Problem 1 心有灵犀 (cooperate)

题目大意：给你一个不超过 10^9 的数字 n 和一个交换次数上限 k，每次操作对这个数字 n 的其中两位进行交换，比如 201 可以换成 102，让你进行 k 次操作，求出交换后最大的数字和最小的数字的差的绝对值。

要点：1. 某一位的数字可以和它本身进行交换

2. 交换的数字不可以有前导零（即第一位不可以是 0）

解法：如果这个数字是 n 位数，那么其交换不超过 n-1 次就可以变成最大值和最小值，可以根据这个点进行剪枝。题目给的数字不超过 10^9，那么进行全排列直接暴力就好了，时间复杂度最多 9!

然而，这道题的关键是大家很容易误以为是贪心，而一般贪心　错的：举个栗子，k=2 时的 970979，贪心求出最大值是 999077，但实际上可以达到的最大值是 999770。所以这题不是个简单的贪心。估计这能坑倒一堆人。

Problem 2 不服来战 (challenge)

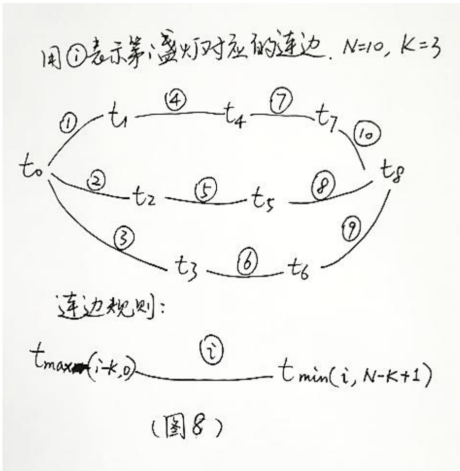
题目大意：你有一列 N 盏灯，初始时有些是开的，有些是关的。每盏灯有各自的权值。每次操作你可以改变任意连续 K 盏灯的开关状态。你可以操作任意多次，求最终最大的亮着的灯的权值和。

本题有 60 分的部分分可以用搜索和状态压缩动态规划获得。对于满分解法，下面给出殊途同归的两种思路。

解法一：不妨把改变 i~$(i+K–1)$这 K 盏灯的状态的操作叫做操作 i。可以发现，我们总共有操作 $1,2,\cdots,(N-K+1)$这些操作，而且每种操作最多只进行 1 次。（显然，多次操作没有意义）于是我们可以把是否进行操作 i 记为 $s[i]$，操作了则为 1，否则为 0。那么，对于第 i 盏灯，能影响到它最终是否亮起来的操作应该是操作$(i-K+1)$~操作 i（注意处于开头和结尾的灯稍有不同，准确地说是操作 $\max(i-K+1, 1)$到操作 $\min(i, N-K+1)$。这里为了简便先讨论一般的情况）。如果 $s[(i-K+1)\sim i]$这一段有奇数个 1，或者说，$s[i–K+1]$ xor $s[i–K+2]$ xor $\cdots$ xor $s[i] = 1$，则 i 号灯最终的状态与初始相反的。

这时我们会发现，与 i 号灯状态相关联的操作太多了，考虑到部分和，我们可以这样做。定义一个数组 $t[i] = s[1]$ xor $s[2]$ xor $\cdots$ xor $s[i]$，约定 $t[0] = 0$。这是个部分和数组，把“一段异或起来”这个操作简化了，变成两个数异或起来。也就是说，如果 $t[i]$ xor $t[i-K] = 1$，那么 i 号灯最终的状态与初始相反。

