二分，三分，哈希，高精度，位运算，模拟退火

曾锦吉

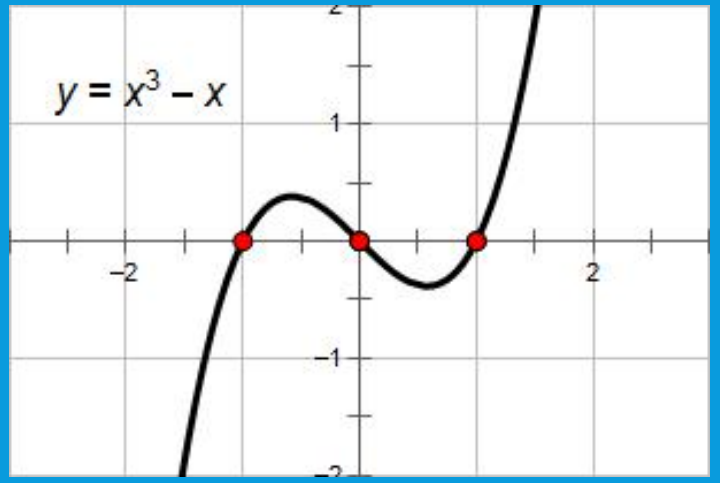
什么是二分法

高中数学必修一上给的定义是：

对于在区间 $[a, b]$ 上连续不断且 $f(a)f(b) < 0$ 的函数 $y = f(x)$，通过不断地把函数 $f(x)$ 零点所在区间一分为二，使区间的两个端点逐步逼近零点，进而得到零点近似值的方法，叫做二分法(Bisection)。

「连续」「变号」「近似值」

同时，只能求出一组解。



标准二分法

根据刚刚的描述我们可以写出代码：

```
double Find_Zero_Point(double left, double right, double precesion) {
    // f(left) * f(right) < 0
    if (right - left < precesion) return left;
    double mid = (left + right) / 2;
    if (f(mid) == 0) return mid;
    if (f(mid) * f(right) > 0) return Find_Zero_Point(left, mid, precesion);
    else return Find_Zero_Point(mid, right, precesion);
}
```

标准二分法

可以改写为非递归形式：

```
double Find_Zero_Point(double left, double right, double precesion) {
    // f(left) * f(right) < 0
    while (right - left > precesion) {
        double mid = (left + right) / 2;
        if (f(mid) == 0) return mid;
        if (f(mid) * f(right) > 0) right = mid;
        else return left = mid;
    }
    return left;
}
```

NOIP 中出现过的二分法

使用或者可以使用二分法解答的 NOIp 题：

NOIp 2010 关押罪犯

NOIp 2011 聪明的质检员

NOIp 2012 借教室

NOIp 2012 疫情控制

NOIp 2014 跳石头

NOIp 2015 运输计划

可以说是十分频繁了。

算法竞赛中的二分法

直接应用二分法的有两种：

1. 二分查找；
2. 二分答案。

二分法通常可以和其他算法一起考察，通常都成为其它算法的载体。

难度范围非常广，下至 NOIp 第一题，上至 NOI 压轴题。

二分法应用十分广泛，很多其他算法都有间接用到二分算法的情况。

二分查找算法

二分查找是一种在有序数组中查找某一特定元素的查找算法。

查找过程从数组的中间元素开始：

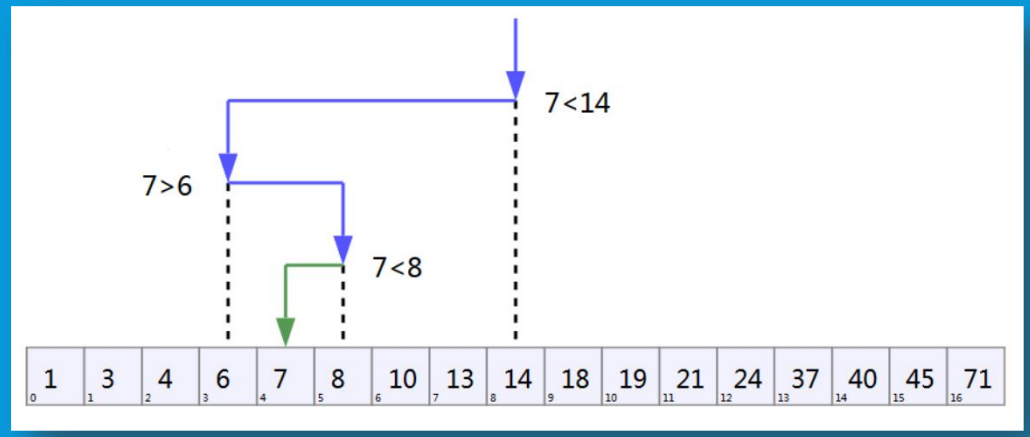
如果中间元素正好是要查找的元素，则查找过程结束；

如果某一特定元素大于或者小于中间元素，则在数组大于或小于中间元素的那一半中查找，而且跟开始一样从中间元素开始比较。

如果在某一步骤数组为空，则代表找不到。

「连续」「单调」「确切答案」

解通常是唯一的，或者不存在。



Source:
https://upload.wikimedia.org/wikipedia/commons/8/83/Binary_Search_Depiction.svg

二分查找算法

根据刚刚的描述我们可以写出代码：

```
int Binary_Search(int a[], int left, int right, int value) {
    if (left > right) return -1;
    int mid = (left + right) / 2;
    if (a[mid] == value) return mid;
    if (a[mid] > value) return Binary_Search(a, left, mid - 1, value);
    else return Binary_Search(a, mid + 1, right, value);
}
```

二分查找算法

可以改写为非递归的形式：

```
int Binary_Search(int a[], int left, int right, int value) {
    while (left <= right) {
        int mid = (left + right) / 2;
        if (a[mid] == value) return mid;
        if (a[mid] > value) right = mid - 1;
        else return left = mid + 1;
    }
    return -1;
}
```

二分查找算法

更简短的形式，同时保证返回的是第一个等于的值：

```
int Binary_Search(int a[], int left, int right, int value) {
    while (left < right) {
        int mid = (left + right) / 2;
        if (a[mid] >= value) right = mid;
        else left = mid + 1;
    }
    return (a[left] == value) ? left : -1;
}
```

使用这种写法需要注意边界，避免陷入死循环。

二分查找算法

事实上，在效率要求不高或者有优化的情况下，可以使用 STL 来进行二分查找。

std::lower_bound() 返回有序表里第一个大于等于给定值的指针或迭代器；

std::upper_bound() 返回有序表里第一个大于给定值的指针或迭代器；

std::binary_search() 返回给定值是否在有序表里存在。

二分查找算法

数组用法：

```
int a[5] = {0, 2, 4, 6, 8};
printf("%d %d %d\n", lower_bound(a, a + 5, 2) - a, upper_bound(a, a + 5, 2) - a,
binary_search(a, a + 5, 2));
printf("%d %d %d\n", lower_bound(a, a + 5, 3) - a, upper_bound(a, a + 5, 3) - a,
binary_search(a, a + 5, 3));
```

