Hello World ! --codeforces

背包dp 融合信息、改变状态

题意(CF#19#B)：一个小偷去商店买N件商品，只有一个售货员，售货员处理商品i用时ti(ti>=0)，售价ci。这个小偷可以趁售货员处理商品的时候偷走剩下还没处理的任意商品，一秒钟可以偷走一件，而处理商品的顺序是小偷决定的。你的任务就是帮助小偷决定给售货员处理商品的顺序，使得小偷要交的费用最少，输出最少费用

思路：直接记录题解的一些点每个ti加一，问题就变为“n件物品，重量ti>0, 价钱ci>0。取若干件物品，要求满足总重量不小于n时，总价钱最少”

一直想不明白ti+1为什么正确以及怎么想出来的。neko刚才跟我说他刚才是怎么AC的的过程，受益良多

首先说正确说法：我们记{X}为付钱的物品，{Y} = N - X为偷的物品，

则需要满足 Σtx >= Y，x∈{X}

应该要想到 |X| + Σtx >= N

想DP的时候发现对于dp[i][j]表前i个物品耗费时间为j的最小花费，既要记录最小耗费Σcx，又要记录{X}的个数（因为要Σtx >= N-|X|），就不知道怎么办。因为我们一般的DP不会需要记录前i件物品取了多少件。

继而就发现，我们可以把式子|X| + Σtx >= N变为 Σ(tx+1) >= N，而且还跟{Y}无关！所以t+1这个想法就出来了。

有了t+1，问题就变成0/1背包了。

拓展：其实这就是一种“融合信息“的方法，把”耗费时间”和”选取的物品个数”这两个信息融合到一个量里面.

memset(f, 127, sizeof(f));

f[0] = 0;

for(int i = 1; i <= n; i++) {

t[i]++;

for(int j = n+2000; j >= 0; j--) {

if(j >= t[i]) f[j] = min(f[j], f[j-t[i]]+c[i]);

// f[j] = min(f[j], f[max(0, j-t[i])]+c[i]);

}

}

for(int i = n; i <= n+2000; i++) ans = min(ans,f[i]);

数位DP 实现

题意(CF#258B)：有包含4或7的数称为Lucky number。现在Little Elephant所在的党派要与其它6个党派竞选，我们分别用[1..m]内的一个数字来代表每个党派，当然不同党派的代表数字不同。现要求Little Elephant的党派的代表数字包含的4/7的数目要比其余6个党派的代表数字包含的4/7数目的总和大，问[1..m]有多少中符合要求的方案。

思路：数位DP预处理出cnt[x]表“[1..m]内有包含x个4/7的数字有多少个”。 但我就是这里不会...=\_= 在cf上找代码看，刚好有个好人不仅风格好，还有中文注释~~Thanks~~

后面的就直接递推搞之。

直接卡看代码。

//DP state [position][less][count].

long long solve(int rest, int luckyNums)

//计算其他6个人幸运数和小于(luckyNums)的情况数

{

if(rest == 0) return 1;

long long res = 0;

for(int i = 0; i <= luckyNums; i++) if(cnt[i] > 0) {

cnt[i]--;

//若选择幸运数长度为i的数,下一个人最多能选择的长度为luckyNums - i

res = (res + ((cnt[i] + 1) \* solve(rest-1, luckyNums-i)) % MOD) % MOD;

cnt[i]++;

}

return res;

}

int dfs(int pos, int luckyNums, int pre, bool full)

// 返回值为 0 ~ m 中 lucky nuberm 个数是 luckyNums(第二个参数)的数有多少个

{

if(pos == -1) return pre == 0;

if(pre < 0) return 0;

if(!full && dp[pos][luckyNums][pre] != -1) return dp[pos][luckyNums][pre];

int res = 0, end = full ? digit[pos] : 9;

for(int i = 0; i <= end; i++)

res += dfs(pos-1, luckyNums, pre - (i==4 || i==7), full && i == end);

return full ? res : dp[pos][luckyNums][pre] = res;

}

int calc(int n, int luckyNums)

{

int idx = 0;

while(n) {

digit[idx++] = n % 10;

n /= 10;

}

return dfs(idx, luckyNums, luckyNums, true);

}

int main()

{

int n;

memset(dp, -1, sizeof(dp));

scanf("%d", &n);

//求出结果(枚举第一个人的幸运数长度)

for(int i = 0; i <= 9; i++) cnt[i] = calc(n, i);

if(cnt[0]) cnt[0]--;

long long ans = 0;

for(int i = 1; i <= 9; i++)

if(cnt[i] > 0)

ans = (ans % MOD + cnt[i] \* solve(6, i-1) % MOD) % MOD;

printf("%I64d\n", ans);

}

树转线性

题意(CF258E)：题目：给出一棵树，每次操作给出两个结点a,b，表示将以a,b为根的两棵子树的所有结点，都加入一个相同的数i，i表示操作序号。然后查询所有的点，和自己至少有1个相同数字的结点有多少个

思路：一开始在树上各种乱搞，结果还是失败了。其实一看到这种树上RMQ的问题应该第一反应就是树转线性、树链剖分什么的，我是傻逼吗>M<..

