
Acm Template

Macmillan School of Magic

Macmillan Expedition

Gene_Liu



November 9, 2019

Contents

0 常规操作	1
1 数学	7
1.1 快速幂与快速乘	7
1.1.1 快速幂	7
1.1.2 矩阵快速幂	7
1.1.3 快速乘取模	8
1.2 筛	9
1.2.1 埃氏筛	9
1.2.2 线性筛	10
1.2.3 杜教筛	10
1.3 GCD 与逆元	12
1.3.1 GCD	12
1.3.2 Exgcd 与逆元	12
1.3.3 O(n) 打表求逆元	13
1.4 唯一分解	13
1.5 中国剩余定理	14
1.5.1 中国剩余定理	14
1.5.2 中国剩余定理拓展	15
1.6 求组合数	16
1.6.1 Lucas (模数为质数)	16
1.6.2 递推求组合数	17
1.7 线性基	17
1.7.1 线性基模板	17
1.7.2 无向图异或路径问题	19
1.7.3 异或为 X 的子集长度和	21
1.7.4 前缀线性基	23
1.7.5 线段树维护线性基交	24
1.8 二次剩余	26
1.9 常见数学式子	27
1.10 数论板子集合	27
1.11 斐波那契数列循环节	29
1.12 划分数	32
1.13 卡特兰数	35
1.14 高斯素数	35
2 博弈	37
2.1 SG 函数	37
3 动态规划	39
3.1 01 背包	39
3.2 完全背包	40
3.3 多重背包	43
3.4 状压 dp	44
3.4.1 子集枚举	44
3.4.2 计算一对元素贡献	45
3.5 区间 dp	47
3.5.1 区间 dp 例题 1	47
3.5.2 区间 dp 例题 2	48
3.6 数位 dp	49
3.6.1 数位 dp 模板	49
3.6.2 巧妙数位 dp 例题	50
3.6.3 双数 dp 问题	51
3.6.4 三数 dp 问题	53
3.6.5 三进制状压	55
3.6.6 数字平方和	57
3.6.7 被数位整除	59
3.6.8 数位逆序对之和	60

3.7 概率 dp	62
3.8 斜率优化 dp	64
3.8.1 斜率优化例题	64
3.8.2 动态维护下凸壳	65
3.9 斯坦纳树	67
3.9.1 斯坦纳树模板	67
3.9.2 斯坦纳森林	69
3.9.3 斯坦纳树路径输出	71
3.10 长链剖分	73
3.10.1 k 级祖先	73
3.10.2 O(1) 优化树形 dp 模板	75
3.10.3 长链剖分维护后缀和	76
3.10.4 长链剖分 + 线段树	78
3.10.5 三元组两两距离相等	81
3.11 巧妙 dp 题	83
3.11.1 亏欠型 DP	83
3.11.2 数位 DP+ 状压 DP	84
4 数据结构	88
4.1 单调栈	88
4.2 单调队列	89
4.3 并查集	90
4.3.1 普通并查集	90
4.3.2 带权并查集	90
4.3.3 带权并查集与背包	92
4.3.4 按秩合并并查集	94
4.4 Hash	96
4.5 KMP	98
4.6 字典树	99
4.7 ST 表	100
4.8 树状数组	100
4.9 线段树	101
4.9.1 动态开点与区间修改 lazy	101
4.9.2 区间递减序列和	101
4.9.3 线段树维护图连通	103
4.9.4 线段树维护树直径	105
4.9.5 区间不同 gcd 个数	107
4.10 扫描线	108
4.10.1 面积并	108
4.10.2 面积交	110
4.10.3 周长并	112
4.10.4 包星星问题	114
4.10.5 覆盖奇数次的面积	116
4.11 主席树	118
4.11.1 静态主席树	118
4.11.2 区间中不同数的个数	120
4.11.3 单点修改主席树	122
4.12 李超树	124
4.12.1 李超树模板题	124
4.12.2 离散化 + 维护最大最小直线	125
4.12.3 李超树 + 两个 lazy	128
4.12.4 树剖 + 李超树 + 区间直线最大值	131
4.13 启发式合并	134
4.13.1 DSU 模板	134
4.13.2 例题 1	135
4.13.3 例题 2	137
4.14 Splay	139
4.14.1 普通 Splay	139
4.14.2 区间 Splay	141

4.14.3 Splay (三个 lazy 标记)	144
4.15 LCT	148
4.15.1 LCT 模板题	148
4.15.2 边权 LCT	153
4.15.3 LCT 最大连续和	155
4.15.4 最小差值生成树	158
4.15.5 维护实边与虚边	160
4.15.6 TopTree	162
4.16 树链剖分	167
4.16.1 点剖模板	167
4.16.2 树上路径颜色段数量	170
4.16.3 动态开点树剖	173
4.16.4 树剖换根	175
4.16.5 边剖 + 主席树	178
4.17 CDQ	181
4.17.1 陌上开花 (三维偏序)	181
4.17.2 动态逆序对	182
4.17.3 矩阵前缀和	184
4.18 莫队	186
4.18.1 普通莫队	186
4.18.2 莫队求组合数前缀和	188
4.18.3 带修改莫队	188
4.18.4 树上带修改莫队	190
4.18.5 回滚莫队 (增加)	193
4.18.6 回滚莫队 (减少)	195
4.19 虚树	196
4.20 十字链表	199
4.21 柯朵莉树	201
4.22 AC 自动机	203
4.22.1 AC 自动机模板	203
4.22.2 AC 自动机 +DP1	205
4.22.3 AC 自动机 + 最短路	208
4.22.4 AC 自动机 +DP2	210
4.22.5 AC 自动机 + 状压 DP	213
4.22.6 AC 自动机 + 缩点	216
4.22.7 阿狸打字机	218
4.23 回文串问题	221
4.23.1 Manacher 模板	221
4.23.2 回文自动机模板	222
4.23.3 模板题	224
4.23.4 Manacher 例题	226
4.23.5 Manacher+ 回文自动机	227
4.23.6 两颗回文自动机	229
4.23.7 马拉车 + 字典树	230
4.24 后缀数组	232
4.24.1 后缀数组模板	232
4.24.2 单字符串常见问题	234
4.24.3 最长公共子串 (双串)	237
4.24.4 长度不小于 k 的公共子串个数	240
4.24.5 n 字符串问题	243
4.25 整体二分	243
4.25.1 无修改第 K 大	243
4.25.2 带修改第 K 大	245
4.25.3 线段树 + 整体二分	247
4.25.4 并查集 + 整体二分	249
4.26 分块	251

5 图论	254
5.1 最短路	254
5.1.1 Dijkstra	254
5.1.2 Dijkstra 记录路径	255
5.1.3 分层图	257
5.1.4 spfa	259
5.1.5 spfa 判断负环	260
5.1.6 最短路计数问题	261
5.2 最小生成树	263
5.3 最近公共祖先	264
5.3.1 LCA (dfs 版本)	264
5.3.2 LCA (bfs 版本)	265
5.4 拓扑排序	267
5.5 欧拉路	269
5.6 双连通分量	270
5.6.1 边双缩点以及求割边	270
5.6.2 边双缩点 LCA	272
5.6.3 点双缩点以及求割点	276
5.6.4 点双例题	279
5.7 强连通分量	280
5.7.1 强连通分量及缩点	280
5.7.2 2-SAT	282
5.8 二分图匹配	284
5.9 第 K 小问题	286
5.9.1 S 到 T 的第 K 短路	286
5.9.2 有向全图第 k 小路径	288
5.9.3 无向全图第 k 小最短路	289
5.9.4 无向全图第 k 小团	291
5.10 简单图构造	292
6 网络流	295
6.1 最大流	295
6.1.1 最小点覆盖	295
6.1.2 最大流 + 二分	297
6.1.3 最大流路径输出	300
6.1.4 最大权闭合子图	302
6.1.5 最小路径覆盖	305
6.2 费用流	308
7 计算几何	311
7.1 计算几何板子	311
7.2 凸包三分	315
7.3 锐角三角形计数	317
7.4 点与凸包的切线	318
8 其他	321
8.1 三分	321
8.2 bitset 优化暴力	321
8.3 IDA*	323
8.4 左右手路径	324
8.5 圆周率小数点后第 n 位	325
8.6 BM 板子	326
8.6.1 BM 板子	326
8.6.2 BM 递推式	328
8.7 高精度	329
8.7.1 高精度	329
8.7.2 大数模板	330

0 常规操作

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define per(i,a,b) for(int i = a; i >= b; i--)
5 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
6 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
7 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
8     << " , " << z1 << ":" << z2 << endl;
9 typedef long long ll;
10 const int N = 1e5+100;
11 const int M = 1e5+100;
12 const db EPS = 1e-9;
13 using namespace std;
14
15 void dbg() {cout << "\n";}
16 template<typename T, typename... A> void dbg(T a, A... x) {cout << a << ' ' ; dbg(x...)}
17 #define logs(x...) {cout << #x << " -> "; dbg(x);}
18
19 /*快速读入、输出*/
20 namespace fastIO {
21     #define BUF_SIZE 10000000          // 预定义缓冲区大小，读入数据大的话就多开点
22     #define LL long long
23     //fread->read
24     bool IOerror=0;
25     inline char nc(){
26         static char buf[BUF_SIZE],*p1=buf+BUF_SIZE,*pend=buf+BUF_SIZE;
27         if (p1==pend){
28             p1=buf; pend=fread(buf,1,BUF_SIZE,stdin);
29             if (pend==p1){IOerror=1;return -1;}
30         }
31         return *p1++;
32     }
33     inline bool blank(char ch){return ch==' '||ch=='\n'||ch=='\r'||ch=='\t';}
34     inline void read(int &x){
35         bool sign=0; char ch=nc(); x=0;
36         for (;blank(ch);ch=nc());
37         if (IOerror) return;
38         if (ch=='-')sign=1,ch=nc();
39         for (;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
40         if (sign)x=-x;
41     }
42     inline void read(LL &x){
43         bool sign=0; char ch=nc(); x=0;
44         for (;blank(ch);ch=nc());
45         if (IOerror) return;
46         if (ch=='-')sign=1,ch=nc();
47         for (;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
48         if (sign)x=-x;
49     }
50     inline void read(double &x){
51         bool sign=0; char ch=nc(); x=0;
52         for (;blank(ch);ch=nc());
53         if (IOerror) return;
54         if (ch=='-')sign=1,ch=nc();

```

```

55     for (;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
56     if (ch=='.'){
57         double tmp=1; ch=nc();
58         for (;ch>='0'&&ch<='9';ch=nc())tmp/=10.0,x+=tmp*(ch-'0');
59     }
60     if (sign)x=-x;
61 }
62 inline void read(char *s){
63     char ch=nc();
64     for (;blank(ch);ch=nc());
65     if (IOerror)return;
66     for (;!blank(ch)&&!IOerror;ch=nc())*s++=ch;
67     *s=0;
68 }
69 inline void read(char &c){
70     for (c=nc();blank(c);c=nc());
71     if (IOerror){c=-1;return;}
72 }
73 #undef LL
74 #undef BUF_SIZE
75 };
76 using namespace fastIO;
77
78 inline void write(int x)
79 {
80     static const int maxlen = 100;
81     static char s[maxlen];
82     if(x < 0) {putchar('-');x=-x;}
83     if(!x){putchar('0'); return;}
84     int len = 0; for(;x;x/=10) s[len++] = x % 10+'0';
85     for(int i = len-1; i >= 0; --i) putchar(s[i]);
86 }
87
88 int n; read(n); write(n); printf("\n");
89 /*-----*/
90
91 /*疯狂加速*/
92 #pragma GCC optimize(2)
93 #pragma GCC optimize(3)
94 #pragma GCC optimize("Ofast")
95 #pragma GCC optimize("inline")
96 /*-----*/
97
98 /*__int128快速读入、输出*/
99 //读入
100 inline char nc() {
101     static char buf[100000], *p1 = buf, *p2 = buf;
102     return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 100000, stdin), p1 == p2) ? EOF :
103         *p1++;
104 }
105 template <typename T> bool rn(T& v) {
106     static char ch;
107     while (ch != EOF && !isdigit(ch)) ch = nc();
108     if (ch == EOF) return false;
109     for (v = 0; isdigit(ch); ch = nc()) v = v * 10 + ch - '0';
110     return true;
111 }
112 //输出
113 template <typename T> void o(T p) {

```

```

113     static int stk[70], tp;
114     if (p == 0) {
115         putchar('0');
116         return;
117     }
118     if (p < 0) {
119         p = -p;
120         putchar('-');
121     }
122     while (p) stk[++tp] = p % 10, p /= 10;
123     while (tp) putchar(stk[tp--] + '0');
124 }
125 /*-----*/
126
127 /*离散化*/
128 sort(base+1,base+1+n);
129
130 int tot = unique(base+1,base+1+n)-base-1;
131
132 int find(ll x){
133     return lower_bound(base+1,base+1+tot,x)-base;
134 }
135 /*-----*/
136
137
138 /*生成随机数*/
139 #include <ctime>
140 srand(time(0));
141 int x = rand()%(int)1000+1; //1~1000范围
142
143 #include<random>
144 #include<ctime>
145 std::mt19937 rnd(time(0));
146 printf("%lld\n",rnd()); //周期长度为2^19937-1, 比rand()优秀很多
147 /*-----*/
148
149
150 /*浮点数取模*/
151 void FLoat_Mod(long double a, long double b){
152     printf("%.9Lf\n",a-b*(long long)(a/b));
153 }
154 /*-----*/
155
156 /* 相关于char[]的输入输出 */
157 char s[105];
158 scanf("%s", s); // 读入一个字符串, 且首指针放到s+0
159 scanf("%s%s%s", s+1, s+21, s+41); // 读入三个字符串, 且首指针分贝放到s+1、s+21、s+41, 注意本
160 样例若前两个字符串长度超过19, 就会在内存有重叠而造成使用出错
161 scanf("\n[%^\n]*c",s); // 读入一行(第一种形式), 首指针为s+0。也可以换为"%[^\\n]*c", %*c表示
162 忽略后一个字符
163 //注意此种读法会忽略每行开始的空格
164 //^表示非, [%\\n]表示读入换行字符就结束读入, %*c表示该输入项读入后不赋予任何变量
165 scanf("%[^\\n]", s); getchar(); // 读入一行(第二种形式), 首指针为s+0。getchar()表示把\\n去掉
166 printf("%s %s", s+0, s+1); // 分别表示从第0、1位开始输出, 直到遇到'\\0', 所以想截断输出一个字符
167 串还可以将字符串的某个位置置为'\\0'即可输出前半部分
168 /*-----*/
169
170 /* scanf */

```

```

168 scanf("%*[ ]%[^/]/%d", t[i].s, &t[i].a); // %*[] 表示越过[]中的字符, %[a-z]表示读入字符串, 直到  

    遇到不是a-z中的字符为止  

169 // %[^a] 表示读入字符串知道遇到字符a为止, 但a并没有被读入  

170 getline(cin, s); // s为一个string, 推荐使用 getline 进行整行读入  

171 /*-----*/  

172  

173 /* 相关于string的输入输出 */  

174 string s; s.resize(100); // 想用scanf读string必须预先设置大小  

175 scanf("%s", &s[0]); // 读入到string  

176 scanf("\n%[^\\n]*c", &s[0]); // 读入1行到string  

177 char tmp[100];  

178 scanf("%s", tmp+1); s=tmp+1; // 先读入到char[]再令string=char[]  

179 printf("%s", s.c_str()); // 输出string  

180 /*-----*/  

181  

182  

183 /*map中存放string*/  

184 mp[s.c_str()] //最好传入string形式, 避免发生错误  

185 std::unordered_map<int,int> tmp; mp.swap(tmp); //设置一个空的map用于交换, 即mp.clear()  

186 mp.reserve(1000); //预留map中元素个数  

187 mp.rehash(1000); //给元素个数预留足够的bucket用于hash  

188 /*-----*/  

189  

190  

191 /*string函数*/  

192 /* 数字转字符串 */  

193 strings s = to_string(2323.232);  

194 /* 取长度 */  

195 int sz = s.size();  

196 /* 取下标字符 */  

197 char c = s[2];  

198 /* 截取字符串 */  

199 string ss = s.substr(2, 3); // 从第2个字符开始截取长度为3的串  

200 /* 查找串 */  

201 size_t found = s.find("23", 0); // 从位置0开始找到第1个串"23", 找到返回下标, 否则found=  

    string::npos  

202 size_t found = s.rfind("23", s.size()-1); // 从右侧开始找到第1个串"23", 至多找到至pos  

203 size_t可以强转为int, int pos = (int)s.find("23",0);  

204 int pos = (int)s.find(':'); //string也可以直接查找字符  

205 /* 拼接 */  

206 s += "aaaa";  

207 /*-----*/  

208  

209  

210 /*bitset函数*/  

211 /* 声明 */  

212 bitset<16> a; // 空, 全零 0000000000000000  

213 bitset<16> b(0xffff); // 整数参数 0011111111111111  

214 bitset<16> c("00101"); // 字符串 0000000000000101  

215  

216 /* 赋值 */  

217 a[0] = 1; // 低位第0位设为1  

218 a.set(0, 1); // 低位第0位设为1  

219 a.set(0); // 低位第0位设为1  

220 a.set(0, 0); // 低位第0位设为0  

221 a.set(); // 设为全1  

222 a.reset(); // 设为全0, 该函数同样有两个参数, 参数1为pos, 参数2为value(为空时是0)  

223 a.flip(1); // 翻转第1位  

224 a.flip(); // 翻转整个01串

```

```

225     a<<=1;          // a左移1位
226
227     /* 遍历 */
228     cout << a << endl;           // cout输出整个01串
229     printf("%s\n", a.to_string().c_str()); // printf输出整个01串, 较为麻烦一些
230     cout << a[0] << endl;         // cout输出低位第0位
231     printf("%d\n", a[0]==1);      // printf输出低位第0位
232
233     /* 判位、换型 */
234     a.count();                  //统计1的个数
235     a = a1 & a2;              //按位与
236     a = a1 | a2;              //按位或
237     a = a1 ^ a2;              //按位异或
238     a = ~a1;                  //按位补
239     a = a1 << 3;              //移位
240     a.text(0);                // 判断第0位是否为1
241     a.any();                  // 判断是否至少1位为1
242     s.none();                 // 判断是否全0
243     int one = a.count();       // 获得1的个数
244     string s = a.to_string();  // 转换为字符串
245     unsigned long = a.to_ulong(); // 转换为无符号整形
246     unsigned long long = a.to_ullong(); // 转换为无符号整形
247     /*-----*/
248
249
250     /*vector*/
251     /* 声明 */
252     vector<int> v;
253     /* 排序 */
254     sort(v.begin(),v.end());
255     /* 遍历方式一 (推荐) */
256     for(auto &tp : v){}
257     /* 遍历方式二 (不推荐) */
258     if(v.size()) { //一定要确保v不为空, 否则会RE
259         rep(i,0,v.size()-1){}
260     }
261     /* resize() 和 reserve() 函数 */
262     v.reserve(100); //改变了vector的capacity, 但是没有改变其空间大小
263     v.resize(100); //同时改变了vector的capacity和size, 且原有元素不会被覆盖
264     /*-----*/
265
266
267     /*set函数*/
268     /* 声明 */
269     set<int> s;
270     /* 赋值 */
271     s.insert(1);
272     s.insert(2);
273     /* 遍历 */
274     if(s.find(1) != s.end()) printf("had");
275     for(set<int>::iterator it=s.begin();it!=s.end();it++) printf("%d", *it); // 全部遍历
276     for(auto &x : s) printf("%d\n", x);
277     int sz = s.size(); // 大小
278     /* 清空 */
279     s.clear();
280     s.erase(x); //x为s中存的内容
281     *(s.begin()); //取出set中首位元素
282     multiset<int>::iterator it1 = st.begin(), it2 = st.end();
283     it2--; //取出set/multiset首尾元素

```

```

284     set<pair<int,int> >::iterator it = st1.lower_bound(make_pair(-pos,0)); //it是指针
285     lower_bound: 找到第一个大于等于这个数的值, 若要找第一个小于的可以考虑往set里丢负值
286     /* multiset删除 */
287     st.erase(st.find(x)) //只删一个x
288     st.erase(x) //删除所有x
289     /*-----*/
290
291
292     /*regex分割*/
293     regex re("[\\.,!\\? ]"); //全局变量, .与?需要转义, 此处分割的内容有'.'、','、'!'、'?'、' '
294     //分割之后, 最后的一个字符串可能会含有大量空格
295     /* 切割单词后放入一个vector */
296     vector<string> ans{
297         sregex_token_iterator(yy.begin()+pos+1, yy.end(), re, -1),
298         sregex_token_iterator()
299     };
300     for(auto &it: ans) printf("%s\n", it.c_str());
301     /*-----*/
302
303     /*线段树节点编号*/
304     inline int get_id(int l,int r) {return (l+r)|(l!=r);}
305     /*-----*/
306
307     /*二进制中01操作*/
308     template<class T> int getbit(T s, int i) { return (s >> i) & 1; }
309     template<class T> T onbit(T s, int i) { return s | (T(1) << i); }
310     template<class T> T offbit(T s, int i) { return s & (~(T(1) << i)); }
311     template<class T> int cntbit(T s) { return __builtin_popcount(s); }
312     /*-----*/
313
314     /*运行时间*/
315     #include <ctime> //程序开头
316     clock_t startTime,endTime;
317     startTime = clock(); //计时开始
318
319     //程序末尾
320     endTime = clock(); //计时结束
321     cout << "The run time is: " <<(double)(endTime - startTime) / CLOCKS_PER_SEC << "s"
322     << endl;
323     /*-----*/

```

1 数学

1.1 快速幂与快速乘

1.1.1 快速幂

```

1 //快速幂
2 #include <cstdio>
3 #include <iostream>
4 typedef long long LL;
5 using namespace std;
6
7 LL poww(LL a,LL b)
8 {
9     LL base = a,ans = 1;
10    while(b!=0)
11    {
12        if(b&1)
13            ans *= base;
14        base *= base;
15        b >>= 1;
16    }
17    return ans;
18 }
19
20 LL poww(LL a,LL b,LL p)
21 {
22     LL base = a%p,ans = 1;
23     while(b!=0)
24     {
25         if(b&1)
26             ans = (ans*base)%p;
27         base = (base*base)%p;
28         b >>= 1;
29     }
30     return (ans%p);
31 }
32
33 int main()
34 {
35     LL h = poww(11,11);
36     cout<<h<<endl;
37     return 0;
38 }
```

1.1.2 矩阵快速幂

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 using namespace std;
5 const int mod = 9973;
6 typedef long long ll;
7
8 int n,m,t;
9 //i、k、j只能作为临时变量
10 struct Matrix{
11     ll a[35][35];
12     Matrix():memset(a,0,sizeof(a)){};
```

```

13     Matrix operator * (const Matrix y)
14     {
15         Matrix ans;
16         for(int i = 1;i <= n;i++) //行
17             for(int j = 1;j <= n;j++) //列
18                 for(int k = 1;k <= n;k++)
19                     ans.a[i][j] = (ans.a[i][j]+a[i][k]*y.a[k][j])%mod;
20
21         return ans;
22     }
23
24     Matrix operator + (const Matrix y)
25     {
26         Matrix ans;
27         for(int i = 1;i <= n;i++)
28             for(int j = 1;j <= n;j++)
29                 ans.a[i][j] = (a[i][j]+y.a[i][j])%mod;
30
31         return ans;
32     }
33
34     Matrix q_pow(Matrix x,int k)
35     {
36         Matrix ans;
37         for(int i = 1;i <= n;i++) ans.a[i][i] = 1;
38         while(k)
39         {
40             if(k&1)
41                 ans = ans*x;
42             x = x*x;
43             k = k>>1;
44         }
45         return ans;
46     }
47
48     int main()
49     {
50         cin>>t;
51         while(t--)
52         {
53             scanf("%d%d",&n,&m);
54             Matrix p;
55             for(int i = 1;i <= n;i++)
56                 for(int j = 1;j <= n;j++)
57                     scanf("%lld",&p.a[i][j]);
58             Matrix ans = q_pow(p,m);
59             long long ans1 = 0;
60             for(int i = 1;i <= n;i++)
61                 ans1 = (ans.a[i][i]+ans1)%mod;
62             printf("%lld\n",ans1);
63         }
64         return 0;
65     }

```

1.1.3 快速乘取模

方法一：
a b p 范围都在 1e18

求 a^*b 的结果

为了不让结果爆 ll，则将 a 进行二进制分解，然后递推 $*b$ 即可

方法二：

用 long double 存数，long double 的有效数字有 18-19 位

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define rep(i,a,b) for(int i = a; i <= b; i++)
6 using namespace std;
7 typedef long long ll;
8
9 ll mul(ll a,ll b,ll p)
10 {
11     a %= p, b %= p;
12     long long c = (long double)a*b/p;
13     long long ans = a*b-c*p;
14     //此处会溢出，但是由于二者相差的部分不会太大，所以前面的数字都是一样的
15     //因此溢出不但不影响答案，还是符合我们的要求的
16     if(ans < 0) ans+=p;
17     else if(ans >= p) ans-=p;
18     return ans;
19 }
20
21 int main()
22 {
23     ll a,b,p;
24     while(~scanf("%lld%lld%lld",&a,&b,&p))
25     {
26         printf("%lld\n",mul(a,b,p));
27     }
28     return 0;
29 }
```

1.2 篩

1.2.1 埃氏篩

```

1 #include <cstdio>
2 #include <iostream>
3 #include <algorithm>
4 #include <cstring>
5 #define rep(i,a,b) for(int i = a;i <= b;i++)
6 using namespace std;
7 const int N = 1.5*1e7+1000;
8
9 int b[N],tot,prime[N];
10
11 int main()
12 {
13     tot = 0; //素数个数
14
15     //埃氏篩
16     for(int i = 2; i <= 1e5; i++)
17     {
18         if(b[i] == 0)
```

```

19         prime[++tot] = i; //哪几个是素数
20     else continue;
21     int base = i;
22     while(base <= 1e5){
23         base += i;
24         b[base] = 1; //这个数是不是素数，为1则非素数
25     }
26 }
27 return 0;
28 }
```

1.2.2 线性筛

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define rep(i,a,b) for(int i = a; i <= b; i++)
6 using namespace std;
7 const int N = 1e7+1000;
8
9 int v[N],prime[N],tot; //v[i]: i的最小质因子
10
11 void primes(int x) //线性筛
12 {
13     memset(v,0,sizeof v);
14     tot = 0;
15     rep(i,2,x){
16         if(!v[i]) v[i] = i, prime[++tot] = i;
17         rep(j,1,tot)
18             if(prime[j] > v[i] || prime[j] > x/i) break;
19             else v[i*prime[j]] = prime[j];
20     }
21 }
22
23 int main()
24 {
25     int n = 1e5;
26     primes(n);
27     rep(i,1,tot) printf("%d\n",prime[i]);
28     return 0;
29 }
```

1.2.3 杜教筛

给定一个 n ($n \leq 2^{31} - 1$), 求 ans_1 和 ans_2

$$ans_1 = \sum_{i=1}^n \varphi(i)$$

$$ans_2 = \sum_{i=1}^n \mu(i)$$

```

1 #include<bits/stdc++.h>
2 #include<tr1/unordered_map>
3 #define N 6000010
4 using namespace std;
5 template<typename T>inline void read(T &x)
6 {
```

```

7     x=0;
8     static int p;p=1;
9     static char c;c=getchar();
10    while(!isdigit(c)){if(c=='-')p=-1;c=getchar();}
11    while(isdigit(c)) {x=(x<<1)+(x<<3)+(c-48);c=getchar();}
12    x*=p;
13 }
14 bool vis[N];
15 int mu[N],sum1[N],phi[N];
16 long long sum2[N];
17 int cnt,prim[N];
18 tr1::unordered_map<long long,long long>w1;
19 tr1::unordered_map<int,int>w;
20 void get(int maxn)
21 {
22     phi[1]=mu[1]=1;
23     for(int i=2;i<=maxn;i++)
24     {
25         if(!vis[i])
26         {
27             prim[++cnt]=i;
28             mu[i]=-1;phi[i]=i-1;
29         }
30         for(int j=1;j<=cnt&&prim[j]*i<=maxn;j++)
31         {
32             vis[i*prim[j]]=1;
33             if(i%prim[j]==0)
34             {
35                 phi[i*prim[j]]=phi[i]*prim[j];
36                 break;
37             }
38             else mu[i*prim[j]]=-mu[i],phi[i*prim[j]]=phi[i]*(prim[j]-1);
39         }
40     }
41     for(int i=1;i<=maxn;i++)sum1[i]=sum1[i-1]+mu[i],sum2[i]=sum2[i-1]+phi[i];
42 }
43 int djsmu(int x)
44 {
45     if(x<=6000000)return sum1[x];
46     if(w[x])return w[x];
47     int ans=1;
48     for(int l=2,r;l>=0&&l<=x;l=r+1)
49     {
50         r=x/(x/l);
51         ans-=(r-l+1)*djsmu(x/l);
52     }
53     return w[x]=ans;
54 }
55 long long djsphi(long long x)
56 {
57     if(x<=6000000)return sum2[x];
58     if(w1[x])return w1[x];
59     long long ans=x*(x+1)/2;
60     for(long long l=2,r;l<=x;l=r+1)
61     {
62         r=x/(x/l);
63         ans-=(r-l+1)*djsphi(x/l);
64     }
65     return w1[x]=ans;