输出结果：

```
1 2 1
2 2 0
```

二分查找算法

vector 用法：

```
lower_bound(a.begin(), a.end(), 2)
```

vector 的迭代器是可以减的；你可以像数组一样对它进行减运算来获取下标。

二分查找算法

set / map / … 用法：

```
a.lower_bound(2)
```

set / map / … 的迭代器是不可减的。

二分查找与标准二分法的异同

把数组看成函数的话，二分查找就是定义在整数区间上的二分法。

相比之下，二分查找额外地要求单调；同时能够找到准确答案。

应用

除了 NOIp 初赛题，题目通常不会直接考察二分查找算法的。

在 NOIp 题目里，更经常出现的是在作为辅助其他算法的存在。

例如：NOIp 1999 导弹拦截 / NOIp 2004 合唱队形）：

优化最长上升子序列的动态规划转移。

对于上午的第一题和第二题，均是有二分查找的正解写法。

二分答案

二分答案，即通过题目中包含的单调性对答案进行二分。

大多数二分答案可以看做是标准二分法套了一个奇奇怪怪的函数。

解答需要使用二分答案的题目，最重要的是观察出单调性。

二分答案的共性

通常来说，二分答案题目为下列两种中的一种：

1. 给定一个评价函数，求评价函数的最小值 / 最大值 。
2. 给定一个条件，要求在满足条件的同时，使得代价最小。

计算函数或者判断符合条件需要消耗极长的时间；

或者评价函数的值域太大了，不能一一计算。

二分答案图解

二分答案的写法

二分答案也只是一种简单的二分，而且写法也非常套路，以求最小值为例：

```
while (left < right) {
    int mid = (left + right) / 2;
    if (check(mid)) right = mid;
    else left = mid + 1;
}
answer = left;
```

对于大多数题目，重写 check 函数足以解决问题。

二分答案的套路

二分答案有着非常套路化的做法。

通常只需两步：

1. 观察单调性，找到需要计算的条件；
2. 二分，并验证条件。

二分答案的难点通常落在发现单调性和验证条件上。

验证条件通常都会涉及到离线的思想。

二分答案题面中的套路

通常来说，题面中有，或者暗含类似于：

- 最大值最小 / 最小值最大
- 最靠近某个值
- 最小的能满足条件的代价

那么通常都可以往二分答案想。

NOIP2011 聪明的质检员

小 T 是一名质量监督员，最近负责检验一批矿产的质量。这批矿产共有 n 个矿石，从 1 到 n 逐一编号，每个矿石都有自己的重量 w_i 以及价值 v_i 。

检验矿产的流程是：

1. 给定 m 个区间 $[L_i, R_i]$；
2. 选出一个参数 W；
3. 对于一个区间 $[L_i, R_i]$，计算矿石在这个区间上的检验值 y ：

$$y_i = (\sum_j 1) \times (\sum_j v_j), j \in [L_i, R_i] \text{且} w_j \geq W$$

也就是 w 大于等于 W 的个数和乘以价值和。

这批矿产的检验结果 Y 为各个区间的检验值之和 。即： $Y = \sum_{i=1}^{n} y_i$

他想通过调整参数 W 的值，让检验结果尽可能的靠近标准值 S ，即使得 $S - Y$ 的绝对值最小。

$1 \leq n, m \leq 2 \times 10^5$ ， $0 < w_i, v_i \leq 10^6$ ， $0 < S \leq 10^{12}$ 。

NOIP2011 聪明的质检员

寻找单调性：可以发现 W 越大， Y 越小。

此时， $Y = f(W)$ 是一个单调不升的函数；

问题转变为求函数 $|f(W) - S|$ 的最小值，也就是求 $f(W) - S$ 零点或者最接近零点的点。二分法解决即可。

NOIP2011 聪明的质检员

如何计算 $f(W)$？

NOIP2011 聪明的质检员

如何计算 $f(W)$？

根据 W，可以离线求出有贡献的 w_i 以及个数的前缀和；

查询一个区间时，取出两个前缀和相减乘起来就好了。

复杂度：$O(n\log n)$

NOIP2015 跳石头

一年一度的“跳石头”比赛又要开始了！

这项比赛将在一条笔直的河道中进行，河道中分布着一些巨大岩石。组委会已经选择好了两块距离为 L 的岩石作为比赛起点和终点。在起点和终点之间，有 N 块岩石(不含起点和终点的岩石)。在比赛过程中，选手们将从起点出发，每一步跳向相邻的岩石，直至到达终点。

为了提高比赛难度，组委会计划移走一些岩石，使得选手们在比赛过程中的最短跳跃距离尽可能长。由于预算限制，组委会至多从起点和终点之间移走 M 块岩石(不能移走起点和终点的岩石)。

$0 \le M \le N \le 50000$, $1 \le L \le 1000000000$。

NOIP2015 跳石头

寻找单调性：答案关于移动石头的个数单调。

移动石头的个数越多，答案越大。

我们令 $f(x)$ 表示，当最小距离不低于 x 时，最小需要移动多少颗石头。

二分找出最后一个 $f(x) \leq M$ 的 x 即为答案。

NOIP2015 跳石头

如何求 $f(x)$？

NOIP2015 跳石头

如何求 $f(x)$？

一种贪心的策略是，从头开始，如果这一块石头与上一块石头的距离小于 x，就直接把这一块石头移去。最后统计移去的石头的总个数。

为什么这样做是对的?

可用反证法证明。

复杂度：$O(n\log L)$

NOIP2012 借教室

我们需要处理 n 天的借教室信息，其中第 i 天学校有 r_i 个教室可供租借。

共有 m 份订单，每份订单用三个正整数描述，分别为d_j, s_j, t_j，表示某租借者需要从第 s_j 天到第 t_j 天租借教室（包括第 s_j 天和第 t_j 天），每天需要租借 d_j 个 教室。

借教室的原则是先到先得，也就是说我们要按照订单的先后顺序依次为每份订单分配教室。如果在分配的过程中遇到一份订单无法完全满足，则需要停止教室的分配，通知当前申请人修改订单。这里的无法满足指从第 s_j 天到第 t_j 天中有至少一天剩余的教室数量不足 d_j 个。

现在我们需要知道，是否会有订单无法完全满足。如果有，需要通知哪一个申请人修改订单。

$1 \le n, m \le 10^6$, $0 \le r_i, d_j \le 10^9$, $1 \le s_j \le t_j \le n$。

NOIP2012 借教室

寻找单调性：前 i 天的条件一定比前 $i+1$ 天条件更容易。

证明很简单，如果前 i 天都不满足了，那么前 $i+x$ 天都不可能被满足。

考虑二分答案。

NOIP2012 借教室

如何验证前 i 天是否满足条件?

