

算法基础——递归分治stl


邓丝雨



分治

- 天下大势，合久必分，分久必合
- 孙子曰：凡治众如治寡，分数是也；斗众如斗寡，形名是也；三军之众，可使必受敌而无败者，奇正是也；兵之所加，如以礮投卵者，虚实是也。——《孙子兵法》



- 
- 当我们求解某些问题时，由于这些问题要处理的数据相当多，或求解过程相当复杂，使得直接求解在时间上相当长，或者根本无法直接求出。对于这类问题，我们往往先把它**分解成几个子问题**，找到求出这几个子问题的解法后，再找到合适的方法，把它们组合成求整个问题的解法。如果这些子问题还较大，难以解决，可以再把它们分成几个更小的子问题，以此类推，直至可以直接求出解为止。这就是分治策略的基本思想。



递归的概念：

- 递归的定义是：
- ——参见递归！
- 实际上，递归的意思就是，一个函数在执行时再次调用函数“本身”。





- `int f(int x)`
- `{`
- `if(x==0) return 1;`
- `return x*f(x-1);`
- `}`



- `int fib(int x)`
- `{`
- `return (x <= 1)? 1: fib(x - 1) + fib(x - 2);`
- `}`



牛客竞赛
AC.NOWCODER.COM



例1：汉诺塔问题

- 汉诺塔（又称河内塔）问题是源于印度一个古老传说的益智玩具。大梵天创造世界的时候做了三根金刚石柱子，在一根柱子上从下往上按照大小顺序摞着64片黄金圆盘。大梵天命令婆罗门把圆盘从下面开始按大小顺序重新摆放在另一根柱子上。并且规定，在小圆盘上不能放大圆盘，在三根柱子之间一次只能移动一个圆盘。





- 求把所有圆盘从A柱移到C柱的最小步数 和 具体方案



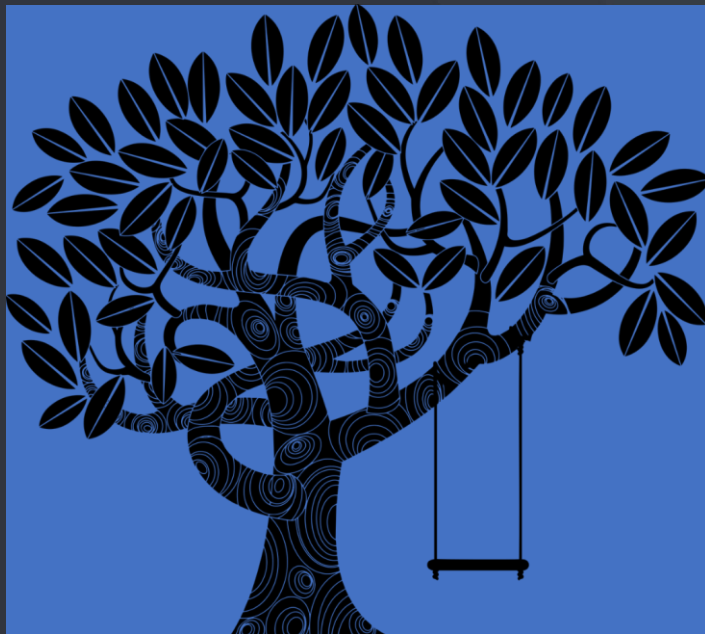
- 要把 n 个从A移到C
 - 前 $n-1$ 个从A移到B
 - 第 n 个从A移到C
 - 前 $n-1$ 个从B移到C

