刷题笔记

1. 树状数组维护区间的和、最值：
    a) 维护区间的和还蛮简单的，就是记住 trees[x]保存的是 trees[x-lowbit(x)+1, x]的和，然后更新的时候 while(x<=n)不断地给 trees[x]加上 k，然后用 x+=lowbit(x)来找父节点即可。而查询区间前缀和是 while(x)，不断地给 ans 加上 trees[x]，然后 x = x-lowbit(x)不断地找前继节点即可。可以康康模板代码里写的。查询区间[l, r]内的和的时候用前缀和(r) – 前缀和(l)即可。
    b) 维护区间的最值：

        这个就比较复杂了，因为最值没有加和性。

        对于树状数组不变（即没有单点修改，只有建立的过程），其 tadd 函数可以沿用维护区间和的代码，只是每一次需要变为 trees[x] = max(trees[x], k)。而如果是有单点修改的话，这样就是错的。试设想，如果你把一个本应该是某区间内最大值的点修改成很小的 k，那么你用 trees[x] = max(trees[x], k)这样的赋值，最大值根本就不会被修改，仍是 trees[x]（因为它更大）。当然，如果你知道每次修改都只会把 a[x]的值改大，你仍可以只用普通的 tadd()函数来更新（如最长下降子序列这样的问题）。否则，单点修改变得很复杂：

        我们只能记结论了，对于 x 这个点被修改，即我们首先手动修改了 a[x] = k，然后调用 update(x)去更新树状数组的值。而 x 这点在没有 x += lowbit(x)之前，只会影响到 x-1、x-2…. x-2^k 这些，其中 2^k ＜lowbit(x)且2^{k+1}≥lowbit(x)。于是我们可以写出下列更新函数：

```
void update(int x)
{
    int lx, i;
    while(x<=n)
    {
        trees[x] = a[x]; //之前已经手动更新好 a[x]了
        lx = lowbit(x);
        for(int i=1; i<lx; i<<=2)
            trees[x] = max(trees[x], trees[x-i]);
        x += lowbit(x);
    }
}
```

        这个算法复杂度是 O((nlogn)^2)其实也很少使用，树状数组更常用的是静态区间的最值维护。

        其中建立树状数组的方法和维护区间和一样，之后不再更新。而查询的时候不能用前缀和相减，因为最大值没有区间可减性。仔细思考会发现，trees[x]储存的是[x-lowbit(x)+1, x]中的最大值，因此我们在查询某一区间的最值时，只需要能找到一种方法遍历那个区间就好了。设我们查询的是[x, y]区间，在查询时当 y-lowbit(y)≥x 的时候，trees[y]就是这一段区间的最值，即 ans = max(trees[y], ans)而当这个条件不满足但 y 仍然大于等于 x 时，我们需要把 ans 设为 max(ans, a[y])然后继续用先前的条件更新 ans。于是可以写出以下代码：

```
int tsearch(int x, int y)
```

```
{
    int ans = -1;
    while(y >= x)
    {
        ans = max(ans, a[y--]);
        for(; y-lowbit(y)>=x; y-=lowbit(y))
            ans = max(ans, trees[y]);
    }
    return ans;
}
```