NOIP2012 借教室

如何验证前 i 天是否满足条件?

由于天数已经固定了，需求也就固定了。

问题转化为，有很多次离线区间加，最后要求判断每一个元素是否小于 d 。

离线区间加我们可以差分后使用前缀和的技巧来解决。

复杂度：$O((n+m)\log m)$。

当然，本题还可以直接使用数据结构解决。

NOIP2010 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~N。

M 对罪犯之间积怨已久，如果客观条件具备则随时可能爆发冲突。我们用“怨气值” 来表示某两名罪犯之间的仇恨程度。

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。

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最大值最小。

求出最优方案下，影响力最大的事件的影响力最小值。

$N \leq 20000$，$M \leq 100000$。

NOIP2010 关押罪犯

寻找单调性：满足更少分配要求比满足更多的要求更容易。

因此，我们一定是首先去满足影响力最大的那些冲突。

考虑二分答案。

NOIP2010 关押罪犯

如何验证某些条件是否能够全部满足?

NOIP2010 关押罪犯

如何验证某些条件是否能够全部满足?

将需要满足的分配的条件看成边，我们需要判断这张图是不是二分图。

使用 DFS / BFS，黑白染色即可。

复杂度：$O((n+m)\log m)$。

当然，本题还可以直接使用带权并查集 / “敌人集合”并查集解决。

NOIP2015 运输计划

L 国有 n 个星球，还有 $n-1$ 条双向航道，每条航道建立在两个星球之间，这 $n-1$ 条航道连通了 L 国的所有星球。对于航道 j，任意飞船驶过它所花费的时间为 t_j，并且任意两艘飞船之间不会产生任何干扰。

小 P 掌管一家物流公司，该公司有很多个运输计划，每个运输计划形如：有一艘物流飞船需要从 u_i 号星球沿最快的宇航路径飞行到 v_i 号星球去。

L 国国王同意小 P 把某一条航道改造成虫洞，飞船驶过虫洞不消耗时间。

在虫洞的建设完成前小 P 的物流公司就预接了 m 个运输计划；在虫洞建设完成后， 这 m 个运输计划会同时开始，所有飞船一起出发。

如果小 P 可以自由选择将哪一条航道改造成虫洞，试求出小 P 的物流公司 m 个运输计划都完成所需要的最短时间是多少?

$N \leq 300000$，$M \leq 300000$。

NOIP2015 运输计划

寻找单调性：满足更大最大值比满足更小最大值的要求更容易。

因此，我们一定要在那些代价大于某个值的运输计划上找到一条共有的边。

考虑二分答案。

NOIP2015 运输计划

如何验证某个最大值是否可行?

NOIP2015 运输计划

如何验证某个最大值是否可行?

考虑对大于这个最大值的运输计划求边的交集。

如果交集中有不小于 (最大运输计划代价 – 目标最大值) 的边，那么这个最大值是可行的。

NOIP2015 运输计划

如何验证某个最大值是否可行?

考虑对大于这个最大值的运输计划求边的交集。

如果交集中有不小于 (最大运输计划代价 – 目标最大值) 的边，那么这个最大值是可行的。

求交集有两种方案:

1. 直接标记

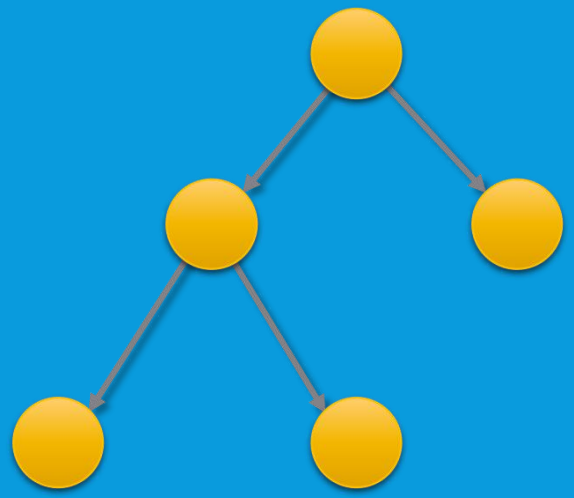
2. 对树链进行操作

NOIP2015 运输计划

直接标记：

如果二分再使用类似树链剖分的处理树链的办法很可能导致超时。

我们考虑直接在树上进行标记。

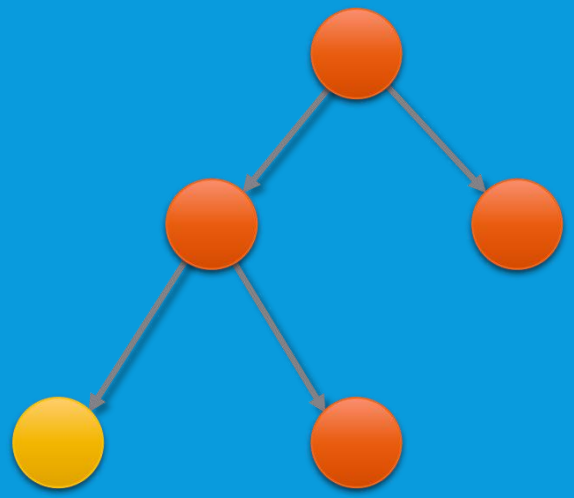


NOIP2015 运输计划

直接标记：

如果二分再使用类似树链剖分的处理树链的办法很可能导致超时。

我们考虑直接在树上进行标记。

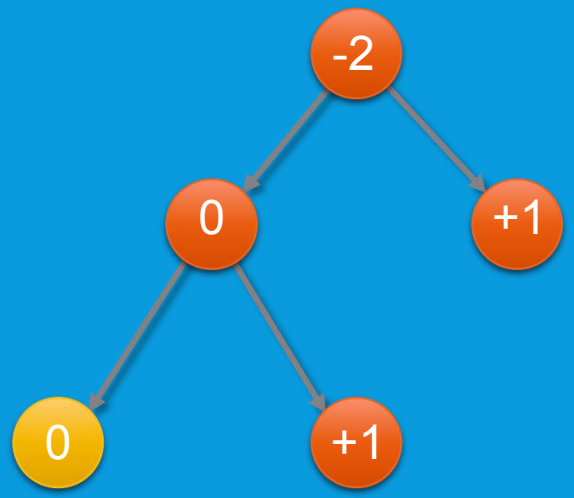


NOIP2015 运输计划

直接标记：

如果二分再使用类似树链剖分的处理树链的办法很可能导致超时。

我们考虑直接在树上进行标记。

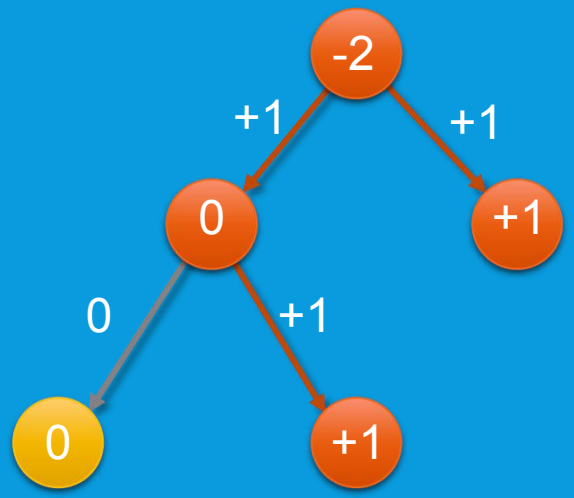


NOIP2015 运输计划

直接标记：

如果二分再使用类似树链剖分的处理树链的办法很可能导致超时。

我们考虑直接在树上进行标记。



NOIP2015 运输计划

直接标记：

如果二分再使用类似树链剖分的处理树链的办法很可能导致超时。

我们考虑直接在树上进行标记。

标记 LCA、起点和终点，一条边被标记的次数就是子树内的点上值的和；

最后进行一次 DFS 求得交集。

复杂度：$O((n+m)\log m)$

NOIP2015 运输计划

对树链进行操作：

有一个结论是，任意两条树链的交集一定是一条树链，或者为空；

同时，如果有交，树链的交集的端点一定在：

S_1, S_2, T_1, T_2

$LCA(S_1, S_2), LCA(S_1, T_2), LCA(S_2, T_1), LCA(S_2, T_2)$

之中取得。

可以用反证法证明。

NOIP2015 运输计划

因此，我们可以在求四次 LCA 的时间复杂度内完成一次求交。

可以预处理 LCA 来消去一个 log；因此时间复杂度只有一个 log。

复杂度：$O((n+m)\log m)$

NOIP2015 运输计划

我们可以发现，二分答案是不必要的，从按运输计划长度从高到低求交即可。

可以使用 Tarjan 算法预处理 LCA；

可以使用均摊复杂度的暴力处理查询树链最大值。

因此，复杂度是线性的。

复杂度：$O(n+m)$。

NOIP2012 疫情控制

H国有 n 个城市，这 n 个城市构成一棵树，1号城市是首都（根节点）。

H国的首都爆发了一种危害性极高的传染病。当局为了控制疫情，不让疫情扩散到边境城市（叶子节点），决定动用军队在一些城市建立检查点，使得从首都到边境城市的每一条路径上都至少有一个检查点，边境城市也可以建立检查点。但特别要注意的是，首都是不能建立检查点的。

在H国的一些城市中已经驻扎有军队，且一个城市可以驻扎多个军队。一支军队可以在有道路连接的城市间移动，并在除首都以外的任意一个城市建立检查点。一支军队经过一条道路所需要的时间为道路的长度（单位：小时）。不同的军队可以同时移动。

请问最少需要多少个小时才能控制疫情，如果无法控制疫情则输出-1。

$2 \le m \le n \le 50000$，$0 < w < 10^9$

NOIP2012 疫情控制

首先先判断无解：

如果首都的出度大于军队数量，那么一定控制不住；

否则，一定控制得住。

10 分到手，excited!

之后的讨论，均是在有解的情况下进行的。

NOIP2012 疫情控制

寻找单调性：时间越长，越容易满足条件。

对时间进行二分答案。

NOIP2012 疫情控制

如何判断指定时间内，军队是否能够完成控制的要求?

NOIP2012 疫情控制

如何判断指定时间内，军队是否能够完成控制的要求?

发现性质：军队越往根走作用越大——因为所能够控制的子树变大了。

NOIP2012 疫情控制

如何判断指定时间内，军队是否能够完成控制的要求?

发现性质：军队越往根走作用越大——因为所能够控制的子树变大了。

考虑贪心：

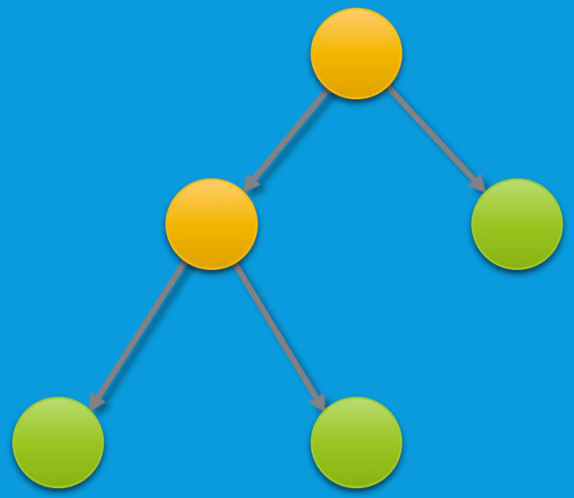
每个军队先贪心地向上走，直到在规定时间内无法走到父亲城市，或者父亲城市是根为止。