```

```

66 }
67 int main()
68 {
69     int t,n;
70     read(t);
71     get(6000000);
72     while(t--)
73     {
74         read(n);
75         printf("%lld %d\n",djsphi(n),djsmu(n));
76     }
77     return 0;
78 }
```

1.3 GCD 与逆元

1.3.1 GCD

```

1 typedef long long ll;
2
3 ll gcd(ll a,ll b)
4 {
5     return b == 0 ? a:gcd(b,a%b);
6 }
```

1.3.2 Exgcd 与逆元

```

1 #include <cstdio>
2 #include <iostream>
3 #include <algorithm>
4 #include <cstring>
5 #define rep(i,a,b) for(int i = a;i <= b;i++)
6 using namespace std;
7 typedef long long ll;
8 const ll mod = 9973;
9
10 //求ax+by = gcd(a,b)的特解
11 ll exgcd(ll a,ll b,ll &x,ll &y)
12 {
13     if(b == 0) {x = 1,y = 0; return a;} //此处a为gcd(a,b)
14     ll d = exgcd(b,a%b,x,y);
15     ll z = x; x = y; y = z-y*(a/b);
16     return d;
17 }
18 // ax+by = c的通解可以表示为 x = (c/d)x1+k*(b/a) 【d为gcd(a,b)】
19 //                                y = (c/d)y1-k*(a/d) 【k为正整数】
20
21 //调用mod_reverse就是求逆元，求a在%n意义下的逆元
22 ll mod_reverse(ll a,ll n)
23 {
24     ll x,y; //求a的逆元，a*ai 与 1 % mi同余，则a*ai-mi*k = 1; a与mi已知，求ai
25     ll d = exgcd(a,n,x,y);
26     if(d == 1) return (x%n+n)%n; //保证逆元为正
27     else return -1;
28 }
```

费马小定理求逆元
1. 模数 p 为素数

$$a^{-1} = a^{p-2}$$

2. 模数 p 不为素数
 $a^{-1} = a^{\phi(p)-1}$

```

1 int main()
2 {
3     ll x;
4     while(~scanf("%lld",&x))
5     {
6         printf("%lld\n",mod_reverse(x,mod));
7     }
8     return 0;
9 }
10 }
```

1.3.3 $O(n)$ 打表求逆元

```

1 #include <cstdio>
2 #include <cmath>
3 using namespace std;
4 typedef long long ll;
5 const int N = 1e5 + 5;
6
7 int inv[N];
8
9 void inverse(int n, int p) { //O(n)求1-n所有逆元
10     inv[1] = 1;
11     for (int i=2; i<=n; ++i) {
12         inv[i] = (ll)(p - p / i) * inv[p%i] % p;
13     }
14 }
```

1.4 唯一分解

```

1 #include <cstdio>
2 #include <iostream>
3 #include <algorithm>
4 #include <cstring>
5 #define rep(i,a,b) for(int i = a;i <= b;i++)
6 using namespace std;
7 const int N = 1.5*1e7+1000;
8
9 int b[N],c[N][2]; //c[i][0] —— 表示第 i 个唯一分解数
10 //c[i][1] —— 表示第 i 个唯一分解数的指数
11 int prime[N],tot;
12 int n;
13
14 //唯一分解
15 void getFactors(ll x)
16 {
17     ll tmp = x;
18     for(ll i = 1; prime[i]*prime[i] <= tmp; i++)
19     {
20         if(tmp%prime[i] == 0)
21         {
22             c[++tot][0] = prime[i];
23             while(tmp%prime[i] == 0){
```

```

24             tmp /= prime[i];
25         c[tot][1]++;
26     }
27 }
28 if(tmp != 1){
29     c[++tot][0] = tmp;
30     c[tot][1] = 1;
31 }
32 }
33 }
34
35 int main()
36 {
37     scanf("%d",&n);
38     tot = 0;
39
40     //埃氏筛
41     for(int i = 2; i <= 1e5; i++)
42     {
43         if(b[i] == 0)
44             prime[++tot] = i;
45         else continue;
46         int base = i;
47         while(base <= 1e5){
48             base += i;
49             b[base] = 1;
50         }
51     }
52     rep(i,1,n)
53     {
54         int x;
55         scanf("%d",&x);
56         getFactors(x);
57     }
58     return 0;
59 }
```

1.5 中国剩余定理

1.5.1 中国剩余定理

```

1 //中国剩余定理
2 #include <cstdio>
3 #include <iostream>
4 #include <algorithm>
5 #include <cstring>
6 #define rep(i,a,b) for(int i = a;i <= b;i++)
7 using namespace std;
8
9 //求ax+by = gcd(a,b)的特解
10 ll exgcd(ll a,ll b,ll &x,ll &y)
11 {
12     if(b == 0) {x = 1,y = 0; return a;} //此处a为gcd(a,b)
13     ll d = exgcd(b,a%b,x,y);
14     ll z = x; x = y; y = z-y*(a/b);
15     return d;
16 }
17 // ax+by = c的通解可以表示为 x = (c/d)x1+k*(b/a) 【d为gcd(a,b)】
18 //                                y = (c/d)y1-k*(a/d) 【k为正整数】
```

```

19
20 //调用mod_reverse就是求逆元，求a在%n意义下的逆元
21 ll mod_reverse(ll a,ll n)
22 {
23     ll x,y;          //求a的逆元，a*ai 与 1 % mi同余，则a*ai-mi*k = 1; a与mi已知，求ai
24     ll d = exgcd(a,n,x,y);
25     if(d == 1) return (x%n+n)%n; //保证逆元为正
26     else return -1;
27 }
28
29 //求ans%m1=a1,ans%m2=a2,...中的ans, 其中m1,m2,m3...互质
30 int CRT(int a[],int m[],int cn){
31     int M = 1;
32     int ans = 0;
33     for(int i=1; i<=cn; i++)
34         M *= m[i];           //M为所有的除数相乘
35     for(int i=1; i<=cn; i++){
36         int Mi = M / m[i];
37         int x = mod_reverse(Mi, m[i]); //求Mi在%m[i]意义下的逆元
38         ans = (ans + Mi * x * a[i]) % M; //最后求出的结果在1-M之间
39         //注意此处的Mi*x*a[i]如果有爆long long的风险，则需要调用快速乘来进行乘法运算
40     }
41     if(ans < 0) ans += M; //ans是特解, ans的通解 = ans+K*M (K为整数)
42     return ans;
43 }

```

1.5.2 中国剩余定理拓展

```

1 //拓展中国剩余定理 —— m1,m2,m3之间不互质
2 #include<iostream>
3 #include<cstdio>
4 #include<climits>
5 #include<cstring>
6 #include<algorithm>
7 using namespace std;
8 #define LL long long
9 const int maxn=1e5+5;
10 int n;
11 LL exgcd(LL a,LL b,LL &x,LL &y){
12     if(!b){x=1,y=0;return a;}
13     LL re=exgcd(b,a%b,x,y),tmp=x;
14     x=y,y=tmp-(a/b)*y;
15     return re;
16 }
17 LL m[maxn],a[maxn];
18 LL work(){
19     LL M=m[1],A=a[1],t,d,x,y;int i;
20     for(i=2;i<=n;i++){
21         d=exgcd(M,m[i],x,y); //解方程
22         if((a[i]-A)%d) return -1; //无解
23         x=(a[i]-A)/d,t=m[i]/d,x=(x%d+t)%t; //求x
24         A=M*x+A,M=M/d*m[i],A%=M; //日常膜一膜(划掉) 模一模, 防止爆
25     }
26     A=(A%M+M)%M;
27     return A;
28 }
29 int main()
30 {

```

```

31     int i,j;
32     while(scanf("%d",&n)!=EOF){
33         for(i=1;i<=n;i++)scanf("%lld%lld",&m[i],&a[i]);
34         printf("%lld\n",work());
35     }
36     return 0;
37 }
```

1.6 求组合数

1.6.1 Lucas (模数为质数)

```

1 #include <cstdio>
2 #include <iostream>
3 #include <algorithm>
4 #include <cstring>
5 #define rep(i,a,b) for(int i = a;i <= b;i++)
6 using namespace std;
7 typedef long long ll;
8
9 //求ax+by = gcd(a,b)的特解
10 ll exgcd(ll a,ll b,ll &x,ll &y)
11 {
12     if(b == 0) {x = 1,y = 0; return a;} //此处a为gcd(a,b)
13     ll d = exgcd(b,a%b,x,y);
14     ll z = x; x = y; y = z-y*(a/b);
15     return d;
16 }
17 // ax+by = c的通解可以表示为 x = (c/d)x1+k*(b/a) 【d为gcd(a,b)】
18 //                                y = (c/d)y1-k*(a/d) 【k为正整数】
19
20 //调用mod_reverse就是求逆元，求a在%p意义下的逆元
21 ll mod_reverse(ll a,ll p)
22 {
23     ll x,y; //求a的逆元，a*ai 与 1 % mi同余，则a*ai-mi*k = 1; a与mi已知，求ai
24     ll d = exgcd(a,p,x,y);
25     if(d == 1) return (x%p+p)%p; //保证逆元为正
26     else return -1;
27 }
28
29 ll fact(ll n, ll p){//n的阶乘 % p
30     ll ret = 1;
31     for (int i = 1; i <= n ; i++) ret = ret * i % p ;
32     return ret ;
33 }
34
35 ll comb(ll n, ll m, ll p){//C(n, m) % p, 将n和m限制在1-p之内
36     if (m < 0 || m > n) return 0;
37     //调用逆元和求解阶乘
38     return fact(n, p) * mod_reverse(fact(m, p), p) % p * mod_reverse(fact(n-m, p), p) %
39     p;
40 }
41 ll Lucas(ll n, ll m, ll p) //求组合数C(n,m)%p
42 {
43     return m ? Lucas(n/p, m/p, p) * comb(n%p, m%p, p) % p : 1;
44 }
45
46
```

```

47 int main()
48 {
49 /*注意事项：
50 C(n,m)%p用Lucas定理求解，p必须为质数，并且p < 1e5，这个限制主要是针对如果p大于1e5，那就算n和m
在1-p之内，如果n和m大于1e5，也难以求解
51 本模板还可以进一步进行加速，那就是预处理每个数的阶乘
52 */
53 ll n,m,p;
54 int T;
55 scanf("%d",&T);
56 while(T--)
57 {
58     scanf("%lld%lld%lld",&n,&m,&p);
59     printf("%lld\n",Lucas(n+m,m,p));
60 }
61 return 0;
62 }
```

1.6.2 递推求组合数

$$C(n, m) = C(n - 1, m - 1) + C(n - 1, m)$$

```

1 C[1][0] = C[1][1] = 1;
2
3 for (int i = 2; i < N; i++){
4     C[i][0] = 1;
5     for (int j = 1; j < N; j++)
6         C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]);
7 }
```

1.7 线性基

1.7.1 线性基模板

一、定义

1. 设数集 T 为一组数任意子集异或得到的集合。
2. T 的线性基是 T 的一个子集 $A = a_1, a_2, a_3, \dots, a_n$ 。 A 中元素互相 xor 所形成的异或集合，等价于原数集 T 的元素互相 xor 形成的异或集合。可以理解为将原数集进行了压缩。

二、性质

1. 线性基没有异或为 0 的子集。
2. 线性基的异或集合中每个元素的异或方案唯一，其实这个跟性质 1 是等价的。
3. 线性基二进制最高位互不相同。
4. 线性基中元素互相异或，异或集合不变。

三. 求 x 在 Q 中的排名 (允许重复)

1. 共有 cnt 个基底，则每个数出现的次数均为 $2^{n-cnt} - 1$ 。因为没被选入基底的数有 $n - cnt$ 个，其能用基底唯一表示，再把基底对应选中异或即为 0 向量，因此共有 $2^{n-cnt} - 1$ 个零向量。
2. x 由第 $a_0, \dots, a_i, \dots, a_m$ 个向量异或而来，没有参与的向量置 0，令 $P = a_0, \dots, a_i \dots, a_m$ 的二进制表示。因此如果 0 可取，则 x 排名为 $(P - 1) * (2^{n-cnt} - 1) + 1$ ；如果 0 不可取，则 x 排名为 P 。

四. 求 T 集合中第 i 位为 1 的数的个数 H

T 集合大小为 $2^{|Q|}$ ，令 $ALL | = p[i], i \in [0, 62]$ 。若 ALL 第 i 位为 1，则 $H = 2^{|Q|-1}$ ，否则 $H = 0$ 。

```

1 struct Linear_Basis{
2     ll p[65],d[65];
```

```

3     int cnt;
4     Linear_Basis() {memset(p, 0, sizeof p);}
5     ~Linear_Basis() {}
6     bool ins(ll x){ //向线性基中插入一个数
7         for(int i = 62; i >= 0; i--) {
8             if(x&(1ll<<i)){
9                 if(!p[i]) {p[i] = x; break;}
10                x ^= p[i];
11            }
12        }
13        return x > 0ll;
14    }
15    ll MAX(ll x){ //求线性空间与ans异或的最大值
16        for(int i = 62; i >= 0; i--)
17            if((x^p[i]) > x) x ^= p[i];
18        return x;
19    }
20    //如果是求一个数与线性基的异或最小值，则需要先rebuild，再从高位向低位依次进行异或
21    ll MIN(){
22        rep(i, 0, 62)
23            if(p[i]) return p[i];
24    }
25    //将线性基改造成每一位相互独立，即对于二进制的某一位i，只有pi的这一位是1，其它都是0
26    //注意此处只是避免多个基位上同时有第i位，但是可能会出现p[5]同时有第5位和第3位，但是p[3]为空
27    void rebuild(){
28        cnt = 0;
29        for(int i = 62; i >= 0; i--)
30            for(int j = i-1; j >= 0; j--)
31                if(p[i]&(1ll<<j))
32                    p[i]^=p[j];
33        rep(i, 0, 62)
34            if(p[i]) d[cnt++] = p[i];
35    }
36    //求线性基能够组成的数中的第K大
37    ll Kth(ll k){
38        ll ret = 0;
39        if(k >= (1ll<<cnt)) return -1; //k大于子集总数，找不到
40        for(int i = 62; i >= 0; i--)
41            if(k&(1ll<<i)) ret ^= d[i];
42        return ret;
43    }
44    //合并两个线性基
45    Linear_Basis& merge(const Linear_Basis &xx){
46        for(int i = 62; i >= 0; i--)
47            if(xx.p[i]) ins(xx.p[i]);
48        return *this;
49    }
50 }LB;
51
52 //两个线性基求交
53 Linear_Basis merge(Linear_Basis a, Linear_Basis b){
54     Linear_Basis A = a, tmp = a, ans; //tmp不断构建A+(B\ans)
55     ll cur,d;
56     rep(i, 0, 33) //从低到高，使得不存在一个基底可以仅由(tmp\A)表示
57         if(b.p[i]){ //b中有这个基底
58             cur = 0, d = b.p[i];
59             per(j, i, 0)
60                 if((d>>j)&1){
61                     if(tmp.p[j]){

```

```

62             d ^= tmp.p[j], cur ^= A.p[j];
63             if(d) continue;
64             ans.p[i] = cur; //cur的第i位不为0
65         }
66         else tmp.p[j] = d, A.p[j] = cur;
67         //如果不能被表示，A的赋值是为了让高位中含有j这位的基底下放到A中j的位置
68         break;
69     }
70 }
71 return ans;
72 }
```

1.7.2 无向图异或路径问题

题意：给出一个 n 个点， m 条边的无向图。定义一个三元组 (u, v, s) ，表示 u 到 v 的一条路径异或和为 s ，该三元组对答案的贡献为 s 。求出所有三元组的贡献和。 $(1 \leq n \leq 10^5, 0 \leq m \leq 2 * 10^5)$

思路：

直接求答案显然不现实，而由于这是一道异或问题，因此考虑对每一个数位求贡献。首先对连通分量求一个 dfs 树，即求出从根节点到每一个点的异或和，因此两点路径异或和就等于两个点的异或和再异或。 dfs 时再求出连通分量中所有环的异或和，将异或和插入线性基中。然后依次统计每一个数位对答案的贡献。

对于数位 i 来说，该数位统计到答案中只有两种情况。第一种情况，两点异或和在数位 i 处为 1，则求出线性基数集合中所有第 i 位为 0 的数个数。

第二种情况，两点异或和在数位 i 处为 0，则求出线性基数集合中所有第 i 位为 1 的数个数。

将两种情况分别统计累加到答案中，即可完成此题。此处需要注意，判断第 i 位为 1 的数能否由线性基组成，只需要将线性基中的 $p[i]$ 依次取或得到 ALL ，若 ALL 中第 i 位为 1，则可以构造，否则不可。

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
6 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
7           << " , " << z1 << ":" << z2 << endl;
8 typedef long long ll;
9 typedef double db;
10 const int N = 1e5+100;
11 const int M = 4*1e5+100;
12 const ll mod = 1e9+7;
13 const db EPS = 1e-9;
14 using namespace std;
15 int n,m,head[N],tot,vis[N];
16 ll rt[65],dis[N];
17 vector<ll> ve;
18 struct Node{
19     int to,next;
20     ll w;
21 }e[M];
22 ll ans = 0;
23
24 void add(int x,int y,ll w){
25     e[++tot].to = y, e[tot].next = head[x], head[x] = tot, e[tot].w = w;
```

```

26 }
27
28 struct Linear_Basis{
29     ll p[65],cnt;
30     Linear_Basis() {memset(p,0,sizeof p); cnt = 0;}
31     ~Linear_Basis() {}
32     void init(){
33         memset(p,0,sizeof p);
34         cnt = 0;
35     }
36     bool ins(ll x){
37         // LOG1("x",x);
38         for(int i = 62; i >= 0; i--){
39             if(x&(1ll<<i)){
40                 if(!p[i]) {p[i]=x; cnt++; break;}
41                 x ^= p[i];
42             }
43         }
44         return x > 0;
45     }
46     void rebuild(){
47         cnt = 0;
48         for(int i = 62; i >= 0; i--)
49             for(int j = i-1; j >= 0; j--)
50                 if(p[i]&(1ll<<j)) p[i]^=p[j];
51         rep(i,0,62)
52             if(p[i]) cnt++;
53     }
54 }LB;
55
56 void dfs(int x,ll tp){
57     if(vis[x]) return;
58     vis[x] = 1; ve.push_back(tp); dis[x] = tp;
59     for(int i = head[x]; i; i = e[i].next){
60         int y = e[i].to;
61         if(vis[y]){
62             ll xp = tp^e[i].w^dis[y];
63             // LOG2("y",y,"xp",xp);
64             LB.ins(xp);
65             continue;
66         }
67         dfs(y,tp^e[i].w);
68     }
69 }
70
71 int main()
72 {
73     scanf("%d%d",&n,&m); tot = 1;
74     rep(i,1,m){
75         int a,b; ll c; scanf("%d%d%lld",&a,&b,&c);
76         add(a,b,c); add(b,a,c);
77     }
78     rep(i,1,n)
79         if(!vis[i]){
80             LB.init(); ve.clear();
81             dfs(i,0);
82             ll thp = (ll)ve.size();
83             // LOG1("thp",thp);
84             memset(rt,0,sizeof rt);

```

```

85         for(auto &v:ve){
86             int tp = 0; ll hp = v;
87             while(hp){
88                 if(hp&1) rt[tp]++;
89                 tp++; hp /= 2ll;
90             }
91         }
92         LB.rebuild();
93         ll Total = 1;
94         rep(H,1,LB.cnt) Total = (Total*2ll)%mod;
95         ll ALL = 0;
96         rep(H,0,62) ALL |= LB.p[H];
97         // LOG1("cnt",LB.cnt);
98         rep(k,0,62){
99             ll hp1 = rt[k], hp2 = thp-rt[k]; //有和没有
100            ll tp = 0;
101            if(ALL&(1ll<<k)){
102                tp = 1;
103                rep(H,1,LB.cnt-1) tp = (tp*2ll)%mod;
104            }
105            // LOG3("Total",Total,"k",k,"tp",tp);
106            //01、10
107            ll now1 = (hp1*(hp1-1ll)/2ll+hp2*(hp2-1ll)/2ll)%mod, now2 = (hp1*hp2)%
108            mod;
109            ans = (ans+(1ll<<k)%mod*now1%mod*tp%mod)%mod;
110            ans = (ans+(1ll<<k)%mod*now2%mod*(Total-tp)%mod)%mod;
111        }
112        printf("%lld\n",ans);
113        return 0;
114    }

```

1.7.3 异或为 X 的子集长度和

意: 给出一个集合 A , 求出 A 中所有异或和为 0 的子集的长度和。($1 \leq n \leq 2 * 10^6, 0 \leq a_i \leq 10^{18}$)

思路: 首先线性基中并没有可以直接求出长度和的方法, 因此我们需要考虑其他方法进行求解。

既然无法从整体上进行求解, 从单个异或和为 0 的子集入手也不现实, 因此考虑最小单元 a_i 。即求出每一个 a_i 对答案的贡献。此处思考的入手点在于将题目要求解的值不断细分, 从整体到子集再到子集中的一个元素。

对于每一个 a_i 求贡献, 问题就简化了很多。首先对集合 A 求线性基, 则有 cnt 个元素成功插入, $n - cnt$ 个元素未插入, 则该 $n - cnt$ 个元素对答案的贡献为 $(n - cnt) * 2^{n - cnt}$ 。

再对于线性基中的 cnt 个元素求贡献。每次取出 cnt 中的一个元素 a_x , 将剩下的 $n - 1$ 个元素插入线性基中, 新形成的线性基元素个数为 T 。若新形成的线性基中无法插入 a_x , 则 a_x 对答案对贡献为 2^{n-1-T} 。至此本题结束。

总结: 本题中还需涉及从线性基中删除一个元素和合并线性基。合并线性基不用多说, 删除一个元素则是记录每一个元素插入线性基的位置, 从最后一个元素往回不断删除, 每次将插入位置置 0 即可。

本题还可以进行修改, 修改为求 A 中所有异或和为 X 的子集长度和。其余地方都无需改动, 只需要将“判断新形成的线性基中是否可以插入 a_x ”修改为“判断新形成的线性基中是否可以插入 $a_x \ xor \ X$ ”即可。

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;

```

```

5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
6 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
7 << " , " << z1 << ":" << z2 << endl;
7 typedef long long ll;
8 typedef double db;
9 const int N = 1e5+100;
10 const int M = 1e5+100;
11 const ll mod = 1e9+7;
12 const db EPS = 1e-9;
13 using namespace std;
14
15 int n;
16 ll a[N],ans;
17 vector<pair<ll,int> > v;
18
19 struct Linear_Basis{
20     ll p[65]; int cnt;
21     Linear_Basis() {memset(p,0,sizeof p); cnt = 0;}
22     ~Linear_Basis() {}
23     void init() {memset(p,0,sizeof p); cnt = 0;}
24     int ins(ll x){
25         int pos = -1;
26         for(int i = 62; i >= 0; i--)
27             if(x&(1ll<<i)){
28                 if(!p[i]) {p[i] = x; pos = i; cnt++; break;}
29                 else x ^= p[i];
30             }
31         return pos;
32     }
33     Linear_Basis& merge(Linear_Basis x){
34         for(int i = 62; i >= 0; i--)
35             if(x.p[i]) ins(x.p[i]);
36         return *this;
37     }
38 }LB1,LB2;
39
40 ll poww(ll a,ll b){
41     ll base = a, tp = 1;
42     while(b){
43         if(b&1) tp = (tp*base)%mod;
44         base = (base*base)%mod;
45         b >>= 1;
46     }
47     return (tp%mod);
48 }
49
50 int main()
51 {
52     while(~scanf("%d",&n))
53     {
54         ans = 0; LB1.init(); LB2.init(); v.clear();
55         rep(i,1,n) scanf("%lld",&a[i]);
56         rep(i,1,n){
57             int flag = LB1.ins(a[i]);
58             if(flag != -1) v.push_back(make_pair(a[i],flag));
59             else LB2.ins(a[i]);
60         }
61         int SIZE = v.size();

```

```

62     // LOG1("SIZE",SIZE);
63     ll tp = 1;
64     if(n-SIZE-1 >= 1) tp = poww(2ll,n-SIZE-1);
65     ans = (ans+((ll)n-(ll)SIZE)*tp%mod)%mod;
66     for(int i = SIZE-1; i >= 0; i--){
67         LB1.p[v[i].second] = 0; LB1.cnt--;
68         Linear_Basis LB3 = LB1;
69         LB3 = LB3.merge(LB2);
70         tp = n-1-LB3.cnt;
71         int flag = LB3.ins(v[i].first);
72         if(flag == -1){
73             ll hp = poww(2ll,tp);
74             ans = (ans+hp)%mod;
75         }
76         LB2.ins(v[i].first);
77     }
78     printf("%lld\n",ans);
79 }
80 return 0;
81 }
```

1.7.4 前缀线性基

题意: 给定 n 个数字, q 个询问, 每次询问对应一个 l, r , 询问区间 $[l, r]$ 中所有子集异或和的最大值。 $(1 \leq n \leq 5 * 10^5, 1 \leq q \leq 5 * 10^5)$

思路: 处理前缀线性基。 $p[i][j]$ 表示前 i 个数的第 j 位的线性基, $pos[i][j]$ 表示前 i 个数的第 j 位的线性基由哪一位数字得到, 保留最靠右的位置。

每次插入一个数字, 如果可以更新 $pos[i][j]$ 则更新, 并将 $p[i][j]$ 替换下来, 用于后续的更新。查询 $[l, r]$ 最值只需将 $p[r]$ 取出, 所有 $pos[r][j] \geq l$ 均为有效基。

总结: 前缀线性基, 维护区间线性基。

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
6 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
7     << " , " << z1 << ":" << z2 << endl;
8 typedef long long ll;
9 typedef double db;
10 const int N = 5*1e5+100;
11 const int M = 1e5+100;
12 const db EPS = 1e-9;
13 using namespace std;
14
15 int n,q,p[N][27],pos[N][27],a[N];
16
17 int main()
18 {
19     scanf("%d",&n);
20     rep(i,1,n) scanf("%d",&a[i]);
21     rep(i,1,n){
22         rep(k,0,24) p[i][k] = p[i-1][k], pos[i][k] = pos[i-1][k];
23         int ti = i;
```