2. 【递归】NOI1777:文件结构“图”：http://noi.openjudge.cn/ch0202/1777/
这道题是给定一个文件结构的字符串，要你按照给定的格式和规则输出文件夹的结构。其实这样的类似树形的逻辑的题目都可以用递归来解决。规则是有目录就优先输出目录，然后再输出同级文件。其实把递归的状态参数定义好就很简单了：start、end、prefix，表示从输入的 start 位开始到 end 位，前缀是 prefix 进行这一部分的输出，每一部分都是先找出匹配的子文件夹的 start 和 end（括号匹配算法），然后按照出现的顺序 sort 一下（规则限制），再递归调用自身（prefix 要加一下），然后再输出同级的文件（规则限制）。
这边要注意的有两点：
    a) 递归调用自身的时候，不能把匹配到的文件夹全都去调用，因为会出现文件夹里套文件夹，这样会被调用两次，所以在递归调用之前，要先把匹配到的文件夹头尾被另外一个文件夹包住的那些文件夹都删掉，留下的就是本级最外层的文件夹了。
    b) 不能把从 start 到 end 的所有文件都输出，而是要输出那些和本级文件夹同级的文件，具体就是文件的位置不能是被包含在 a)算出的那些文件夹的里面的。
3. 【分治】SJTUOJ1219：重要逆序：https://acm.sjtu.edu.cn/OnlineJudge/problem/1219
之前有做过逆序的题目，大概思路就是在归并的时候看到左边的第 i 个数比右边的第 j 个数大，就把逆序数 cnt 加上 mid-i+1，其中 mid 是左边数组的最后一个的下标。这是因为我们按照升序排序了(因为归并排序就是不断地交换逆序达到正序(升序）的)，所以左边数组的第 i 个比右边数组第 j 个大，那左边数组 i 之后的也肯定比右边数组的第 j 个数大了，因此加上 mid-i+1 然后把 j++这样。
但是重要逆序是 i<j 且 a[i]>2*a[j]，这样就不能直接在归并的过程中计算逆序了，因为左边的第 i 个比右边的第 j 个大时，并不一定说左边的第 i 个比右边的第 j 个的两倍大，但是我们又把右边的 j++了，导致左边后面大的没办法和右边的第 j 个比了，因此出错。具体的例子就是：左 359，右 13，本应把 9 和 3 比，但若直接在归并的时候去做，在 5 的时候就把 j++了，9 无法和 3 比，就少了一个重要逆序了。
解决方法是在归并之前先用一个 while 循环计算重要逆序的个数。相当于一次 merge 的过程用了两次 while 循环，复杂度变为 2 倍，但还是 nlogn 级别的。
4. 【分治】NOI 1708:麦森数: http://noi.openjudge.cn/ch0404/1708/
这题其实就是个快速幂，不过要求取结果的最后 500 位，也就相当于模10^{500}，所以要用到高精度。快速幂里面把 a 和 ans 设成高精度，然后只要重载乘法就行了。注意把高精度乘法里面 ans 的 length 强行设成 500，这样就可以间接做到模10^{500}。注意不能强行不取模然后算最终的答案!!!因为位数增长到一定长度的时候乘法运算的复杂度会非常高……所以还是要取模的（虽然模的是10^{500}这么大的数）。比较

有技术含量的（其实也没啥）就是如何判断2^p有几位。根据下面的推导可以直接算出，2^p有 int(p*(log(2)/log(10))+1 位。

还有就是不知道为什么把 BigInteger 作为 fastExp 的参数会不能跑……然后我直接把快速幂的逻辑写到 main 里去了，现在还没解决这个问题。

已知：$10^x < y < 10^{x+1}$

那么 y 就是 x+1 位而！

$\Rightarrow 10^x < 2^p < 10^{x+1}$

$\Rightarrow x\ln 10 < p\ln 2 < (x+1)\ln 10$

$\Rightarrow p\frac{\ln 2}{\ln 10} - 1 < x < p\frac{\ln 2}{\ln 10}$

$\therefore x+1 = \text{int}(p*(\ln 2/\ln 10))+1$

即就是 int(p*(ln2/ln10))+1 位而！

5. 【KMP】洛谷：https://www.luogu.com.cn/problem/CF1029A
这道题比较新颖，说是给定一个字符串 t，要求你构建一个字符串 s，s 中有 k 个 t 子串，要求输出最短的那个 s。其实思路就是把，（开头为下标 0 的前缀子串 = 结尾为最后一个下标的后缀子串）的最长的前后缀子串求出，然后重复（从前缀结尾+1 的位置到最后）的子串 k-1 次就行了。那么如何求得这个前缀子串呢？就用 KMP 算法的失效函数求法即可。
6. 【KMP】洛谷 P3435 [POI2006]OKR-Periods of Words
https://www.luogu.com.cn/problem/P3435
这道题也比较新，而且~~需要阅读理解~~。其大意是，一个字符串 s，它的“真前缀”定义为非空且非 s 的前缀。s 的一个 period prefix（周期前缀）Q 定义为，Q 是 s 的真前缀，s 又是 concatenate(Q, Q)的前缀。然后现在给定一个小于 100w 长度的字符串 s，要你计算 s 的所有前缀 ps 的最大周期前缀Q_{ps}的和，即：sum(Q_{ps}) for ps 是 s 的前缀。经过一番分析可以知道，假设 ps 的最后一个字符的下标是 m，这个最大周期前缀其实就是 ps 的长度，减去(KMP 算出的那个数组的 nxt[m]一直递归到最头上的非-1 值 v + 1)，当然如果一开始就是-1，那就不能算，因为有规定Q_{ps}是 ps 的真前缀。那么我们先对 s 做预处理算出那个 nxt 数组，然后再对从 1 到 k（s 的长度）的每一个 m 去算 get_v(m-1)即可，这个 get_v(m-1)有个小技巧，那就是使用类似并查集 find 操作的路径压缩。因为我们只关心递归到最头上的非-1 值 v，所以可以在递归回溯的时候把路径上的点的值都设为那个 v。这样效率快了 50 倍…就没有 TLE 了！
7. 【二分图最大(小)权匹配】POJ2195: Going Home
http://poj.org/problem?id=2195