得到树的先序dfs序列后，对子树操作就是对一个区间操作。再看看题意，只要求“有一个相同”就可以，也就是说如果我们把一个操作看作对一>个区间加一个数值，那么询问就是问[1,n]内有多少个add[]不为0的数。还是按树的先序dfs更新维护线段树，每次查询有多少个点被覆盖就行了

个人觉得维护线段树那部分代码还是挺精彩挺妙的！我是自己写不出来，然后看题解上的，特别是push\_up()

void push\_up(int u)

{

if(f[u].lazy) {

f[u].sum = f[u].r - f[u].l + 1;

} else {

if(f[u].l == f[u].r) f[u].sum = 0;

else f[u].sum = f[L(u)].sum + f[R(u)].sum; //妙呀...

}

}

void Update(int u, int l, int r, int val)

{

if(l == f[u].l && f[u].r == r) { //妙 //这是树dfs后的序列喔~

f[u].lazy += val;

push\_up(u);

return ;

}

int mid = MID(f[u].l, f[u].r);

if(r <= mid) Update(L(u), l, r, val);

else if(mid < l) Update(R(u), l, r, val);

else Update(L(u), l, mid, val), Update(R(u), mid+1, r, val);

push\_up(u);

}

void calc(int u, int father)

{

for(int k = node[u]; k != -1; k = oper[k].next) {

int v = oper[k].v;

Update(1, in[v], out[v], 1);

}

cnt[u] = f[1].sum;

if(cnt[u] > 0) cnt[u]--;

for(int k = head[u]; k != -1; k = edge[k].next) {

int v = edge[k].v;

if(v != father) calc(v, u);

}

for(int k = node[u]; k != -1; k = oper[k].next) {

int v = oper[k].v;

Update(1, in[v], out[v], -1);

}

}

void predfs(int u, int father)

{

in[u] = ++Index;

for(int k = head[u]; k != -1; k = edge[k].next) {

int v = edge[k].v;

if(v != father) predfs(v, u);

}

out[u] = Index;

}

int main()

{

int query, u, v;

scanf("%d%d", &n, &query);

init();

for(int i = 1; i < n; i++) {

scanf("%d%d", &u, &v);

add\_edge(u, v), add\_edge(v, u);

}

predfs(1, 1);

build(1, 1, Index);

edge\_num = 0;

for(int i = 0; i < query; i++) {

scanf("%d%d", &u, &v);

add\_oper(u, v), add\_oper(v, u), add\_oper(u, u), add\_oper(v, v); //

}

calc(1, 1);

for(int i = 1; i <= n; i++) printf("%d%c", cnt[i], i == n ? '\n' : ' ');

}

DP 好题 实现

题意(CF#265E)：n个小球排成一列，有颜色ci，权值vi.要求选出若干的小球，选出来的小球相对顺序不变，这个序列的序列权值最大.序列{Xi}权值计算方法：

(1)小球xi在第一个或者与xi前面的小球同色，即i==1或者c[xi]==c[xi+1]，则序列选择和加v[xi]\*A

(2)否则序列权值加v[xi]\*B

其中A，B是给出的常量，|vi|<=10^5, |A|,|B|<=10^5

思路：之前个人赛的题呀，到现在还是不会做......看题解貌似很简单，但是感觉自己写的话就会栽在细节处理和逻辑转换上...>\_<

定义dp[c]为以颜色c结尾的最优值 //是的，我根本没有想过是这样定义状态的好不好，很妙好不好...

易知dp[c]至于之前颜色c和其它颜色的最大值有关，那定义fir,sec为之前出现的最大值和次大值的颜色，分类讨论(c=ci)：

(1)跟前面的颜色相同，即dp[c] = max(dp[c], dp[c] + vi\*A)

(2)跟前面的颜色不同，

c != fir，则用dp[fir]更新dp[c]肯定是最优的，dp[c] = max(dp[c], dp[fir] + vi\*B)

c == fir，因为我们在DP过程中一直维护fir!=sec（除了初始化），所以c == fir也就是c != sec，这时用dp[sec]更新dp[c]是最优的

dp[c] = max(dp[c], dp[sec] + vi\*B)

然后维护fir和sec其实很简单，我之前想复杂了>\_<

if(c != fir) {

if(dp[c] > dp[fir]) sec = fir, fir = c;

else if(dp[c] > dp[sec]) sec = c;

}

相邻0/1序列交换 思路题

昨天做CF D题，是neko搞出来的，赛后看了一下standing上别人的代码，记下对这道题的理解。

problem link: http://codeforces.com/contest/353/problem/D

首先来一个最短的代码：

for(int i = 0; i < n; i++) {

if(str[i] == 'M') x++;

else if(x) y = max(y+1, x);