例2：给出二叉树的前序中序求后序





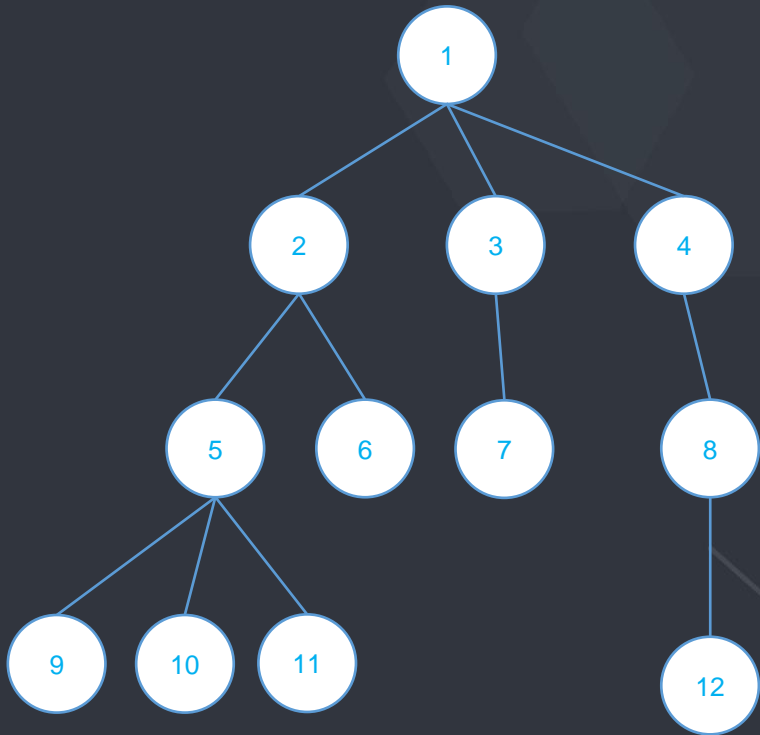
什么是树?





树

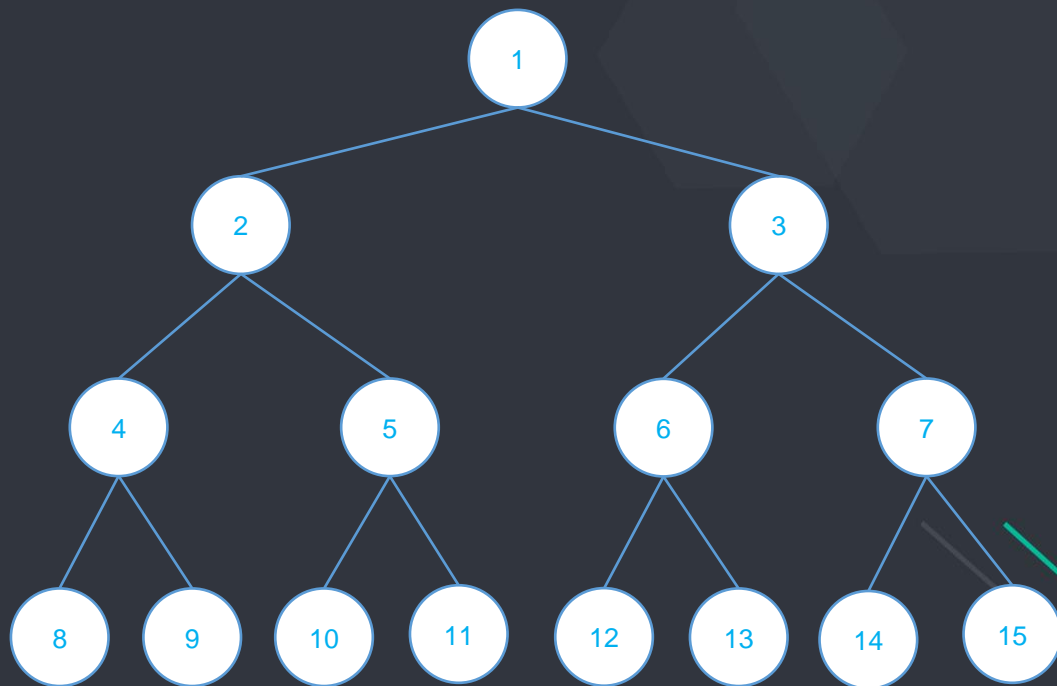
- 树是一种数据结构，它是由 n ($n \geq 1$) 个有限节点组成一个具有层次关系的集合。把它叫做“树”是因为它看起来像一棵倒挂的树，也就是说它是根朝上，而叶朝下的。它具有以下的特点：
- 每个节点有零个或多个子节点
- 没有父节点的节点称为根节点
- 每一个非根节点有且只有一个父节点；除了叶子节点外，每个子节点可以分为多个不相交的子树





二叉树

- **二叉树**：每个节点最多含有两个子树的树称为二叉树；
- **满二叉树**：除最后一层无任何子节点外，每一层上的所有结点都有两个子结点的二叉树。
- **完全二叉树**：除最后一层外，每一层上的节点数均达到最大值；在最后一层上只缺少右边的若干结点。