这个过程可以用倍增实现。

NOIP2012 疫情控制

此时，先将父亲不是根的军队从下至上标记；

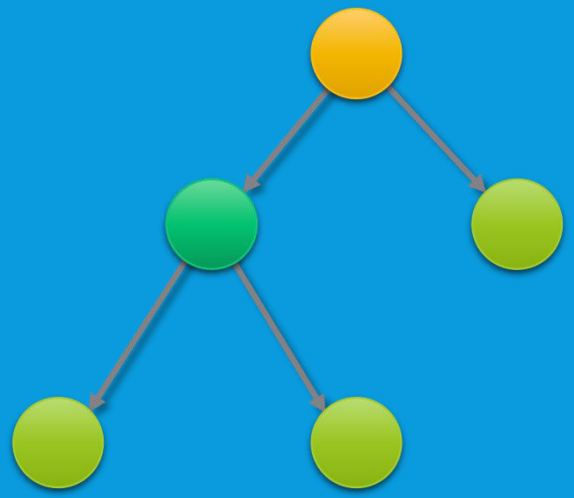
如果一个节点的儿子被标记了，那么它也应该被标记。



NOIP2012 疫情控制

此时，先将父亲不是根的军队从下至上标记；

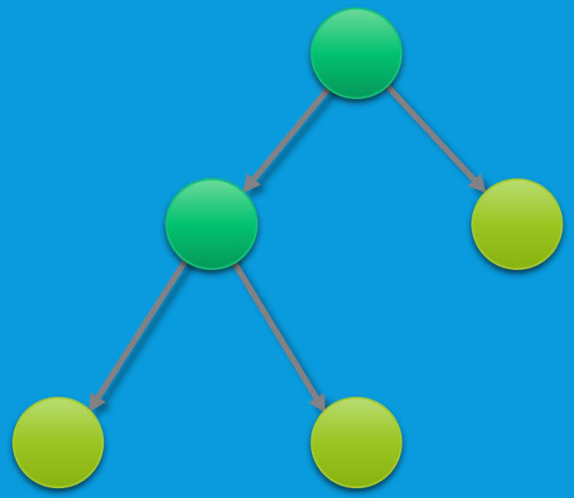
如果一个节点的儿子被标记了，那么它也应该被标记。



NOIP2012 疫情控制

此时，先将父亲不是根的军队从下至上标记；

如果一个节点的儿子被标记了，那么它也应该被标记。



NOIP2012 疫情控制

父亲已经是根的军队可以考虑去支援其他城市。

把这些的军队按照剩余时间和目标点扔进两个数组里，和目标点到根节点的距离从小到大排序。

剩余时间从小到大拿出军队：

如果那个军队所在的点还没有被标记，所以就标记这个城市；

否则就尝试找个离根节点最近的目标点分配给它。

通过查看最后根节点的出边是否全部被标记了判断当前答案是否可行。

NOIP2012 疫情控制

这道题细节非常多，实现时需要严谨；不然容易写错。

时间复杂度：$O(n\log n\log W)$。

小结

二分法解决问题套路明显，思路清晰。

在设计二分答案的检验函数时，应该尽可能地考虑清楚复杂度，并适当优化。

常用的检验算法有：

贪心

动态规划

进行数据结构操作（通常是类似于前缀和的离线的形式）

网络流

小结

同时，不是要死磕在二分法上：

如果验证方法足够优秀，完全可以抛弃二分，直接使用优秀的方法求得答案。

例如：直接求交集的运输计划

拓展

某些数据结构和算法实质上也是利用了二分法来解决问题。

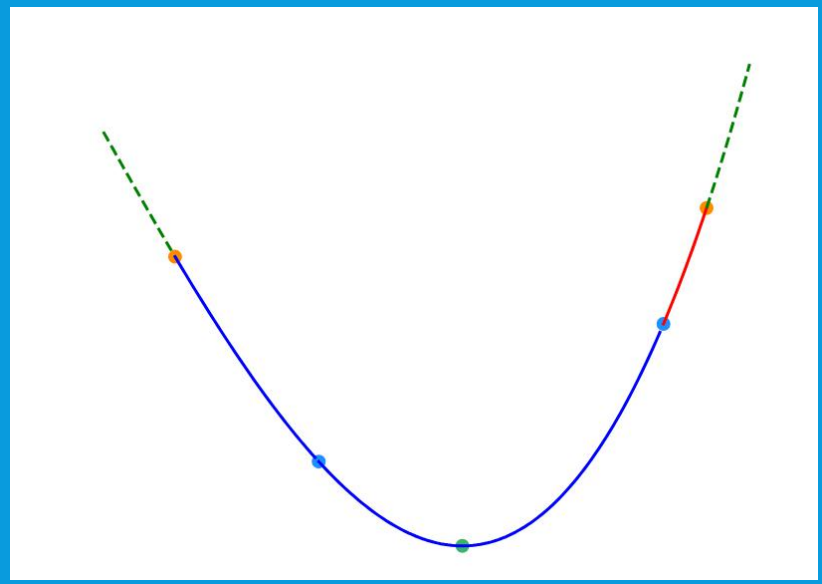
- 二叉搜索树（线段树，平衡树……）
- 倍增算法
- 高精度运算的除法的一种实现形式
- 各类分治算法的“一分为二”是否也算呢……

更高更妙的二分？

- 分数规划
- 整体二分

什么是三分

三分法与二分法的基本思想类似，但每次操作需在当前区间 [l,r]（下图中除去虚线范围内的部分）内任取两点 lmid,rmid(lmid < rmid)（下图中的两蓝点）。如下图，如果 f(lmid)<f(rmid)，则在 [rmid,r]（下图中的红色部分）中函数必然单调递增，最小值所在点（下图中的绿点）必然不在这一区间内，可舍去这一区间。反之亦然。



三分

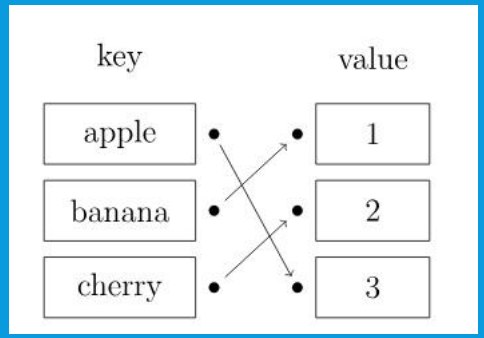
```
while (r - l > eps) {
  mid = (l + r) / 2;
  lmid = mid - eps;
  rmid = mid + eps;
  if (f(lmid) < f(rmid))
    r = mid;
  else
    l = mid;
}
while (left+eps<right)
          l_mid = left+(right-left)/3;
          r_mid = right-(right-left)/3;
          f (check(l_mid) > check(r_mid))
          {    right = r_mid; }
         else {left = l_mid;       }
       }
```

例题：

多组测试数据. n个选手参加比赛. 比赛分为跑步和骑车两项,但 总长固定. 已知每位选手进行每种项目时的速度,求是否可以通 过调整各项目长度使某指定选手获胜. 如果可以获胜还需输出与 第二名的耗时差最大值及此时各项目长度. 输出保留两位小数. 其中$n \leqslant 20$, $t \leqslant 100$

哈希算法

哈希表又称散列表，一种以「key-value」形式存储数据的数据结构。所谓以「key-value」形式存储数据，是指任意的键值 key 都唯一对应到内存中的某个位置。只需要输入查找的键值，就可以快速地找到其对应的 value。可以把哈希表理解为一种高级的数组，这种数组的下标可以是很大的整数，浮点数，字符串甚至结构体。



哈希算法

哈希算法将输入内容进行映射,输出映射结果. 对于两个不同的输入内容映射到相同位置,有多种不同处理方法. 设计良好的哈希算法有利于减少冲突

哈希函数可以处理整数、实数、字符串、排列组合，树等数据

整数的HASH函数

最常见的是直接取余法

- $h(k) = k \bmod M$，M 为Hash表的容量
- 为了保证随机性，应该尽量使 k 的每一位都对 h(k) 产生影响
- M 应选取不太接近 2 的幂的素数
- 例如 k = 100, M = 12, 那么 h(k) = 4