这道题主要是你要能看出它是考二分图的最小权匹配，因为假设有 n 个房子和 m 个人，这道题要计算 m 个人进 n 个房子的距离之和的最小值，其实有点像寒假做的那个滴滴打车的题，算最短等待时间（怪不得 zkp 会说用最小费用最大流做…）。难点其实是建二分图啦，我的思路是用 BFS 去找第 i 间房子到第 person_num 个人的最小距离，然后存放在 weights[i][person_num]里面，之后再用这个数组去 add 边，建立链式前向星模式的图。比较特殊的是要建立源汇点 s 和 t，s 和 t 分别要和 X 点集(房子)和 Y 点集(人)连接，并设容量为 1，cost 为 0，并且也要注意同时添加反向边。建完图之后就跑最小费用最大流的模板就好咯。

8. 【图论】Window Pains：http://dsalgo.openjudge.cn/graph/3/
   ummm 我的想法变成模拟解法了，大概思路就是不断地循环看看能不能清除（还原）某一 2*2 区域的窗口，具体地就是能不能把 4 个像素都变成左上角的那个数字。然后不断循环直至没有再能清理的，然后跳出循环判断是不是窗口的每一个像素点都被清除过，如果是的话，就是合法的 presentation，否则不是。
   看了题解，发现这其实不是模拟，而是图论……在窗口交叉的地方，比如 0,1 这个位置，可以填 1 也可以是 2。所以如果是 1，说明必然是先点了窗口 2 再点了窗口 1，这样可以建立一条有向边。给所有点都建立了有向边之后，跑拓扑排序判环即可。若有环则是不合法的窗口。建图就是对于每一个像素点，用一个 vector 保存其可以填写的数字，然后这个地方是哪个数字，就把剩下的数字指一条边到这个数字上，就建完图了==
9. 【最小生成树】The Unique MST：http://dsalgo.openjudge.cn/graph/11/
   太草了，debug 了快 40 分钟……这道题题意是让你求出某一张图它的最小生成树是不是唯一（unique）的，思路很简单，稍微想想就出来了，其实就是先 Kruscal 算出最小生成树，然后在 n-1 条边里依次删边，剩下的再去跑 Kruscal，如果算出的最小生成树值和第一次的一样，那就说明最小生成树不唯一。实现的时候很坑啊，我一开始是用下标记录第一次生成的最小生成树的边是哪些，然后再去删。但是！！！！Kruscal 要把边集排序啊！！！！！排完序之后下标就变了…其实我真傻，真的，删边的手段搞成把 length 设为 0x7fffffff……其实没必要，只要用一个 bool valid 数组记录一下删了哪条边就好了…这样就不用每一次 Kruscal 都要重新排序了，也可以采用记录下标的方法来记录最小生成树的边是哪些了……
   哎，出思路之后写代码不要太急，三思而后行吧。
10. 【欧拉回路】http://dsalgo.openjudge.cn/graph/7/
    判断无向图的欧拉回路不仅仅是要判断所有节点的度都为偶数，还要判断图是否连通……图是否连通不能仅仅通过判断度是否为 0！！！因为这只能找出孤立节点！！！而是应该用并查集或者 BFS 去判通……
11. 【DP】
    http://noi.openjudge.cn/ch0405/191/
    这道题……一开始做的时候犯了一个错误，就是……不能直接像杨辉三角那样直接算路径数！因为如果是遇到没有钉子的地方直接下落，概率不是平分的！也就是说不能直接把其当成一条路径！！！只有全部都等概率往左往右落下的时候才是杨辉三角 QAQ。正确的做法是，dp[i][j]表示第 i 层的第 j 个位置会落下的小球数。到了 n 层会有 2^(n-1)个小球，我们要求的是第 n+1 层的第 m+1 列最后会有多少球。一开始初始化小球个数为 1<<n，然后第 i 行遇到钉子就左右平分一半小球，分到第 i+1 行，遇到没有钉子的地方直接全部小球掉到 i+2 行。注意初始化小球个数的时候要：ull dp[1][1] = (ull)((ull)1<<(ull)n)，不然会出问题。哎…概率没学好啊！