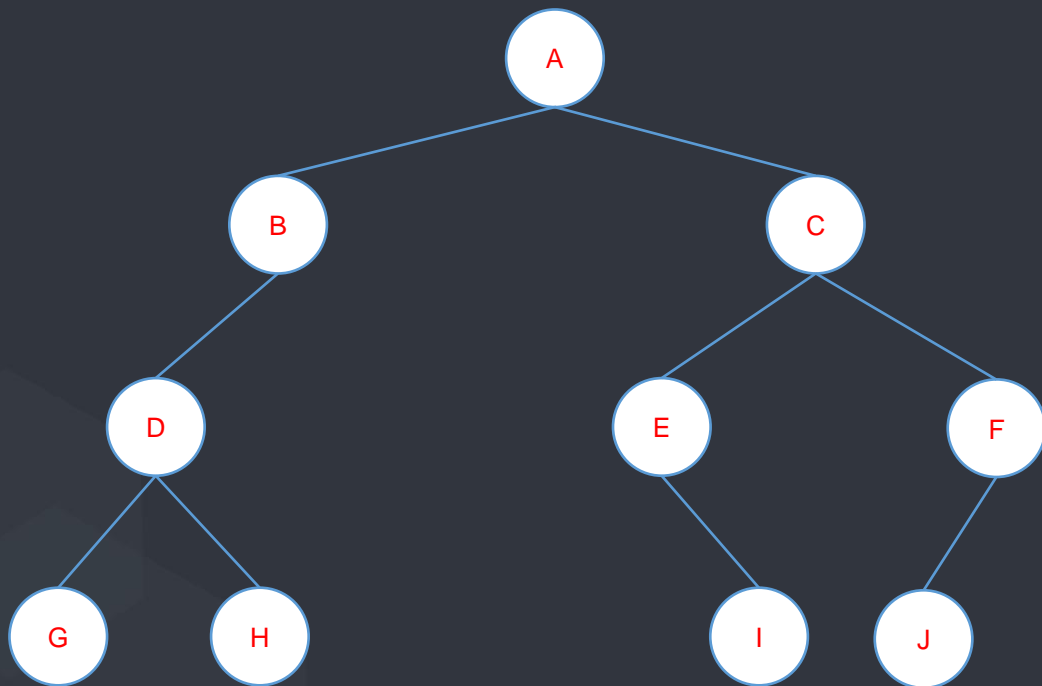
二叉树的先序遍历、中序遍历、后序遍历和层序遍历

- 先序遍历也叫做先根遍历、前序遍历，首先访问根结点然后遍历左子树，最后遍历右子树。在遍历左、右子树时，仍然先访问根结点，然后遍历左子树，最后遍历右子树，如果二叉树为空则返回。
- 中序遍历首先遍历左子树，然后访问根结点，最后遍历右子树。在遍历左、右子树时，仍然先遍历左子树，再访问根结点，最后遍历右子树。
- 后序遍历首先遍历左子树，然后遍历右子树，最后访问根结点，在遍历左、右子树时，仍然先遍历左子树，然后遍历右子树，最后遍历根结点。
- 层序遍历就是按二叉树的每一层的顺序来遍历,也就是先访问根,然后访问第一层,接着访问第二层





二叉树的四种遍历





例2：已知前序中序求后序

Eg:前:ABCD FE 中:BAD FCE





扩展：204382 中序序列

- 给定一棵有 n 个结点的二叉树的先序遍历与后序遍历序列，求其中序遍历序列。
- 若某节点只有一个子结点，则此处将其看作左儿子结点





- 有了这个条件之后：先序遍历中根的右边第一个一定是他的左儿子，后续遍历中夹在左儿子和根之间的是右子树。然后我们就可以区分左右儿子了。





例3：归并排序&快速排序

- 三种 $O(n^2)$ 的排序：冒泡、选择、插入
- 三种不基于比较的排序：桶、基数、计数





例3：归并排序&快速排序

- 归并排序：每次把待排序区间一分为二，将两个子区间排序，然后将两个已经排好序的序列合并
- 快速排序：选择一个基准，将小于基准的放在基准左边，大于基准的放在基准右边，然后对基准左右都继续执行如上操作直到全部有序。