把关键字 k 平方，中间几位受 k 的每一位影响最大

由此想到平方取中法

- 把关键字 k 平方，取出中间的 r 位作为 h(k) 的值
- 此时 $M = 2^r$

例子：

- r = 4, k = 100
- $k = (1100100)_2$
- $k^2 = (10011\ 1000\ 10000)_2$
- $h(k) = (1000)_2 = 8$

整数的HASH函数

另一种方法：**利用无理数**

乘积取整法

用 k 乘以一个(0,1)中的无理数 A

取出小数部分，乘以 M

再取出整数部分作为 h(k)

例：k = 100, A = 0.61803…, M = 12

h(k) = 9

整数的HASH函数

比较这三种方法：

直接取余法：

- 实现容易
- 效果受 M 影响大

乘积取整法

- M 可以任取，效果好
- 速度奇慢

平方取中法

- 速度快
- 不容易推广

结论：一般的竞赛应用，直接取余法足矣

字符串的HASH函数

字符串信息量巨大，无法直接定址

可以把字符串看成 256 进制的大整数，套用直接取余法

M 选得好的话，效果还是可以接受的

例子：str = “IOI2005”, M = 23

str = 0x494F4932303035, M = 0x17

h(str) = str % M = 13

h(“NOI2005”) = 1

h(“IOI2004”) = 12

出现了扎堆，这是直接取余法的必然现象

字符串的HASH函数

更主流的是多项式哈希

对于一个长度为 l 的字符串 s 来说，我们可以这样定义多项式 Hash 函数：、

$f(s) = \sum_{i=1}^{l} s[i] \times b^{l-i} \pmod{M}$。

例如，对于字符串 xyz，其哈希函数值为 xb^2+yb+z。

字符串的HASH函数

```
using std::string;
const int M = 1e9 + 7;
const int B = 233;
typedef long long ll;
int get_hash(const string& s) {
  int res = 0;
  for (int i = 0; i < s.size(); ++i) {
    res = ((ll)res * B + s[i]) % M;
  }
  return res;
}
bool cmp(const string& s, const string& t) {
  return get_hash(s) == get_hash(t);
}
```

字符串HASH函数应用

多次询问子串哈希

字符串匹配

最长回文子串

最长公共子字符串

确定字符串中不同子字符串的数量

例题：雪花

有n片雪花,每片雪花由六个整数描述.两片雪花相同当且仅当它们的描述旋转或翻转后相同,求是否有两片相同的雪花.

其中$0<n\leqslant 1e5$

例题：[HNOI2014]抄卡组

多组测试数据.输入N个含通配符的字符串,询问它们是否可能相同.

其中N≤1e5,测试数据组数=10,N×最长字符串长度≤2×1e8.

例题：回文串

【题目描述】

令$F(A,B)$ 表示选择一个串 的非空前缀A 和串B 的非空后缀 使得将串S 和串T 拼接起来之后是回文串的方案数。

现在给定两个串A 和B，令A_i 表示串A 的第i 长的后缀，B_i 为串B 的第i 长的前缀。

有Q 组询问，第i 组询问给定x_i 和y_i，对每组询问求$F(A_{x_i}, B_{y_i})$ 的值。

【数据规模】

对于100%的数据，字符串中只出现小写字母。

子任务编号	子任务分值	$max(\|A\|,\|B\|)$	Q	数据类型
1	10	≤ 40	≤ 200	C
2	15	≤ 2000	≤ 2000	A
3	7	$\leq 10^5$	$=1$	C
4	19	≤ 2000	≤ 2000	C
5	20	$\leq 8\times 10^5$	$\leq 10^5$	B
6	26	$\leq 8\times 10^5$	$\leq 10^5$	C
7	3	$\leq 8\times 10^5$	$\leq 10^5$	C

数据类型，A：数据随机；B：串 随机且$|B| \leq 10^4$；C：无特殊性质。

高精度

在平常的实现中，高精度数字利用字符串表示，每一个字符表示数字的一个十进制位。因此可以说，高精度数值计算实际上是一种特别的字符串处理。

读入字符串时，数字最高位在字符串首（下标小的位置）。但是习惯上，下标最小的位置存放的是数字的 最低位，即存储反转的字符串。这么做的原因在于，数字的长度可能发生变化，但我们希望同样权值位始终保持对齐（例如，希望所有的个位都在下标 [0]，所有的十位都在下标 [1]……）；同时，加、减、乘的运算一般都从个位开始进行（回想小学的竖式运算），这都给了「反转存储」以充分的理由。

高精度加

高精度加法，其实就是竖式加法啦。

也就是从最低位开始，将两个加数对应位置上的数码相加，并判断是否达到或超过 10。如果达到，那么处理进位：将更高一位的结果上增加 1，当前位的结果减少 10。

```
void add(int a[], int b[], int c[]) {
  clear(c);
for (int i = 0; i < LEN - 1; ++i) {
    // 将相应位上的数码相加
    c[i] += a[i] + b[i];
    if (c[i] >= 10) {
      c[i + 1] += 1;
      c[i] -= 10;
    }
  }
}
```

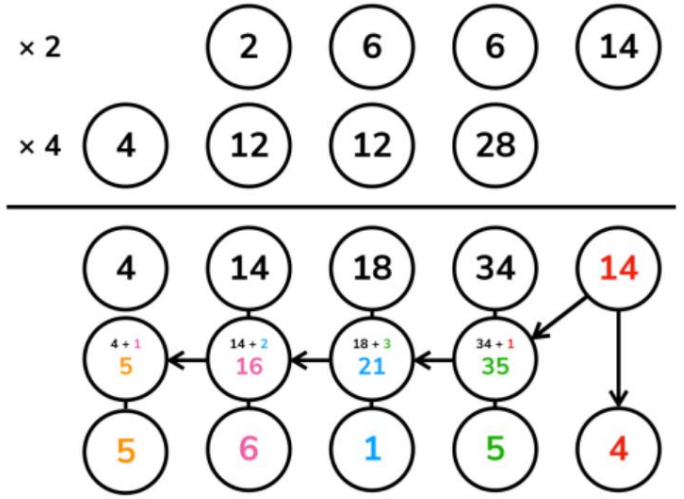
高精度减

高精度减法，也就是竖式减法啦。

```
void sub(int a[], int b[], int c[]) {
  clear(c);

  for (int i = 0; i < LEN - 1; ++i) {
    // 逐位相减
    c[i] += a[i] - b[i];
    if (c[i] < 0) {
      // 借位
      c[i + 1] -= 1;
      c[i] += 10;
    }
  }
}
```