```

23     for(int k = 24; k >= 0; k--) {
24         if(a[i] & (1<<k)) {
25             if(!p[i][k]) {p[i][k] = a[i]; pos[i][k] = ti; break;}
26             if(pos[i][k] < ti) swap(p[i][k], a[i]), swap(pos[i][k], ti);
27             a[i] ^= p[i][k];
28         }
29     }
30 }
31 scanf("%d", &q);
32 while(q--) {
33     int l, r; scanf("%d%d", &l, &r);
34     int ans = 0;
35     for(int k = 24; k >= 0; k--) {
36         if(pos[r][k] >= l && ans < (ans^p[r][k]))
37             ans = ans^p[r][k];
38     }
39     printf("%d\n", ans);
40 }
41 return 0;
42 }
```

1.7.5 线段树维护线性基交

引理：若 V_1, V_2 是线性空间， B_1, B_2 分别是他们的一组基，令 $W = B_2 \cap V_1$ ，若 $B_1 \cup (B_2 \setminus W)$ 线性无关，则 W 是 $V_1 \cap V_2$ 的一组基。

证明：

采用反证法，假设 $\exists v \in V_1 \cap V_2$ ，且 v 不能由 W 线性组合表示。由于 v 一定可以由 W 和 $B_2 \setminus W$ 线性组合表示，即 $v = xor(B_1) = xor(W) \wedge xor(B_2 \setminus W)$ ，转换可得 $xor(B_2 \setminus W) = xor(B_1) \wedge xor(W)$ ，因此 $B_2 \setminus W$ 与 B_1 线性相关，与题设条件不符，因此假设不成立，原定理正确。

求解：

因此问题就变成了如何求取 W 。令 $tmp = B_1 \cup (B_2 \setminus W)$ ， tmp 初始等于 B_1 ，我们在构建 tmp 的过程中求取 W ，因此我们不断取出 B_2 的基底加入 tmp 。假设当前取出的元素为 x ，若 x 不能被 tmp 表示，则直接在 tmp 中加入 x 。否则 x 一定能被 tmp 线性组合而成，设 $x = xor(B_1) \wedge xor(tmp \setminus B_1)$, $cur = xor(B_1) = x \wedge xor(tmp \setminus B_1)$ ，即 cur 既可以由 B_1 表示也可以由 B_2 表示，因此直接将 cur 加入 W 即可。

枚举 x 时，按照从低到高的顺序进行枚举，保证 x 不能仅由 $tmp \setminus B_1$ 表示。

问题描述：给出 n 个线性基，每次询问给出 l, r, x ，询问区间 $[l, r]$ 的线性基交能否线性组合出 x 。

问题解决：直接线段树维护区间线性基交即可。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 #define per(i,a,b) for(int i = a; i >= b; i--)
4 typedef long long ll;
5 const int N = 50000+100;
6 using namespace std;
7
8 int n,m;
9 struct Linear_Basis{
10     ll p[50];
11     Linear_Basis() {memset(p,0,sizeof p);}
12     ~Linear_Basis() {}
13     bool ins(ll x,int jud){
14         for(int i = 33; i >= 0; i--){
15             if(x&(1ll<<i)){
```

```

16         if(!p[i]) {
17             if(jud) p[i] = x;
18             break;
19         }
20         x ^= p[i];
21     }
22 }
23 return x > 0ll;
24 }
25 }t[2*N],base[N];
26 inline int get_id(int l,int r) {return (l+r)|(l!=r);}
27
28 Linear_Basis merge(Linear_Basis a, Linear_Basis b){
29     Linear_Basis A = a, tmp = a, ans; //tmp不断构建A+(B\ans)
30     ll cur,d;
31     rep(i,0,33) //从低到高, 使得不存在一个基底可以仅由(tmp\A)表示
32     if(b.p[i]){ //b中有这个基底
33         cur = 0, d = b.p[i];
34         per(j,i,0)
35         if((d>>j)&1){
36             if(tmp.p[j]){
37                 d ^= tmp.p[j], cur ^= A.p[j];
38                 if(d) continue;
39                 ans.p[i] = cur; //cur的第i位不为0
40             }
41             else tmp.p[j] = d, A.p[j] = cur;
42             //如果不能被表示, A的赋值是为了让高位中含有j这位的基底下放到A中j的位置
43             break;
44         }
45     }
46     return ans;
47 }
48
49 void build(int l,int r){
50     int now = get_id(l,r);
51     if(l == r) {t[now] = base[l]; return;}
52     int mid = (l+r)>>1;
53     build(l,mid); build(mid+1,r);
54     t[now] = merge(t[get_id(l,mid)],t[get_id(mid+1,r)]);
55 }
56
57 bool query(int l,int r,int tl,int tr,ll x){
58     int now = get_id(l,r);
59     if(tl <= l && r <= tr) return !t[now].ins(x,0);
60     int mid = (l+r)>>1;
61     if(tr <= mid) return query(l,mid,tl,tr,x);
62     else if(tl > mid) return query(mid+1,r,tl,tr,x);
63     else return query(l,mid,tl,tr,x) && query(mid+1,r,tl,tr,x);
64 }
65
66 int main()
67 {
68     scanf("%d%d",&n,&m);
69     rep(i,1,n){
70         int sz; scanf("%d",&sz);
71         rep(j,1,sz){
72             ll tp; scanf("%lld",&tp);
73             base[i].ins(tp,1);
74         }

```

```

75     }
76     build(1,n);
77     rep(i,1,m){
78         int l,r; ll x; scanf("%d%d%lld",&l,&r,&x);
79         if(query(1,n,l,r,x)) printf("YES\n");
80         else printf("NO\n");
81     }
82     return 0;
83 }
```

1.8 二次剩余

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
6 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
    << " , " << z1 << ":" << z2 << endl;
7 typedef long long ll;
8 typedef double db;
9 const db EPS = 1e-9;
10 using namespace std;
11 ll pow_mod(ll a, ll k, ll p) {
12     ll ret = 1;
13     while(k) {
14         if(k & 1)    ret = ret * a % p;
15         a = a * a % p;
16         k >>= 1;
17     }
18     return ret % p;
19 }
20 ll modsqr(ll a, ll n) { //求解  $x^2 \equiv a \pmod{n}$ 
21     ll b, k, i, x;
22     if(a == 0) return 0;
23     if(n == 2) return a % n;
24     if(pow_mod(a, (n - 1) / 2, n) == 1) {
25         if(n % 4 == 3) {
26             x = pow_mod(a, (n + 1) / 4, n);
27         } else {
28             for(b = 1; pow_mod(b, (n - 1) / 2, n) == 1; ++b);
29             i = (n - 1) / 2;
30             k = 0;
31             do {
32                 i /= 2;
33                 k /= 2;
34                 if((pow_mod(a, i, n) * (ll)pow_mod(b, k, n) + 1) % n == 0) k += (n -
1) / 2;
35             } while(i % 2 == 0);
36             x = (pow_mod(a, (i + 1) / 2, n) * (ll)pow_mod(b, k / 2, n)) % n;
37         }
38         if(x * 2 > n) x = n - x;
39         return x;
40     }
41     return -1;
42 }
43 int main() {
```

```

44     int T;
45     ll p = 1000000007ll;
46     ll inv2 = pow_mod(2ll, p - 2, p);
47     scanf("%d",&T);
48     while (T--) {
49         ll b, c;
50         scanf("%lld %lld",&b,&c);
51         ll a = ((b * b % p) - (4ll * c % p) + p) % p;
52         ll x = modsqr(a, p);
53         if(x == -1) {
54             puts("-1 -1");
55         } else {
56             ll y = p - x;
57             ll X = inv2 * (x + b) % p;
58             ll Y = inv2 * (y + b) % p;
59             printf("%lld %lld\n", min(X, Y), max(X, Y));
60         }
61     }
62     return 0;
63 }
```

1.9 常见数学式子

$$1. F(n) = \sum_{x=1}^n x * 2^x$$

错位相减，直接求出答案 $F(n) = (n - 1) * 2^{n+1} + 2$ 。

$$2. F(n) = \sum_{x=1}^n 2^x * x^2$$

先构造 $f(x) = \sum_{i=1}^n x^i * i^2$ ，即直接求出 x 的通式，然后再将 x 替换成 2。

利用高数计算无穷级数的方法，先构造 $\frac{f(x)}{x}$ ，然后先积分再求导，得出答案 $F(n) = 2^{n+1} * (n^2 - 2 * n + 3) - 6$ 。

3. 常见组合数公式

$$\sum_{i=1}^n 2^i i^2 = (n^2 - 2n + 3)2^{n+1} - 6$$

$$\sum_{i=1}^n 2^i i = (n - 1)2^{n+1} + 2$$

$$1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(n+1)(2*n+1)}{6}$$

$$1^3 + 2^3 + 3^3 + \dots + n^3 = \frac{n^2(n+1)^2}{4}$$

$$\sum_{i=0}^k C_{n+i}^n = C_{n+k+1}^{n+1}$$

$$\sum_{i=1}^n iC_n^i = n * 2^{n-1}$$

$$\sum_{i=1}^n i^2 C_n^i = n * (n + 1) * 2^{n-2}$$

$$\sum_{i=1}^n (-1)^{i-1} \frac{C_n^i}{i} = \sum_{i=1}^n \frac{1}{i}$$

$$\sum_{i=0}^n (C_n^i)^2 = C_{2n}^n$$

1 return 0;

1.10 数论板子集合

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
6 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
7 << " , " << z1 << ":" << z2 << endl;
8 typedef long long ll;
9 typedef double db;
10 const int N = 1e5+100;
11 const int M = 1e5+100;
12 const db EPS = 1e-9;
13 using namespace std;
14
15 //gcd
16 ll gcd(ll a,ll b)
17 {
18     return b == 0 ? a:gcd(b,a%b);
19 }
20
21 //快速幂
22 ll poww(ll a,ll b){
23     ll base = a, ans = 1ll;
24     while(b){
25         if(b&1) ans = (ans*base)%mod;
26         base = (base*base)%mod;
27         b >= 1;
28     }
29     return ans;
30 }
31
32 //线性筛
33 void primes(int x){
34     tot = 0;
35     memset(v,0,sizeof v);
36     rep(i,2,x){
37         if(!v[i]){
38             v[i] = i, prime[++tot] = i;
39         }
40         rep(j,1,tot){
41             if(prime[j] * i > x) break;
42             v[i*prime[j]] = prime[j];
43             if(i % prime[j] == 0) break;
44         }
45     }
46 }
47
48 //唯一分解
49 void getFactors(ll x){
50     ll tmp = x; tc = 0;
51     for(ll i = 1; prime[i]*prime[i] <= tmp; i++){
52         if(tmp%prime[i] == 0){
53             c[++tc][0] = prime[i]; c[tc][1] = 0;
54             while(tmp%prime[i] == 0){
55                 tmp /= prime[i];
56                 c[tc][1]++;
57             }
58         }
59     }
60 }

```

```

58     }
59     if(tmp != 1){
60         c[++tc][0] = tmp;
61         c[tc][1] = 1;
62     }
63 }
64
65 //费马小定理  $a^{p-2}$  为逆元
66 ll comp(ll base,ll x){ //求组合数, C(base,x)
67     ll ans1 = 1, ans2 = 1;
68     for(ll i = base; i >= base-x+1ll; i--) ans1 = (ans1*i)%mod;
69     for(ll i = 1ll; i <= x; i++) ans2 = (ans2*i)%mod;
70     ll tp = (ans1*poww(ans2,mod-2ll))%mod;
71     return tp;
72 }
73
74 //递推求组合数
75 c[1][0] = c[1][1] = 1; //c[i][j] = C(i,j)
76 for(int i = 2; i <= 3000; i++){
77     c[i][0] = 1;
78     for(int j = 1; j <= i; j++)
79         c[i][j] = (c[i-1][j]+c[i-1][j-1])%mod;
80 }

```

1.11 斐波那契数列循环节

$(f[1], f[2]) - (f[3], f[4]) - (f[5], f[6]) - (f[7], f[8])$, 给定 n 求 m , 满足 $f[1+m]\%n = f[1]\%n$
在此题中, 由于每次增加两个数, 因此最后需要将答案/2。

```

1 #include <bits/stdc++.h>
2
3 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
4 using namespace std;
5
6 #define ll unsigned long long
7
8 ll gcd(ll a, ll b) {
9     return b == 0 ? a : gcd(b, a % b);
10 }
11
12 ll lcm(ll a, ll b) {
13     return a / gcd(a, b) * b;
14 }
15
16 struct Matrix {
17     ll mat[2][2];
18
19     Matrix() { memset(mat, 0, sizeof mat); }
20 };
21
22 Matrix mul_M(Matrix a, Matrix b, ll mod) {
23     Matrix ans;
24     for (int i = 0; i < 2; ++i) {
25         for (int j = 0; j < 2; ++j) {
26             ans.mat[i][j] = 0;
27             for (int k = 0; k < 2; ++k) {
28                 ans.mat[i][j] += a.mat[i][k] * b.mat[k][j] % mod;

```

```

29             if (ans.mat[i][j] >= mod) ans.mat[i][j] -= mod;
30         }
31     }
32 }
33 return ans;
34 }
35
36 Matrix pow_M(Matrix a, ll n, ll mod) {
37     Matrix ans;
38     for (int i = 0; i < 2; ++i) ans.mat[i][i] = 1;
39     Matrix tmp = a;
40     while (n) {
41         if (n & 1) ans = mul_M(ans, tmp, mod);
42         tmp = mul_M(tmp, tmp, mod);
43         n >>= 1;
44     }
45     return ans;
46 }
47
48 ll pow_m(ll a, ll n, ll mod) {
49     ll ans = 1, tmp = a % mod;
50     while (n) {
51         if (n & 1) ans = ans * tmp % mod;
52         tmp = tmp * tmp % mod;
53         n >>= 1;
54     }
55     return ans;
56 }
57
58 const int maxn = 1e6 + 10;
59
60 int prime[maxn];
61
62 void getprime() {
63     memset(prime, 0, sizeof prime);
64     for (int i = 2; i <= maxn; ++i) {
65         if (!prime[i]) prime[prime[0]] = i;
66         for (int j = 1; j <= prime[0] && prime[j] * i <= maxn; ++j) {
67             prime[prime[j] * i] = 1;
68             if (i % prime[j] == 0) break;
69         }
70     }
71 }
72
73 ll factor[100][2];
74 int fatcnt;
75
76 int getfactors(ll x) {
77     fatcnt = 0;
78     ll tmp = x;
79     for (int i = 1; prime[i] * prime[i] <= tmp; ++i) {
80         factor[fatcnt][1] = 0;
81         if (tmp % prime[i] == 0) {
82             factor[fatcnt][0] = prime[i];
83             while (tmp % prime[i] == 0) {
84                 factor[fatcnt][1]++;
85                 tmp /= prime[i];
86             }
87             fatcnt++;
88         }
89     }
90 }
```

```

88         }
89     }
90     if (tmp != 1) {
91         factor[fatcnt][0] = tmp;
92         factor[fatcnt++][1] = 1;
93     }
94     return fatcnt;
95 }
96
97 int legendre(ll a, ll p) {
98     if (pow_m(a, (p - 1) >> 1, p) == 1) return 1;
99     else return -1;
100 }
101
102 int f0 = 1;
103 int f1 = 1;
104
105 ll getfib(ll n, ll mod) {
106     if (mod == 1) return 0;
107     Matrix A;
108     A.mat[0][0] = 0;
109     A.mat[1][0] = 1;
110     A.mat[0][1] = 1;
111     A.mat[1][1] = 1;
112     Matrix B = pow_M(A, n, mod);
113     ll ans = f0 * B.mat[0][0] + f1 * B.mat[1][0];
114     return ans % mod;
115 }
116
117 ll fac[maxn];
118
119 ll G(ll p) {
120     ll num;
121     if (legendre(5, p) == 1) num = p - 1;
122     else num = 2 * (p + 1);
123     int cnt = 0;
124     for (ll i = 1; i * i <= num; ++i) {
125         if (num % i == 0) {
126             fac[cnt++] = i;
127             if (i * i != num) {
128                 fac[cnt++] = num / i;
129             }
130         }
131     }
132     sort(fac, fac + cnt);
133     ll ans;
134     for (int i = 0; i < cnt; ++i) {
135         if (getfib(fac[i], p) == f0 && getfib(fac[i] + 1, p) == f1) {
136             ans = fac[i];
137             break;
138         }
139     }
140     return ans;
141 }
142
143 ll find_loop(ll n) {
144     getfactors(n);
145     ll ans = 1;
146     for (int i = 0; i < fatcnt; ++i) {

```

```

147     ll record = 1;
148     if (factor[i][0] == 2)record = 3;
149     else if (factor[i][0] == 3)record = 8;
150     else if (factor[i][0] == 5)record = 20;
151     else record = G(factor[i][0]);
152     for (int j = 1; j < factor[i][1]; ++j) {
153         record *= factor[i][0];
154     }
155     ans = lcm(ans, record);
156 }
157 return ans;
158 }
159
160 int main() {
161     __;
162     getprime();
163     ll n;
164     while (cin >> n) {
165         ll ans = find_loop(n);
166         cout << ans / 2ll << endl;
167     }
168     return 0;
169 }
```

1.12 划分数

一、适用问题

只要涉及一些难以用组合公式直接求得的数的划分问题。比如将 n 个数划分成多个不大于 m 的划分方法，且不存在相同数。或者划分成若干奇数或偶数的问题。

划分要求千变万化，主要识别点在于将划分数类比为盒子与球的问题，类比成多个球放到多个盒子中，且球和盒子都没有区别。

这类问题与一般的划分问题都不同，需要二维 dp 求解，有时还需要处理前缀和进行计算。

二、算法举例

(1) 将 n 个数划分成多个不大于 m 的数，可以存在相同整数

$f[n][m]$ 表示整数 n 的划分中，每个数都不大于 m 的划分方案。转移时可以分成两种情况，第一种情况，划分方案中每个数都小于 m ，因此划分方案为 $f[n][m - 1]$ 。

第二种情况，划分方案中至少有一个数为 m ，即在 n 中减去 m ，得到 $f[n - m][m]$ 。

$$f[n][m] = f[n][m - 1] + f[n - m][m]$$

(2) 将 n 个数划分成多个不大于 m 的数，不能存在相同整数

这个问题和上一个问题唯一的区别就在于不能存在相同整数，因此只需要修改第二种情况即可实现正确的转移。

$$f[n][m] = f[n][m - 1] + f[n - m][m - 1]$$

(3) 将 n 个数划分成 k 个数的划分方案

考虑 k 个数中是否存在 1。第一种情况从 n 个数中取出 k 个，保证剩下 k 组中每组数大于等于 2。第二种情况则保证 k 份中有至少一个 1，即先取出一个 1 分成 $k - 1$ 组。

$$f[n][k] = f[n - k][k] + f[n - 1][k - 1]$$

(4) 将 n 个数划分成不超过 k 个数的划分方案

此处枚举的是划分方案中是否有 k 组, $f[n][k]$ 分别由划分组数小于 k 和划分组数等于 k 两种情况转移而来。

$$f[n][k] = f[n - k][k] + f[n][k - 1]$$

(5) 将 n 个数划分若干奇数/偶数的划分方案

$g[n][m]$: 将 n 划分为 m 个偶数的方案

$f[n][m]$: 将 n 划分为 m 个奇数的方案

求 $g[n][m]$ 只需从 n 中取出 m 个, 然后将 $f[n - m][m]$ 中的每个数 +1, 从奇数变成偶数即可。求 $f[n][m]$ 则需要考虑划分方案中有无 1 的情况, 因为直接从 $g[n - m][m]$ 转移过来, 会丢失有 1 的这种情况。

$$g[n][m] = f[n - m][m]$$

$$f[n][m] = f[n - 1][m - 1] + g[n - m][m]$$

(6) 将 n 个数划分成若干个数, 使这若干个数的乘积最大

1. 对于任意大于等于 4 的正整数 m , 可以划分成 $m = m_1 + m_2$, 使得 $m_1 * m_2 \geq m$ 。

证明: 令 $m_1 = (\text{int})m/2, m_2 = m - m_1$, $m_1 \geq 2, m_2 \geq m_1$ 且 $m_2 > 2$, 则 $m_1 * m_2 \geq 2 * m_2 \geq m$ 。

2. 因此不断这样拆分下去, 最后剩下的数一定是很多个 2 和 3 的组合, 即 $ans = 2^r * 3^s$, 且 $r \leq 2$ 。

若 $r > 2$, 则至少有 3 个因子为 2, 而 $2 * 2 * 2 < 3 * 3$, 且 $2 + 2 + 2 = 3 + 3$, 因此 $r \leq 2$ 。

3. 至此为止, 不难发现答案只需对小于 4 的数分类讨论, 对于大于 4 的数, 枚举 2 的最少个数即可。

$$n = 3 * r, ans = 3^r$$

$$n = 3 * r + 1, ans = 3^{r-1} * 2 * 2$$

$$n = 3 * r + 2, ans = 3^r * 2$$

(7) 将 n 个数划分成若干个不同的数, 使这若干个数的乘积最大

1. 拆分成的数的个数越多, 乘积越大。因此找到第一个大于等于 n 的 $S_x = \sum_{i=2}^x i$ 。

2. 若 $S_x = n$, 则 S_x 即为划分答案。

3. 若 $S_x = n + 1$, 则将 S_x 中的 2 去掉, 令 x 变成 $x + 1$, 即为最大值。

4. 若 $S_x > n + 1, k = S_x - n$, 则答案为将 S_x 序列中的 k 去掉, 剩下数构成最终答案。

三、例题

题意: 给出一个 H 行 N 列的网格, (r, c) 指第 r 行第 c 列的格子。现在需要在网格中选择 N 个格子涂成黑色。这 N 个格子需要满足一定的条件。

1. $(1, 1)$ 需要涂黑。

2. 如果 (r, a) 与 (r, b) 涂黑了, 则 (r, k) 为黑。 $a < k < b$, 即每行涂黑的格子必须是连续的。

3. 如果 (r, c) 涂黑了, 则 $(r - 1, c)$ 如果存在, 也必须涂黑。

4. 如果 (r, c) 涂黑了, 并且现在没有任何 $k < c$ 使得 (r, k) 是黑的, 则 $(r + 1, c)$ 为白。

稍加模拟就会发现满足如上条件的格子组成了一个金字塔形状。如下图所示。

先要求给出一个 H 和 N , 求出有多少种不同的涂法满足条件, 答案模 $10^9 + 7$ 。 $(1 \leq H, N \leq 10^5)$

思路: 首先不难发现, 若想满足题意进行格子涂色, 则每行涂色的格子数必定逐行递减, 因此可以发现涂色格子的行数一定在 \sqrt{N} 级别, 即最多不会超过 400。

然后观察这个问题, 这是个计数问题, 比较常见的考虑就是使用 dp 进行转移计算答案。而使用 dp 的话, 我们的选择就无非是按行从上往下 dp 转移, 或者按列从左往右 dp 转移。(不过比赛的时候绝大多数想到的都只有按行从上往下)

如果按行从上往下转移，那怎么想都需要一个三维 dp , $dp[i][j][k]$, 第 i 行, 有 j 个格子, 还剩 k 个格子没有填的方案数, 然后就可以全队自闭了。思路陷在这里如果出不来就基本是告别 AC 了。

既然行不可行, 我们来考虑列转移, 从左往右进行列的 dp 转移。我们从最高点将左右两边的金字塔划开, 发现左边的金字塔列是不断递增上来的, 即前一个与后一个要么高度不变, 要么高度 +1。而右边的金字塔则只是递增, 并没有严格的 +1 关系。

因此我们考虑对于左右两边进行分开 dp 。左边状态为 $f[i][j]$, 表示 i 个数, 最大的数为 j 的方案个数, 并且在转移过程严格要求一个数列非递减关系。而定义右边状态为 $s[i][j]$, 表示 i 个数, 所有的数小于等于 j 的方案个数。然后来分别思考转移方程。

左半部分转移方程, 由于必须满足划分的数为非递减关系, 因此我们枚举前一个数为 j 或 $j - 1$, 因此得到如下转移方程。

$$f[0][0] = 1$$

$$f[i][j] = f[i - j][j] + f[i - j][j - 1]$$

右半部分转移方程, 只需满足所有数小于等于 j , 因此可以类比之前的经典模型, i 个数进行划分, 最大的数不超过 j 的划分方案数。

$$s[0][i] = 1$$

$$s[i][j] = s[i][j - 1] + s[i - j][j]$$

最后, 枚举高度, 然后再枚举左半部分格子数统计答案即可。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 const int N = 1e5+100;
4 const int mod = 1e9+7;
5 using namespace std;
6
7 int n,h,f[N][460],s[N][460];
8
9 void init(){
10     //f[i][j]: i个数, 最大的数为j, 方案个数
11     f[0][0] = 1;
12     rep(i,1,100000)
13         rep(j,1,min(i,455)){
14             f[i][j] = (f[i-j][j]+f[i-j][j-1])%mod;
15         }
16
17     //s[i][j]: i个数, 所有的数小于等于j, 方案个数
18     rep(i,0,455) s[0][i] = 1;
19     rep(i,1,100000){
20         rep(j,1,455){
21             s[i][j] = s[i][j-1];
22             if(i >= j) s[i][j] = (s[i][j]+s[i-j][j])%mod;
23         }
24     }
25 }
26
27 int main()
28 {
29     init();
30     scanf("%d%d",&h,&n);
31     long long ans = 0;
32     //一开始先枚举高度, 整个题的思路也是这样, 但是根据内存连续性的问题, 必T无疑...我也成功找了一个小

```

```

时bug才找到...
33 //今后枚举数组时，务必记得先枚举行，再枚举列，谨记今日教训
34 rep(i,1,n)
35     rep(j,1,min(h,455))
36     ans = (ans+1ll*f[i][j]*s[n-i][j-1])%mod;
37     printf("%lld\n",ans);
38     return 0;
39 }

```

1.13 卡特兰数

一、基础公式

(1) 定义式

$$f[n] = \sum_{i=0}^{n-1} f[i] * f[n - 1 - i]$$

(2) 组合数公式 (用生成函数推导定义式)

$$1. f[n] = C_{2n}^n - C_{2n}^{n-1}$$

$$2. f[n] = \frac{C_{2n}^n}{n+1}$$

$$3. f[n] = \frac{4n-2}{n+1} * f[n-1]$$

(3) 卡特兰数列

$$1. f[0] = 1, f[1] = 1, f[2] = 2, f[3] = 5\dots$$

$$2. 1,1,2,5,14,42,132,429,1430,4862,16796,58786,208012\dots$$

二、应用方向

(1) 括号匹配问题

n 个左括号, n 个右括号, 对于每一个位置, 左括号数大于等于右括号数的方案总数。

等价于 n 个 1, n 个 -1, 每个位置前缀和大于等于 0 的方案总数。

两种理解方向:

1. $f[n] = \sum_{i=0}^{n-1} f[i] * f[n - 1 - i]$ 。枚举第一次前缀和为 0 的位置, 假如第 $2 * x$ 个点为第一次前缀和为 0 的点, 则固定第一个数为 1, 第 x 个数为 -1, 则对答案贡献为 $f[x-1] * f[n-x]$ 。

2. $f[n] = C_{2n}^n - C_{2n}^{n-1}$ 。总方案数为 C_{2n}^n , 现需求不符合条件的方案数, 将问题转化为网格上的折线问题。第 i 次在 (i, j) 处, 第 $i+1$ 次在 $(i+1, j+1)$ 或 $(i+1, j-1)$ 处, 终点为 $(2n, 0)$ 。不符合条件则说明折线上出现了 $(x, -1)$ 这个点, x 为第一次到达 -1 的点, 我们将 x 点之后的折线沿 $y = -1$ 对称过来, 则终点为 $(2n, -2)$, 则一共有 $n-1$ 个 1, $n+1$ 个 -1, 即不合法的方案总数为 C_{2n}^{n-1} , 因此 $f[n] = C_{2n}^n - C_{2n}^{n-1}$ 。

(2) 出栈次序问题

一个无穷大的栈, 进栈序列为 1, 2, 3... n , 求有多少个不同的出栈序列。

(3) 多边划分三角形问题

将一个凸多边形区域分成三角形区域的方案数。

在圆上选择 $2 * n$ 个点, 将这些点对连接起来使得所得到的 n 条线段不想交的方案数。

(4) 二叉树计数问题

给定 n 个节点, 能构成多少种形状不同的二叉树。

先取一个点作为顶点, 然后左边依次可以取 $0 \dots n-1$ 个点, 右边则可以取 $n-1 \dots 0$ 个点, 相乘再累加即可得到答案。