归并

```
6 void _merge(int l, int mid, int r)
7 {
8     int p1 = l, p2 = mid + 1;
9     for (int i = l; i <= r; i++)
10     {
11         if ((p1 <= mid) && ((p2 > r) || (a[p1]) <= a[p2]))
12         {
13             b[i] = a[p1];
14             p1++;
15         }
16         else {
17             b[i] = a[p2];
18             p2++;
19         }
20     }
21     for (i = l; i <= r; i++) a[i] = b[i];
22 }
```

```
24 void erfen(int l , int r)
25 {
26     int mid = (l + r) >> 1;
27     if (l < r)
28     {
29         erfen(l , mid);
30         erfen(mid + 1, r);
31     }
32     _merge(l, mid, r);
33 }
```



牛客竞赛

AC.NOWCODER.COM



快排

```
10 void quick_sort(int l, int r)
11 {
12     int i = l, j = r;
13     int mid = (l + r) / 2;
14     int x = a[mid];
15     while (i <= j)
16     {
17
18         while (a[i] < x) i++;
19         while (a[j] > x) j--;
20         if (i <= j)
21         {
22             swap(a[i], a[j]);
23             i++; j--;
24         }
25     }
26     if (l < j) quick_sort(l, j);
27     if (i < r) quick_sort(i, r);
28 }
```





求逆序对个数

- 给你一个序列，求出这个序列的逆序对个数
- 逆序对：对于一个包含 N 个非负整数的数组 $A[1..n]$ ，如果有 $i < j$ ，且 $A[i] > A[j]$ ，则称 $(A[i], A[j])$ 为数组 A 中的一个逆序对。





求一个序列的第k大数

```
5  int finding(int l, int r, int k)
6  {
7      if (l == r) return a[l];
8      int i = l, j = r;
9      int mid = (l + r) / 2;
10     int x = a[mid];
11     while (i <= j)
12     {
13         while (a[i] < x) i++;
14         while (a[j] > x) j--;
15         if (i <= j)
16         {
17             swap(a[i], a[j]);
18             i++; j--;
19         }
20     }
21     if (k <= j) return finding(l, j, k);
22     else if (i <= k) return finding(i, r, k);
23     else return a[k];
24 }
```





例4：求最大子串和

- 给你一个序列，求其最大连续的一段和
- （有动态规划解法，这里暂时不讨论）





例4：求最大子串和

- 把序列分成两部分：
- 左边的最大和
- 右边的最大和
- 左边包含最右点的最大和 + 右边包含最左点的最大和





```
10 int maxs(int n, int l, int r)
11 {
12     int mid = (l+r) / 2;
13     if(l == r) return a[l];
14     int ans = max(maxs(n, l, mid), maxs(n, mid+1, r));
15     int ll = 0, rr = 0, tmp = 0;
16     for(int i = mid + 1 ; i <= r; i++)
17     {
18         tmp += a[i];
19         rr = max(rr, tmp);
20     }
21     tmp = 0;
22     for(int i = mid ; i >= l; i--)
23     {
24         tmp += a[i];
25         ll = max(ll, tmp);
26     }
27     ans = max(ans, ll+rr);
28     return ans;
29 }
```





NC50999 表达式计算4

- 给出一个表达式,其中运算符仅包含 $+$, $-$, $*$, $/$, $^$ (加 减 乘 整除 乘方) 要求求出表达式的最终值
- 数据可能会出现括号情况,还有可能出现多余括号情况
- 数据保证不会出现 $\geq 2^{31}$ 的答案
- 数据可能会出现负数情况