高精度乘



高精度减法，也就是竖式乘法啦。（1337*42）

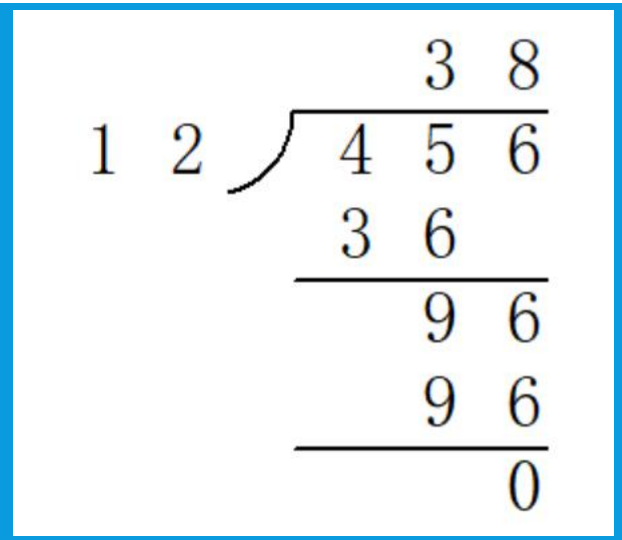
```
void mul(int a[], int b[], int c[]) {
  clear(c);
  for (int i = 0; i < LEN - 1; ++i) {
    // 这里直接计算结果中的从低到高第 i 位，且一并处理了进位
    // 第 i 次循环为 c[i] 加上了所有满足 p + q = i 的 a[p] 与 b[q] 的乘积之和
    // 这样做的效果和直接进行上图的运算最后求和是一样的，只是更加简短的一种实现方式
    for (int j = 0; j <= i; ++j) c[i] += a[j] * b[i - j];

    if (c[i] >= 10) {
      c[i + 1] += c[i] / 10;
      c[i] %= 10;
    }
  }
}
```

高精度除

高精度减法，也就是竖式除法啦。

```
// 被除数 a 以下标 last_dg 为最低位，是否可以再减去除数 b 而保持非负
// len 是除数 b 的长度，避免反复计算
bool greater_eq(int a[], int b[], int last_dg, int len) {
  // 有可能被除数剩余的部分比除数长，这个情况下最多多出 1 位，故如此判断即可
  if (a[last_dg + len] != 0) return true;
  // 从高位到低位，逐位比较
  for (int i = len - 1; i >= 0; --i) {
    if (a[last_dg + i] > b[i]) return true;
    if (a[last_dg + i] < b[i]) return false;
  }
  // 相等的情形下也是可行的
  return true;
}
```



高精度除

```
void div(int a[], int b[], int c[], int d[]) {
  clear(c);
  clear(d);
  int la, lb;
  for (la = LEN - 1; la > 0; --la)
    if (a[la - 1] != 0) break;
  for (lb = LEN - 1; lb > 0; --lb)
    if (b[lb - 1] != 0) break;
  if (lb == 0) {
    puts("> <");
    return;
  } // 除数不能为零
  // c 是商
  // d 是被除数的剩余部分，算法结束后自然成为余数
  for (int i = 0; i < la; ++i) d[i] = a[i];
  for (int i = la - lb; i >= 0; --i) {
    // 计算商的第 i 位
    while (greater_eq(d, b, i, lb)) {
      // 若可以减，则减
      // 这一段是一个高精度减法
      for (int j = 0; j < lb; ++j) {
        d[i + j] -= b[j];
        if (d[i + j] < 0) {
          d[i + j + 1] -= 1;
          d[i + j] += 10;
        }
      }
      // 使商的这一位增加 1
      c[i] += 1;
      // 返回循环开头，重新检查
    }
```

位运算

按二进制位进行运算

位运算的运算对象是二进制的位

位运算速度快，效率高，节省存储空间

只能对整型数据(包括字符型)进行位运算

负数以补码形式参与运算

注意与逻辑运算区别

位运算

运算符	名称	举例	优先级
~	按位取反	~flag	(高)
			(算术运算符)
<<	左移	a << 2	
>>	右移	b >> 3	
			(关系运算符)
&	按位与	flag & 0x37	
^	按位异或	flag ^ 0xC4	
|	按位或	flag | 0x5A	(低)
			(赋值运算符)

位运算常见操作

功能	示例	C语言
去掉最后一位	(101101->10110)	x>>1
在最后加一个0	(101101->1011010)	x<<1
在最后加一个1	(101101->1011011)	(x<<1)+1
把最后一位变成1	(101100->101101)	x|1
把最后一位变成0	(101101->101100)	(x|1)-1
最后一位取反	(101101->101100)	x^1
把右数第k位变成1	(101001->101101,k=3)	x|(1<<(k-1))
把右数第k位变成0	(101101->101001,k=3)	x&~(1<<(k-1))
右数第k位取反	(101001->101101,k=3)	x^(1<<(k-1))
取末三位	(1101101->101)	x&7
取末k位	(1101101->1101,k=5)	x&((1<<k)-1)
取右数第k位	(1101101->1,k=4)	(x>>(k-1))|1
把末k位变成1	(101001->101111,k=4)	x|((1<<k)-1)
末k位取反	(101001->100110,k=4)	x^((1<<k)-1)
把右边连续的1变成0	(100101111->100100000)	x&(x-1)
把右起第一个0变成1	(100101111->100111111)	x|(x+1)
把右边连续的0变成1	(11011000->11011111)	x|(x-1)
取右边连续的1	(100101111->1111)	(x^(x+1))>>1
去掉右起第一个1的左边	(100101000->1000)	x&(x^(x-1))
去掉右起第一个1的左边	(10010100->1000)	x&-x

最后两个树状数组会用到

位运算的运用

位运算一般有三种作用：

1.高效地进行某些运算，代替其它低效的方式。

2.表示集合（常用于 状压 DP）。

3.题目本来就要求进行位运算。

如果需要操作的集合非常大，可以使用 bitset 。

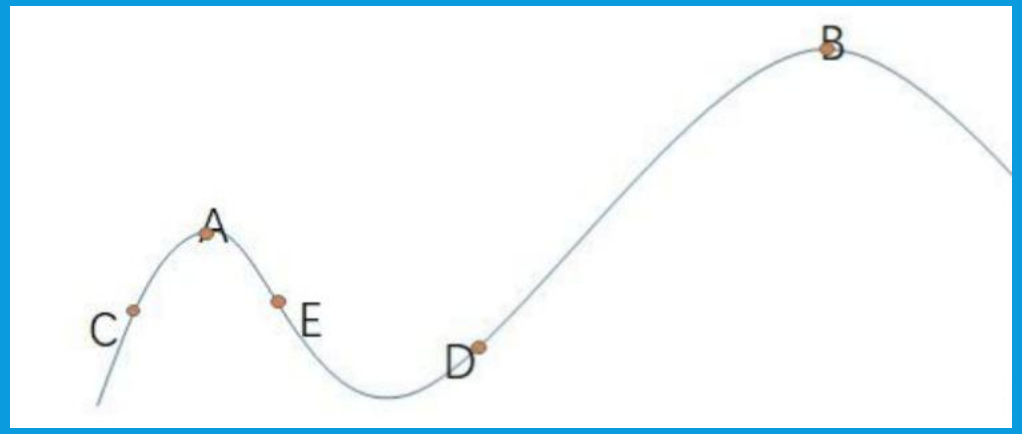
最短HAMILTON路径

给定一张 n(n≤20) 个点的带权无向图，点从0 ~ n−1标号，求起点 0 到终点 n-1 的最短Hamilton路径。 Hamilton路径的定义是从 0 到 n-1 不重不漏地经过每个点恰好一次