}

printf("%d\n". y);

x是前面boys的累加，a表前面最近一个girl，b表当前girl，y0是a所需要的时间，y1是b所需要的时间。

<1>首先，y0+1和x是y1的两个下界。

证明：不论a、b相对位置如何，x是y1的一个下界都没有争议吧，因为b至少要跟x个人交换嘛； 若a、b是相邻的，则显而易见y1 = y0 + 1；若a、b不相邻，当然，下界还是(y0+1, x)~~对了，我这里说的“下界”只是为了说明y1肯定比x和y0+1大

<2>两个下界的最大值就是当前girl b所需要的时间

证明：如果b能畅通无阻地走到其目的地，则所需时间为x；

如若不能，说明在某个时刻a会挡住b，即a、b贴在一起。OK，既然知道她们俩在某个时刻会贴在一起，而移动又都是同步的，那y1 = y0 +１

　　这种思路的巧妙性在于，它不去讨论两个girls中间有多少个boys，而是直接就分成上述两种情况，掩盖了各种讨论的种种细节。

这种做法给我的启发，遇到这种需要分很多种情况的题，不妨换种思路，换种分情况的法子。像这道题，因为交换是同步的，然后每一次交换都把序列变得乱七八糟的没有规律。如果找到“不论怎样，每个girl的相对顺序不变”，“当前girl所需的时间只跟前面那个girl移动的时间和前面总boys数有关”这两个性质，大概就能分析出来了。

ps : if(x) .... 这是为了判断前导‘F’的情况。

再来一发：

for(int i = 0; i < n; i++) {

if(str[i] == 'F') {

time = i - female++;

if(time) time += continousFemale++;

} else {

continousFemale = max(0, continousFemale-1);

}

ans = max(ans, time);

}

这份代码我并没理解到它的精髓。

continousFemale其实就是当前女孩所需要等待的时间（所需时间 = 移动到目的地的步数 + 等待时间），有种同性相加，异性相消的感觉...

随便记点小理解吧~ str[i] == 'M'，说明后面的女生又可以少一个空闲时间了(假设continousFemale > 0)，因为多了一个str[i]要交换呀...

我根据这份代码猜测出一个结论：把girl看成1，boy看成-1，得到一个1/-1序列。若有0~i-1的前缀和sum都大于等于0，则对于当前位置i上的女生，她的等待时间为sum。

如果遇到前缀和小于0的时候，我们把前面砍掉，sum清空为0，重新从下一个位置开始计算一个新的前缀和sum。

只是猜测，不知对错，不知怎么证明，但我觉得要是搞懂这个点应该是很有价值的。有想出证明的大神路过求告知~~

第3分比较常规，跟第2分类似：

for(int i = 0; i < n; i++) {

if(str[i] == 'F') to[i] = dst++;

else to[i] = -1;

}

delay[0] = 0;

for(int i = 1; i < n; i++) {

if(to[i] == i) delay[i] = 0;

else if(str[i] == str[i-1] && str[i] == 'F') delay[i] = delay[i-1] + 1;

else if(str[i] == str[i-1] && s[i] == 'M') delay[i] = max(0, delay[i-1] - 1);

else delay[i] = delay[i-1];

}

for(int i = 0; i < n; i++)

if(str[i] == 'F' && to[i] != i)

ans = max(ans, i - to[i] + delay[i]);

贪心 优先队列 堆 括号序列

题意(CF#3D)：给一个有'(',')','?'的括号序列，要求把'?'替换成括号，使得该括号序列合法。并且每个'?'替换成左括号和右括号有不同花费(ai,bi)。问替

换之后的最小总花费

思路：

用计数器法来保证括号序列合法

摘：

I.如果是一个没有问题的括号串要怎么判断是否是个合法的匹配？一般的方法是用栈，如果中途出现了’)'但是栈中空，那么就失败了。或者所>有括号读完了，栈中还有元素，那么也是失败的。其实我们可以只用一个计数器来代替栈，效果完全一样。count计数器，遇到’(‘增1，遇到’)'减1，如果>串还没读完count出现了小于0的值，那么和栈的第一种情况是等价的，失败。如果读完了整个串，count不为0（那只会为正数），那么和栈的第二种情况是

等价的

II.贪心的前提是一定要保证这个问题有解。一开始将所有’?'换成’)'，然后先计算一个花费

1.替换完后从左到右读这个串，从上面说的方法去计数

2.如果读到’(‘，那么这个’(‘原来一定不是括号，并且count++

3.如果读到’)'，count–，但是要判断这个’)'是不是由’?'变来的，如果是的话，将这个元素放入一个堆中，堆中的每个元素要记录两个信息，>一个信息就是它来自于串中的那个位置，另一个信息等下再说

4.如果count<0，说明出现了无法匹配的情况，但这时候不是退出宣布失败，而是看看能不能补救，之所以会发生不匹配，可能是因为一开始的>时候将所有'?'变成了')'导致的，也就是说看看之前是否出现过'?'，如果没有出现过，那么说明这个不匹配是不关'?'的事的，那么只好退出宣告失败了。