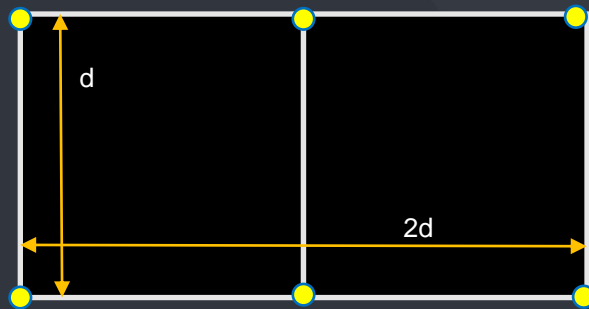
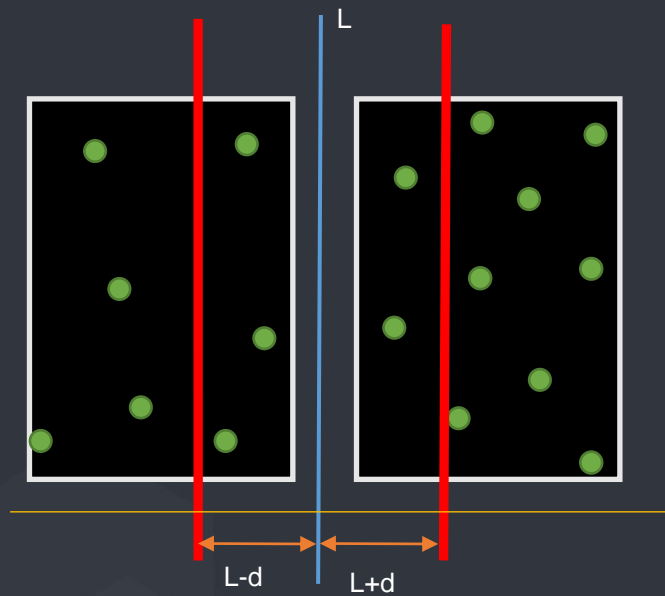




最近点对Raid

- 在二维平面上的 n 个点中，找出距离最近的两个点。







```
31 double near_dis(int l, int r)
32 {
33     if(r == l+1) return dis(p[l], p[r]);
34     if(l+2 == r) return min(dis(p[l], p[r]), min(dis(p[l], p[l+1]), dis(p[l+1], p[r])));
35     int mid = (l + r) >> 1;
36     double ans = min(near_dis(l, mid), near_dis(mid+1, r));
37     int cnt = 0;
38     for(int i = l; i <= r; i++)
39         if(p[i].x >= p[mid].x - ans && p[i].x <= p[mid].x + ans)
40             a[cnt++] = i;
41     sort(a, a+cnt, compy);
42     for(int i = 0; i < cnt; i++)
43         for(int j = i + 1; j < cnt; j++)
44         {
45             if(p[a[j]].y - p[a[i]].y >= ans) break;
46             ans = min(ans, dis(p[a[i]], p[a[j]]));
47         }
48     return ans;
49 }
```





补充：倍增

- 倍增，从字面的上意思看就是成倍的增长。指我们在处理一个问题的时候，如果问题的规模很大，线性递推无法满足时间和空间复杂度的要求，就可以通过成倍的增长，先求解状态空间中在 2 的整数次幂位置上的值作为代表，再对答案进行求解拼接。





求a的b次方模p的值

- 思路: $a^{10} = a^{(1010)_2} = a^{1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0} = a^{1 \times 2^3} \times a^{1 \times 2^1}$
- 我们可以通过倍增的方法先计算模p意义下的 $a^1, a^2, a^4, a^8, a^{16}, a^{32}$ 再把它们根据b的二进制拆分乘起来。



二分、三分、分数规划



二分

- 单调函数求0点





二分查找

- 在一个**单调有序**的集合中查找元素，每次将**集合**分为左右两部分，判断解在哪个部分中并调整**集合**上下界，重复直到找到目标元素。
- 例如在以下序列中查找55

0	5	13	19	22	41	55	68	72	81	98
---	---	----	----	----	----	----	----	----	----	----

55	68	72	81	98
----	----	----	----	----

55	68
----	----

55





C++ STL的二分查找函数

- `binary_search` 返回bool值,是否存在
 - `lower_bound` 返回可插入的最小位置的迭代器
 - 即返回第一个符合条件的元素位置
 - `upper_bound` 返回可插入的最大位置的迭代器
 - 即返回最后一个符合条件的元素位置
-
- Eg:
 - `lower_bound(a, a+11, 55)`





例1:

- 给一串 n 个单调递增的数，有 q 次询问 $\geq x$ 且 $\leq y$ 的数有多少个
- 数据规模: $1 \leq n \leq 10^5$ $1 \leq q \leq 50000$

```
14  int find_low(int x)
15  {
16      int mid, l = 1, r = n;
17      while (l <= r)
18      {
19          mid = (l + r) >> 1;
20          if (a[mid] < x) l = mid + 1;
21          else r = mid - 1;
22      }
23      return l;
24  }
25  int find_up(int x)
26  {
27      int mid, l = 1, r = n;
28      while (l <= r)
29      {
30          mid = (l + r) >> 1;
31          if (a[mid] <= x) l = mid + 1;
32          else r = mid - 1;
33      }
34      return l;
35  }
```