```
1 return 0;
```

1.14 高斯素数

一、高斯整数

$a = x + y * i$ ($x, y \in Z$)，则 a 为高斯整数。

a 的范为 $N(a) = |a^2| = x^2 + y^2$ 。

若存在高斯整数 y ，使得 $ay = 1$ ，则 a 为高斯整数中的乘法可逆元， y 为 a 的逆。

高斯整数 a 是可逆元的充要条件是 $N(a) = 1$ ，高斯整数中只有 4 个可逆元，分别是 -1 1 i $-i$ 。

$a b$ 为高斯整数， $a = by$ ，则 a 和 b 等价，即 $a = b - b ib - ib$ 。

二、高斯素数

定义：设 ϕ 为高斯整数中的非零非可逆元，则 ϕ 为高斯素数。即 ϕ 的因子或者为可逆元，或者是与 ϕ 等价的高斯整数。

ϕ 为高斯整数，且 $N(\phi) = p$ 为素数，则 ϕ 必定为高斯素数。

若 ϕ 为高斯素数，则其共轭元也是高斯素数。

三、高斯素数判断

高斯整数 $a + bi$ 是素数，当且仅当：

1. $a b$ 中有一个是零，另一个数的绝对值是形如 $4 * n + 3$ 的素数。
2. $a b$ 均不为零，而 $a^2 + b^2$ 为素数。

四、费马平方和定理

奇质数能表示为两个平方数之和的充分必要条件是该质数被 4 除余 1，即 $4 * n + 1$ 。

例题：找出 $[l, r]$ 中的素数 t ，满足 $t = a^2 + b^2$ ($a, b \in N^*$)，输出这种素数总数。

```
1 return 0;
```

2 博弈

2.1 SG 函数

题意：有 n 堆魔法珠，第 i 堆有 a_i 颗。选择 n 堆魔法珠中数量大于 1 的任意一堆。记该堆魔法珠数量为 p ， p 有 $b_1 b_2 b_3 \dots b_m$ 这 m 个小于 p 的约数。然后将这一堆魔法珠变成 m 堆，每堆各有 $b_1 b_2 b_3 \dots b_m$ 颗魔法珠。最后选择这 m 堆的任意一堆，令其消失，不可操作者输，问谁能获胜。 $(1 \leq n \leq 100, 1 \leq a_i \leq 1000)$

思路：不难发现整个游戏就是由多个魔法珠堆组成的，因此 $SG[x]$ 表示某一堆魔法珠，个数为 x 时的 SG 值。求 $SG[x]$ 时，若 x 有 $b_1 b_2 b_3 \dots b_m$ 这 m 个小于 x 的约数，则可以达到的后继状态一共有 m 个，即去掉任意一堆达到的状态。而每一个状态中仍有 $m - 1$ 个堆，即 $m - 1$ 个子游戏，因此每一个状态的 SG 值为这 $m - 1$ 个子游戏 SG 值的异或和。具体过程见代码。

```

1 #include <csdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
8 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
9 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
10    << " , " << z1 << ":" << z2 << endl;
11 typedef long long ll;
12 typedef double db;
13 const int N = 1000+10;
14 const int M = 1e5+100;
15 const db EPS = 1e-9;
16 using namespace std;
17
18 int sg[N],n;
19
20 int solve(int x)
21 {
22     if(sg[x] != -1) return sg[x];
23     int vis[2010]; memset(vis,0,sizeof vis);
24     int ans[2010]; //记录每一个约数
25     int tp = 0, ct = 0;
26     rep(i,1,x-1){
27         if(x%i == 0){
28             ans[++ct] = solve(i);
29             tp ^= ans[ct];
30         }
31     }
32     rep(i,1,ct) vis[tp^ans[i]] = 1; //枚举每一个子状态
33     rep(i,0,2000)
34         if(vis[i] == 0) return sg[x] = i;
35 }
36
37 int main()
38 {
39     memset(sg,-1,sizeof sg);
40     sg[1] = 0;
41     while(~scanf("%d",&n)){
42         int ans = 0;
43         rep(i,1,n){
44             int xx; scanf("%d",&xx);

```

```
44         ans ^= solve(xx);
45     }
46     if(ans == 0) printf("rainbow\n");
47     else printf("freda\n");
48 }
49 return 0;
50 }
```

3 动态规划

3.1 01 背包

题意：一个比赛有 n 个裁判，每个裁判有一个权值 a_i ，并且给出通过或不通过。有一个判断值 p ，若所有给出通过的裁判的权值和大于等于 p ，那么整体就算通过。给出第二组裁判，权值为 b_i ，判断值为 q ，判断所有 2^n 种判断下这两组给出的结果是否完全相同，如果不相同，给出一种不相同的情况。

思路：比赛的时候完全没思路，一个劲的贪心... 赛后才知道这题其实是个 dp， $dp[i][j]$ 表示前 i 个人，序列 a 中选的人的值到达 j 的所有情况中，序列 b 中对应人的权值之和的最大值。因此当 $dp[i][j] \geq q$ ，而 $j < p$ 时，即是错误的情况。然后再判断一遍 b 序列即可。此处还有一个地方，记错误方案时如果是用 pre 数组不断回溯的话，会 T。正确做法是对于每个 $dp[j]$ 记一个 bitset 的状态，检查出错误后，直接输出即可。

```

1 #include <csdio>
2 #include <iostream>
3 #include <cstring>
4 #include <bitset>
5 #include <algorithm>
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
8 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
9 ;
10 typedef long long ll;
11 typedef double db;
12 const db EPS = 1e-9;
13 using namespace std;
14 const int N = 200;
15 const int M = 1e6+1000;
16 int n,a[N],b[N],dp[M],am,bm,tp[M],ans[N];
17 //dp[i]表示a组中，价值为i的所有组合中，b组最大价值的一组
18 bitset<110> base1[M],base2[M];
19
20 void solve()
21 {
22     rep(i,1,n) ans[i] = 0;
23     rep(i,1,n){
24         for(int j = am-1; j >= a[i]; j--){
25             if(j == a[i]){
26                 if(dp[j] < b[i]) dp[j] = b[i], base1[j].reset(),base1[j].set(i);
27             }
28             if(dp[j] < dp[j-a[i]]+b[i] && dp[j-a[i]]!=0){
29                 dp[j] = dp[j-a[i]]+b[i];
30                 base1[j] = base1[j-a[i]];
31                 base1[j].set(i);
32             }
33             if(dp[j] >= bm){
34                 printf("NO\n");
35                 rep(i,1,n) printf("%d", (int)base1[j][i]);
36                 printf("\n");
37                 return;
38             }
39         }
40     }
41     rep(i,1,n){
42         for(int j = bm-1; j >= 1; j--){
43             if(j == b[i]){
44                 if(tp[j] < a[i]) tp[j] = a[i], base2[j].reset(),base2[j].set(i);

```

```

45     }
46     if(j < b[i]) continue;
47     if(tp[j] < tp[j-b[i]]+a[i] && tp[j-b[i]]!=0){
48         tp[j] = tp[j-b[i]]+a[i];
49         base2[j] = base2[j-b[i]];
50         base2[j].set(i);
51     }
52     if(tp[j] >= am){
53         printf("NO\n");
54         rep(i,1,n) printf("%d", (int)base2[j][i]);
55         return;
56     }
57 }
58 }
59 printf("YES\n");
60 }
61
62 int main()
63 {
64     scanf("%d", &n);
65     scanf("%d", &am);
66     rep(i,1,n) scanf("%d", &a[i]);
67     scanf("%d", &bm);
68     rep(i,1,n) scanf("%d", &b[i]);
69     solve();
70     return 0;
71 }
```

3.2 完全背包

题意：

音符格式转换成长度最小且字典序最小的格式。一共有 7 种音节，分别是 $R1$ 、 $R2$ 、 $R4$ 、 $R8$ 、 $R16$ 、 $R32$ 、 $R64$ ，分别表示 1 、 $\frac{1}{2}$ 、 $\frac{1}{4}$ 、 $\frac{1}{8}$ 、 $\frac{1}{16}$ 、 $\frac{1}{32}$ 、 $\frac{1}{64}$ 拍。对于 $R1R2$ 即为 $\frac{3}{2}$ 拍，也可以表示成 $R1..$ 。即可将相邻的一个音节化为 ‘.’，因此 $R2...$ 表示 $R2R4R16R32$ 。

此题给出一个音符表示格式，要求转化为长度最短且字典序最小的方式。

例如 $R1R4R16$ 转化为 $R16R1R4$ 。

思路：

比赛的时候遇到这题，错误地当成了贪心问题进行考虑，导致始终无法 AC。

现在我们来观察这个问题，这个表达式所有音节的拍数相加之和是不变的，而每一个音节单元的拍数和长度也都是确定的，因此等价于一个完全背包问题。

因此如果此题没有要求给出一个字典序最小的方案，只要求最短长度的话。 $dp[i]$ 表示音符表达式拍数为 i 所需的最短长度，然后直接跑一个完全背包即可。

但是这题还要求求出长度最小时字典序最小，因此我们需要考虑状态方程。我一开始的错误做法是两维 *for* 循环，第一维枚举用到了第 i 个音节单元，第二维枚举当前拍数 j ，然后在转移时若是小于，直接转移，若是等于，则比较当前音节单元字典序与 $dp[j]$ 的最后一个音节单元字典序。这样会 wa，因为无法保证字典序最小。

但是如果换一种枚举方式就可以通过此题，第一维 *for* 循环枚举音节拍数 i ，第二维枚举使用了第 j 个音节单元，然后 == 部分的转移仍比较当前 j 与 $dp[i]$ 的上一个音节单元字典序即可通过此题。原因在于求取 $dp[i]$ 的时候，已经使用了所有的音节单元，而前一种枚举方式并没有用上所有的音节单元所以导致出错。

当然此题也可以字典序倒序枚举前 i 个音节单元，也可以通过此题。

处理完这部分之后，就是完全背包的路径输出了，由于完全背包 $dp[i][j]$ 是由 $dp[i][j - w[i]]$ 转移而来，即同维转移而来，因此不需要需要 01 背包的二维记录路径，直接使用一维进行路径记录，即上一次由哪个音节单元转移而来即可。

总结：

此题考查了完全背包的两个内容，一是完全背包求取过程中加上了对于字典序最小的求解，二是完全背包的一维路径输出。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <string>
5 #include <bitset>
6 #include <algorithm>
7 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
8 #define rep(i,a,b) for(int i = a; i <= b; i++)
9 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
10 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
11 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
12     << " , " << z1 << ":" << z2 << endl;
13 typedef long long ll;
14 typedef double db;
15 const int N = 3200000+10;
16 const int M = 1e5+100;
17 const db EPS = 1e-9;
18 using namespace std;
19
20 char s[N];
21 int len,n,tot,dp[N],pre[N],ttt[N];
22 // int str[N][33];
23 // string str[N];
24 //v: 拍数, w: 长度
25 //len: 总拍数
26 //用最少的长度凑满总拍数
27 //dp[i]: 表示凑满拍数为i所需的最少长度
28 struct Node{
29     int num,sum,cnt,v,w; //num: 编号, sum: .个数, cnt: 选了多少个, v: 拍数, w: 字符长度
30     string ss;
31     bool operator < (Node xx) const {
32         if(num == xx.num) return sum > xx.sum;
33         else return (ss < xx.ss);
34     }
35 }ans[30];
36
37 void solve()
38 {
39     memset(dp,0x3f,sizeof dp);
40     dp[0] = 0;
41     rep(j,1,len) //音符长度
42         rep(i,1,tot){ //枚举所有音符单元
43             if(j >= ans[i].v){
44                 if(dp[j] > dp[j-ans[i].v]+ans[i].w){
45                     dp[j] = dp[j-ans[i].v]+ans[i].w;
46                     pre[j] = i;
47                 }
48                 else if(dp[j] == dp[j-ans[i].v]+ans[i].w){
49                     if(pre[j] > i){ //之前的字典序更大

```

```

50                     dp[j] = dp[j-ans[i].v]+ans[i].w;
51                 }
52             }
53         }
54     }
55 }
56 int tp = len;
57 while(tp > 0){ //完全背包一维记录路径，直接回溯
58     ans[pre[tp]].ctt++;
59     tp = tp-ans[pre[tp]].v;
60 }
61 rep(i,1,tot)
62     rep(j,1,ans[i].ctt)
63         cout << ans[i].ss;
64 cout << endl;
65 }
66
67 int main()
68 {
69     // freopen("e.in","r",stdin);
70     // freopen("e.out","w",stdout);
71     scanf("%s",s+1); n = strlen(s+1);
72     int num;
73     //获取拍数总长度
74     rep(i,1,n){
75         num = 0;
76         if(s[i] == 'R'){
77             i++;
78             while(s[i] != 'R' && i <= n && s[i] != '.')
79             {
80                 num = num*10+s[i]-'0';
81                 i++;
82             }
83         }
84         int tt = 64/num;
85         // LOG2("i",i,"tt",tt);
86         len += tt;
87         if(s[i] == '.'){
88             while(s[i] != 'R' && i <= n){
89                 tt /= 2;
90                 len += tt;
91                 i++;
92             }
93         }
94         i--;
95     }
96     //构造所有音符单元
97     rep(i,1,7){
98         rep(j,0,i-1){
99             int tp = 1<<(i-1);
100            ++tot;
101            ans[tot].v += tp;
102            rep(k,1,j){
103                tp /= 2, ans[tot].v += tp;
104            }
105            ans[tot].w = j+1;
106            if(i <= 3) ans[tot].w += 2;
107            else ans[tot].w += 1;
108            ans[tot].sum = j;

```

```

109         ans[tot].num = 1<<(7-i);
110     }
111 }
112 rep(i,1,tot){
113     ans[i].ss = "\0";
114     string tpp;
115     tpp += 'R';
116     tpp += to_string(ans[i].num);
117     rep(j,1,ans[i].sum) tpp += ".";
118     ans[i].ss = tpp;
119 }
120 sort(ans+1,ans+1+tot);
121 //按字典序构造音符单元 —— 非关键操作，也可直接打表输入
122 Node tp = ans[7];
123 rep(i,8,10) ans[i-1] = ans[i];
124 ans[10] = tp;
125 // rep(i,1,tot)
126     // LOG3("i",i,"ss",ans[i].ss,"v",ans[i].v);
127 solve();
128 return 0;
129 }
```

3.3 多重背包

```

1 #include <cstdio>
2 #include <iostream>
3 #include <algorithm>
4 #include <cstring>
5 #define rep(i,a,b) for(int i = a;i <= b;i++)
6 using namespace std;
7 const int N = 500;
8
9 int v[N],w[N],num[N]; //每袋的价格、重量和袋数
10 int dp[N]; //dp[i]:总经费为i时所能买的最多重量的大米
11 int n,total; //大米种类、总经费
12
13 void ZeroOnePack(int cost,int weight)
14 {
15     for(int i = total; i >= cost;i--)
16         dp[i] = max(dp[i],dp[i-cost]+weight);
17 }
18
19 //完全背包，代价为cost，获得的价值为weight
20 void CompletePack(int cost,int weight)
21 {
22     for(int i = cost;i <= total;i++) //total为上界
23         dp[i] = max(dp[i],dp[i-cost]+weight);
24 }
25
26 //多重背包，代价为cost，获得的价值为weight
27 void MultiplePack(int cost,int weight,int amount)
28 {
29     if(cost*amount >= total) CompletePack(cost,weight); //相当于取任意袋数，变为完全背包
30     else{
31         int k = 1;
32         while(k < amount){
33             ZeroOnePack(k*cost,k*weight); //二进制拆分
34             amount -= k;
```

```

35         k<<=1;
36     }
37     ZeroOnePack(amount*cost,amount*weight); //把剩余个数用01背包求
38     //这个很重要，不要忘记了
39 }
40 }
41
42 int main()
43 {
44     int T;
45     scanf("%d",&T);
46     while(T--)
47     {
48         memset(dp,0,sizeof dp); //memset别忘了，很重要！
49         scanf("%d%d",&total,&n);
50         rep(i,1,n)
51             scanf("%d%d%d",&v[i],&w[i],&num[i]);
52         rep(i,1,n)
53             MultiplePack(v[i],w[i],num[i]);
54         printf("%d\n",dp[total]);
55     }
56     return 0;
57 }
```

3.4 状压 dp

3.4.1 子集枚举

题意：

一共 N 个人，给出任意两个人之间的胜负关系，你的编号是 M 。现在需要安排一棵竞赛树使得 M 能够胜出，问使竞赛树高度最小且 M 获胜的安排方案一共有多少个。 $(1 \leq N \leq 16)$

思路：

根据题意以及数据范围，可以很明显的发现这是一个状压 dp ，因此我们来考虑 dp 的状态。

既然是状态，那肯定要记录当前的状态，即选了哪些人，然后还要记录当前的胜利者，以及当前树的高度，因此 $dp[i][j][k]$ 表示 i 状态下，胜出者为 j ，树高度为 k 的安排方案数。

然后采用记忆化搜索，求取 $dp[i][j][k]$ 时将 i 分为两个子状态 x, y ，然后递归求取 $dp[x][j][k-1]$ ，枚举 m 为 y 状态下的胜利者并且会输给 j ，求取 $dp[y][m][k]$ 。具体细节见代码。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <cmath>
5 #include <algorithm>
6 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
7 #define rep(i,a,b) for(int i = a; i <= b; i++)
8 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
9 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
10 ;#define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
11 << " , " << z1 << ":" << z2 << endl;
12 typedef long long ll;
13 typedef double db;
14 const int N = 1e5+100;
15 const int M = 1e5+100;
```

```

15 const db EPS = 1e-9;
16 using namespace std;
17
18 template<class T> int getbit(T s, int i) { return (s >> i) & 1; }
19 template<class T> T onbit(T s, int i) { return s | (T(1) << i); }
20 template<class T> T offbit(T s, int i) { return s & (~(T(1) << i)); }
21 template<class T> int cntbit(T s) { return __builtin_popcount(s); }
22
23 int c[20][20],n,win,h[20];
24 ll dp[1<<16][17][7];
25
26 ll solve(int stat,int m,int height){
27     if(dp[stat][m][height] != -1) return dp[stat][m][height];
28     if(cntbit(stat) == 1){
29         if(getbit(stat,m-1)) return dp[stat][m][height] = 1ll;
30         else return dp[stat][m][height] = 0;
31     }
32     for(int i = (stat-1)&stat; i >= 1; i = (i-1)&(stat)){ //枚举子集
33         int x = i, y = stat-i;
34         if(!getbit(x,m-1) || h[cntbit(x)] > height-1 || h[cntbit(y)] > height-1)
35             continue;
36         // if(height-1 > cntbit(x) || height-1 > cntbit(y)) continue;
37         ll ans1 = solve(x,m,height-1);
38         ll ans2 = 0;
39         rep(j,1,n)
40             if(getbit(y,j-1) && c[m][j] == 1) ans2 += solve(y,j,height-1);
41         if(dp[stat][m][height] == -1) dp[stat][m][height] = ans1*ans2;
42         else dp[stat][m][height] += ans1*ans2;
43     }
44     return max(0ll,dp[stat][m][height]);
45 }
46 int main()
47 {
48     freopen("f.in","r",stdin);
49     freopen("f.out","w",stdout);
50     scanf("%d%d",&n,&win);
51     rep(i,1,n)
52         rep(j,1,n) scanf("%d",&c[i][j]);
53     rep(i,1,16) h[i] = (log(i-0.5)/log(2))+2;
54     memset(dp,-1,sizeof dp);
55     printf("%lld\n",max(0ll,solve((1<<n)-1,win,h[n])));
56     return 0;
57 }

```

3.4.2 计算一对元素贡献

题意：

有一个 n 个数字的序列，数字范围在 1 ~ 20 之间，现在要对这些数字进行重新排列，使得相同数字出现在一个连续区间，如 113322、2222233444 都是符合题意的序列。重新排列这个序列的操作只有交换两个相邻的数字，现问最少需要交换多少次可以使这个序列变成符合题意的序列。 $(2 \leq n \leq 4 * 10^5)$

思路：

由于所有数字都在 1 ~ 20 之间，因此应该对于有多少个数字进行考虑，而不是对 n 进行考虑。

如果直接考虑数字的移动会将问题复杂化，因为考虑的内容会非常多。因此我们直接构建一个最终序列，然后求原序列转化到这个最终序列的贡献。

先说如何求，再说如何构建。因为如果不会求的话，也不知道构建的入手点在哪。对于求贡献，此处主要对于一对元素进行考虑。

对一对元素进行考虑，即仅考虑 1、2 时，序列为 1 1 2 1 1 2 1 2 2，我们令 $val[i][j]$ 表示元素 i 在元素 j 之前对答案的影响，因此只需要考虑对每个 1 来说前面有多少个 2，即 $val[1][2] = 1 + 1 + 2 = 4$, $val[2][1] = 2 + 4 + 5 + 5 = 16$ 。

知道如何求贡献之后，我们来考虑如何解决这个问题。令 $dp[i][S]$ 表示当前放置第 i 个元素，前面放的元素状压的状态为 S ，因此我们枚举第 i 个元素为 j , $dp[i][S] = \min(dp[i-1][S \wedge (1 \ll j)])$ 。计算贡献时可以使用一对元素的考虑方法，需要进行预处理。

总结：

这其实是一道计算贡献的状压 dp 问题，主要难点应该在于从考虑过程转变为考虑结果，然后再思考如何根据初始序列和结果序列直接得到答案。再之后就是直接通过状压再加上一定的预处理通过此题。

考虑结果序列的贡献中，一对元素的考虑方式比较巧妙，类似于逆序对的感觉，主要抓住的特点在于一个元素在序列中的位置本质上是由它前面有多少个元素决定的，因此想到了一对元素的思考方式来解决问题。

```

1 #include <bits/stdc++.h>
2 #define mem(a,b) memset(a,b,sizeof a);
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define per(i,a,b) for(int i = a; i >= b; i--)
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 typedef long long ll;
7 typedef double db;
8 const int N = 4e5+100;
9 const ll inf = 1e15;
10 const db EPS = 1e-9;
11 using namespace std;
12
13 void dbg() {cout << "\n";}
14 template<typename T, typename... A> void dbg(T a, A... x) {cout << a << ' ' ; dbg(x...)}
15 #define logs(x...) {cout << #x << " -> "; dbg(x);}
16
17 int n,a[N],b[N],tot,num[25],cnt[1<<20];
18 ll dp[22][1<<20],val[25][25];
19
20 int main()
21 {
22     rep(S,0,(1<<20)-1){
23         int hp = S, c1 = 0;
24         while(hp){
25             if(hp&1) c1++;
26             hp /= 2;
27         }
28         cnt[S] = c1;
29     }
30     scanf("%d",&n);
31     rep(i,1,n){
32         scanf("%d",&a[i]);
33         b[++tot] = a[i];
34     }
35     sort(b+1,b+1+tot);
36     tot = unique(b+1,b+1+tot)-b-1;
37     rep(i,1,n){
38         num[a[i]]++;
39         int pos = lower_bound(b+1,b+1+tot,a[i])-b;
40         rep(j,1,tot)
41             if(b[j] != a[i]) val[pos][j] += num[b[j]];
42     }
}

```

```

43     rep(i,0,tot)
44         rep(S,0,(1<<tot)-1) dp[i][S] = inf;
45     dp[0][0] = 0;
46     rep(i,1,tot){
47         rep(j,1,tot){ //枚举放的东西
48             rep(S,0,(1<<tot)-1){ //前面的状态
49                 if(S&(1<<(j-1)) && cnt[S] == i){
50                     ll base = dp[i-1][S^(1<<(j-1))];
51                     rep(k,1,tot)
52                         if(S&(1<<(k-1))){
53                             base += val[k][j];
54                         }
55                     dp[i][S] = min(dp[i][S],base);
56                 }
57             }
58         }
59     }
60     printf("%lld\n",dp[tot][(1<<tot)-1]);
61     return 0;
62 }
```

3.5 区间 dp

3.5.1 区间 dp 例题 1

题意：

给定一个序列。将序列中的一个数字消去的代价是与这个数字相邻的两个数字的 gcd，问将所有数字消去的最小代价。注意这个序列是环形的。

思路：

首先我们比较容易发现这是一个区间 DP 问题，于是问题就变成了如何列区间 DP 状态。一开始考虑的是 $dp[i][j]$ 表示区间 $[i,j]$ 全部消去的最小代价，然后在区间 $[i,j]$ 中枚举第一个消去的 k 进行更新。然后会发现一个问题，如何先消的是 j ，那么 j 是右端点，因此 j 两端的元素是不确定的，因此这个转移方程不对。

因此我们来重新考虑这道题。由于左右端点不确定，因此我们重新定义 dp 状态， $dp[i][j]$ 表示区间 $[i, j]$ 中所有数全部消除，最后剩下 i 和 j 的最小代价。则在区间中枚举 k ， $dp[i][j] = dp[i][k]+dp[k][j]+\gcd(i,j)$ 。由于是个环形序列，因此需要将长度扩展两倍进行 dp 。最后的答案就是枚举最后剩下的两个点，然后找最小值即可。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define rep(i,a,b) for(int i = a; i <= b; i++)
6 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
7 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
8 ;
9 typedef long long ll;
10 typedef double db;
11 const db EPS = 1e-9;
12 using namespace std;
13 const int N = 300;
14
15 int n,a[N],dp[N][N];
16
17 int gcd(int a,int b)
18 {
19     return b == 0 ? a:gcd(b,a%b);
20 }
```

```

20
21 int main()
22 {
23     while(~scanf("%d",&n))
24     {
25         if(n == 0) break;
26         rep(i,1,n) scanf("%d",&a[i]);
27         rep(i,n+1,2*n) a[i] = a[i-n];
28         rep(len,1,n)
29             rep(j,1,2*n){
30                 int x = j, y = x+len-1;
31                 if(y > 2*n) continue;
32                 if(y == x+1 || y == x) dp[x][y] = 0;
33                 else{
34                     dp[x][y] = 10000;
35                     rep(k,x+1,y-1) dp[x][y] = min(dp[x][y],dp[x][k]+dp[k][y]+gcd(a[x],a
[y]));
36                 }
37                 // printf("dp[%d][%d]:%d\n",x,y,dp[x][y]);
38             }
39         int ans = 10000;
40         rep(i,1,n)
41             rep(j,i+1,i+n-1){
42                 if(ans > dp[i][j]+dp[j][i+n]+gcd(a[i],a[j])){
43                     ans = dp[i][j]+dp[j][i+n]+gcd(a[i],a[j]);
44                     // LOG1("ans",ans);
45                     // LOG2("i",i,"j",j);
46                 }
47             }
48             printf("%d\n",ans);
49     }
50     return 0;
51 }
```

3.5.2 区间 dp 例题 2

题意：

一个序列，选手 A、B 轮流从序列中从左端或者右端选一段区间，然后区间和加到自己的权值中。两个选手都会按照最优的方式进行选取，问先手 A 最多可以比 B 多拿多少。 $(1 \leq n \leq 100)$

思路：

很明显这是一道 DP 问题，又因为只能从左端点或右端点拿，因此不难想到用区间 DP 的方法来解决此题。

既然是区间 DP，那么最常见的状态就是 $DP[i][j]$ 表示对于区间 $[i, j]$ ，先手最多领先后手多少。又因为区间和是一定的，因此已知选手 A 获得的价值就可以知道选手 B 获得的价值，因此修改状态为 $DP[i][j]$ 表示区间 $[i, j]$ ，先手最多可以获得多少价值。

因此 $DP[i][j] = \max(\sum[i][j] - DP[x][j], \sum[i][j] - DP[i][y]) \quad i < x, y < j$ ，由于 n 比较小，直接枚举区间长度，从小区间到大区间进行转移即可。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
```

```

8 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
9 ;
10 typedef long long ll;
11 typedef double db;
12 const db EPS = 1e-9;
13 using namespace std;
14 const int N = 200;
15
16 int n,a[N],sum[N],dp[N][N];
17 //dp[i][j]: 表示区间[i,j]选手能够获得的最大价值
18
19 int main()
20 {
21     while(~scanf("%d",&n))
22     {
23         if(!n) break;
24         memset(dp,0,sizeof dp);
25         rep(i,1,n) scanf("%d",&a[i]);
26         rep(i,1,n) sum[i] = sum[i-1]+a[i];
27         rep(len,1,n)
28             rep(i,1,n-len+1){
29                 int j = i+len-1;
30                 dp[i][j] = sum[j]-sum[i-1];
31                 rep(k,1,len-1){
32                     dp[i][j] = max(sum[j]-sum[i-1]-dp[i+k][j],dp[i][j]);
33                     dp[i][j] = max(sum[j]-sum[i-1]-dp[i][j-k],dp[i][j]);
34                 }
35                 int ans = dp[1][n]-(sum[n]-sum[0]-dp[1][n]);
36                 printf("%d\n",ans);
37             }
38     }
39     return 0;
}

```

3.6 数位 dp

3.6.1 数位 dp 模板

通常来说，数位 dp 问题都是通过 *dfs* 解决的，因为 *dfs* 的做法更容易理解，也能一定程度地简化代码。

对于 *dfs* 求解的数位 dp 问题，其中设置的状态为 $f[pos]...$ ， pos 表示最后 pos 位没有填， $...$ 表示的是从 pos 位之前继承来的信息，然后 $f[pos]...$ 的数值表示仅考虑这 pos 位所对答案产生的贡献。

这 $...$ 所表示的状态通常需要根据题意来进行定义，比较个性化，也是数位 dp 的核心难点。但数位 dp 问题还是非常套路化的，你只需要根据题意想明白想要计算后 pos 位的信息，到底需要从前几位继承哪些信息，想明白这个之后就可以直接套上 *dfs* 的模板进行求解了。

下面的习题给出的都是非套路化问题，具有一定的难度，初学者建议先写一些模板题再来进行挑战。

最后给出 *dfs* 问题的大致模板。(某一模板题的 AC 代码)

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 typedef long long ll;
4 using namespace std;
5
6 ll f[21][21][2010],a[30]; //左-右
7