但是如果之前出现了'?'，那么还可以补救，那么就将前面的'?'换成'(',就可以抵消掉当前这个')'.但是前面可能有多个'?'，要怎么选？贪心就是贪在这>里。

试想，一开始将所有'?'变为')'并计算了花费 +b，现在把一个')'变回'('，那么花费的变化就是 cost - b + a = cost + (a-b) , 也就是说要让cost越小，让(a-b)越小。所以之前堆记录的另外一个信息就是每个括号的a-b值，每次从推堆中取出一个'?'就是取出a-b值最小的那个元素。

取出一个'?'将其从')'变为'('的时候，会对count值产生影响，本来是')'变'('，count += 2

数位DP 压缩空间

题意：统计区间内的beautiful number，beautiful number定义：一个正整数能整除它的每个位上的数字(a positive integer number is beautiful if and only if it is divisible by each of its nonzero digits.)

思路:容易想到要整除每位上的数字，即整除每位上的数字的最小公倍数。

比较难想到的一点是压缩：dp[pos][sum][lcm]表前面位的值为sum，最小公倍数的lcm的合法数。但是sum会超int，开不下这么大的数组。看题>解知道需要压缩。下面讲压缩：

int MOD = LCM(1,2,..9) = 2520

按照定义，x为beautiful number：

x % LCM{digit[x]} = 0

即 x % MOD % LCM{digit[x]} = 0

而在逐位统计时，假设到了pre\*\*\*(pre指前面一段已知的数字，而\*是任意变)

(preSum \* 10 + i) % MOD % LCM(preLcm, i)

= (preSum \* 10 % MOD + i % MOD) % LCM(preLcm, i)

另外，还把[1..2520]中能整除2520的数hash成一个下标，因为只有x能整除2520，才有可能在DP做LCM时算出LCM = x。这样，又把dp数组第三维压到了[48]

void preprocess()

{

int lcm = 1;

for(int i = 1; i <= 9; i++) lcm = LCM(lcm, i);

MOD = lcm;

int j = 0;

for(int i = 1; i <= MOD; i++) if(MOD % i == 0) idx[i] = j++;

}

LL dfs(int pos, int preSum, int preLcm, bool full)

{

if(pos == -1) return preSum % preLcm == 0;

if(!full && dp[pos][preSum][ idx[preLcm] ] != -1) return dp[pos][preSum][ idx[preLcm] ];

LL res = 0;

int end = full ? digit[pos] : 9;

for(int i = 0; i <= end; i++) {

int newSum = (preSum \* 10 + i) % MOD; //TODO 妙呀

int newLcm = (i == 0 ? preLcm : LCM(preLcm, i));

res += dfs(pos-1, newSum, newLcm, full && i == end);

}

if(!full) dp[pos][preSum][ idx[preLcm] ] = res;

return res;

}

LL calc(LL n)

{

int Index = init(n);

return dfs(Index-1, 0, 1, true);

}

题很简单，但WA让我更理解块排的算法.........

思考一下为什么下面两种排序结果不一样。。。待续。。。下棋先。。。

if(abs(a.x) != abs(b.x)) return abs(a.x) < abs(b.x);

return ABS(a.y) < ABS(b.y);

//if(a.x != b.x) return abs(a.x) < abs(b.x);

//return ABS(a.y) < ABS(b.y);

树的构造题

题意(CF#190D)：给一棵树的没个几点赋于Rank'A'-'Z'，使得任意两个有相同Rank的节点x和y之间至少有一个节点z有Rank[z] > Rank[x] = Rank[y]

思路:<1>递归分治

<2>对于一棵树，找到其中心点root，给root赋予当前最高Rank

<3>把root删掉，则这棵树就被拆成若干棵子树

<4>递归到每颗子树，做同样的事情，即回到<1>

一开始我是想找一遍树的中心，然后用分层思想，一层一个Rank从高到低下来。易证这不是最优的，比方说遇到1-2-3-...-100这个链就还会挂。

后来看题解，想了很久都没想出来......

刚才在活动室的时候跟Tim讨论了一会儿，想了一种方法：还是找到树的中心root，递归到子节点sub，如果sub是有多棵子树的分支节点，则Rank[sub] = Rank[root] + 1，递归下去；如果sub是只有一棵子树的节点，也就是接下来是一条链，则'Z' .. Rank[root]+1依次赋予这条链上的节点，直到遇>到分叉口或者‘Z'用完.