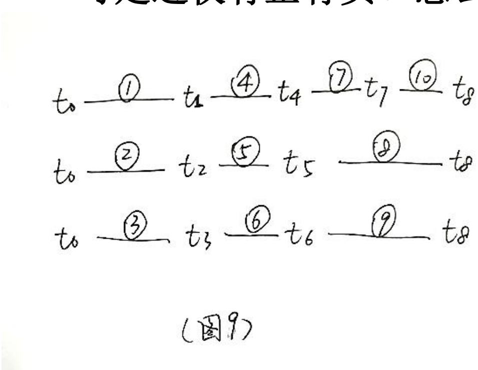
注意到 s 和 t 是**一一对应**的，给我一个操作方案 s，我能求出部分和 t；给我一组 t，我可以还原出你的操作方法，也就是，还原出哪些 $s[i]$是 1。这样一来，我们可以完全抛弃 s，而只考虑 t。问题变成：现在，你要随便给 $t[1], \ldots, t[N–K+1]$赋值 0 和 1，然后，我判断每一盏灯的状态是否改变，把亮的灯加起来。目标是最后总的亮度最大。



可能有些同学已经注意到，**把问题转化成图会看得更清楚**。把 $t[i]$看成结点，把灯看成边，连接着两个关系到这盏灯最终状态的点。举一个例子，N=10，K=3，那么这时连边的情况如图 8。需要注意的是排在前面和后面的几盏灯（如前文所述），比如第 2 盏灯，只有操作 1,2 能影响它，于是与它连着的结点是 t[0]和 t[2]。

不妨一开始把开着的灯的权值全部加起来，看看最优答案应该如何进行调整。现在，我们要给图中每个点标上 0 或 1。每盏灯，即每条边，如果连着的两个点，被标上的数字不同，就会使这盏灯状态翻转，对答案产生贡献（这一贡献正负都有可能，取决于这盏灯初始是否是开的），不妨把这个贡献作为每条边的边权。我们的目标是，把图中的所有点分成两类，然后使得“跨越”这两类点的边的边权和最大。似乎已经看到希望了！想到了什么？割集？

可是边权有正有负！怎么办？



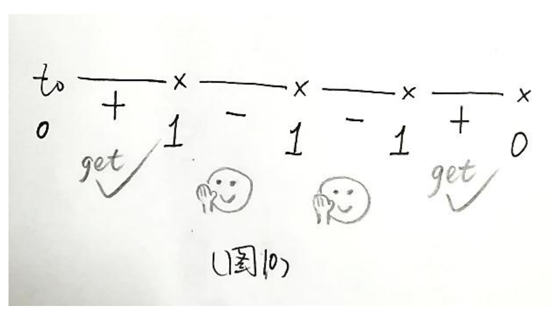
别慌！仔细观察这个图，可以发现，**每一行的决策可以独立进行**，也就是说，这个图可以分成几条链（图 9）。

链与链之间，**除了开头和结尾，其他部分并不互相影响**。假定 $t[0] = 0$（事实上 $t[0]$是什么并不重要，只要给每条链规定好相同的就行），不难想到对每条链进行动态规划。单独考虑一条链，决策每个点究竟应该取 0 还是 1（也就是，每个点应该拆成两个状态）。如果当前点的 01 状态从前一个点不同的 01 状态转移过来，这一步就会产生贡献。为每条链算好最后一个点取 01 状态时这条链的最优解。最后，看看最后一个点究竟取 0 还是 1，对应每条链的最优答案加起来最大。

这样就把这题解决了！

别急，还可以再想一想，能不能改进？不用 *DP* 行不行？

假如没有“每条链的结尾必须相同”这一限制，好像可以贪心地只把所有为正数的边取到！（如图 10）



这就是说，这条链上所有的灯都被点亮了！而且，正数的个数等于这条链上“还没有点亮的灯的数目”，像这样贪心地取，如果这个数目为偶数，那么最后一个点会被标成 0，否则被标成 1。（因为每遇到一个整数，01 就要转换一次）