例2:

- 给你N个机器和M个任务，每个任务有两个值花费时间x和难度y，每个机器也有两个值最大工作时间x1和最大工作难度y1，机器可以胜任某个工作的条件是 $x_1 \geq x \ \&\& \ y_1 \geq y$ ，机器胜任一个工作可以拿到 $x*500+2*y$ 的钱，现在问你怎么匹配才能使匹配数最大且钱数最多。
- $X_i \leq 1440 \ y_i \leq 100$
- $1 \leq N \leq 100000, 1 \leq M \leq 100000$





二分答案+检验

- 对于难以直接确定解的问题,采取**二分枚举+检验**的思想将求解类问题转换为验证类问题
- $\geq K$ 可行
- $< K$ 不可行
- 目标就是求K





例3:

- 有N个牛棚在x轴上,已知他们的坐标.FJ有C只奶牛,每只都必须安排在一个牛棚里,一个牛棚只能容纳一只.但是他们会互相攻击,所以要求距离最近的两个牛棚间的距离最大.
- $2 \leq N \leq 100,000$
- $0 \leq x_i \leq 1,000,000,000$
- $2 \leq C \leq N$





例3:

- 我们可以假设距离最近的两个牛棚间的距离为 x , 你能判断这个 x 是否可行吗?
- 先把 x_i 排序
- 对于一个解 m ,我们验证 m 是否可行.
- 第一个点放置牛.
- 从左向右 $O(N)$ 扫描一遍牛棚,第 i 个点如果和上一个放置点的距离 $\geq m$ 则在第 i 个点放置一头牛,统计总放置的数量是否 $\geq C$.





24
25
26
27
28
29
30
31
32
33
34
35
36
37
38

```
while (scanf("%d%d", &n, &c) != EOF)
{
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
    sort(a + 1, a + n + 1);

    int mid, l = 1, r = a[n];
    while (l <= r)
    {
        mid = (l + r) >> 1;
        if (judge(mid)) l = mid + 1;
        else r = mid - 1;
    }
    cout << r << endl;
}
return 0;
```

```
7 bool pan(int x)
8 {
9     int la = a[1];
10    int cnt = 1;
11    for (int i = 2; i <= n; i++)
12    {
13        if (a[i] - la >= x)
14        {
15            la = a[i];
16            cnt++;
17        }
18        if (cnt >= c) return true;
19    }
20    return false;
21 }
```



牛客竞赛

AC.NOWCODER.COM



例4:

- 有 n 件衣服需要晾干,每件含水量 a_i .每件衣服每分钟自然干1单位的水.每分钟可以对其中任意一件使用吹风机,其可以减少 k 的水.求晾干所有衣服的最少时间.
- $1 \leq n \leq 100\ 000$
- $1 \leq a_i \leq 10^9$
- $1 \leq k \leq 10^9$





- 设某次二分出的一个值是mid:
- 1、对于一件ai值小于等于mid的衣服, 直接晾干即可;
- 2、对于一件ai值大于mid值的衣服, 最少的用时是用吹风机一段时间, 晾干一段时间, 设这两段时间分别是x1和x2, 那么有 $mid = x1 + x2$, $ai \leq k * x1 + x2$, 解得 $x1 \geq (ai - mid) / (k - 1)$, 所以对 $(ai - mid) / (k - 1)$ 向上取整就是该件衣服的最少用时。
- 我们把所有的使用吹风机时间加起来, 这个总和应该 $\leq mid$





- 解决最大值最小 or 最小值最大的常见方法
- 贪心——国王游戏
- 二分
- 动态规划



牛客竞赛

AC.NOWCODER.COM



例5:

- 题意：给你N行4列的数，从每一列选一个数，问使他们的和为0的情况有多少种
($N \leq 4000$)





- 既然有四列，那么我们可以分别计算前两列和后两列的和（只需要 $n*n*2$ 次运算），然后对后两列的和排序，那么我们对于每一个前两列的和都可以二分找到后两列的和中与之相加为0的个数，这样的复杂度就是 $O(n*n*\log(n))$ 是可以接受的，