这个方法被Tim的数据否决掉了：偏二叉树情况...

int find\_center(int u)

{

ans = -1;

dfs(u, 1, u);

ans = -1;

dfs(ansi, 1, ansi);

ans /= 2; //直径除2为半径

while(ans--) ansi = pre[ansi];

return ansi;

}

void dye(int u, char rank)

{

int root = find\_center(u); //<2>

color[root] = rank; //<3>

for(int Size = adj[root].size(), i = 0; i < Size; i++) {

int v = adj[root][i];

if(color[v]) continue;

dye(v, rank+1); //<4>

}

}

思路题

题意(CF#190C)：初始位置为(0,0)，目标位置(a,b)，移动序列为'UDLR'组成的字符串，移动时必须按字符串左到右的顺序操作，和重复这个操作序列多次。问有

没有可能到达(a,b)？

思路：这种题的思路很简单，但比赛时的思路有点点小麻烦，而我一遇上稍微麻烦的题就跪，这个是超级打缺陷！！！ 哎，还是太弱了。

我找我一份看起来最简洁的代码。它利用了题目的一个性质：如果在第一次执行到操作序列S的第i个操作S[i]时，位置为(x, y)，而执行完一次完整的

S后位置为(cx, cy)，则再第二次执行到S[i]时位置应该时(x+cx, y+cy), 第k次是(x+k\*cx, y+k\*cy)。 这一点我确实没明确地意识到。

int main()

{

int a,b;

string s;

cin >> a >> b >> s;

int x = 0, y = 0;

for(int len = s.length(), i = 0; i < len; i++) {

vx.push\_back(x);

vy.push\_back(y);

switch (s[i]) {

case 'U' : y++; break;

case 'D' : y--; break;

case 'L' : x--; break;

case 'R' : x++; break;

}

}

for(int Size = vx.size(), i = 0; i < Size; i++) {

int dx = vx[i], dy = vy[i];

int k = 0;

if(x != 0) k = (a-dx) / x;

else if(y != 0) k = (b-dy) / y;

if(k < 0 ) k = 0;

if(dx + k\*x == a && dy + k\*y == b) {

cout << "Yes" << endl;

return 0;

}

}

cout << "No" << endl;

return 0;

}

LCS 思路题 贪心 巧妙

题意(CF#187#D)：给出两个重复的字符串[A, b], [C, d]，A,C分别是两串的重复单元，b,d分别是重复次数，问第二个串在第一个串出现的次数（非严格连续子序列） note : [s,p]表示s重复p次

思路：这道题太妙了。首先，[C,d]在[A,b]出现的次数 = [C, ∞]串与[A,b]串的最长公共子序列 / [len(c)\*d]

知道了这个以后，接下来就是求[C, ∞]串与[A,b]串的最长公共子序列的字数了：

<1>先预处理出以C串的每个位置开头时，在A串中能匹配到的字数（因为有可能以C[i]开头，一个A串能循环匹配C串多次，所以这个字数很可能大于len(c)），我用get[i]表示“开始C[i]，匹配一个A串，能匹配多少个字母”。

<2>接下来算[C, ∞]串与[A,b]串的最长公共子序列的字数，最好对照代码看。words表当前位置能够匹配到的字母数，也表示下一轮跟A匹配时的起始位

置（想一想，为什么）

和gyy讨论了好久，未果。一开始是想找出循环节，然后前后部分搞一下，就可以了。找循环节的时候是想直接从[A,b]的开始找，第一次能完全

匹配[C,d]的地方就是循环节了，可是gyy给出了反例，显示要第二次完全匹配到[C,d]的部分才是最小循环节...于是就囧了。

连CF上的题解都说最好对照着代码看...

构造题

题意 (CF#183C): 给一个n, 求出3个0 ~ n-1的全排列{a}, {b}, {c}, 使得(ai + bi) % n == ci % n

第一次打cf. 这是赛后在"neko小队"里一个人的解法.

{a} = 0,1,2,3,4...,n-1

a移动一下位置就可以得出b了...下面的代码是移动一位, 其实移动多少位都可以.即ai == b(i+move).

数学题 构造题 \*\*未完全弄懂细节\*\*

题意 (CF#180E): 给出长为n的unique序列s. 要求把s切分成两个长度为n的序列a, b, 使得: (1 <= i <= n)

(1)ai, bi都大于0

(2)si = ai + bi

(3)a,和b都是`almost unique`序列. 所谓的almost unnique序列是指, 当且仅当移除不超过[n/3]个元素后, 剩下的序列是unique序列. ps: [x] 是向上取整

思路: 看题解知道的规律 :

'An equivalent definition for almost unique, is arrays with at least ⌊ 2n / 3⌋ different elements. The idea is to split s into three parts. In the first part, we give uniqueness to a. In the second part, we give uniqueness to b. In the third part, we give uniqueness to both.'