模拟退火

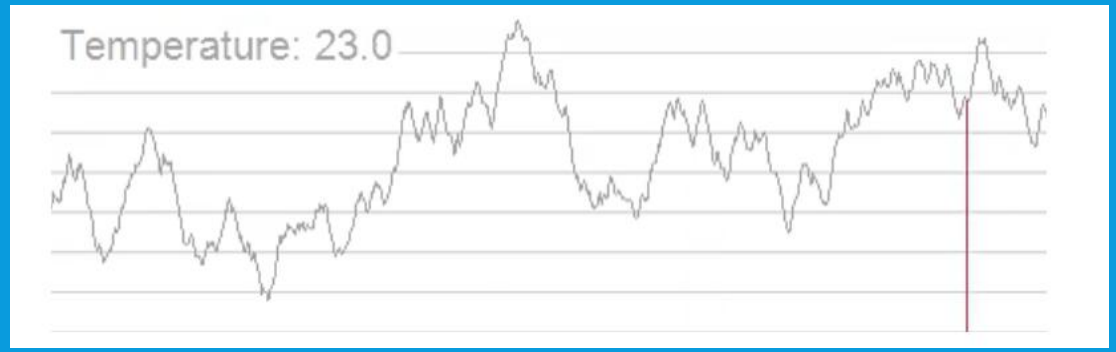
模拟退火是一种随机化算法。当一个问题的方案数量极大（甚至是无穷的）而且不是一个单峰函数时，我们常使用模拟退火求解。

从爬山算法讲起，爬山算法是一种简单的贪心搜索算法，该算法每次从当前解的临近解空间中选择一个最优解作为当前解，直到达到一个局部最优解。



模拟退火

模拟退火算法从某一较高初温出发，伴随温度参数的不断下降,结合一定的概率突跳特性在解空间中随机寻找目标函数的全局最优解，即在局部最优解能概率性地跳出并最终趋于全局最优。



模拟退火

用一句话概括：如果新状态的解更优则修改答案，否则以一定概率接受新状态。我们定义当前温度为 T，新状态 S‘ 与已知状态 S（新状态由已知状态通过随机的方式得到）之间的能量（值）差为ΔE，若$\Delta E \geq 0$则发生状态转移（修改最优解）的概率

$$P(\Delta E) = \begin{cases} 1, & S' \text{ is better than } S, \\ e^{\frac{-\Delta E}{T}}, & \text{otherwise.} \end{cases}$$

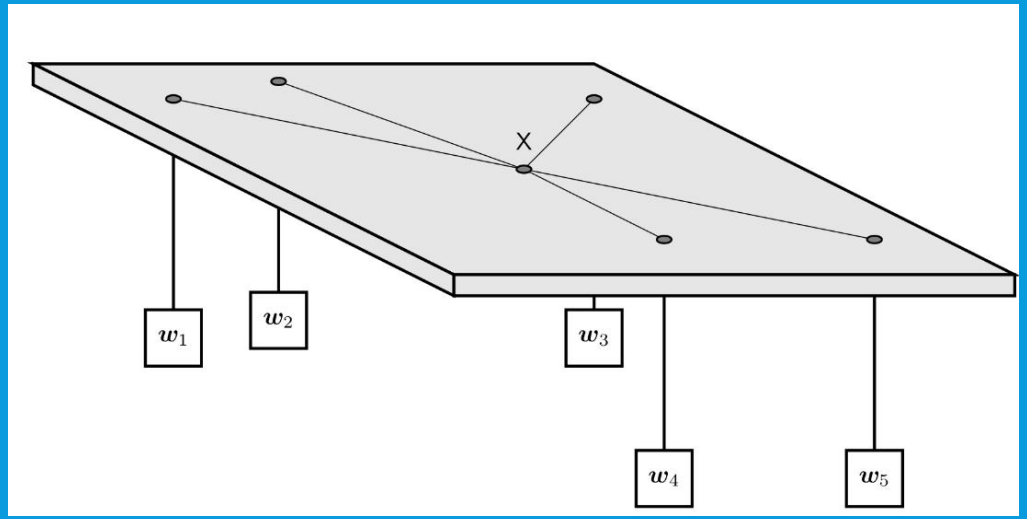
例题：暴打XXX

如图，有 n 个重物，每个重物系在一条足够长的绳子上。

每条绳子自上而下穿过桌面上的洞，然后系在一起。图中 x 处就是公共的绳结。假设绳子是完全弹性的（即不会造成能量损失），桌子足够高（重物不会垂到地上），且忽略所有的摩擦，求绳结 x 最终平衡于何处。

注意：桌面上的洞都比绳结 x 小得多，所以即使某个重物特别重，绳结 x 也不可能穿过桌面上的洞掉下来，最多是卡在某个洞口处。

$-10000 \le x_i, y_i \le 10000, 0 < w_i \le 1000$



模拟退火

```
double t=3000;//温度要足够高
  while (t>1e-15)//略大于0
  {
     double ex=ansx+(rand()*2-RAND_MAX)*t;//随机产生新的答案
     double ey=ansy+(rand()*2-RAND_MAX)*t;
     double ew=energy(ex,ey);
     double de=ew-answ;
     if (de<0)//如果此答案更优，就接受
     {ansx=ex;ansy=ey;answ=ew;}
     else if(exp(-de/t)*RAND_MAX>rand())//否则根据多项式概率接受
     {ansx=ex; ansy=ey;}
     t*=down;
  }
```

均分数据

已知 *n* 个正整数 *a*1,*a*2...*an* 。今要将它们分成 *m* 组，使得各组数据的数值和最平均，即各组数字之和的均方差最小。均方差公式如下：

$$\sigma = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(\overline{x} - x_i)^2}, \overline{x} = \frac{1}{m}\sum_{i=1}^{m} x_i$$

其中 σ 为均方差，$\overline{x}$ 为各组数据和的平均值，*xi* 为第 *i* 组数据的数值和。
求最小的均方差为多少?

样例输入：
6 3
1 2 3 4 5 6

m≤*n*≤20，2≤*m*≤6

谢谢大家！