```

```

8 ll dfs(int pos,int balan,int k,bool flag){
9     //位置 平衡点 左边继承来的数值 有无继承
10    if(pos == 0){
11        if(k == 0) return 1;
12        else return 0;
13    }
14    if(!flag && f[pos][balan][k+1000] != -1) return f[pos][balan][k+1000];
15    ll ans = 0;
16    int end = flag?a[pos]:9;
17    rep(i,0,end){
18        ll tp = dfs(pos-1, balan, k+(pos-baln)*i, flag && i == end);
19        ans += tp;
20    }
21    if(!flag) f[pos][balan][k+1000] = ans;
22    return ans;
23 }
24
25 ll solve(ll n){
26     //求a数组
27     if(n == -1) return 0;
28     int pos = 0;
29     memset(a,0,sizeof a);
30     while(n){
31         a[++pos] = n%10;
32         n /= 10;
33     }
34     ll ans = 0;
35     rep(i,1,pos) ans += dfs(pos,i,0,1); //对每一个平衡点分开求
36     ans -= pos-1;
37     return ans;
38 }
39
40 int main()
41 {
42     //初始化
43     rep(i,0,20)
44         rep(j,0,20)
45             rep(k,0,2000) f[i][j][k] = -1;
46     //读入
47     int _; scanf("%d",&_);
48     while(_--){
49         ll L,R; scanf("%lld%lld",&L,&R); L--;
50         printf("%lld\n",solve(R)-solve(L));
51     }
52     return 0;
53 }
```

3.6.2 巧妙数位 dp 例题

题意：

给定 k 与 b , 求出所有 k 在 $0(2^b-1)$ 范围内的倍数, 将这些倍数二进制中的 1 求 sum 和, 模 $1e9+9$ 输出。

思路：

首先, 这是一个在数位上的 dp, 重点就在于如何描述每个数的状态。

发现数的范围很大, 想要直接描述是不可能的。但是 k 的范围很小, 只有 1000, 因此考虑存储这个数

然后就可以列出 dp 方程, $dp[i][j]$ 表示前 i 个二进制位, $mod k = j$ 的个数, 再用 $ans[i][j]$ 表示前 i 个二进制位, $mod k = j$ 的每一种情况二进制拆分后 1 的总和。

因为在 mod 意义下的加减都是可以的。因此对于第 i 个位置，我们只需考虑此处为 0 还是 1，只有两个状态，然后就可以列出转移方程。

$$\begin{aligned} dp[i][j] &= dp[i-1][j] + dp[i-1][(j-poww[i]+k)] \\ ans[i][j] &= ans[i-1][j] + ans[i-1][(j-poww[i]+k)] \end{aligned}$$

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define rep(i,a,b) for(int i = a; i <= b; i++)
6 using namespace std;
7 const int mod = 1e9+9;
8
9 int k,b,dp[150][1100],poww[200],ans[150][1100];
10
11 int main()
12 {
13     scanf("%d%d",&k,&b);
14     poww[1] = 1%k;
15     rep(i,2,130) poww[i] = (poww[i-1]*2)%k;
16     dp[1][0] = 1, dp[1][1] = 1;
17     ans[1][0] = 0, ans[1][1] = 1;
18     if(k == 1) dp[1][0] = 2, dp[1][1] = 0, ans[1][0] = 1, ans[1][1] = 0;
19     rep(i,2,b)
20         rep(j,0,k-1){
21             dp[i][j] = (dp[i-1][j]+dp[i-1][(j-poww[i]+k)%k])%mod;
22             ans[i][j] = ((ans[i-1][j]+ans[i-1][(j-poww[i]+k)%k])%mod+dp[i-1][(j-poww[i]+k)%k])%mod;
23         }
24     printf("%d\n",ans[b][0]);
25     return 0;
26 }
```

3.6.3 双数 dp 问题

题意:

给出区间 $[l, r]$ ，查询多少对 (a, b) 满足如下条件。 $(0 \leq l \leq r \leq 10^9)$

1. $a + b = a \text{ xor } b$
2. $l \leq a \leq r$
3. $l \leq b \leq r$

思路:

CF div2 的最后一题，比赛的时候有 $1h$ 来考虑这个问题。当时主要在思考这个题想要考察的是什么内容，思考过数位 dp ，但是没有做过 *pair* 类型的数位 dp ，于是就没有从这个角度继续往下深入思考。因此剩下的大部分时间都在思考是不是一道结合某些数据结构的思维题，说实话，感觉数据结构开始限制我的思维了，什么题都老从数据结构考虑，这样非常容易被治，必须要改！

继续回到该题，此题其实想要询问的就是 $a \& b = 0$ 的 *pair* 对数。因此我们处理出一个 $solve(x, y)$ 函数，表示 $a \in [0, x], b \in [0, y]$ ，符合条件的 *pair* 对数，所以最终答案就是 $solve(r, r) - solve(l-1, r) - solve(r, l-1) + solve(l-1, l-1)$ 。

接下来就是如何计算 $solve(x, y)$ 函数，其实维护 x 和 y 各自的数组，两个 *flag*，然后套上最基本的 *dfs* 板子，稍微改一下就可以过了。

总结:

此题其实应该是两个数同时数位 dp 的裸题，套上了一个最基本的容斥。而具体的函数实现过程还是比较套路的，并不难思考。

做不出来的原因也主要是没有从数位 dp 这个角度继续往下深挖，是自己思考的片面性错失了 AC。

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 typedef long long ll;
5 typedef double db;
6 const int N = 1e5+100;
7 const int M = 1e5+100;
8 const db EPS = 1e-9;
9 using namespace std;
10
11 void dbg() {cout << "\n";}
12 template<typename T, typename... A> void dbg(T a, A... x) {cout << a << ' '; dbg(x...)}
13 #define logs(x...) {cout << #x << " -> "; dbg(x);}
14
15 ll f[40],a[40],b[40]; //左-右
16
17 ll dfs(int pos,bool flag1,bool flag2){
18     if(pos == 0) return 1;
19     if(!flag1 && !flag2 && f[pos] != -1) return f[pos];
20     ll ans = 0;
21     int end1 = flag1?a[pos]:1;
22     int end2 = flag2?b[pos]:1;
23     if(!flag1 && !flag2){
24         ans += 3ll*dfs(pos-1,0,0);
25     }
26     else if(!flag1){
27         rep(i,0,end2){
28             if(i == 0) ans += 2ll*dfs(pos-1,0,flag2 && i == end2);
29             else ans += dfs(pos-1,0,flag2 && i == end2);
30         }
31     }
32     else if(!flag2){
33         rep(i,0,end1){
34             if(i == 0) ans += 2ll*dfs(pos-1,flag1 && i == end1,0);
35             else ans += dfs(pos-1,flag1 && i == end1,0);
36         }
37     }
38     else{
39         rep(i,0,end1){
40             rep(j,0,end2){
41                 if(i != 1 || j != 1) ans += dfs(pos-1,flag1 && i == end1,flag2 && j == end2);
42             }
43         }
44     }
45     if(!flag1 && !flag2) f[pos] = ans;
46     return ans;
47 }
48
49 ll solve(ll x,ll y){
50     if(x == -1 || y == -1) return 0;
51     int p1 = 0, p2 = 0;
52     memset(a,0,sizeof a);
53     memset(b,0,sizeof b);
54     while(x){

```

```

55         a[++p1] = x%2;
56         x /= 2;
57     }
58     while(y){
59         b[++p2] = y%2;
60         y /= 2;
61     }
62     return dfs(max(p1,p2),1,1);
63 }
64
65 int main()
66 {
67     int _; scanf("%d",&_);
68     memset(f,-1,sizeof f);
69     while(_--){
70         ll L,R; scanf("%lld%lld",&L,&R);
71         ll ans = solve(R,R)-solve(L-1,R)-solve(R,L-1)+solve(L-1,L-1);
72         printf("%lld\n",ans);
73     }
74     return 0;
75 }
```

3.6.4 三数 dp 问题

题意:

给出四个数, $L R A B$, 表示 $x \in [L, R]$, $y \in [0, A]$, $z \in [0, B]$, 求 $(x \wedge y) + (y \wedge z) + (z \wedge x)$ 的最大值。 $(0 \leq L \leq R \leq 10^{18}, 0 \leq A, B \leq 10^{18})$

思路:

这个问题考察的是涉及三个数字的数位 dp, 首先我们来思考一下如何表示状态。

此题是求取最大值, 因此很明显需要从数位的角度入手进行考虑, 所以状态的设置一定会包含各个数位的状态, 我们另 $f[pos][p1][p2][p3][p4]$ 表示仅考虑最后 pos 位, 从前面转移来的状态为 $p1 p2 p3 p4$ 时答案的最大值。

1. $p1$ 为 1 表示比 L 大
2. $p2$ 为 1 表示比 R 小
3. $p3$ 为 1 表示比 A 小
4. $p4$ 为 1 表示比 B 小

然后在 dfs 的时候枚举 pos 位放的值, 然后更新答案即可。代码中将每个数拆成了二进制进行计算, 因为仅考虑 0 1 可以简化问题。

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
6 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
    << " , " << z1 << ":" << z2 << endl;
7 typedef long long ll;
8 typedef double db;
9 const int N = 1e5+100;
10 const int M = 1e5+100;
11 const db EPS = 1e-9;
12 using namespace std;
13
14 ll f[70][2][2][2][2],a1[70],a2[70],a3[70],a4[70];
```

```

15
16 ll pow_mod(ll a,ll b){
17     ll base = a, ans = 1;
18     while(b){
19         if(b&1) ans *= base;
20         base *= base;
21         b >= 1;
22     }
23     return ans;
24 }
25
26 ll dfs(int pos,bool f1,bool f2,bool f3,bool f4){
27     //比L大，比R小，比A小，比B小
28     if(pos == 0) return 0;
29     ll ans = f[pos][f1][f2][f3][f4];
30     if(ans != -1) return ans;
31
32     int b1 = 0, b2 = 1, d1 = 0, d2 = 1, e1 = 0, e2 = 1;
33     if(!f1 && a1[pos] != 0) b1 = 1;
34     if(!f2 && a2[pos] != 1) b2 = 0;
35     if(!f3 && a3[pos] != 1) d2 = 0;
36     if(!f4 && a4[pos] != 1) e2 = 0;
37     bool k1,k2,k3,k4;
38     rep(i,b1,b2)
39         rep(j,d1,d2)
40             rep(k,e1,e2){
41                 k1 = f1, k2 = f2, k3 = f3, k4 = f4;
42                 ll x = (i^j)+(j&k)+(k^i);
43                 x *= pow_mod(2,pos-1);
44                 // printf("*****\n");
45                 // LOG3("i",i,"j",j,"k",k);
46                 // LOG2("pos",pos,"x",x);
47                 if(i > a1[pos]) k1 = 1;
48                 if(i < a2[pos]) k2 = 1;
49                 if(j < a3[pos]) k3 = 1;
50                 if(k < a4[pos]) k4 = 1;
51                 ans = max(ans,x+dfs(pos-1,k1,k2,k3,k4));
52             }
53     return (f[pos][f1][f2][f3][f4] = ans);
54 }
55
56 ll solve(ll L,ll R,ll A,ll B){
57     memset(a1,0,sizeof a1);
58     memset(a2,0,sizeof a2);
59     memset(a3,0,sizeof a3);
60     memset(a4,0,sizeof a4);
61     int pos = 0;
62     while(L || R || A || B){
63         ++pos;
64         a1[pos] = L & 1;
65         a2[pos] = R & 1;
66         a3[pos] = A & 1;
67         a4[pos] = B & 1;
68         L /= 2; R /= 2; A /= 2; B /= 2;
69     }
70     return dfs(pos,0,0,0,0);
71 }
72
73 int main()

```

```

74 {
75     int _; scanf("%d",&_);
76     while(_--){
77         rep(i,0,65)
78             rep(j,0,1)
79                 rep(k,0,1)
80                     rep(t1,0,1)
81                         rep(t2,0,1)
82                             f[i][j][k][t1][t2] = -1;
83         ll L,R,A,B; scanf("%lld%lld%lld%lld",&L,&R,&A,&B);
84         printf("%lld\n",solve(L,R,A,B));
85     }
86     return 0;
87 }
```

3.6.5 三进制状压

题意:

查找区间 $[l, r]$ 中，符合如下条件的数字的个数：

1. 所有出现过的偶数位，都出现了奇数次
2. 所有出现过的奇数位，都出现了偶数次

$(1 \leq l \leq r \leq 10^{19})$

思路:

这个问题主要的困难之处在于如何表示那些没有出现过的数字，如果用二进制状压的方式显然无法表示那些从未出现过的数字。

因此我们考虑使用三进制状压的方式，0 表示没有出现过，1 表示出现了奇数次，2 表示出现了偶数次。

因此我们设置状态数组 $f[pos][S]$ ，表示还有 pos 个位置没有填数，之前填了的数字继承下来的状态为 S ，符合题干条件的数的个数。

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
6 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
    << " , " << z1 << ":" << z2 << endl;
7 typedef long long ll;
8 typedef double db;
9 const int N = 1e5+100;
10 const int M = 1e5+100;
11 const db EPS = 1e-9;
12 using namespace std;
13
14 ll f[21][60000],a[25],tp[25];
15
16 ll pow_mod(ll a,ll b){
17     ll base = a, ans = 1;
18     while(b){
19         if(b&1) ans *= base;
20         base *= base;
21         b >>= 1;
22     }
23     return ans;
```

```

24 }
25
26 ll dfs(int pos,ll S,bool flag){ //S-各数位出现次数的状压
27     //flag只是表示是否表示了该位的全部情况
28     if(pos == 0){
29         int jud = 1;
30         memset(tp,0,sizeof tp); int tot = -1;
31         while(S){
32             tp[++tot] = S % 3;
33             S /= 3;
34         }
35         rep(i,0,9){
36             if(tp[i] == 0) continue;
37             if(i%2){ //奇数
38                 if(tp[i] != 2) {jud = 0; break;}
39             }
40             else{ //偶数
41                 if(tp[i] != 1) {jud = 0; break;}
42             }
43         }
44         if(jud) return 1;
45         else return 0;
46     }
47     if(!flag && f[pos][S] != -1) return f[pos][S];
48     int end = flag?a[pos]:9;
49     ll ans = 0;
50     rep(i,0,end){
51         if(i == 0 && S == 0) ans += dfs(pos-1,S,flag && i == end); //不能把前面的前导0继承
过来
52         else{
53             int tot = -1, p = 0; ll hp = S;
54             while(hp){
55                 ++tot;
56                 if(tot == i) {p = hp%3; break;}
57                 hp /= 3ll;
58             }
59             hp = S;
60             if(p == 0 || p == 1) hp += pow_mod(3,i);
61             else hp -= pow_mod(3,i);
62             ans += dfs(pos-1,hp,flag && i == end);
63         }
64     }
65     // LOG3("pos",pos,"S",S,"ans",ans);
66     if(!flag) f[pos][S] = ans;
67     return ans;
68 }
69
70 ll solve(ll n){
71     if(n == 0) return 1;
72     int pos = 0;
73     memset(a,0,sizeof a);
74     while(n){
75         a[++pos] = n % 10;
76         n /= 10;
77     }
78     return dfs(pos,0,1);
79 }
80
81 int main()

```

```

82 {
83     ll base = pow_mod(3,10);
84     // LOG1("base",base);
85     rep(i,0,20)
86         rep(j,0,base)
87             f[i][j] = -1;
88     int _; scanf("%d",&_);
89     while(_--){
90         ll L,R; scanf("%lld%lld",&L,&R); L--;
91         printf("%lld\n",solve(R)-solve(L));
92     }
93     return 0;
94 }
95 /*
96 数位DP主要就是数位移动时，对答案贡献的求取
97 在数位移动时，要仔细考虑各个细节点，包括前导0等信息
98 */

```

3.6.6 数字平方和

题意:

寻找区间 $[l, r]$ 中所有与 7 无关的数字的平方和。与 7 有关需要符合下述三个条件之一:

1. 整数中某一位是 7
2. 整数的每一位加起来的和是 7 的整数倍
3. 这个整数是 7 的整数倍
($1 \leq l \leq r \leq 10^{18}$)

思路:

如果这题只是单纯地求个数，那就是一个普通数位 dp 问题，但此题要求的是数字平方和，因此我们需要对每一个数位进行考虑。

当我们枚举 pos 位时，先递归到 $pos - 1$ 位，返回一个结构体 B ，表示若 $pos - 1$ 位前全为空时满足题意的 cnt 、 $sum1$ 、 $sum2$ ，即个数、和、平方和。

设当前结构体为 A ，因此 $A.cnt = A.cnt + B.cnt$, $A.sum1 = A.sum1 + B.sum1 + B.cnt * 10^{pos-1}$, $A.sum2 = A.sum2 + B.sum2 + 10^{pos-1} * 10^{pos-1} + 2 * 10^{pos-1} * B.sum1$ 。

总结:

数位 dp 的主要难点还是在于列出状态，然后采用递归的思想来思考如何根据第 $pos - 1$ 位的答案推出 pos 位的答案。

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
6 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
    << " , " << z1 << ":" << z2 << endl;
7 typedef long long ll;
8 typedef double db;
9 const int N = 1e5+100;
10 const int M = 1e5+100;
11 const ll mod = 1e9+7;
12 const db EPS = 1e-9;
13 using namespace std;
14
15 struct Node{

```

```

16     ll cnt,sum1,sum2; //个数、和、平方和
17 }dp[25][10][10];
18 ll a[30];
19
20 ll pow_mod(ll a, ll b, ll p){
21     ll base = a, ans = 1;
22     while(b){
23         if(b&1) ans = (ans*base)%p;
24         base = (base*base)%p;
25         b >>= 1;
26     }
27     return ans;
28 }
29
30 Node dfs(int pos, int pre1, int pre2, bool flag){ //pre1: 前面数位之和, pre2: 前面数位*贡献
31     //之和
32     //flag = 1, 前面为答案继承而来
33     if(pos == 0){
34         if(pre1 == 0 || pre2 == 0) return {0,0,0};
35         else return {1,0,0};
36     }
37     if(!flag && dp[pos][pre1][pre2].cnt != -1) return dp[pos][pre1][pre2];
38     Node ans = {0,0,0}, tmp;
39     int end = flag?a[pos]:9;
40     rep(i,0,end){
41         if(i == 7) continue;
42         tmp = dfs(pos-1, (pre1+i)%7, (pre2*10ll+i)%7, flag && i == end);
43         ans.cnt = (ans.cnt+tmp.cnt)%mod;
44         ans.sum1 = (ans.sum1+tmp.sum1+(tmp.cnt*(ll)i)%mod*pow_mod(10,pos-1,mod)%mod)%mod;
45         ans.sum2 = (ans.sum2+tmp.sum2+(2ll*tmp.sum1*(ll)i)%mod*pow_mod(10,pos-1,mod)%mod)%mod;
46         ans.sum2 = (ans.sum2+(ll)i*(ll)i*pow_mod(10,pos-1,mod)%mod*pow_mod(10,pos-1,mod)%mod*tmp.cnt%mod)%mod;
47     }
48     if(!flag) dp[pos][pre1][pre2] = ans;
49     return ans;
50 }
51
52 ll solve(ll n){
53     int pos = 0;
54     memset(a,0,sizeof a);
55     while(n){
56         a[++pos] = n%(10ll);
57         n /= 10ll;
58     }
59     return dfs(pos,0,0,1).sum2;
60 }
61
62 int main()
63 {
64     rep(i,0,20)
65         rep(j,0,6)
66             rep(k,0,6) dp[i][j][k].cnt = -1;
67     int _; scanf("%d",&_);
68     while(_--){
69         ll L,R; scanf("%lld%lld",&L,&R); L--;
70         printf("%lld\n", (solve(R)-solve(L)+mod)%mod);
71     }
72 }
```

```

71     return 0;
72 }
```

3.6.7 被数位整除

题意:

一个正整数被称为漂亮数, 当且仅当其能够被其所有非零数位整除, 现需要求 $[l, r]$ 中漂亮数的个数。 $(1 \leq l \leq r \leq 9 \times 10^{18})$

思路:

整除所有非零数位, 即整除非零数位的 LCM 。因此我们需要在 dfs 的过程中, 记录出现数位的 LCM , 以及记录当前数字的大小。

这里会出现一个问题, 即当前数字非常大, 直接记录不可行, 因此我们需要将当前数字对 2520 取模, 即 19 所有数的 LCM 。思考到这一步, 剩下的就是一些代码细节了, 可以参考一下下述代码。

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
6 ;
6 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
7 << " , " << z1 << ":" << z2 << endl;
7 typedef long long ll;
8 typedef double db;
9 const int N = 1e5+100;
10 const int M = 1e5+100;
11 const db EPS = 1e-9;
12 using namespace std;
13
14 ll a[30],b[3000],tot,f[20][55][3000];
15
16 ll gcd(ll a,ll b){
17     return b == 0 ? a:gcd(b,a%b);
18 }
19
20 int find(ll x){
21     return lower_bound(b+1,b+1+tot,x)-b;
22 }
23
24 ll dfs(int pos,int lcm,ll m,bool flag){
25     if(pos == 0){
26         if(m%b[lcm] == 0) return 1;
27         else return 0;
28     }
29     if(!flag && f[pos][lcm][m] != -1) return f[pos][lcm][m];
30     int end = flag?a[pos]:9;
31     ll ans = 0;
32     rep(i,0,end){
33         if(i == 0) ans += dfs(pos-1,lcm,(m*10ll+i)%2520ll),flag && i == end;
34         else{
35             ll tp = gcd(b[lcm],i);
36             ll hp = b[lcm]*i/tp;
37             int xp = find(hp);
38             ans += dfs(pos-1,xp,(m*10ll+i)%2520ll),flag && i == end;
39         }
40     }
}
```

```

41     if(!flag) f[pos][lcm][m] = ans;
42     return ans;
43 }
44
45 ll solve(ll n){
46     if(n == 0) return 1;
47     int pos = 0;
48     memset(a, 0, sizeof a);
49     while(n){
50         a[++pos] = n % 10;
51         n /= 10;
52     }
53     return dfs(pos, 1, 0, 1);
54 }
55
56 int main()
57 {
58     rep(i, 1, ((1 << 9) - 1)){
59         ll ans = 1;
60         rep(j, 1, 9){
61             if(i & (1 << (j - 1))){
62                 ll tp = gcd(ans, j);
63                 ans = ans * j / tp;
64             }
65         }
66         b[++tot] = ans;
67     }
68     sort(b + 1, b + 1 + tot);
69     tot = unique(b + 1, b + 1 + tot) - b - 1;
70     rep(i, 0, 19)
71         rep(j, 0, 50)
72             rep(k, 0, 2900) f[i][j][k] = -1;
73     int _; scanf("%d", &_);
74     while(_--){
75         ll L, R; scanf("%lld%lld", &L, &R); L--;
76         printf("%lld\n", solve(R) - solve(L));
77     }
78     return 0;
79 }

```

3.6.8 数位逆序对之和

题意:

给出区间 $[l, r]$, 求所有数字中数位逆序对之和。 $(1 \leq l \leq r \leq 10^8)$

思路:

这是一个涉及到组合计数的数位 dp 问题, 初次看到肯定觉得很棘手, 但是我们可以将这个问题不断地进行分解。

我们先考虑 pos 个位置, 每个位置可以填 $[0, 9]$ 时, 所有可能的数的数位逆序对之和。这个问题不难处理, 直接从 pos 中取两个位置组成逆序队, 其它位置任意取, 答案为 $ans1(pos) = C(pos, 2) * 45 * 10^{pos-2}$ 。

然后再考虑如果 pos 个位置, 第一个位置不能为 0 时的答案。我们先计算第一个位置和之后位置的贡献, 再计算后面 $pos - 1$ 位置产生的贡献。因此 $ans2(pos) = 36 * C(pos - 1, 1) * 10^{pos-2} + 9 * ans1(pos - 1)$ 。

处理完这两个子问题之后, 我们直接按照数位 dp 的套路 dfs 求解即可。如果不是很清楚, 可以查看代码进行进一步了解。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 typedef long long ll;
4 using namespace std;
5
6 void dbg() {cout << "\n";}
7 template<typename T, typename... A> void dbg(T a, A... x) {cout << a << ' ' ; dbg(x...)}
8 #define logs(x...) {cout << #x << " -> "; dbg(x);}
9
10 ll a[30],POS;
11
12 //快速幂
13 ll calc(ll a,ll b){
14     if(b < 0) return 0;
15     ll ans = 1, base = a;
16     while(b){
17         if(b&1) ans *= base;
18         base *= base;
19         b >= 1;
20     }
21     return ans;
22 }
23
24 //数字长度为pos且第一个数字不为0
25 ll calc1(ll pos){
26     ll ans = 0;
27     ans += 36ll*(pos-1ll)*calc(10,pos-2); //第一个数不为0, 后续数字与其组成的贡献
28     ans += 9ll*(pos-2ll)*(pos-1ll)*45ll*calc(10,pos-3)/2ll; //除第一个数字之外的贡献
29     return ans;
30 }
31
32 //数字长度为pos且第一个数字可以为0
33 ll calc2(ll pos){
34     return pos*(pos-1ll)*45ll*calc(10,pos-2)/2ll;
35 }
36
37 ll dfs(int pos,int *base,bool flag,ll hp){
38     ll ans = 0;
39     if(pos == 0){
40         return hp;
41     }
42     if(!flag){
43         ans = calc2(pos);
44         rep(i,0,8) ans += (ll)base[i]*(ll)pos*(ll)(9-i)*calc(10,pos-1);
45         ans += hp*calc(10,pos);
46         return ans;
47     }
48     int end = a[pos];
49     ll tmp = 0;
50     rep(i,0,end){
51         if(POS == pos && i == 0) continue; //保证起始位不为0
52         if(i > 0) tmp += (ll)base[i-1];
53         base[i]++;
54         ans += dfs(pos-1,base,flag && i == end,hp+tmp);
55         base[i]--;
56     }
57     return ans;
58 }

```

```

59
60 ll solve(ll n){
61     if(n <= 9) return 0;
62     POS = 0; memset(a,0,sizeof a);
63     while(n){
64         a[++POS] = n%10ll;
65         n /= 10ll;
66     }
67     int base[11];
68     rep(i,0,9) base[i] = 0;
69     ll ans = dfs(POS,base,1,0);
70     rep(i,2,POS-1) ans += calc1(i);
71     return ans;
72 }
73
74 int main(){
75     int _; scanf("%d",&_);
76     rep(Ca,1,_){
77         ll x,y; scanf("%lld%lld",&x,&y);
78         x--;
79         printf("Case %d: %lld\n",Ca,solve(y)-solve(x));
80     }
81 }
```

3.7 概率 dp

题意:

给出一个有向无环图，从 1 号点向 n 号点前进。每次等概率地转移到后继节点，也有可能留在当前节点。定义每一天的花费是当前走过的天数，即第 i 天的花费是 i ，求从起点走到终点的总花费。 $(2 \leq n \leq 10^5, 1 \leq m \leq 2 * 10^5)$

思路:

令 $day[i]$ 表示从第 i 个点到达第 n 个点的期望天数， $cost[i]$ 表示从第 i 个点到达第 n 个点的期望花费， $deg[i]$ 表示第 i 个点的出度。

$$day[i] = \sum_{j \in deg[i]} \frac{day[j]}{deg[i]+1} + \frac{day[i]}{deg[i]+1} + 1$$

$$day[i] = \sum_{j \in deg[i]} \frac{day[j]}{deg[i]} + \frac{deg[i]+1}{deg[i]}$$

然后再来求取 $cost[i]$ 。

$$cost[i] = \sum_{j \in deg[i]} \frac{cost[j]}{deg[i]+1} + \frac{cost[i]}{deg[i]+1} + day[i]$$

$$cost[i] = \sum_{j \in deg[i]} \frac{cost[j]}{deg[i]} + \frac{deg[i]*(deg[i]+1)}{deg[i]}$$

为什么求 $cost[i]$ 的最后要加上一个 $day[i]$ ，因为当前之后还有 $day[i] - 1$ 天，每天的期望要加上一个 1，即当前走的这一步。因此 $day[i] - 1 + 1 = day[i]$ 。

接下来就是把边反向跑拓扑序转移即可。

总结:

概率 dp 题目，主要难点在于考虑清所有的情况，精准地列出式子。

有时最终的式子也有可能由多个式子组成，需要求出多个变量，需要对问题进行一定的拆分。