NC19916 [CQOI2010]扑克牌

- 你有 n 种牌，第 i 种牌的数目为 c_i 。另外有一种特殊的牌：joker，它的数目是 m 。你可以用每种牌各一张来组成一套牌，也可以用一张joker和除了某一种牌以外的其他牌各一张组成1套牌。比如，当 $n=3$ 时，一共有4种合法的套牌： $\{1,2,3\}$, $\{J,2,3\}$, $\{1,J,3\}$, $\{1,2,J\}$ 。给出 n , m 和 c_i ，你的任务是组成尽量多的套牌。每张牌最多只能用在一副套牌里（可以有牌不使用）。





- 直接贪心很难构造又因为是满足单调性的（套数大于答案的都不可行，小于等于答案的都可行）我们直接二分答案考虑验证。

如果当前待验证的数字的 x （即判断当前手上的牌够不够组成 x 套）——首先，张数大于等于 x 的牌只需要每一套用一张就行，小于 x 的牌差几张就需要补几张joker，于是我们可以计算出究竟要补几张joker。如果最后需要补的joker数比手上有的joker数多，说明joker不够，不行。如果最后需要补的joker数比 x 还大，说明必然有两个joker在一套牌里，也不行。其他情况都是可以的。





NC14301 K-th Number

- 对数列A的每个区间求第K大，并将第k大插入到B中，再求B的第M大。





- 把求值变成验证——

首先二分答案，每次二分的时候我们得到一个 x ，这个时候我们利用尺取，去得到第 K 大数大于 x 的区间一共有多少，如果大于 $m-1$ 个说明 x 取小了。

那么第 k 大数大于 x 的区间一共有多少怎么求？第 k 大数大于 x 的其实也就是大于 x 的数至少有 k 个，当我们枚举区间左界 L ，我们可以从 L 往右扫描到第一个大于 x 的数有 k 个的点 R ，右界在这个位置及其之后的区间大于 x 的点的个数都大于等于 k 个。而当 L 右移一个位置， R 显然只会右移不会左移，所以 L 和 R 对每一个数字都只访问一遍，这样，时间复杂度是 $O(n)$ 的了。加上二分就是 $O(n \log n)$ 了。





三分

类似二分的定义Left和Right

$\text{mid} = (\text{Left} + \text{Right}) / 2$

$\text{midmid} = (\text{mid} + \text{Right}) / 2$;

如果mid靠近极值点, 则 $\text{Right} = \text{midmid}$;

否则(即midmid靠近极值点), 则 $\text{Left} = \text{mid}$;



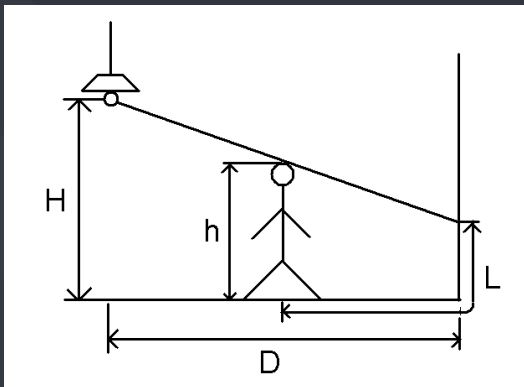
牛客竞赛

AC.NOWCODER.COM



例1:

- 如图，人左右走动，求影子 L 的最长长度。
- 根据图，很容易发现当灯，人的头部和墙角成一条直线时(假设此时人站在A点)，此时的长度是影子全在地上的最长长度。当人再向右走时，影子开始投影到墙上，当人贴着墙，影子长度即为人的高度。所以当人从A点走到墙，函数是先递增再递减，为凸性函数，所以我们可以用三分法来求解。





例2:

- 在一个2维平面上有两条传送带，每一条传送带可以看成是一条线段。两条传送带分别为线段AB和线段CD。lxhgww在AB上的移动速度为 P ，在CD上的移动速度为 Q ，在平面上的移动速度 R 。现在lxhgww想从A点走到D点，他想知道最少需要走多长时间。





01分数规划

- 有一堆物品，每一个物品有一个收益 a_i ，一个代价 b_i ，我们要求一个方案选若干个物品使得所选择的 $\frac{\sum a_i}{\sum b_i}$ 最大。









Thanks