证明该结果正确:

以a为例,

第一阶段的a = 0, 1, ..., [n/3]-1

第二阶段的忽略, 有[2n/3] - [n/3] = [n/3]个, 即使这时候这[n/3]个数都相同, 且第一阶段有一个数也与之相同, 最多也只需要>删除[n/3]个数就能使a是unique

接下来只要证明第三阶段没有数与之前的数相等即可. 第三阶段的公式是ai = si - (n-i-1) , 令i=2n/3 + r, si=2n/3 + k, 则ai = 2n/3 + k + 2n/3 + r + 1 - n = n/3 + k + r + 1

因为第一阶段的最后一个是([n/3]-1), 所以第三阶段的ai与第一阶段最后一个数的差为:ai - ([n/3]-1) = k+r+2 >= 2, 即说明>第三阶段的a不会与之前阶段的结果有重合, 所以只需要[n/3]个即可满足a是almost unique.

b与a不同点是, 第二阶段的数不会和第一阶段的数重合, 但是有可能第三阶段的第一个数和第二阶段的第一个重合, 所以也是最多只需要删[n/3]>即可.

Floyd 的理解 图论 启发题

题意(CF#179D): 给出一个有N个点的带权完全图G, 再给出一个删除序列{d}, 要按照这个序列把图G中与点di相关的边删掉, 求每次删边之前的所有两点路径之和

思路：加深了我对floyd 算法的理解!

for(k = 0; k < n; k++)

for(i = 0; i < n; i++)

for(j = 0; j < n; j++)

f[i][j] = update(f[i][j], f[i][k] + f[k][j])

上面的代码中, 是每次都枚举k作为中间节点, 更新i-->j的值.即相当于k是新加进来的点.

floyd是不断加k点的过程， 那么题目其实就是floyd 思想的逆过程.

for(int k = add[n-1], l = n-1; l >= 0; --l, k = add[l]) {

for(int i = 0; i < n; i++) if(i != k)

for(int j = 0; j < n; j++) if(i != j && j != k)

if(dis[i][j] > dis[i][k] + dis[k][j])

dis[i][j] = dis[i][k] + dis[k][j];

long long sum = 0;

for(int i = n-1; i >= l; i--)

for(int j = n-1; j >= l; j--)

sum += dis[add[i]][add[j]];

ans.push\_back(sum);

}

组合数学 容斥定理

题意(CF#179B): 给出可能包含数字或'?'的长度为n的字符串a,b, 问: a,b字符串里面的'?'可以随意以数字(1-9)替换, 有多少种方法数使得a,b为incomparable. 所谓的incomparable是指满足条件: 存在i, j(1 <= i,j <= n), 使得ai > bi 且 aj < bj

思路: 第一次写容斥的题目.

sum\_ab不可比 = 总情况数 + ab相等情况数 - a大于等于b情况数 – b大于等于a情况数

xor性质

题意(CF#177E)：求一个排列(0^n)使得0^p0 + 1^p1 + 2^p2 + ... + n^pn 最大

思路：

(1)自己做的时候是打表找规律，不过没找到，唯一一个游泳的规律是，当n为偶数的时候，排列的第一个是0。其实也发现了序列的数貌似有“对偶

”性质，不过这对数学渣的我没意义>O<

(2)后来看题解，题解说贪心，利用互补性质。

Aha 了一下，因为是0~n的排列，若n为奇，则利用互补性质搞个两两配对就行了，若n为偶，则空出0位，利用到了<1>打表出来的规律。

一上来我就用刚才打表的程序跑了一下这个思路，就是输出一下 (bit[len]-1) ^ ans[i] 是不是等于i，但发现不等于，想了好没想出来。

又去看题解，才知道我把“互补”理解错了，互补的两个数的位数要相等，而我直接用 bit[len]-1来搞，那当然不能得到互补数啦，

要bti[length(i)]-1 == i ^ ans[i]。既然互补要两个数的位数相等，那4的互补是3，他们的位数不相等撒？

所以，这里就出现了我对“互补”的另一个误解，纠正：对于互补对pair<a,b>, patch(a) == patch(b) == Bit[len(max(a, b))]-1，而不>是patch(a) == Bit[len(a)]-1. 这个时候要取最大位的。所以程序里有一点要特别注意，循环数倒过来跑的，因为这样能保证对于一个互补pair<a,b>,a<b，第一次遍历到的一定是b，所以len(pair) == len(b) 而不是len(pair) == len(a)!!!

因为是两两配对，任意两对不互相影响，而因为每一对都互补，都使得xor操作取得了最大值(不难想)，所以sum最大。

TODO : 题解有直接就得出了sum = n\*(n+1)，不理解

for(int i = n; i >= 0; i--) {

if(ans[i] != 0) continue; //因为ans[a] != 0就能肯定和a同在一个pair里的b(a<b)已经被遍历过，那时候取道的len才是互补pair>的位数，这个时候取的len(a)可能小，不正确。

int j = bit[len(i)] - 1 - i;

ans[i] = j;

ans[j] = i;

sum += 2\*(bit[len(i)]-1);//2\*(i ^ ans[i]); //两种写法都可以

}

思路 操作维护

题意：对于一个序列{pi}，定义操作f(p,k)为将序列按k个分为一组，最后一组若不够分，则含有n%k个元素，然后对于每一个小组循环左移一位。k从2到N依次轮转。

思路：这种题要么就是能找到规律，直接出结果，要么就时考验对于每一轮操作f(p, k)该如何用时间复杂度较小的方法来维护。

这里是法2.