```

1 #include <bits/stdc++.h>
2 #define mem(a,b) memset(a,b,sizeof a);
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
```

```

4 #define per(i,a,b) for(int i = a; i >= b; i--)
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 typedef long long ll;
7 typedef double db;
8 const int N = 4e5+100;
9 const int M = 5e5+100;
10 const db EPS = 1e-9;
11 using namespace std;
12
13 void dbg() {cout << "\n";}
14 template<typename T, typename... A> void dbg(T a, A... x) {cout << a << ' '; dbg(x...)}
15 ;}
16
17 #define logs(x...) {cout << #x << " -> "; dbg(x);}
18
19 int n,m,tot,head[N],deg[N],base[N];
20 db day[N],cost[N];
21 struct Node{
22     int to,next;
23 }e[M];
24
25 void add(int x,int y){
26     e[++tot].to = y, e[tot].next = head[x], head[x] = tot;
27 }
28
29 void init(){
30     tot = 1;
31     rep(i,0,n){
32         head[i] = deg[i] = 0;
33         day[i] = cost[i] = 0;
34     }
35 }
36
37 void solve(){
38     queue<int> q;
39     while(q.size()) q.pop();
40     rep(i,1,n) base[i] = deg[i];
41     rep(i,1,n)
42         if(!deg[i]) q.push(i);
43     while(q.size()){
44         int x = q.front(); q.pop();
45         if(base[x] != 0){
46             day[x] += (db)(base[x]+1)/(db)base[x];
47             cost[x] += ((db)day[x]*(db)(base[x]+1))/(db)base[x];
48         }
49         for(int i = head[x]; i; i = e[i].next){
50             int y = e[i].to;
51             deg[y]--;
52             day[y] += (db)day[x]/(db)base[y];
53             cost[y] += (db)cost[x]/(db)base[y];
54             if(!deg[y]) q.push(y);
55         }
56     }
57     printf("%.2f\n",cost[1]);
58 }
59
60 int main()
61 {
62     int _; scanf("%d",&_);
63     while(_--){

```

```

62     scanf("%d%d", &n, &m);
63     init();
64     rep(i, 1, m) {
65         int u, v; scanf("%d%d", &u, &v);
66         deg[u]++;
67     }
68     solve();
69 }
70 return 0;
71 }
```

3.8 斜率优化 dp

3.8.1 斜率优化例题

题意：

n 个需要被处理的任务，机器启动时间为 s。每个任务都有时间 T_i 和花费 C_i ，计算方法为完成这批任务所需的时间是各个任务需要时间的总和。注意，同一批任务将在同一时刻完成，新的一批任务开始时，机器需要重新启动。确定一个分组方案，使得总费用最小。【每批任务包含相邻的若干任务】

思路：

$$1 \leq n \leq 3 \times 10^5, 0 \leq S, C_i \leq 512, -512 \leq T_i \leq 512.$$

首先我们需要列出 dp 方程， $dp[i]$ 表示完成前 i 个任务的最小费用。此处由于涉及到 s ，因此我们需要用到一个叫做“费用提前计算”的思想。因为 s 对于后续的每个任务都有影响，因此我们应当将 s 对后续任务的影响提前计算。

假设 $dp[i]$ 由 $dp[j]$ 更新而来，则 $dp[i] = dp[j] + sumT[i]*(sumC[i]-sumC[j]) + s*(sumC[N]-sumC[j])$ ；将转移方程拆开，则可以得到 $dp[j] - s*sumC[j] = sumT[i]*sumC[j] + dp[i] - s*sumC[N] - sumT[i]*sumC[i]$ ，由此我们可以发现令横坐标 $x = sumC[j]$ ，纵坐标 $y = dp[j] - s*sumC[j]$ ，则对于每一个 j ，都可以在平面中确定一个 (x, y) 坐标，而直线的斜率也是固定的，为 $sumT[i]$ 。

因此在平面中的这么多点中，我们需要维护一个下凸壳，更新 $dp[i]$ 的时候，只需要在下凸壳中，二分 x 点与 $x+1$ 点的斜率是否小于 $sumT[i]$ ，找到一个点，该点左边的斜率小于 k ，右边斜率大于 k ，则该点即为该下凸壳中的最优点。

再谈一下如何维护下凸壳，用一个数组，末尾为 qt ，则判断 $qt-1$ 与 x 的斜率是否小于等于 $qt-1$ 与 qt 的斜率，如果小于等于，则需要弹出 qt ，然后不断继续往下更新。注意此处如果是等于也需要弹出。

本题在维护凸壳时，可能会爆 long long，需要转成 long double。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define rep(i,a,b) for(int i = a; i <= b; i++)
6 using namespace std;
7 const int N = 1e6+100;
8 typedef long long ll;
9
10 int n, qt;
11 ll s, T[N], C[N], sumT[N], sumC[N], dp[N], q[N], xx[N], yy[N];
12
13 void calc(int x) //在凸壳中二分最优点，斜率优化
14 {
15     int l = 1, r = qt-1, ans = q[l];
16     ll k = sumT[x];
17     while(l <= r)
18     {
```

```

19         int mid = (l+r)>>1;
20         if((long double)(yy[q[mid+1]]-yy[q[mid]]) <= (long double)(xx[q[mid+1]]-xx[q[
21             mid]])*(long double)k)
22             ans = q[mid+1], l = mid+1;
23         else r = mid-1;
24     }
25     dp[x] = yy[ans]-k*xx[ans]+s*sumC[n]+sumT[x]*sumC[x];
26 }
27 void solve()
28 {
29     qt = 1, q[1] = 0;
30     xx[0] = yy[0] = 0;
31     rep(i,0,n) xx[i] = 0, yy[i] = 0;
32     rep(i,1,n)
33     {
34         calc(i);
35         xx[i] = sumC[i], yy[i] = dp[i]-s*sumC[i];
36         while(qt >= 2){
37             //重点在于 >=, > 会wa
38             if((long double)(yy[q[qt]]-yy[q[qt-1]])*(long double)(xx[i]-xx[q[qt-1]]) >=
39                 (long double)(yy[i]-yy[q[qt-1]])*(long double)(xx[q[qt]]-xx[q[qt-1]]))
40                 qt--;
41             else break;
42         }
43         qt++; q[qt] = i;
44     }
45     printf("%lld\n",dp[n]);
46 }
47 int main()
48 {
49     while(~scanf("%d%lld",&n,&s))
50     {
51         sumT[0] = sumC[0] = 0;
52         rep(i,1,n){
53             scanf("%lld%lld",&T[i],&C[i]);
54             sumT[i] = sumT[i-1]+T[i];
55             sumC[i] = sumC[i-1]+C[i];
56         }
57         solve();
58     }
59     return 0;
60 }
```

3.8.2 动态维护下凸壳

题意：

给定一颗树，单向边，给出每个点的价值。然后任选树中一个点 i 进入，从 j 点出来，获得的价值为 $\text{val}[i]*1+\text{val}[i+1]*2+\dots+\text{val}[j]*(j-i+1)$ ，也可以选择不进入，节点价值有正有负，求最多可以获得多少价值。

思路：

这题可以简化成在一个序列上，找出一段 $[x,y]$ ，求 $\text{SUM}(\text{val}[i]^{*(i-x+1)})$ 的最大值，即 $1*\text{val}[1]+2*\text{val}[2]+3*\text{val}[3]+\dots$ 的问题。

我们可以列一下 dp 方程， $dp[i]$ 表示以 i 为右端点的区间和最大值，假设 $dp[i]$ 的最优解是 $[j+1, i]$ 这一段，则 $dp[i] = f[i]-f[j]-(\text{sum}[i]-\text{sum}[j])*j$ ， $\text{sum}[i]$ 表示从根节点到 i 路径上所有点的权值之和， $f[i]$ 表示从根节点到 i 路径上所有点按照 $1*\text{val}[1]+2*\text{val}[2]+3*\text{val}[3]$ 的方式得到的累加和。

将 dp 方程拆成斜率优化的形式，可以得到 $f[j]-j*sum[j] = -sum[i]*j+f[i]-dp[i]$ ，而 $dp[i]$ 就是答案。因此令横坐标为 j ，纵坐标为 $f[j]-j*sum[j]$ ，在图中标出这些点，维护一个下凸壳，然后在下凸壳中二分左边斜率最接近 $-sum[i]$ 的点即为最优点。因此问题变成了如何维护下凸壳，正常的维护方法是用单调栈，添加一个点之后就将其他的点一一弹出，直到该点到达合理位置。但由于现在是在树上维护凸壳，因此下凸壳的序列会不断变化，因此我们需要进行动态维护。

可以发现每次往下凸壳中加入一个点，这个点最终都会到达原凸壳中的某一个位置，因此每次添加一个点，只是 $O(1)$ 修改，然后我们在修改完之后再改回来，然后再维护原来的凸壳长度，即可实现动态维护。如何求该点最终到达的位置，只需要进行二分，找到点 x 与 $x+1$ 之间斜率最接近 x 与新点之间斜率的点 x ， $x+1$ 处即为新点最终的更新位置。因此本题就是两个二分 +dfs 结束。

```

1 #include <csdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define rep(i,a,b) for(int i = a; i <= b; i++)
6 using namespace std;
7 typedef long long ll;
8 const int N = 1e5+1000;
9
10 int n,q[N],fa[N],head[N],tot,qh,qt;
11 ll val[N],ans,x0[N],y0[N];
12 struct Edge{
13     int to,next;
14 }e[N];
15
16 void init()
17 {
18     tot = 1;
19     rep(i,0,n) head[i] = 0;
20 }
21
22 void add(int x,int y)
23 {
24     e[++tot].to = y, e[tot].next = head[x], head[x] = tot;
25 }
26
27 bool jud(int a,int b,int c)
28 {
29     return ((y0[a]-y0[b])*(x0[c]-x0[b]) > (y0[c]-y0[b])*(x0[a]-x0[b]));
30 }
31
32 int calc1(int l,int r,int x) //维护凸壳
33 {
34     int tp = r+1; r = r-1;
35     while(l <= r)
36     {
37         int mid = (l+r)>>1;
38         if(jud(q[mid+1],q[mid],x)) tp = mid+1, r = mid-1;
39         else l = mid+1;
40     }
41     return tp;
42 }
43
44 void calc2(int l,int r,ll k,ll f) //斜率优化dp, 找到第一个右边斜率比k大的点
45 {
46     int tp = q[l]; r--;
47     while(l <= r)
48     {

```

```

49         int mid = (l+r)>>1;
50         if(y0[q[mid+1]]-y0[q[mid]] < k*(x0[q[mid+1]]-x0[q[mid]])) tp = q[mid+1], l =
51             mid+1;
52         else r = mid-1;
53     }
54     ans = max(ans,-y0[tp]+k*x0[tp]+f);
55 }
56 void dfs(int x,int dep,ll sum,ll f) //从1号点往下dfs
57 {
58     x0[x] = dep, y0[x] = f-dep*sum;
59     calc2(qh,qt,-sum,f);
60     int oldh = qh, oldt = qt;
61     int pos = calc1(qh,qt,x), old = q[pos]; q[pos] = x, qt = pos;
62     for(int i = head[x]; i; i = e[i].next)
63     {
64         int y = e[i].to;
65         dfs(y,dep+1,sum+val[y],f+(dep+1)*val[y]);
66     }
67     qh = oldh, qt = oldt, q[pos] = old;
68 }
69
70 int main()
71 {
72     int T; scanf("%d",&T);
73     while(T--)
74     {
75         init();
76         ans = 0;
77         scanf("%d",&n);
78         rep(i,1,n) scanf("%lld",&val[i]);
79         rep(i,2,n) scanf("%d",&fa[i]), add(fa[i],i);
80         qh = qt = 1; q[1] = 0;
81         dfs(1,1,val[1],val[1]);
82         printf("%lld\n",ans);
83     }
84     return 0;
85 }
```

3.9 斯坦纳树

3.9.1 斯坦纳树模板

适用问题:一个图中,有若干个关键点,将这几个关键点连在一起的最小花费。直观的理解,就是带关键节点的最小生成树。

时间复杂度: $O(n * 3^k + c * |E| * 2^k)$ 。 n 为点数, $|E|$ 为边数, k 为关键点数, c 为 $spfa$ 常数, 前一部分为子集枚举的复杂度, 第二部分为枚举边松弛的复杂度。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 const int N = 30+5;
4 const int K = 10;
5 const int inf = 1e8;
6 using namespace std;
7
8 int dp[N][1<<K], st[N], endS;
9 bool vis[N][1<<K];
10 queue<int> q;
```

```

11 //dp[i][state]: 以i为根, 关键点在当前斯坦纳树中的连通状态为state的最小花费。
12
13 //初始化函数
14 void init(){
15     //定义末状态
16     endS = (1<<K)-1;
17
18     //初始化
19     rep(i,0,n)
20         rep(S,0,endS) dp[i][S] = inf, vis[i][S] = 0;
21
22     //求每个点的状态, 假设前k个点为特殊点
23     rep(i,1,k) st[i] = 1<<(i-1);
24
25     //dp函数关键点赋初值
26     rep(i,1,k) dp[i][st[i]] = 0;
27 }
28
29 //斯坦纳树第二部分转移
30 void SPFA(int state){
31     while(q.size()){
32         int x = q.front(); q.pop(); vis[x][state] = 0;
33         //枚举连边
34         for(int i = head[x]; i; i = e[i].next){
35             int y = e[i].to;
36             //松弛操作
37             if(dp[y][st[y]|state] > dp[x][state]+e[i].w){
38                 dp[y][st[y]|state] = dp[x][state]+e[i].w;
39                 //状态保持一致或已经在队列中了
40                 if((st[y]|state) != state || vis[y][state]) continue;
41                 vis[y][state] = 1;
42                 q.push(y);
43             }
44         }
45     }
46 }
47 }
48
49 //斯坦纳树主函数
50 void steinerTree(){
51     rep(S,1,endS){
52         rep(i,1,n){
53             //i为关键节点, 判断i是否在状态S中
54             if(st[i] && (S|st[i]) != S) continue;
55             //第一部分转移, 枚举子集
56             for(int sub = S&(S-1); sub; sub = (sub-1)&S){
57                 int x = st[i]|sub, y = st[i]^(S-sub);
58                 if(dp[i][x] != inf && dp[i][y] != inf)
59                     dp[i][S] = min(dp[i][S],dp[i][x]+dp[i][y]);
60             }
61             if(dp[i][S] != inf)
62                 q.push(i), vis[i][S] = 1;
63         }
64         //第二部分转移, 对于每个状态进行一次转移
65         SPFA(S);
66     }
67 }

```

3.9.2 斯坦纳森林

题意: n 个点, m 条边, 前 k 个点为居住处, 后 k 个点为避难处。现在 k 个居民需要从居住处逃避到避难处, 问最少的连边的花费。注意只要满足居民有避难处可待即可, 因此是个森林。 $(1 \leq n \leq 50, 0 \leq m \leq 1000, 1 \leq k \leq 5, 2*k \leq n)$

思路: 这其实是个斯坦纳森林模板题。首先正常的求出斯坦纳树的结果, 即 $dp[i][state]$ 表示第 i 个点为树根, 关键点连通状态为 $state$ 的最小花费。

然后再设置一个新的函数 $f[1 << K]$, $f[state]$ 表示关键节点连通状态为 $state$ 时的最小代价。由于答案可以是森林, 因此枚举 $state$ 的子状态进行转移即可。

$dp[i][state] = \min(dp[i][state], dp[i][sub] + dp[i][state - sub])$ 。这里需要注意要检验状态 $state$ 是否合法, $state$ 状态合法当且仅当该状态中居住处点个数与避难处点个数相同。

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl;
6 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
7     << " , " << z1 << ":" << z2 << endl;
7 typedef long long ll;
8 typedef double db;
9 const db EPS = 1e-9;
10 const int N = 50+5;
11 const int M = 2*1000+100;
12 const int inf = 1e8;
13 using namespace std;
14
15 int n,m,k,head[N],dp[N][1<<10],f[1<<10],endS,st[N],tot;
16 bool vis[N][1<<10];
17 struct Node{
18     int to,next,w;
19 }e[M];
20 queue<int> q;
21
22 void add(int x,int y,int w){
23     e[++tot].to = y, e[tot].next = head[x], head[x] = tot, e[tot].w = w;
24 }
25
26 void init(){
27     tot = 1;
28     rep(i,0,n) head[i] = 0;
29     endS = (1<<(2*k))-1;
30     rep(i,0,n) memset(vis[i],0,sizeof vis[i]);
31     rep(i,0,n) st[i] = 0;
32     rep(i,0,n)
33         rep(S,0,endS) dp[i][S] = inf, vis[i][S] = 0;
34     rep(i,1,k) st[i] = 1<<(i-1);
35     rep(i,n-k+1,n) st[i] = 1<<(i-(n-k)+k-1);
36     rep(i,1,n)
37         if(st[i]) dp[i][st[i]] = 0;
38     rep(S,0,endS) f[S] = inf;
39 }
40
41 void SPFA(int state){
42     while(q.size()){

```

```

43     int x = q.front(); q.pop(); vis[x][state] = 0;
44     for(int i = head[x]; i; i = e[i].next){
45         int y = e[i].to;
46         // LOG2("x",x,"y",y);
47         if(y == x) continue;
48         if(dp[y][st[y]|state] > dp[x][state]+e[i].w){
49             dp[y][st[y]|state] = dp[x][state]+e[i].w;
50             if((st[y]|state) != state || vis[y][state]) continue;
51             q.push(y); vis[y][state] = 1;
52         }
53     }
54 }
55 }

56 void steinerTree(){
57     rep(S,1,endS){
58         rep(i,1,n){
59             //i在S中
60             if(st[i] && (st[i]|S) != S) continue;
61             //枚举子集
62             for(int sub = (S-1)&S; sub; sub = (sub-1)&S){
63                 int x = st[i]|sub, y = st[i]|(S-sub);
64                 // LOG2("x",x,"y",y);
65                 if(dp[i][x] != inf && dp[i][y] != inf)
66                     dp[i][S] = min(dp[i][S],dp[i][x]+dp[i][y]);
67             }
68             if(dp[i][S] != inf){
69                 q.push(i), vis[i][S] = 1;
70                 // LOG3("i",i,"S",S,"dp[i][S]",dp[i][S])
71             }
72         }
73     }
74     SPFA(S);
75 }
76 }

77 int check(int state){
78     int cnt = 0;
79     rep(i,0,2*k-1){
80         if(state&(1<<i))
81             cnt += i<k?1:-1;
82     }
83     return cnt == 0 ? 1:0;
84 }
85 }

86 void solve(){
87     rep(S,1,endS)
88         if(check(S)){
89             // LOG1("S",S);
90             rep(i,1,n)
91                 f[S] = min(f[S],dp[i][S]);
92         }
93     rep(S,1,endS)
94         for(int sub = S&(S-1); sub; sub = (sub-1)&S){
95             int x = sub, y = S-sub;
96             if(check(x) && check(y))
97                 f[S] = min(f[S],f[x]+f[y]);
98         }
99     if(f[endS] == inf) printf("No solution\n");
100    else printf("%d\n",f[endS]);
101 }
```

```

102 }
103
104 int main()
105 {
106     int _; scanf("%d",&_);
107     while(_--){
108         scanf("%d%d%d",&n,&m,&k);
109         init();
110         rep(i,1,m){
111             int x,y,w; scanf("%d%d%d",&x,&y,&w);
112             add(x,y,w); add(y,x,w);
113         }
114         steinerTree();
115         solve();
116     }
117     return 0;
118 }
```

3.9.3 斯坦纳树路径输出

题意: $n * m$ 的格子, 每个格子上的值表示这个点上的志愿者个数, 若该点为 0, 表示该点为景区。问最少需要多少个志愿者, 可以连通所有的景区, 需要输出路径。 $(1 \leq n, m, k \leq 10)$

思路: 答案直接上板子即可得到。(板子还是理解完记住比较好, 养成学一个记住一个的好习惯)。路径输出的话, 我们考虑一下 dp 的两部分转移方程。

可以发现 $(i, j, state)$ 构成了一个状态, (i, j) 表示坐标。每个状态可能由一个或两个子状态转移而来, 因此我们用 $pre[i][j][state]$ 结构体记住两个子状态, 输出路径的时候 dfs 递归所有子状态进行格点标记即可。

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
6 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
7 #define LOG4(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
8 #define LOG5(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
9 #define LOG6(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
10 #define LOG7(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
11 #define LOG8(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
12 #define LOG9(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
13
14 int n,m,a[N][N],st[N][N],cnt,dp[N][N][1<<N],endS,mp[N][N];
15 bool vis[N][N][1<<N];
16 int dir[4][2] = {{-1,0},{1,0},{0,1},{0,-1}};
17 queue<pair<int,int> q;
18 struct Node{
19     int x,y,s1,s2;
20     Node() {x = y = s1 = s2 = 0;}
21     Node(int a,int b,int c,int d):x(a),y(b),s1(c),s2(d) {}
22 }pre[N][N][1<<N];
23
24 void init(){
25     rep(i,1,n)
26         rep(j,1,m){
```

```

27         scanf("%d", &a[i][j]);
28         if(a[i][j] == 0) st[i][j] = 1<<cnt, cnt++;
29     }
30     endS = (1<<cnt)-1;
31     rep(i,0,n)
32         rep(j,0,m)
33             rep(S,0,endS) dp[i][j][S] = inf, vis[i][j][S] = 0;
34 //DP赋初值
35     rep(i,0,n)
36         rep(j,0,m)
37             if(st[i][j]) dp[i][j][st[i][j]] = 0;
38 }
39
40 void SPFA(int state){
41     while(q.size()){
42         int x = q.front().first, y = q.front().second; q.pop(); vis[x][y][state] = 0;
43         rep(i,0,3){
44             int tx = x+dir[i][0], ty = y+dir[i][1];
45             if(tx < 1 || tx > n || ty < 1 || ty > m) continue;
46             if(dp[tx][ty][state|st[tx][ty]] > dp[x][y][state]+a[tx][ty]){
47                 dp[tx][ty][state|st[tx][ty]] = dp[x][y][state]+a[tx][ty];
48                 //更新前驱
49                 pre[tx][ty][state|st[tx][ty]] = {x,y,state,state};
50                 if((state|st[tx][ty]) != state || vis[tx][ty][state]) continue;
51                 q.push(make_pair(tx,ty)); vis[tx][ty][state] = 1;
52             }
53         }
54     }
55 }
56
57 void steinertree(){
58     rep(S,1,endS){
59         rep(i,1,n)
60             rep(j,1,m){
61                 if(st[i][j] && (st[i][j]|S) != S) continue;
62                 for(int sub = S&(S-1); sub; sub = (sub-1)&S){
63                     int x = sub|st[i][j], y = st[i][j]|(S-sub);
64                     if(dp[i][j][x] != inf && dp[i][j][y] != inf){
65                         if(dp[i][j][S] > dp[i][j][x]+dp[i][j][y]-a[i][j]){
66                             dp[i][j][S] = dp[i][j][x]+dp[i][j][y]-a[i][j];
67                             pre[i][j][S] = {i,j,x,y};
68                         }
69                     }
70                 }
71                 if(dp[i][j][S] != inf)
72                     q.push(make_pair(i,j)), vis[i][j][S] = 1;
73             }
74         SPFA(S);
75     }
76 }
77
78 void dfs(int x,int y,int state){
79     if(vis[x][y][state] || x < 1 || x > n || y < 1 || y > m || !state) return;
80     mp[x][y] = 1; vis[x][y][state] = 1;
81     int tx = pre[x][y][state].x, ty = pre[x][y][state].y, s1 = pre[x][y][state].s1, s2
82     = pre[x][y][state].s2;
83     if(s1 != s2){
84         dfs(tx,ty,s1); dfs(tx,ty,s2);
85     }

```

```

85     else{
86         dfs(tx,ty,s1);
87     }
88 }
89
90 void solve(){
91     int ans = inf, xp = 0, yp = 0;
92     rep(i,1,n)
93         rep(j,1,m)
94             if(dp[i][j][endS] < ans){
95                 ans = dp[i][j][endS]; xp = i, yp = j;
96             }
97     memset(vis,0,sizeof vis);
98     dfs(xp,yp,endS);
99     printf("%d\n",ans);
100    rep(i,1,n){
101        rep(j,1,m){
102            if(a[i][j] == 0) printf("x");
103            else if(mp[i][j] == 1) printf("o");
104            else printf("_");
105        }
106        printf("\n");
107    }
108 }
109
110 int main()
111 {
112     scanf("%d%d",&n,&m);
113     init();
114     steinertree();
115     solve();
116     return 0;
117 }
```

3.10 长链剖分

3.10.1 k 级祖先

题意：

N 个点的一棵树， M 次询问，每次询问给出一个 x, k ，表示询问点 x 的 k 级祖先。 $(1 \leq N \leq 3 * 10^5, 1 \leq M \leq 1800000)$

思路：

k 级祖先是长链剖分的经典问题，主要应用的性质是“任意一个点的 k 级祖先 y 所在的长链的长度大于等于 k ”。

因此我们对于每个长链的端点维护两个 $vector$ ， $down[x]$ 中存储了点 x 所属长链上的所有节点， $up[x]$ 中存储了点 x 向上寻找能够找到的 $\min(d[x], dep[x])$ 个点。此处 $dep[x]$ 表示节点 x 在树中的深度， $d[x]$ 表示以 x 为端点的长链的长度。

因此我们查询点 x 的 k 级祖先时，只需要判断点 x 所在长链的端点距 x 的距离是否大于 k ，如果大于 k ，则需要使用点 x 长链端点的 up 数组，否则使用 $down$ 数组。大致思路是这样，具体的实现过程中还有一些其它的细节，详见代码。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 typedef long long ll;
4 const int N = 3*1e5+100;
5 const int M = 6*1e5+100;
6 const db EPS = 1e-9;
7 using namespace std;
```

```

8
9 int n,m,tot,head[N],top[N],son[N],len[N],fa[N],f[N][25],d[N],hbit[N];
10 vector<int> up[N], down[N];
11 struct Edge{
12     int to,next;
13 }e[M];
14
15 void add(int x,int y){
16     e[++tot].to = y, e[tot].next = head[x], head[x] = tot;
17 }
18
19 void dfs1(int x,int y){
20     len[x] = 1; d[x] = d[y]+1; f[x][0] = fa[x];
21     for(int i = 1; (1<<i) <= d[x]; i++)
22         f[x][i] = f[f[x][i-1]][i-1];
23     for(int v,i = head[x]; i; i = e[i].next)
24         if((v = e[i].to) != y){
25             fa[v] = x;
26             dfs1(v,x);
27             if(len[son[x]] < len[v]) son[x] = v;
28         }
29     len[x] = len[son[x]]+1;
30 }
31
32 void dfs2(int x,int y){
33     top[x] = y;
34     if(son[x]) dfs2(son[x],y);
35     for(int v,i = head[x]; i; i = e[i].next)
36         if((v = e[i].to) != fa[x] && v != son[x]) dfs2(v,v);
37 }
38
39 int query(int x,int k){
40     if(k == 0) return x;
41     if(d[x] < k+1) return 0;
42     int r = f[x][hbit[k]];
43     if(d[r] == d[x]-k) return r;
44     else{
45         int an = top[r];
46         if(d[an] <= d[x]-k){
47             int len = d[x]-k-d[an];
48             return down[an][len];
49         }
50         else{
51             int len = d[an]-d[x]+k;
52             return up[an][len];
53         }
54     }
55 }
56
57 int main()
58 {
59     scanf("%d",&n); tot = 1;
60     rep(i,1,n-1){
61         int x,y; scanf("%d%d",&x,&y);
62         add(x,y); add(y,x);
63     }
64     scanf("%d",&m);
65     dfs1(1,0);
66     dfs2(1,1);