枚举最后一个点取 0 或者取 1，如果最后一个点的取值恰好与我枚举的相同，那最好；如

果不同，这条链必须作出妥协！为了减少损失，我们只多翻转一次，即加上一个负数（灭掉一盏原来是开的灯），或者减去一个刚才 get 到的正数（也就是少开一盏灯），对应于链上每个点的 01 变化就是：把链中后面的一段的点的 01 全部翻转，以满足最后一个点的要求，但产生了一定的损失。为了让损失最小，我们损失那盏亮度最小的灯（本来我们是可以把所有的灯都打开的，现在不行了）。每条链都如此处理，便可得出最后一个点取 0 和 1 时的答案，比较后取最优。

怎么样？最后做法是不是很简单？

解法二：其实，我们发现，最终的做法似乎和我们的建图没什么关系。思维较好的同学，不从图的角度思考也可以得出正解。考虑操作 i 和操作 i+1 同时按下，这时产生的效果时把第 i 盏灯和第 i+K 盏灯翻转。也就是说，我们得到了一种新的操作，这种操作是，把任意相距为 K 的两盏灯同时翻转。不难发现，以下两种操作可以等价于原题中的操作“翻转任意连续 K 盏灯的状态”：

（1）同时翻转 1~K 盏灯

（2）同时翻转任意距离为 K 的两盏灯

等价性是因为，操作（2）可以由原题中的操作得来；而原题中“翻转任意连续 K 盏灯的状态”可以通过一次（1）和若干次（2）组合而来。

下面我们**只考虑新的操作（1）（2）**。首先，操作（1）是否进行可以枚举。这时，我们**只需考虑操作（2）**。而操作（2）似乎把一列灯也分成了许多组。以 N=10, K=3 为例，

灯 1,4,7,10 是一组；

灯 2,5,8 是一组；

灯 3,6,9 是一组。

每次进行操作（2），等价于翻转一组内相邻的两盏灯！

组与组之间没有影响！

再稍作分析，不难发现，如果一组内“初始时是关的”的灯为偶数个，那么我们可以做到只把它们全部打开，也就是说打开了这一组所有的灯；如果一组内“初始时是关的”的灯为奇数个，这个时候我们不能把所有的灯都打开，只好作出让步，选择放弃亮度最小的那盏灯。每一组都如此处理，最后加起来。

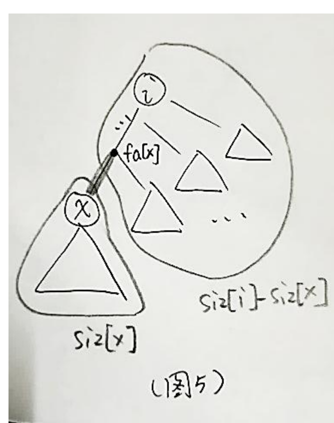
解法一、二的时间复杂度都是 $O(N)$的。

Problem 3 铁路网络 (network.cpp)

题目大意：给你一棵有根树，每条边有边权。实现两种操作：1. 给某一条路径上所有边的权值加上一个数；2. 询问某棵子树内所有点对的距离和。

本题设置了几个部分分梯度，得部分分的做法有不少，复杂度也各有千秋。下面给出比较容易理解的一种满分解法。

从查询的角度思考，如何计算以 i 为根的子树内所有点对的距离和？



不妨设点 x 与它的父亲 $fa[x]$ 相连的边的权值为 $p[x]$，考虑 $p[x]$ 会对那些点对产生贡献？显然是经过 x——$fa[x]$ 这条边的那些点对。记 $siz[x]$ 为以 x 为根的子树的大小。则经过 x——$fa[x]$ 的点对有 $siz[x]\times(siz[i]-siz[x])$ 对（如图 5），于是，子树 i 内的点 x 的贡献就是 $p[x]\times siz[x]\times siz[i]-p[x]\times(siz[x])^2$。**黄色的部分与 i 无关。**如果要计算，我们只需把子树 i 内所有点（不包括 i）的自己的黄色部分加起来，然后就可以求出答案。这是因为

$$\sum(p[x]\times siz[x]\times siz[i]-p[x]\times(siz[x])^2)$$
$$=siz[i]\times\sum(p[x]\times siz[x])-\sum(p[x]\times(siz[x])^2)$$