直接copy大牛的例子：

1 2 3 4 5 6 7 8 9 =>

2 3 1 5 6 3 8 9 7

将被移动的字符看作\*，于是：

\* 2 3 \* 5 6 \* 8 9

2 3 \* 5 6 \* 8 9 \*

这样就可以将左移变为O(1)了

反思：这种利用“相对位置”的“相对论”有启发。发散思维，不让排列被位置(下标)束缚住，利用了“相对位置”这一点将每次循环左移复杂度降到最低

for(int k = 2; k <= n; k++) {

int pos = n % k == 0 ? n-k+1 : k \* (n/k) + 1;

pos += start - 1;

p[start+n] = p[pos];

pos -= k;

while(pos >= start) {

p[pos+k] = p[pos];

pos -= k;

}

start++;

}

排列 构造

题意(CF#176C)：求出一个1-n的全排列，使得P(P(i)) = n+1-i.

思路：设P(x) = y ---> P(y) = P(P(x)) = n+1-x。连续应用此式有：

P(x) = y

P(y) = n+1-x

P(n+1-x) = n+1-y

P(n+1-y) = x

可见用该式4次后又回来了.4个一组，每组能相互跳，特殊地中间可能单出一个来, 不难知P[mid] = mid.

证当n % 4 == 2或3时无解：由上述可知，一组4个，假设n%4==2时有解，则该组元素应该有两个，则需y = n+1-y, x = n+1-x-->x+y = n+1, 联立这3>个方程得x=y.

而x=y的话P(P(x)) = x，说明该组元素只有一个，假设不成立. n%4=3的证明应该时类似的。

反思：题目已给出规则之后，可以先尝试着应用几次规则，看看会不会发现什么规律。

Xor Trie

题意(CF#173E)：给出一串数，在其首尾各取连续的一段，要使其异或和最大。

思路：

暴力思路：枚举i,j，O(n^2)，TLE

Trie：其实在暴力思路中，我们每枚举一个前缀pref[i]，那接下来就只是看在接下来的所有后缀中找一个后缀，使得pref[i] XOR suff[j]最大。

\*\*\*问题就转换成了“给一个数a，另外一堆数找出一个数b，使得a XOR b最大” \*\*\*

想想：肯定要把这些数转换成二进制数，枚举a的最高位，要使异或值最大，就要与a对应的位不同。

想到这里，那就好办了：把这一堆数装进一个字典树中，当然，是从最高位开始装，然后就是在这一棵字典树中尽量找出与 a 当前位不同>的数，直到找到最低位为止，那么当前路径上经过的数就是我们的数 b 了，a 异或 b 也一定是最大的

利用XOR的特性...于是就出现了Trie

自己实现Trie的时候撮了，这里的代码是抄别人的. 竟然用到了Trie，太神了!

class Trie {

public :

Trie \*children[2];

Trie () { children[0] = children[1] = NULL; }

void Insert(long long x, int deep) {

if(deep == -1) return ;

int t = ((x >> deep) & 1LL);

if(!children[t]) {

children[t] = new Trie();

}

children[t]->Insert(x, deep-1);

}

long long Query(long long x, int deep) {

if(deep == -1) return 0;

int t = ((x >> deep) & 1LL);

if(children[!t]) {

return (1LL << deep) + children[!t]->Query(x, deep-1);

} else {

return children[t]->Query(x, deep-1);

}

return 0; //ATTENTION !

}

} ;

Trie root = \*(new Trie());

int main()

{

scanf("%I64d", &n);

for(long long i = 0; i < n; i++) scanf("%I64d", &a[i]);

for(long long i = n-1; i >= 0; i--) suff[i] = suff[i+1] ^ a[i];

root.Insert(0, 63);

long long pre = 0;

long long ans = 0;

for(long long i = 0; i < n; i++) {

ans = max(ans, root.Query(suff[i], 63));

pre ^= a[i];

root.Insert(pre, 63);

}

ans = max(ans, root.Query(0, 63));

printf("%I64d\n", ans);

return 0;

}

单调栈

题意：给出一个序列，要求找到一个区间[l, r]，使得这个区间里的最大数和次最大数的XOR值最大。

思路：一定要好好吸收这个思想。

//贴题解：从另外一个角度考虑，当我确定一个数为次大数以后，那么对应的最大数其实是可以确定下来的了，就是左边第一个比当前数大的，和右边的第一个。这两个数都可以用单调栈来维护，因此复杂度就是O(N)，枚举一遍求最大值即可。

如果a[i]想当区间maximum，则从i处出发从左往右扫，这个过程它都可以作为区间maximum，直到扫到一个比a[i]大的数x，扫描停止。

如果a[i]想当区间second maximum，在上述扫描过程中，我们不是扫到一个比a[i]大的数x吗，这个时候就可以用x作为区间maximum，而a[i]就>可以作为second maximum了。