```

```

67     int ans = 0;
68     rep(i,1,n)
69         if(top[i] == i){
70             int x = i;
71             while(x) {down[i].push_back(x); x = son[x];}
72             int ct = len[i];
73             x = i;
74             while(x && ct) {up[i].push_back(x); x = fa[x]; ct--;}
75         }
76     rep(i,1,300000)
77         rep(j,0,20)
78             if(i & (1<<j)) hbit[i] = max(hbit[i],j);
79     rep(i,1,m){
80         int x,y; scanf("%d%d",&x,&y);
81         x ^= ans; y ^= ans;
82         ans = query(x,y);
83         printf("%d\n",ans);
84     }
85     return 0;
86 }
```

3.10.2 O(1) 优化树形 dp 模板

题意：

n 个点的一棵树，对于树上每个点 x ，求出一个 $ans[x]$ ，表示在 x 的子树中， $ans[x]$ 深度处点的个数最多，若个数相同，令下标最小。 $(1 \leq N \leq 10^6)$

思路：

这是一道长链剖分的经典例题，也可以说是模板题。可以查看这道题的代码，来学会长链剖分的基础操作。

这道题思路比较经典，就是 $O(1)$ 继承重儿子，暴力继承轻儿子，更新的时候记录最大值即可。

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
6 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
7     << " , " << z1 << ":" << z2 << endl;
8 typedef long long ll;
9 typedef double db;
10 const db EPS = 1e-9;
11 const int N = 1e6+100;
12 using namespace std;
13 int n,tot,head[N],ans[N],d[N],son[N];
14 int *f[N],tmp[N],*id=tmp;
15 struct Node{
16     int to,next;
17 }e[2*N];
18
19 void add(int x,int y){
20     e[++tot].to = y, e[tot].next = head[x], head[x] = tot;
21 }
22
23 void dfs1(int x,int fa){
```

```

24     for(int i = head[x]; i; i = e[i].next){
25         int y = e[i].to;
26         if(y == fa) continue;
27         dfs1(y,x);
28         if(d[y] > d[son[x]]) son[x] = y;
29     }
30     d[x] = d[son[x]]+1;
31 }
32
33 void dfs2(int x,int fa){
34     f[x][0] = 1;
35     if(son[x]){
36         f[son[x]] = f[x]+1;
37         dfs2(son[x],x);
38         ans[x] = ans[son[x]]+1;
39     }
40     for(int i = head[x]; i; i = e[i].next){
41         int y = e[i].to;
42         if(y == fa || y == son[x]) continue;
43         f[y] = id; id += d[y];
44         dfs2(y,x);
45         rep(j,0,d[y]-1){
46             f[x][j+1] += f[y][j];
47             if(f[x][j+1] > f[x][ans[x]] || (f[x][j+1] == f[x][ans[x]] && j+1 < ans[x]))
48                 ans[x] = j+1;
49             }
50     if(f[x][ans[x]] == 1) ans[x] = 0;
51 }
52
53 int main()
54 {
55     scanf("%d",&n); tot = 1;
56     rep(i,1,n-1){
57         int x,y; scanf("%d%d",&x,&y);
58         add(x,y); add(y,x);
59     }
60     dfs1(1,0);
61     f[1] = id; id += d[1];
62     dfs2(1,0);
63     rep(i,1,n) printf("%d\n",ans[i]);
64     return 0;
65 }
66 /*
67 题意: n个点的一棵树, 对于树上每个点x, 求出一个ans[x], 表示在x的子树中, ans[x]深度处点的个数最多。
68 */

```

3.10.3 长链剖分维护后缀和

题意:

n 个点的一棵树, 有 q 组询问, 每组询问给出一个 p, k , 询问有多少个有序三元组 (p, b, c) 满足要求。其中 p, b, c 分别为树上三个不同的点, 其中 p 和 b 都是 c 的祖先, 且 p 与 b 之间的距离小于等于 k 。 $(1 \leq n \leq 3*10^5, 1 \leq q \leq 3*10^5, 1 \leq k \leq n)$

思路:

首先我们来分析一下这个三元组的性质。 (p, b, c) , p 和 b 是 c 的祖先, p 与 b 之间距离小于等于 k 。因此 b 可能是 p 的儿子, 也可能 p 的祖先。当 b 是 p 的祖先时, $ans = ans + min(dep[p] - 1, k) * (sz[p] - 1)$; 当 b 是 p 的儿子时, $ans = ans + sz[b] - 1$ 。

因此我们可以发现， b 是 p 的祖先的情况好处理，因此我们只需要处理 b 是 p 的儿子时情况即可。而 b 是 p 的儿子时，对答案的贡献就是 p 子树中所有节点 $sz[b] - 1$ 的和。

因此我们定义 $dp[u][j]$ 表示节点 u 中所有距 u 距离大于等于 j 的节点 $size - 1$ 之和。

$$dp[u][j + 1] = dp[u][j + 1] + dp[v][j]$$

$$dp[u][0] = dp[u][1] + sz[u] - 1$$

然后我们再将所有查询离线，接着进行长链剖分的常规操作， $O(1)$ 继承重儿子，暴力继承轻儿子即可完成此题。

这里需要注意，由于长链剖分的数组只能暂时保存每个节点状态的值，因此我们需要将所有查询离线进行预先处理。

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
6 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
7 << " , " << z1 << ":" << z2 << endl;
7 typedef long long ll;
8 typedef double db;
9 const int N = 3*1e5+100;
10 const db EPS = 1e-9;
11 using namespace std;
12
13 int n,q,tot,head[N],son[N],d[N],dep[N];
14 ll *f[N],tmp[N],*id=tmp,sz[N],ans[N];
15 struct Node{
16     int to,next;
17 }e[2*N];
18 vector<pair<int,int> > Q[N];
19
20 void add(int x,int y){
21     e[++tot].to = y, e[tot].next = head[x], head[x] = tot;
22 }
23
24 void dfs1(int x,int fa){
25     sz[x] = 1; dep[x] = dep[fa]+1;
26     for(int i = head[x]; i; i = e[i].next){
27         int y = e[i].to;
28         if(y == fa) continue;
29         dfs1(y,x); sz[x] += sz[y];
30         if(d[y] > d[son[x]]) son[x] = y;
31     }
32     d[x] = d[son[x]]+1;
33 }
34
35 void dfs2(int x,int fa){
36     if(son[x]){
37         f[son[x]] = f[x]+1;
38         dfs2(son[x],x);
39     }
40     for(int i = head[x]; i; i = e[i].next){
41         int y = e[i].to;
42         if(y == son[x] || y == fa) continue;

```

```

43     f[y] = id; id += d[y];
44     dfs2(y,x);
45     for(int j = 0; j < d[y]; j++)
46         f[x][j+1] += f[y][j];
47 }
48 for(auto &v:Q[x]){
49     int num = v.first, k = v.second;
50     ans[num] = (sz[x]-1ll)*(ll)min(dep[x]-1,k);
51     if(k >= d[x]-1) ans[num] += f[x][1]-f[x][d[x]-1];
52     else ans[num] += (ll)f[x][1]-(ll)f[x][k+1];
53 }
54 f[x][0] = f[x][1]+sz[x]-1ll;
55 }
56
57 int main()
58 {
59     scanf("%d%d",&n,&q); tot = 1;
60     rep(i,1,n-1){
61         int u,v; scanf("%d%d",&u,&v);
62         add(u,v); add(v,u);
63     }
64     dfs1(1,0);
65     f[1] = id; id += d[1];
66     rep(i,1,q){
67         int p,k; scanf("%d%d",&p,&k);
68         Q[p].push_back(make_pair(i,k));
69     }
70     dfs2(1,0);
71     rep(i,1,q) printf("%lld\n",ans[i]);
72     return 0;
73 }
```

3.10.4 长链剖分 + 线段树

题意：

n 个点的一棵树，树上每条边都有一个权值。现要在树中找到一条长度在 $[L, R]$ 范围内的一条路径，使得 $AvgValue = \frac{\sum_{e \in S} v(e)}{|S|}$ 最大，即道路平均价值最大。 $(1 \leq n \leq 10^5, 1 \leq L \leq R \leq n - 1, v_i \leq 10^6)$

思路：

看完题干中的道路平均价值 $AvgValue = \frac{\sum_{e \in S} v(e)}{|S|}$ 最大，应该就能反应过来这是一道 01 分数规划问题。而 01 分数规划问题，解法比较固定，即二分答案，然后令 $Sum = \sum_{e \in S} (v(e) - ans)$ ，其中 ans 为二分的答案。因此将图中所有边的边权减去 ans ，然后在图中找一条长度在 $[L, R]$ 范围内的路径，使得 Sum 最大。若 $Sum \geq 0$ ，则二分右半部分，否则二分左半部分。

因此我们的问题就变成了在树中找一条长度在 $[L, R]$ 范围内的路径，使得路径权值最大。

涉及到了路径长度，因此我们考虑树形 dp ，定义 $dp[u][j]$ 表示从节点 u 向其子树出发，路径长度为 j 时的最大路径权值和。

$$dp[u][j+1] = \max(dp[u][j+1], dp[v][j] + e[i].w)$$

$$ans = \max(ans, dp[v][j] + dp[u][x] + e[i].w)$$

$$L \leq 1 + j + x \leq R$$

从上述的 dp 方程中，我们可以发现我们需要在 $dp[u]$ 的一个范围内寻找最大值，由此想到了利用线段树来寻找最值。

因此我们在原先的长链剖分数组上维护一个线段树，每个节点保存该节点到根的距离，由此 $dp[u][x] = dis[u][x] - dis[u]$ ，就可以避免继承重儿子信息时的区间修改等操作。如若定义 $dp[u][x]$ 为 u 到深度为 x 的节点的距离，则在继承重儿子信息时，无可避免地需要用到区间修改。而定义 $dp[u][x]$ 为根到 u 深度为 x 的节点的距离的话，即可避免修改问题。

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << ", " << y1 << ":" << y2 << endl;
6 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << ", " << y1 << ":" << y2
7 << ", " << z1 << ":" << z2 << endl;
7 typedef long long ll;
8 typedef double db;
9 const int N = 1e5+100;
10 const db EPS = 1e-5;
11 const db inf = 1e14;
12 using namespace std;
13
14 int n,L,R,head[N],tot,d[N],son[N];
15 db *f[N],tmp[N],*id = tmp, ans, maxn[2*N], dis[N];
16 struct Node{
17     int to,next;
18     ll wt;
19     db w;
20 }e[2*N];
21 inline int get_id(int l,int r) {return (l+r)|(l!=r);}
22
23 void add(int x,int y,ll w){
24     e[++tot].to = y, e[tot].next = head[x], head[x] = tot, e[tot].wt = w;
25 }
26
27 void build(int l,int r){
28     maxn[get_id(l,r)] = -inf;
29     if(l == r) return;
30     int mid = (l+r)>>1;
31     build(l,mid); build(mid+1,r);
32 }
33
34 void insert(int l,int r,int pos,db v){
35     int now = get_id(l,r);
36     if(l == r) {maxn[now] = max(maxn[now],v); return;}
37     int mid = (l+r)>>1;
38     if(pos <= mid) insert(l,mid,pos,v);
39     else insert(mid+1,r,pos,v);
40     maxn[now] = max(maxn[get_id(l,mid)],maxn[get_id(mid+1,r)]);
41 }
42
43 db query(int l,int r,int ls,int rs){
44     if(rs < ls) return -inf;
45     int now = get_id(l,r);
46     if(ls <= l && r <= rs) return maxn[now];
47     int mid = (l+r)>>1;
48     db tmp = -inf;
49     if(ls <= mid) tmp = max(tmp,query(l,mid,ls,rs));
50     if(rs > mid) tmp = max(tmp,query(mid+1,r,ls,rs));

```

```

51     return tmp;
52 }
53
54 void dfs1(int x,int fa){
55     for(int i = head[x]; i; i = e[i].next){
56         int y = e[i].to;
57         if(y == fa) continue;
58         dfs1(y,x);
59         if(d[y] > d[son[x]]) son[x] = y;
60     }
61     d[x] = d[son[x]]+1;
62 }
63
64 void dfs2(int x,int fa){
65     int pos = f[x]-tmp;
66     f[x][0] = dis[x];
67     insert(1,n,pos+1,f[x][0]);
68     for(int i = head[x]; i; i = e[i].next){
69         if(son[x] == e[i].to){
70             f[son[x]] = f[x]+1;
71             dis[son[x]] = dis[x]+e[i].w;
72             dfs2(son[x],x);
73         }
74     }
75     for(int i = head[x]; i; i = e[i].next){
76         int y = e[i].to;
77         if(y == fa || y == son[x]) continue;
78         f[y] = id; id += d[y];
79         dis[y] = dis[x]+e[i].w;
80         dfs2(y,x);
81         rep(j,0,d[y]-1){ //找最大值
82             if(j+1 <= R){
83                 db base = query(1,n,max(1,pos+L-j),min(pos+d[x],pos+R-j))-2.0*dis[x];
84                 ans = max(ans,base+f[y][j]);
85             }
86         }
87         rep(j,0,d[y]-1){ //更新最值
88             if(f[y][j] > f[x][j+1]){
89                 f[x][j+1] = f[y][j];
90                 insert(1,n,pos+j+2,f[x][j+1]);
91             }
92         }
93     }
94     db base = query(1,n,pos+L+1,min(pos+d[x],pos+R+1))-dis[x];
95     ans = max(ans,base);
96 }
97
98 int check(db base){
99     ans = -inf;
100    rep(i,2,tot) e[i].w = (db)e[i].wt-base;
101    build(1,n);
102    rep(i,0,n) tmp[i] = 0, dis[i] = 0;
103    id = tmp; f[1] = id; id += d[1];
104    dfs2(1,0);
105    if(ans-0 >= EPS) return 1;
106    else return 0;
107 }
108
109 int main()

```

```

110 {
111     scanf("%d", &n); tot = 1;
112     scanf("%d%d", &L, &R);
113     rep(i, 1, n - 1) {
114         int x, y, ll w; scanf("%d%d%lld", &x, &y, &w);
115         add(x, y, w); add(y, x, w);
116     }
117     dfs1(1, 0);
118     db l = 1, r = 1000000;
119     while(r - l >= EPS) {
120         db mid = (l + r) / 2.0;
121         if(check(mid)) l = mid;
122         else r = mid;
123     }
124     printf("%.3f\n", r);
125     return 0;
126 }
```

3.10.5 三元组两两距离相等

题意：

n 个点的一棵树，求树上的三元无序集合 (a, b, c) ，满足集合中任意两点距离相同的条件的集合个数。 $(1 \leq n \leq 10^5)$

思路：

这是一道树形 dp 用长链剖分优化的典型问题，主要难点在于树形 dp 的状态方程如何设置。

我们首先要思考如何才能不重不漏地计算所有的集合个数。而想要不重不漏地计算个数，就必须找到一个不变量。而在一棵树中，最常见的不变量就是 lca 。因此我们对于树上每一个点，枚举其作为 lca 时对答案的贡献。由于 lca 是固定的，所以枚举 lca 可以避免重复计算。

然后我们考虑对于一个点作为 lca 时，它对答案的贡献是什么？首先对于树形 dp ，我们一定是从左往右枚举子节点的一个过程，当枚举到节点 v 时，对答案的贡献要么是左边子树中的两个节点 + v 子树中的一个节点，不然就是左边子树中的一个节点 + v 子树中的两个节点。只有这两种情况需要进行枚举。

思考到这一步，我们的 dp 状态方程就好定了。首先令 $f[u][j]$ 表示距节点 u 距离为 j 的节点个数，由此我们可以解决子树中一个节点的情况。然后考虑子树中两个节点时，如果设置状态？

由于我们需要两个子节点之间的距离，因此我们令 $g[u][j]$ 表示 u 子树中， (a, b) 点对数量。点对满足条件为 a, b 到其 lca 距离均为 d ，而其 lca 到 u 距离为 $d - j$ ， d 为任意值。这样设置状态的原因是， $d - j + j = d$ ，这样就可以满足三点距离相同了。至此，状态便设置完成，然后考虑如何书写转移方程。

$$f[u][j] = f[u][j] + f[v][j - 1]$$

$$g[u][j] = g[u][j] + g[v][j + 1]$$

上述两条转移式子比较显然，但是对于 g 数组来说，我们只考虑了 $d - j$ 不为 0 的情况，即 u 不为 $lca(a, b)$ 的情况，因此还需要考虑 $u = lca(a, b)$ 的情况，即 (a, b) 分别位于 u 的两个子节点的子树中。

$$g[u][j] = g[u][j] + f[u][j] * f[v][j - 1]$$

至此，我们考虑完了 f, g 两个数组的转移方程。接下来我们需要考虑对答案的贡献如何表示。如上所述，每一个节点作为三点 lca 时，对答案的贡献为左子树两点 + 当前子节点子树中一点，或左子树一点 + 当前子节点子树中两点。由此可以得到下述方程。

$$ans = ans + f[u][j-1] * g[v][j] + g[u][j+1] * f[v][j]$$

现在，我们得到了此题完整的四个转移方程。接下来只有最后一个注意点就是，由于 $g[u][j] = g[u][j] + g[v][j+1]$ ，因此 g 数组我们需要开两倍，避免子节点转移时下标 -1 影响到了其它节点的区间造成越界。

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
6 ;
7 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
8 << " , " << z1 << ":" << z2 << endl;
9
10 typedef long long ll;
11 typedef double db;
12 const db EPS = 1e-9;
13 const int N = 1e5+100;
14 using namespace std;
15
16 int n,tot,head[N],son[N],d[N];
17 ll *f[N], *g[N], tf[N], tg[2*N], *id1 = tf+1, *id2 = tg+1, ans;
18 struct Node{
19     int to,next;
20 }e[2*N];
21
22 void add(int x,int y){
23     e[++tot].to = y, e[tot].next = head[x], head[x] = tot;
24 }
25
26 void dfs1(int x,int fa){
27     for(int i = head[x]; i; i = e[i].next){
28         int y = e[i].to;
29         if(y == fa) continue;
30         dfs1(y,x);
31         if(d[y] > d[son[x]]) son[x] = y;
32     }
33     d[x] = d[son[x]]+1;
34 }
35
36 void dfs2(int x,int fa){
37     f[x][0] = 1;
38     if(son[x]){
39         f[son[x]] = f[x]+1; g[son[x]] = g[x]-1;
40         dfs2(son[x],x);
41     }
42     ans += g[x][0];
43     for(int i = head[x]; i; i = e[i].next){
44         int y = e[i].to;
45         if(y == son[x] || y == fa) continue;
46         f[y] = id1, g[y] = id2+d[y];
47         id1 += d[y], id2 += d[y]*2;
48         dfs2(y,x);
49         rep(j,0,d[y]-1){
50             ans += g[x][j+1]*f[y][j];
51             if(j >= 1) ans += f[x][j-1]*g[y][j];
52         }
53         rep(j,0,d[y]-1){
54
55
56
57
58
59
5
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
50

```

```

51         g[x][j+1] += f[x][j+1]*f[y][j];
52         f[x][j+1] += f[y][j];
53     if(j >= 1) g[x][j-1] += g[y][j];
54 }
55 }
56 }
57
58 int main()
59 {
60     scanf("%d",&n); tot = 1;
61     rep(i,1,n-1){
62         int a,b; scanf("%d%d",&a,&b);
63         add(a,b); add(b,a);
64     }
65     dfs1(1,0);
66     f[1] = id1, g[1] = id2+d[1]; id1 += d[1], id2 += d[1]*2;
67     dfs2(1,0);
68     printf("%lld\n",ans);
69     return 0;
70 }
```

3.11 巧妙 dp 题

3.11.1 亏欠型 DP

题意:

一条街上一共 N 个点，需要在某些点上建路灯，使得整条街被照亮，一个路灯可以照亮左右两个点，每个点都有一个建路灯的花费。现在你还有 K 次机会，可以交换两个路灯的建造费用，求使得整条街被照亮的最小花费。 $(1 \leq N \leq 250000, 0 \leq k \leq 9)$

思路:

如果这道题没有 K 次交换路灯花费的操作的话，问题将会变得简单很多，只需要记录每个点某尾和前一个点的状态即可递推求解。

但是此题有了 K 次交换路灯的条件。仔细观察，可以发现，交换路灯花费时，一定是在便宜的那个点上建路灯，在贵的那个点上不建路灯，交换两个都要建路灯的节点是没有意义的。因此我们考虑将建灯花费先欠着，即对于点 i 来说，我们让这个点灯亮，但是具体花费先欠着，等到后面一个点的时候再还这个费用。

因此我们构建 DP 状态， $dp[i][j][k][l]$ 表示第 i 个点，欠了 j 次，还了 k 次， l 表示节点最后两个点的状态。 $l = 0$ ，亮 + 不亮。 $l = 1$ ，不亮 + 亮。 $l = 2$ ，(亮/不亮)+ 亮。

然后就可以进行转移了，具体转移过程见代码，并不复杂，主要难点在于这个思想。

总结:

亏欠型 DP 的核心关键点在于“欠”与“还”。“欠”对应着预支花费，“还”对应着预付花费，可以有效的解决交换花费的问题。

这种方法的重要思想是“预支”，与此类似的还有一种思想是“反悔”，就是我先把这个值拿过来，如果之后发现更好的，我再把它扔掉。(如网络流中的反向边)

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
6 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
               << " , " << z1 << ":" << z2 << endl;
7 typedef long long ll;
```

```

8  typedef double db;
9  const db EPS = 1e-9;
10 const int N = 250000+100;
11 using namespace std;
12
13 int n,K;
14 ll dp[N][10][10][3],w[N];
15
16 int main()
17 {
18     scanf("%d%d",&n,&K);
19     rep(i,1,n) scanf("%lld",&w[i]);
20     memset(dp,0x3f,sizeof dp);
21     dp[0][0][0][0] = 0; //第i个位置，欠了j个，还了k个
22     rep(i,1,n)
23         rep(j,0,K){
24             rep(k,0,K){
25                 dp[i][j][k][0] = dp[i-1][j][k][2];
26                 if(k > 0) dp[i][j][k][0] = min(dp[i][j][k][0],dp[i-1][j][k-1][2]+w[i]);
27
28                 dp[i][j][k][1] = dp[i-1][j][k][0];
29                 if(k > 0) dp[i][j][k][1] = min(dp[i][j][k][1],dp[i-1][j][k-1][0]+w[i]);
30
31                 rep(h,0,2)
32                     dp[i][j][k][2] = min(dp[i][j][k][2],dp[i-1][j][k][h]+w[i]);
33                 if(j > 0){
34                     rep(h,0,2)
35                         dp[i][j][k][2] = min(dp[i][j][k][2],dp[i-1][j-1][k][h]);
36                 }
37             }
38         }
39     ll ans = 1e17;
40     rep(i,0,K)
41         ans = min(ans,min(dp[n][i][i][0],dp[n][i][i][2]));
42     printf("%lld\n",ans);
43     return 0;
44 }

```

3.11.2 数位 DP+ 状压 DP

题意:

给出一个 n , 现可以从 $1 \sim n$ 中任意选取若干个数字, 要求所有数字不存在重复的数位。例如1, 2, 3 和2, 11 是合法的, 但是1, 2, 10 和2, 5, 12 是不合法的。

现给出一个 n , 询问存在多少个集合满足上述要求, 结果模 $1e9 + 7$ 。 $(1 \leq n \leq 10^9)$

思路:

首先, 不难想到此题需要将 $0 \sim 9$ 的数位状压起来, 然后再进行数位 dp 求解, 最后再进行一个类似于容斥或背包的操作求出答案。

我们定义 $num[i]$ 表示 $1 \sim n$ 中有多少个数字, 数字中包含的 $0 \sim 9$ 的状态恰好为 i 。再定义 $ans[i]$ 表示 $1 \sim n$ 中有多少个不同的满足条件的集合, 集合中数位的状态恰好为 i 。

因此我们先求 $num[i]$, 定义数位 dp 的状态为 $dp[S][pos]$ 表示长度为 pos 的数位, 数位状态为 S 的数的个数。

然后数位 dp 经典操作 dfs 求取 $num[i]$, 中间用 $dp[S][pos]$ 进行记忆化搜索。枚举第 pos 位的数位时, 答案既可以由 $dp[S][pos - 1]$ 更新而来, 也可以由 $dp[S \ xor (1 << i)][pos - 1]$ 更新而来。因为 pos 位已经提供了 i 这个数位, 因

此之后的数位可以提供也可以不提供，因此有两种情况。

所以我们可以只用这个二维的状态即可求解出 $num[i]$ 。比赛时数位 dp 部分写了一个比较复杂的三维状态，最后卡时通过。

然后我们再来考虑如何求取 $ans[i]$ 。两种考虑方法，一种是利用的背包的想法，一种是利用组合的想法。

先讲背包的想法。我们已经求出了 $num[i]$ 表示 $1 \dots n$ 中数位状态为 i 的数的个数。然后可以将 $num[i]$ 中的每一个 i 看成一个物品，然后 $ans[i][j]$ 表示利用前 i 个物品组成的状态为 j 的方案总数。

```

1 ans[0][0] = 1;
2 for(nt i = 0; i < 1024; i++){
3     for(int j = 0; j < 1024; j++){
4         ans[i+1][j] += ans[i][j];
5         if((j&i) == 0){
6             ans[i+1][j|i] += ans[i][j] * num[i];
7         }
8     }

```

再讲组合容斥的想法。定义 $ans[i][j]$ 表示有多少个集合，组成了 i 这个状态，一共含有 j 个不同的数位状态。比如数位 12 这个状态，既可以由 1 和 2 两个状态组成，也可以直接由 12 这一个状态组成。然后我们就可以直接枚举子集进行计算。这里需要注意，枚举子集时，每次由一个单体和一个集合一起算贡献，而不是每次用两个集合算贡献，这样可以尽量避免重复。利用单体算贡献，每个单体被重复计算 k 次，因此最后答案除 k 。

```

1 rep(S, 1, endS){
2     rep(k, 2, 9){
3         ll cnt = 0;
4         for(int sub = S & (S - 1); sub; sub = (sub - 1) & S){
5             int x = sub, y = S - sub;
6             cnt = (cnt + DP[x] * ans[y][k - 1]) % mod;
7         }
8         cnt = cnt * inv[k] % mod;
9         ans[S][k] = (ans[S][k] + cnt) % mod;
10    }
11 }

```

完整代码如下。

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define mem(a,b) memset(a,b,sizeof a);
4 #define rep(i, a, b) for(int i = a; i <= b; i++)
5 #define LOG1(x1, x2) cout << x1 << ":" << x2 << endl;
6 #define LOG2(x1, x2, y1, y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl;
7 #define LOG3(x1, x2, y1, y2, z1, z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << " , " << z1 << ":" << z2 << endl;
8 typedef long long ll;
9 typedef double db;
10 const db EPS = 1e-9;
11 const ll mod = 1e9 + 7;
12 using namespace std;
13
14 ll dp[1 << 11][13], endS, ans[1 << 11][10], DP[1 << 11], inv[100];
15 int len, a[13];
16
17 void init() {
18     mem(dp,-1);
19     mem(ans,0);