而修改操作，就是给一条路径上所有点（准确地说，不含 LCA）的 p 加上一个数。我们要**高效地维护上面黄色部分标出的两种“和”**。这里有两种解法：

解法一：树链剖分 + DFS 序。

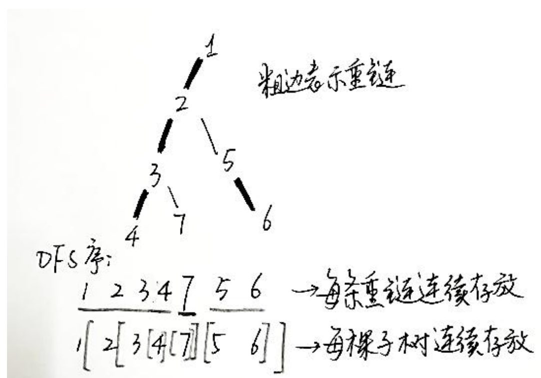
修改操作在树的链上进行，可以考虑使用树链剖分；而查询操作是查询子树的和，可以考虑用 DFS 序。

树链剖分，把重链的结点在线段树中连续存放；

而 DFS 序，把子树结点连续存放。

两者可以兼得吗？

事实上，进行树链剖分的时候，我们记录了每个结点的“重儿子”，求树的 DFS 序时，我们对每个结点都**先搜索它的重儿子再搜索其他儿子**。这样，就可以保证，DFS 序中，每条重链是连续存放的，每棵子树也是连续存放的，可以使用同一棵线段树存放。（不妨参考如图 6 的例子）在线段树中，很容易处理让一段的值都加上一个数和查询一段的和这样的操作。

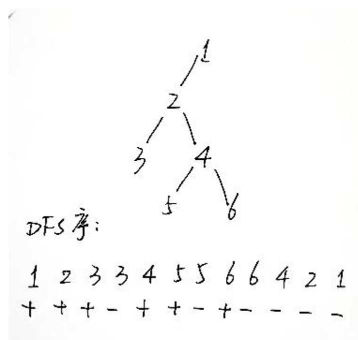


(图6)

这样做是 $O(n(\log_2 n)^2)$ 的。

解法二：直接使用 DFS 序。

DFS 序有时有些巧妙的性质。我们把每个点拆成两个点，分别放在它的开始搜索和结束搜索的时刻，不妨称他们为“正结点”和“负结点”（可以参考图 7 的例子）。



(图7)

这样，任何一条从父亲到子孙的路径都可对应于连续的一段，比如 2---4---6 这条路径，在 DFS 中对应了 2 3 3 4 5 5 6，注意到其中 3 3, 5 5 都是成对出现。由于这里求的是每个点 x 的 $p[x]\times siz[x]$ 和 $p[x]\times(siz[x])^2$ 的值，我们在负结点处让该点的 p 乘上 $-siz[x]$ 和 $-(siz[x])^2$。这样，修改 2---4---6 这条链时，直接改一整段的 p。至于中间成对出现的正负结点，本来它们的 p 不应该修改，但**因为他们在求子树和的时候一定也会成对出现，因此这部分“不应该有的修改”会正负抵消掉**；但是，对 2,4,6 这几个点我们只修改了它们的正结点，这部分修改在后续的查询中会有所贡献。

这种做法是 $O(n\log_2 n)$ 的，代码也比较短。

点评：需要维护查询和修改操作常常可以这样思考：我要记录哪些信息来支持我的查询？我要如何维护我的信息？比如这题，我是直接把子树点对距离和改在子树的根，还是先把子树内每个点的贡献“自己存着”，需要时再查？修改和查询操作的高效率如何两全？这些都应该是我们要考虑的问题。