为什么我们只考虑从右往左扫，那i的右边部分呢？ 其实i的右边部分对于a[i]来说是可以不用考虑的，因为对于后面的j(i < j)，我们同样

也会做上述的扫描，所以“i的右边”变成了“j的左边”，这时候就帮i搞定了“i的右边”。

上述过程用线性暴力扫的话冗余很多，所以用从大到小的单调栈来维护，减少冗余： (1)如果栈顶元素a[x] < a[i]，则区间[x, i] maximum = a[i], second\_muximum = a[x]，将a[x]弹出栈，a[i]继续向下比较

(2)如果栈顶元素a[x] > a[i]，则区间[x, i] maximum = a[x], second\_muxinum = a[i], 将a[i]压栈，停止扫描。因为再下去扫没有意义，扫下去的区间a[i]充当不了maximum和second\_maximum

for(int i = 0; i < n; i++) {

scanf("%d", &x);

while(!S.empty() && x > S.top()) {

ans = max(ans, x ^ S.top());

S.pop();

}

if(!S.empty()) ans = max(ans, x ^ S.top());

S.push(x);

}

XOR 抑或 思路题

题意(CF#169D)：给出区间[l, r]，找出l <= a,b <= r，使得a XOR b最大.

思路：可以证明最优异或结果总是1111..11形式的。这一点我自己做的时候没能高出来。

来试着小证一下：当l == r时，ans = 0;

当l < r时，假设l和r在第k位上的数字（二进制）不同。则k+1,k+2,...,63位的异或值肯定是0，易知b的k右部分可取2^k-1

所以可以这样取a，b：b = r, a取b的异或值的前k位, 后面的位数跟r一样

r:10101100

l:10000110

b:10101100 (r)

a:01010011

int main()

{

long long left, right, ans = 0;

cin >> left >> right;

int i;

for(i = 63; i >= 0; i--)

if((left & (1LL << i)) ^ (right & (1LL << i))) break;

for(i ; i >= 0; i--) ans |= (1LL << i);

cout << ans << endl;

return 0;

}

图论构造题 启发题

题意(CF#167E):n匹马，每匹马有不超过3个敌人。现要把这些马分成两部分，使得对于每条马，和它在同一部分的敌人数量不超过 1个.给出敌对关系，要求一个划分方案，或impossible = -1.

思路：思路是看题解的，题解讲得很清晰 ...

实现也是用他的思路，不过代码比他简洁. enemy[i]是记录跟i在同一部分的敌人数量

摘抄：

做法分析

首先，观察下为什么题目给的数据范围这么奇葩：每条马的敌人的数量不超过 3 个

这有什么用呢？想了很久，画了好几个图，最终确定，这样的条件下，一定是存在一个划分方案，使得每部分中，每条马的敌人数量不超过 1 个。

可以考虑 4 个点的完全图，每个点的度是 3，对应了 3 个敌人，我们完全可以找出一种分配的方案使得这个图的点分成两个点集，那么，对于其余的情况

，肯定也是能够找出一种解的，因为他们的关系比 4 个点的完全图的关系更加的简单

也就是说，输出 -1 的情况是不存在的

接下来就考虑怎么构造出一种分配的方案了

实在是想不出有什么其他的做法了，干脆贪心的找找，类似于 SPFA 的 BFS：

①先假设所有的马都在同一个部分中，把不合格的马（有超过 1个敌人的）入队

②BFS 的过程中，先把当前马移动到另一个部分中，然后再统计它的敌人中，哪些马变得不合法了，把不合法的加入到队列中

这样不断的 BFS，肯定能够找到一种分配方案，但是具体的时间复杂度不知道怎么计算的，迷迷糊糊的就过了......

反思：题解作者在看到“敌人不超过3个”这一条件的时候就觉得这可能是突破点，而我虽然也在意了一下，但没有深入地去思考，也就没有从这>个条件里发掘出什么有用的东西。\*\*\*最不好的一点是，我甚至都没有怎么在纸上划划！！！\*\*\* 作者是一步一步思考出AC来的（看题解就知道），但我就>没有他那种“一步一步”的耐心！改！

void bfs()

{

for(int i = 1; i <= n; i++) {

if(enemy[i] > 1) Q.push\_back(i);

color[i] = 0;

}

while(!Q.empty()) {

int u = Q.front();

Q.pop\_front();

if(enemy[u] <= 1) continue; //有可能他的敌人已经被扔到另一边了，enemy[u]就变得<=1了，这里就不能扔u

color[u] = !color[u]; //把u扔到另一边

enemy[u] = 0;

for(int i = 0; i < adj[u].size(); i++) {

int v = adj[u][i];

if(color[v] == color[u]) enemy[u]++, enemy[v]++;

else enemy[v]--;

if(enemy[v] > 1) Q.push\_back(v);

}

}

}