12. 【DP】最长公共上升子序列的 O(n*m)算法（包含 O(n*m²)及如何优化）
这也算是一道很经典的题目了……一开始只能想到$O(n^2 * m^2)$的超级暴力算法，就是 dp[i][j]表示以 a[i]、b[j]为结尾的最长上升公共子序列长度，然后转移就是，如果 a[i]≠b[j]，dp[i][j]就是 0，如果 a[i]==b[j]，就对于 x<i，y<j，去找满足 a[x]<a[i]、b[y]<b[j]、a[x]= =a[y]的，最大的 dp[x][y]，然后 dp[i][j]就是 dp[x][y]+1。
后来发现没必要这样，因为包含了 a[i]==b[j]的信息，所以可以只设 dp[i][j]是以 a 的前 i 个数，b[j]为结尾的最大上升公共子序列长度。这样，当 a[i]≠b[j]的时候，就有 dp[i][j]=dp[i-1][j]，当 a[i]==b[j]的时候，就有 dp[i][j]等于，从 k from 1 to j-1 里，选出满足 b[k]<b[j]，并且 dp[i-1][k]+1 最大的那个。这个算法是 O(n*m²)的。
再仔细思考就会发现，你在选：

max{dp[i-1][k] | k from 1 to j-1 && b[k]<b[j]}

的时候，其实这时有 b[j] == a[i]，而 k<j，因此在每一轮 i 下，可以维护一个 maxdp 的值，表示在这一轮 i 下当前最大的 dp[i-1][k]的值。因为 k<j，所以每次当 a[i]==b[j]的时候，都可以使用到上一轮 j 维护过的 maxdp，因此当 a[i]==b[j]的时候，只要把 dp[i][j]设为 maxdp+1 即可，而当 a[i]≠b[j]的时候，除了更新 dp[i][j]=dp[i-1][j]之外，还要判断是否有 b[j]<a[i]，如果有的话，就用 dp[i][j]更新 maxdp：

maxdp = max{maxdp, dp[i][j]}

以便为下一轮更新 dp[i][j]做准备。这样的算法就是 O(nm)的！

13. 【不知道该是 DP 还是贪心还是啥？】HDU1422：重温世界杯
http://acm.hdu.edu.cn/showproblem.php?pid=1422
其实就是找一个循环数组的最长连续子序列，这个子序列满足从子序列的开头到结尾的每一个地方的前缀和都≥0。暴力当然 O(n²)没问题，O(n)的算法需要一点技巧。假如不考虑循环数组，只找一个数组中前缀和都≥0 的最长连续子序列，只要用一个数组记录 f 以每一个地方为结尾的最长这个子序列，然后再用一个变量记录当前子序列的前缀和，从头到尾遍历一遍数组，当加上某位置，当前前缀和仍≥0 时，就 f[i] = f[i-1]+1，然后更新前缀和。若加上这个位置，前缀和小于 0 了，说明要新开一个连续子串，而这个位置的值必然是小于 0 的，所以设 f[i]=0 然后当前前缀和重设为 0。这样遍历一遍数组就 ok 了。那如果是循环数组呢？循环数组其实可以当做把原数组复制然后拼接在尾巴上，然后当做是普通数组。当然这样可能会导致算出来的最大连续子序列大于 n（设想假如全部值都是正数），这个问题很好解决，当某个 f[i]=n 时直接跳出循环就可以了。因为 n 已经是最大的答案了，如果找到了一个 n，说明原循环数组一定可以得出 n 的答案，即这个答案是合法的。所以直接跳出循环是 ok 的。