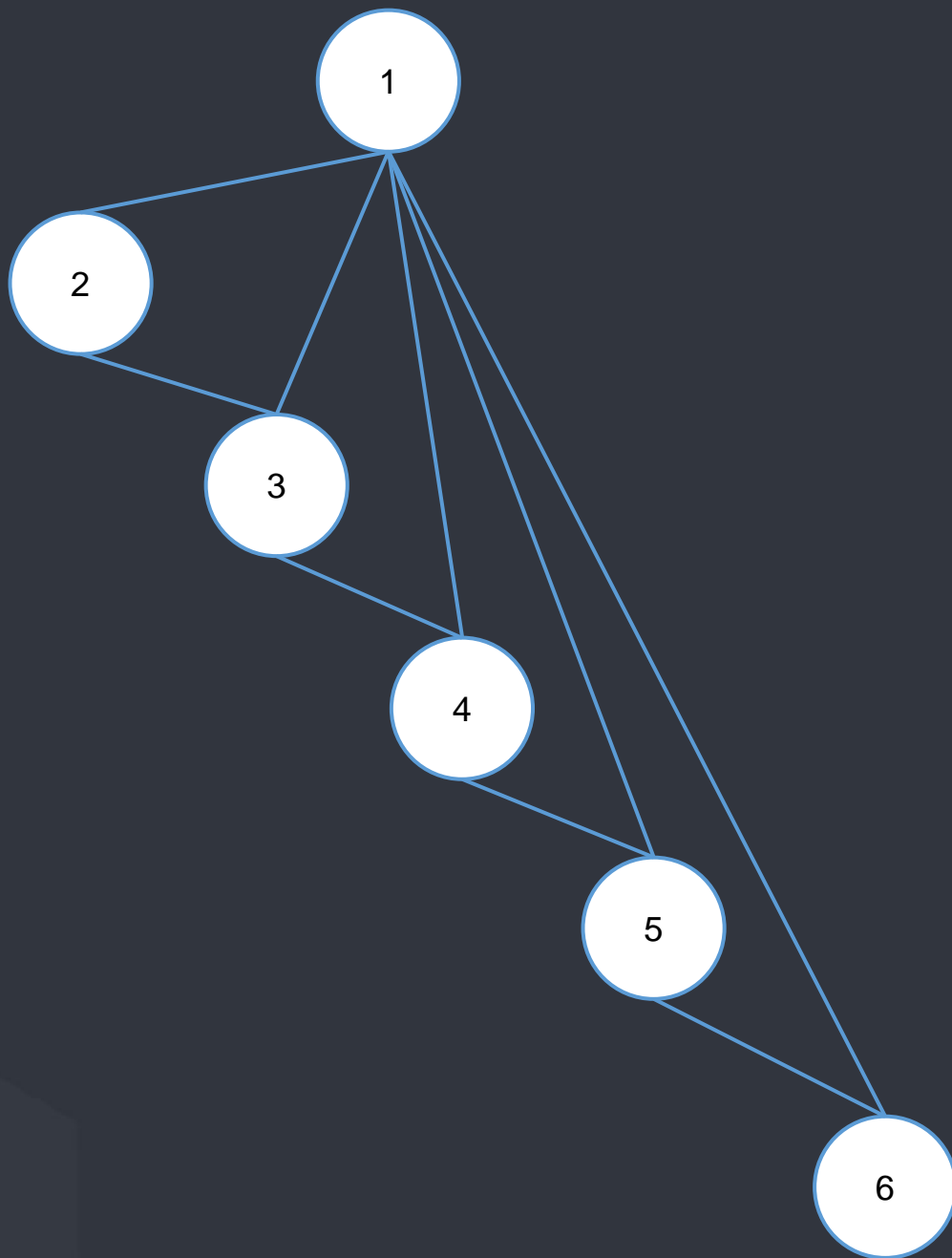
合并5, 9所在的集合



按秩合并



路径压缩



- 如果只有路径压缩，或者只有按秩合并，并查集单次操作的复杂度都是 $O(\log n)$ 。
路径压缩 + 按秩合并的并查集，单次操作的复杂度才是 $O(\alpha(N))$ 。这里 α 是Ackerman函数的某个反函数，在很大的范围内（人类目前观测到的宇宙范围估算有10的80次方个原子，这小于前面所说的范围）这个函数的值可以看成是不大于4的，所以并查集的操作可以看作是常数的。

例1：NC 23803 DongDong认亲戚

- DongDong每年过春节都要回到老家探亲，然而DongDong记性并不好，没法想起谁是谁的亲戚（定义：若A和B是亲戚，B和C是亲戚，那么A和C也是亲戚），她只好求助于会编程的你了

例2: NC 106585 poj 1988 Cube Stacking

- 有 n 个从1到 n 编号的箱子,将每个箱子当做一个栈,对这些箱子进行 p 次操作,每次操作分别为以下两种之一:
- 1、输入 $M \ x \ y$:表示将编号为 x 的箱子所在的栈放在编号为 y 的箱子所在栈的栈顶.
- 2、输入 $C \ x$:计算编号为 x 的所表示的栈中在 x 号箱子下面的箱子数目.



例3: NC 16884 食物链

- 动物王国中有三类动物A,B,C，这三类动物的食物链构成了有趣的环形。A吃B， B吃C， C吃A。 现有N个动物，以1 - N编号。每个动物都是A,B,C中的一种，但是我们并不知道它到底是哪一种。 有人用两种说法对这N个动物所构成的食物链关系进行描述：
- 第一种说法是"1 X Y"，表示X和Y是同类。
- 第二种说法是"2 X Y"，表示X吃Y。
- 此人对N个动物，用上述两种说法，一句接一句地说出K句话，这K句话有的是真的，有的是假的。问有多少句假话。



例4: NC 16591 关押罪犯

- S 城现有两座监狱，一共关押着 N 名罪犯，编号分别为 $1 \sim N$ 。他们之间的关系自然也极不和谐。如果两名怨气值为 c 的罪犯被关押在同一监狱，他们俩之间会发生摩擦，并造成影响力为 c 的冲突事件。
- 每年年末，警察局会将本年内监狱中的所有冲突事件按影响力从大到小排成一个列表，然后上报到S 城Z 市长那里。公务繁忙的Z 市长只会去看列表中的第一个事件的影响力，如果影响很坏，他就会考虑撤换警察局长。
- 在详细考察了 N 名罪犯间的矛盾关系后，警察局长准备将罪犯们在两座监狱内重新分配，以求产生的冲突事件影响力都尽量小。假设只要处于同一监狱内的某两个罪犯间有仇恨，那么他们一定会在每年的某个时候发生摩擦。那么，应如何分配罪犯，才能使Z市长看到的那个冲突事件的影响力最小？这个最小值是多少？



例5:

- 给你一些点，还有一些边，每个点上都有一个权值，然后有一些询问操作，分为两种
- Q a 询问与a直接或者间接想连的点中最大权值的是哪个点，输出那个点
- D a b 删除a b的边



作业:

- <https://ac.nowcoder.com/acm/problem/collection/1206>



顺序:

- NC212914 牛牛与后缀表达式
- NC21874 好串
- NC16430 蚯蚓
- NC50528 滑动窗口
- NC14893 栈和排序
- NC15029 吐泡泡
- NC15688 Operating System
- NC15975 小C的记事本
- NC20806 区区间间间
- NC16663 合并果子
- NC50940 Running Median
- NC50439 tokitsukaze and Soldier
- NC20154 [JSOI2007]建筑抢修
- NC20185 [JSOI2010]缓存交换
- NC17315 背包



- NC14661 简单的数据结构
- NC15128 老子的全排列呢
- NC17508 指纹锁
- NC17889 新建 Microsoft Office Word 文档
- UVA11020 Efficient Solutions
- UVALive7146 Defeat the Enemy
- NC23803 DongDong认亲戚
- NC14545 经商
- NC16884 食物链
- POJ2492 A Bug's Life
- POJ1988 Cube Stacking
- NC14685 加边的无向图
- NC15976 小C的周末

补充：分块



分块

- 分块其实是一种非常优雅的暴力



例1:

- 给出一个长为 n 的数列，以及 n 个操作，每次操作可以对一个区间的每一个数加上一个数，或者查询一个点的值。



例1:

- 暴力算法 修改一次 $O(n)$ 查询一次 $O(1)$
- 问题就出在修改太慢上面



例1:

- 由于每次操作的区间加的是同一个数，考虑将数列中每 m 个元素进行一个打包
- 打包之后，如果是对这个包整体进行一个 $+x$ 的操作，那么就标记一下，暂时不需要给每个数都加上这个数
- 如果是对区间的一部分加上一个数，那么就把区间中每一个数的值都更新一下。



[6,11]+2

下标

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----

各块的标记

				+5+2=+7						+5									
--	--	--	--	---------	--	--	--	--	--	----	--	--	--	--	--	--	--	--	--

各个点的值

0	0	0	0	5	0	0	0	0	0	2	0	0	0	0	5	5	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



- 每次修改改变最多 n/m (m 为块的大小)个区间和最多 $2m-2$ 个单独的位置, 所有单次修改时间复杂度为 $O(n/m) + O(m)$
- 因为 $a + b \geq 2\sqrt{ab}$, 所以 $n/m = m$ 的时候最小, 块的大小取 \sqrt{n}



例2:

- 给出一个长为 n 的数列，以及 n 个操作，每次操作可以对一个区间的每一个数加上一个数，或者查询一个区间的和。