```

```

20     mem(DP, 0);
21 }
22
23 ll dfs(int pos, int state, bool flag, int jud) { //jud表示有无数字
24     if (pos == 0) {
25         if(state == 0 && jud == 1) return 1;
26         else return 0;
27     }
28     if (!flag && dp[state][pos] != -1 && jud == 1) return dp[state][pos];
29     ll base = 0;
30     int end = flag ? a[pos] : 9;
31     rep(i, 0, end){
32         if(jud == 0 && i == 0){ //无数字
33             base = (base + dfs(pos - 1, state, flag && i == end, 0)) % mod;
34         }
35         else{
36             if (((1 << i) & state) == 0) continue;
37             base = (base + dfs(pos - 1, state, flag && i == end, 1)) % mod;
38             base = (base + dfs(pos - 1, state^(1<<i), flag && i == end, 1)) % mod;
39         }
40     }
41     if (!flag && jud == 1) dp[state][pos] = base;
42     return base;
43 }
44
45 void solve(ll n) {
46     len = 0;
47     memset(a, 0, sizeof a);
48     while (n) {
49         a[++len] = n % (10ll);
50         n /= 10ll;
51     }
52     rep(S, 1, endS) DP[S] = dfs(len, S, 1, 0);
53 }
54
55 ll pow_mod(ll a, ll b, ll m) {
56     ll ans = 1;
57     while (b) {
58         if (b & 1) ans = ans * a % mod;
59         a = a * a % mod;
60         b >>= 1;
61     }
62     return ans;
63 }
64
65 int main() {
66     int _;
67     scanf("%d", &_);
68     for (int i = 1; i <= 100; ++i) {
69         inv[i] = pow_mod(i, mod - 2, mod);
70     }
71     endS = (1 << 11) - 1;
72     rep(Ca, 1, _) {
73         init();
74         ll n;
75         scanf("%lld", &n);
76         solve(n);
77         rep(S, 1, endS) ans[S][1] = DP[S];
78         rep(S, 1, endS){

```

```
79     rep(k, 2, 9){
80         ll cnt = 0;
81         for(int sub = S & (S - 1); sub; sub = (sub - 1) & S){
82             int x = sub, y = S - sub;
83             cnt = (cnt + DP[x] * ans[y][k - 1]) % mod;
84         }
85         cnt = cnt * inv[k] % mod;
86         ans[S][k] = (ans[S][k] + cnt) % mod;
87     }
88     ll sum = 0;
89     rep(S, 1, endS) rep(i, 1, 9) sum = (sum + ans[S][i]) % mod;
90     printf("Case %d: %lld\n", Ca, sum);
91 }
92 return 0;
93 }
```

4 数据结构

4.1 单调栈

给出一个序列，长度为 n 。定义区间价值为区间和 * 区间最小值，求出这个序列中的最大区间价值。 $(n \leq 10^5, 0 \leq ai \leq 10^6)$

回忆一下单调栈和单调队列，单调栈——对于序列中每个点，求出序列中左/右边第一个比它大/小的点，单调队列——对于序列中每个点，求出序列中距离该点 K 步范围内的最小/大值。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <stack>
5 #include <algorithm>
6 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
7 #define rep(i,a,b) for(int i = a; i <= b; i++)
8 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
9 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
10 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
     << " , " << z1 << ":" << z2 << endl;
11 typedef long long ll;
12 typedef double db;
13 const int N = 1e5+100;
14 const int M = 1e5+100;
15 const db EPS = 1e-9;
16 using namespace std;
17
18 int n,a[N],tt1[N],tt2[N];
19 ll sum[N];
20 stack<int> st;
21
22 int main()
23 {
24     scanf("%d",&n);
25     rep(i,1,n) scanf("%d",&a[i]);
26     rep(i,1,n) tt1[i] = 0, tt2[i] = n+1;
27     st.push(1);
28     rep(i,2,n){
29         while(st.size() && a[i] < a[st.top()]){
30             int x = st.top();
31             st.pop();
32             tt2[x] = i;
33             // LOG1("x",x);
34         }
35         if(st.size()) tt1[i] = st.top();
36         st.push(i);
37     }
38     rep(i,1,n) sum[i] = sum[i-1]+(ll)a[i];
39     ll ans = 0;
40     int l = 1,r = 1;
41     rep(i,1,n){
42         if(ans < a[i]*(sum[tt2[i]-1]-sum[tt1[i]])){
43             ans = a[i]*(sum[tt2[i]-1]-sum[tt1[i]]);
44             l = tt1[i]+1;
45             r = tt2[i]-1;
46         }
47     }
}

```

```

48     printf("%lld\n",ans);
49     printf("%d %d\n",l,r);
50     return 0;
51 }
```

4.2 单调队列

题意：

给定一串 01 串， m 次询问，每次询问给你一个数 k 。 k 为对于这个 01 串所能进行的最多次操作，每次操作可以将该串中任意一个位置的数移到任意一个其他的位置。

每次询问之后，输出在这个操作数之内，所能达到的最长的连续 0 的长度。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <algorithm>
4 #include <cstring>
5 #define rep(i,a,b) for(int i = a;i <= b;i++)
6 using namespace std;
7 const int N = 1e6+1000;
8
9 char s[N];
10 int m,n,a[N],sum[N],q[N],cnt;
11
12 void solve(int k)
13 {
14     int ans = 0;
15     q[1] = 0;
16     int l = 1, r = 1;
17     rep(i,1,n)
18     {
19         while(l <= r && a[q[r]] >= a[i]) r--; //维护单调队列，先对右端点进行更新
20         q[++r] = i;
21         while(l <= r && sum[i]-sum[q[l]] > k) l++; //再对左端点进行更新
22         ans = max(ans,a[q[r]]-a[q[l]]+k);
23     }
24     printf("%d\n",min(ans,cnt));
25 }
26
27 int main()
28 {
29     while(~scanf("%s",s))
30     {
31         cnt = 0;
32         scanf("%d",&m);
33         int len = strlen(s);
34         n = len;
35         sum[0] = 0;a[0]=0;
36         rep(i,1,len)
37         {
38             if(s[i-1] == '0'){
39                 sum[i] = sum[i-1];
40                 cnt++;
41             }
42             else sum[i] = sum[i-1]+1;
43             a[i] = i-2*sum[i];
44         }
45     }
46 }
```

```

44
45     }
46     rep(i,1,m)
47     {
48         int x;
49         scanf("%d",&x);
50         solve(x);
51     }
52 }
53 return 0;
54 }
```

4.3 并查集

4.3.1 普通并查集

```

1 #include <cstdio>
2 #include <iostream>
3 using namespace std;
4 const int maxn=10000+10;
5
6 int n,m,p[maxn];
7
8 int find(int x) {
9     return p[x]==x?x:p[x]=find(p[x]);
10 }
11
12 void merge(int x,int y){
13     int r1=find(x),r2=find(y);
14     if(r1!=r2){
15         p[r1]=r2;
16     }
17 }
18
19 int main()
20 {
21     cin>>n>>m;
22     for(int i=1;i<=n;i++){
23         p[i]=i;
24     }
25     for(int i=1;i<=m;i++){
26         int x,y;
27         cin>>x>>y;
28         p[x]=y;
29     }
30     return 0;
31 }
```

4.3.2 带权并查集

题意： N 个人在玩剪刀石头布的游戏，其中有一个人是裁判，其余人随机被分到了三个阵营，即剪刀、石头、布。现在有 M 轮游戏， $x <, >, = y$ 表示 x 与 y 之间的关系，其中裁判可以自由变换阵营。问是否可以根据这 M 轮游戏，判断出谁是裁判，输出裁判是谁以及最早在哪一轮可以找到裁判。如果无法判断，则输出 *Can not determine*，如果游戏的情况不符合题意，则输出 *Impossible*。 $(N \leq 500, M \leq 2000)$

思路：一开始做这题的时候，的确有些懵，只能想到如何发现有人变换了阵营，但是不知道如何找到这个人，并且确定这种情况是唯一的。

所以我们可以发现直接考虑整个问题会非常困难，再加上此题数据范围很小，直接做的话复杂度肯定很少，太对不起这个数据范围了。因此我们考虑 N^2 做法，即枚举每个人为裁判。枚举 x 为裁判时，如果 x 恰好为裁判，则并查集合并时不会发生矛盾。我们统计有多少个人为裁判时，并查集合并会发生矛盾，记人数为 cnt ，并且统计每个人为裁判时，发生矛盾的轮数 ri 。

现在我们来考虑输出答案的所有情况。

1. 如果 $cnt = n - 1$ ，则可以唯一确定裁判，并且最早发现该裁判的轮数为 $\max(ri)$ 。
2. 如果 $cnt = n$ ，则输出 *impossible*。
3. 其余情况则为 *Can not determine*。

至于带权并查集的合并类似于食物链问题，是个模 3 剩余系中的加减问题。

```

1 #include <csdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
8 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
9 ;
10 typedef long long ll;
11 typedef double db;
12 const db EPS = 1e-9;
13 using namespace std;
14 const int N = 1000;
15 const int M = 2000+100;
16 int n,m,fa[N],d[N],vis[N],error[N],cnt;
17
18 int find(int x){
19     if(x == fa[x]) return x;
20     int root = find(fa[x]);
21     d[x] = (d[x]+d[fa[x]]+3)%3;
22     return fa[x] = root;
23 }
24 struct Query{
25     int x,y,cc;
26 }q[M];
27
28 int main()
29 {
30     while(~scanf("%d%d",&n,&m))
31     {
32         memset(vis,0,sizeof vis);
33         memset(error,0,sizeof error);
34         cnt = 0;
35         rep(i,1,m){
36             int x,y; char cc;
37             scanf("%d%c%d",&x,&cc,&y);
38             q[i].x = x, q[i].y = y;
39             if(cc == '=') q[i].cc = 0;
40             else if(cc == '<') q[i].cc = 1;
41             else q[i].cc = 2;
42         }
43         rep(i,0,n-1){
44             rep(k,0,n) fa[k] = k, d[k] = 0;
45             rep(j,1,m){
```

```

46         int x = q[j].x, y = q[j].y, cc = q[j].cc;
47         if(x == i || y == i) continue;
48         int fx = find(x), fy = find(y);
49         if(fx != fy){
50             fa[fx] = fy;
51             d[fx] = (d[y]-d[x]+cc+3)%3;
52         }
53     else{
54         int ttp = (d[x]-d[y]+3)%3;
55         if(ttp != cc){
56             cnt++; // 产生矛盾个数
57             error[i] = j;
58             vis[i] = 1;
59             break;
60         }
61     }
62 }
63 }
64 // impossible
65 // not determine
66 // determine
67 if(cnt == (n-1)) { //determine
68     int ans = 0, hm;
69     rep(i,0,n-1){
70         ans = max(ans,error[i]);
71         if(!vis[i]) hm = i;
72     }
73     printf("Player %d can be determined to be the judge after %d lines\n",hm,
74     ans);
75 }
76 else if(cnt == n){ // impossible
77     printf("Impossible\n");
78 }
79 else{ // not determine
80     printf("Can not determine\n");
81 }
82 return 0;
83 }
```

4.3.3 带权并查集与背包

题意：给出 p_1 个天使， p_2 个魔鬼，一共有 n 个问题。每个问题的格式为 $x\ y\ (yes\ or\ no)$ 表示问 $x\ y$ 是否为天使，如果 x 为天使，则会说真话，如果 x 为魔鬼，则会说假话。问根据这 n 个问题，是否可以确定哪些人为天使，如果可以确定，按编号大小输出天使编号。 $(n \leq 1000, p_1, p_2 \leq 300)$

思路：遇到这样的 $yes\ or\ no$ 问题，显然我们需要对于可能出现的情况进行模拟。

1. 假如 x 为天使， y 为天使，则回答 yes 。
2. 假如 x 为天使， y 为魔鬼，则回答 no 。
3. 假如 x 为魔鬼， y 为天使，则回答 no 。
4. 假如 x 为魔鬼， y 为魔鬼，则回答 yes 。

可以发现，如果回答是 yes ，则 x 与 y 属于同一类。如果回答 no ，则 x 与 y 类别相反。因此一个模 2 剩余系的带权并查集合并就可以维护所有人的关系。

因此不难发现，处理完 n 个问题之后，我们会拥有若干个并查集，每个并查集中都会分为两批人，两批人类别不同。

因此问题变成了，从这若干个并查集中随机选一部分，使得最后选中的人数恰好为 p_1 ，并且这种选择方式唯一，则我们可以确定哪些人为天使。因此本题就变成了一个类似背包的问题， $dp[i][j]$ 表示前 i 个并查集选取人数为 j 一共有多少种选择方案， $pre[i][j] = x$ 表示 $dp[i][j]$ 由 $dp[i-1][x]$ 更新而来。到此，本题即可顺利解决。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <map>
5 #include <algorithm>
6 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
7 #define rep(i,a,b) for(int i = a; i <= b; i++)
8 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
9 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
10
11 typedef long long ll;
12 typedef double db;
13 const db EPS = 1e-9;
14 using namespace std;
15 const int N = 700;
16
17 int n,p1,p2,fa[N],d[N],base[N][2],tot,dp[N][600],pre[N][600],vis[N];
18 map<int,int> mp;
19
20 int find(int x)
21 {
22     if(x == fa[x]) return x;
23     int root = find(fa[x]);
24     d[x] = (d[x]+d[fa[x]]+2)%2;
25     return fa[x] = root;
26 }
27
28 void solve()
29 {
30     memset(base,0,sizeof base);
31     memset(vis,0,sizeof vis);
32     mp.clear();
33     tot = 0;
34     rep(i,1,p1+p2){
35         fa[i] = find(i);
36         if(mp.find(fa[i]) == mp.end()) mp[fa[i]] = ++tot;
37         int pos = mp[fa[i]];
38         base[pos][d[i]]++; //0: 相同 1: 不同
39     }
40     memset(dp,0,sizeof dp);
41     memset(pre,0,sizeof pre);
42     dp[1][base[1][1]]++;
43     dp[1][base[1][0]]++;
44     rep(i,2,tot){
45         int minn = min(base[i][0],base[i][1]);
46         rep(j,minn,p1){
47             if(j >= base[i][0] && dp[i-1][j-base[i][0]] != 0)
48                 dp[i][j] += dp[i-1][j-base[i][0]], pre[i][j] = j-base[i][0];
49             if(j >= base[i][1] && dp[i-1][j-base[i][1]] != 0)
50                 dp[i][j] += dp[i-1][j-base[i][1]], pre[i][j] = j-base[i][1];
51         }
52     }
53     if(dp[tot][p1] != 1) printf("no\n");
54     else{
55         int xx = tot, yy = p1;
56     }
57 }
```

```

55     while(xx > 1){
56         if(pre[xx][yy] == yy-base[xx][0]) vis[xx] = 0, yy -= base[xx][0];
57         else if(pre[xx][yy] == yy-base[xx][1]) vis[xx] = 1, yy -= base[xx][1];
58         xx--;
59     }
60     if(xx == 1){
61         if(base[1][0] == yy) vis[1] = 0;
62         else vis[1] = 1;
63     }
64     rep(i,1,p1+p2){
65         int pos = mp[fa[i]];
66         int dd = d[i];
67         if(vis[pos] == 1 && dd == 1) printf("%d\n",i);
68         else if(vis[pos] == 0 && dd == 0) printf("%d\n",i);
69     }
70     printf("end\n");
71 }
72 }
73
74 int main()
75 {
76     while(~scanf("%d%d%d",&n,&p1,&p2))
77     {
78         if((n+p1+p2) == 0) break;
79         rep(i,0,p1+p2) fa[i] = i, d[i] = 0;
80         rep(i,1,n){
81             int x,y; char op[10];
82             scanf("%d%d",&x,&y);
83             scanf("%s",op);
84             int fx = find(x), fy = find(y);
85             // LOG2("x",x,"y",y);
86             // LOG2("fx",fx,"fy",fy);
87             if(fx != fy){
88                 fa[fx] = fy;
89                 if(op[0] == 'n') d[fx] = (2+1+d[y]-d[x])%2;
90                 else d[fx] = (d[y]-d[x]+2)%2;
91             }
92         }
93         solve();
94     }
95     return 0;
96 }
```

4.3.4 按秩合并并查集

题意：

n 个点， m 个操作，操作共两类。1 $u v$ 表示在图中加一条边连接 $u v$ ，2 $u v$ 表示查询 u 与 v 最早是在哪一次加边操作后连通，不连通输出 -1。 $(1 \leq n, m \leq 10^5)$

思路：

维护连通性，最直观的想法就是用并查集来维护连通性。但是如何通过并查集来查看两点最早什么时候连通呢？

首先可以知道并查集维护的其实是一个森林，假如我们不破坏树的结构，即不进行路径压缩，则并查集每次加边，则将边权定义为操作编号，那么两点树上边权最大值就是两点最早连通的加边操作。

因此问题变成如何维护树的结构进行并查集合并，方法就是按秩合并，将小的树合并到大的树上，这样可以保证每个节点最多被合并 $\log(n)$ 次，因此每个节点的高度最多为 $\log(n)$ 。因此对于每次查询，我们可以将两个点到根节点的路

径直接取出，然后查询第一次遇到的位置，输出达到这个位置之前的边权最大值即可。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
8 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
9 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
10 << " , " << z1 << ":" << z2 << endl;
11 typedef long long ll;
12 typedef double db;
13 const int N = 1e5+100;
14 const int M = 1e5+100;
15 const db EPS = 1e-9;
16 using namespace std;
17
18 int f[N],n,m,siz[N],d[N],flag[N],vis[N];
19
20 int find(int x){
21     if(x == f[x]) return x;
22     else return find(f[x]);
23 }
24
25 int solve(int u,int v){
26     vis[u] = 1; flag[u] = 0; int w = 0;
27     while(f[u] != u){
28         w = max(w,d[u]);
29         vis[f[u]] = 1; flag[f[u]] = w;
30         u = f[u];
31     }
32     vis[u] = 1, flag[u] = w;
33     w = 0;
34     if(vis[v] == 1) return flag[v];
35     while(f[v] != v){
36         w = max(w,d[v]);
37         if(vis[f[v]] == 1) return max(w,flag[f[v]]);
38         v = f[v];
39     }
40     if(vis[v] == 1) return max(flag[v],w);
41 }
42
43 void clear(int u){
44     while(f[u] != u) vis[u] = 0, flag[u] = 0, u = f[u];
45     vis[u] = 0, flag[u] = 0;
46 }
47
48 int main()
49 {
50     int _; scanf("%d",&_);
51     while(_--){
52         scanf("%d%d",&n,&m);
53         rep(i,0,n) f[i] = i, d[i] = 0, siz[i] = 1;
54         rep(i,1,m){
55             int op,u,v; scanf("%d%d%d",&op,&u,&v);
56             int xu = find(u), xv = find(v);

```

```

56     if(op == 1 && xu != xv){
57         if(siz[xu] < siz[xv]) f[xu] = xv, d[xu] = i, siz[xv] += siz[xu];
58         else f[xv] = xu, d[xv] = i, siz[xu] += siz[xv];
59     }
60     else if(op == 2){
61         if(xu != xv) printf("-1\n");
62         else{
63             printf("%d\n", solve(u,v));
64             clear(u);
65         }
66     }
67 }
68 }
69 return 0;
70 }
```

4.4 Hash

题意：

给出 n 个串，m 组询问。每组询问均为一个字符串，询问在初始 n 个串中是否存在一个串与该询问串恰好只有一个位置不相同。输出 YES or NO。

Hash 思路：

首先先讲讲 Hash 的算法，利用 bkdr 算法将每个字符串 hash 成一个数值，hash 函数如下：

```

hash: abccac
hash[1] = 0
hash[6] = 0 * 1315 + 1 * 1314 + 2 * 1313 + 2 * 1312 + 0 * 131 + 2
```

此处 hash[6] 即是这个字符串的 hash 值，131 为 seed，即 hash 种子，然后还要取一个模数，即 mod。由此可以发现 hash 值即是字符串中每一个位置的贡献，所以本题要求恰好只有一个位置不相同，即可以枚举不相同位置，减去原有贡献，加上新贡献即可。然后本题就可以解决，但是由于此题的数据卡的很 e xin，所以 seed 和 mod 取的不好的话，会被卡成 zz。

这里补充一下常见的 seed 和 mod，seed 一般取质数，3、5、7、13、131、13131 这些均可，mod 一般也取质数，1e9+7，1e11+7，1e13+7，1e18+7 均可，也可以直接将数据类型取为 unsigned long long，即可对 $2^{64} - 1$ 直接取模。

有一个注意点，在计算过程中，seed*mod 不能超过数据类型的最大值，否则相当于在计算过程中又模上了一个不是 mod 的数，会导致结果错误。

由于本题只有三个字符，所以可以将 seed 定为 3，mod 定一个很大的数，类似于三进制。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <map>
5 #include <algorithm>
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 using namespace std;
8 typedef long long ull;
9 const int N = 6*1e5+1000;
10 const ull mod = 1e11+7;
11 const ull ttp1 = 3;
12 const ull ttp2 = 5;
13 int n,m;
14 ull base[5] = {0,1,2,131313,123};
15 ull seed1[N],seed2[N];
```

```

16 char s[N];
17 struct Node{
18     ull a,b;
19 }gn;
20 map<Node,int> mp;
21
22 bool operator == (Node x, Node y)
23 {
24     if(x.a == y.a && x.b == y.b) return true;
25     else return false;
26 }
27
28 bool operator < (Node x, Node y)
29 {
30     return x.a < y.a;
31 }
32
33 int solve(ull hash1, ull hash2)
34 {
35     int len = strlen(s);
36     // printf("s:%s,len:%d\n",s,len);
37     rep(j,0,len-1)
38     {
39         rep(k,0,2)
40         {
41             if(s[j]-'a' == k) continue;
42             ull tmp1 = hash1;
43             tmp1 = (tmp1+(-base[s[j]-'a']+base[k])*seed1[len-1-j]%mod)%mod;
44             if(tmp1 < 0) tmp1 += mod;
45
46             ull tmp2 = hash2;
47             tmp2 = (tmp2+(-base[s[j]-'a']+base[k])*seed2[len-1-j]%mod)%mod;
48             if(tmp2 < 0) tmp2 += mod;
49
50             gn.a = tmp1, gn.b = tmp2;
51             // printf("k:%d,tmp:%llu\n",k,tmp);
52             if(mp[gn] == 1) return 1;
53         }
54     }
55     return 0;
56 }
57
58 int main()
59 {
60     // printf("mod:%lld\n",mod);
61     mp.clear();
62     seed1[0] = 1;
63     seed1[1] = ttp1;
64     seed2[0] = 1;
65     seed2[1] = ttp2;
66     int _ = 6*1e5+100;
67     rep(i,2,_)
68     {
69         seed1[i] = (seed1[i-1]*ttp1)%mod;
70         if(seed1[i] < 0) seed1[i] += mod;
71
72         seed2[i] = (seed2[i-1]*ttp2)%mod;
73         if(seed2[i] < 0) seed2[i] += mod;
74     }
}

```

```

75     scanf("%d%d",&n,&m);
76     rep(i,1,n)
77     {
78         scanf("%s",s);
79         int len = strlen(s);
80         ull hash1 = 0;
81         ull hash2 = 0;
82         rep(j,0,len-1)
83         {
84             hash1 = (hash1*tpp1%mod+base[s[j]-'a'])%mod;
85             if(hash1 < 0) hash1+=mod;
86
87             hash2 = (hash2*tpp2%mod+base[s[j]-'a'])%mod;
88             if(hash2 < 0) hash2+=mod;
89         }
90         // printf("s:%s,hash:%llu\n",s,hash);
91         gn.a = hash1, gn.b = hash2;
92         mp[gn] = 1;
93     }
94     rep(i,1,m)
95     {
96         scanf("%s",s);
97         int len = strlen(s);
98         ull hash1 = 0;
99         ull hash2 = 0;
100        rep(j,0,len-1)
101        {
102            hash1 = (hash1*tpp1%mod+base[s[j]-'a'])%mod;
103            if(hash1 < 0) hash1+=mod;
104
105            hash2 = (hash2*tpp2%mod+base[s[j]-'a'])%mod;
106            if(hash2 < 0) hash2+=mod;
107        }
108        if(solve(hash1,hash2))
109            printf("YES\n");
110        else printf("NO\n");
111    }
112    return 0;
113 }
114
115 //mod: 1e9+7, seed: 257
116 //mod: 1e18+3, seed: 3、5
117 //mod: 1e11+7, seed: 3、5、1313、...
118 //如果mod*seed会越界的话，那么结果就会错误，因为计算过程中出现了两个seed，多了自动溢出的那个seed

```

4.5 KMP

一、KMP 求解过程

next 数组——最长前缀后缀匹配长度，需要小于串长度

ababa, $next[1] = 0, next[2] = 0, next[3] = 1, next[4] = 2, next[5] = 3$

求解问题：模式串在匹配串的哪些位置出现了，模式串是小串。

二、例题

(1) 求取 num 数组，表示最长的，不超过一半的，前缀与后缀相同的长度。 $(1 \leq n \leq 10^6)$

一个显然的性质: $num[i] \leq num[i - 1] + 1$
因此求取 $num[i]$ 时, 先令 $num[i] = num[i - 1] + 1$, 如果 $num[i]$ 超过一半的长度或者在 $num[i]$ 处发生失配, 则直接跳 $next$ 数组。

(2) 给出 n 个串, 每个串的长度为 m_i , 询问这 n 个串的最长公共子串。此处两个子串相同的条件为, 长度相同且其中一个子串加上一个数即可得到另一个子串。 $(1 \leq n \leq 1000, 1 \leq m_i \leq 100)$

1. 在差分序列上进行 kmp 匹配。
2. 这里由于数据范围比较小, 因此可以直接枚举后缀进行 kmp 匹配求最小值。
3. 但是多个串的最长公共子串其实是一个经典问题, 可以用后缀数组或后缀自动机解决。

```

1 //预处理next数组, 这里注意不要反复求strlen, 这个函数是O(n)复杂度的, 反复求会将复杂度变成 O(n^2)
2 next[0] = -1;
3 for(int i = 1, j; i <= n; i++){
4     for(j = next[i-1]; j >= 0 && s1[j+1] != s1[i]; j = next[j]);
5     next[i] = j+1;
6 }
7 //next[j] <= next[j-1]+1, 不会比前一位+1更大
8 //匹配过程
9 for(int i=1, j=0; i <= m; i++){
10    for(; s2[i] != s1[j+1] && j >= 0; j = next[j]);
11    j++;
12    if(j==n) ans++;
13 }
```

4.6 字典树

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 using namespace std;
5 const int charset = 26;
6 const int max_node = 100000+10;
7
8 struct Trie
9 {
10     int tot, root, child[max_node][charset]; //root:根节点 tot:节点编号
11     //child[i][j]=k, 表示编号为i的节点的第j个孩子是编号为k的节点
12     bool flag[max_node]; //是否以某一个字符为结束
13     Trie()
14     {
15         // memset(child,0,sizeof(child));
16         // memset(flag,0,sizeof(flag));
17         root = tot = 1; //根节点编号为1
18     }
19     void mem() //将字典树初始化
20     {
21         memset(child,0,sizeof(child));
22         memset(flag,0,sizeof(flag));
23         root = tot = 1; //根节点编号为1
24     }
25     void insert(const char *str)
26     {
27         int cur = root;
28         for(int i = 0;str[i];i++)
29         {
30             int x = str[i]-'a';
31             if(child[cur][x]==0)
32                 child[cur][x] = tot;
33             cur = child[cur][x];
34             if(flag[cur])
35                 return;
36         }
37         flag[cur] = true;
38     }
39     void search(const char *str)
40     {
41         int cur = root;
42         for(int i = 0;str[i];i++)
43         {
44             int x = str[i]-'a';
45             if(child[cur][x]==0)
46                 return;
47             cur = child[cur][x];
48         }
49         if(flag[cur])
50             cout << str << endl;
51     }
52     void print()
53     {
54         queue<int> q;
55         q.push(root);
56         while(!q.empty())
57         {
58             int cur = q.front();
59             q.pop();
60             for(int i = 0;i<26;i++)
61             {
62                 if(child[cur][i])
63                     q.push(child[cur][i]);
64             }
65         }
66     }
67 };
68
69 int main()
70 {
71     Trie t;
72     t.mem();
73     t.insert("abc");
74     t.insert("ab");
75     t.insert("a");
76     t.insert("bcd");
77     t.insert("bc");
78     t.insert("b");
79     t.print();
80 }
```

```

31         if(child[cur][x] == 0)
32             child[cur][x] = ++tot;
33         cur = child[cur][x];
34     }
35 // flag[cur] = true; 记录单词以该点结束
36 }
37 bool query(const char *str)
38 {
39     int cur = root;
40     for(int i = 0; str[i]; i++)
41     {
42         int x = str[i]-'a';
43         if(child[cur][x] == 0) return false;
44         cur = child[cur][x];
45     }
46     return true;
47 //查询单词时应该 return flag[cur];
48 }
49 }tre;

```

4.7 ST 表

```

1 int st[N][25];
2
3 //数组范围为[0,len-1]
4 void init(){
5     for(int i = 0; i < len; i++) st[i][0] = a[i];
6     for(int j = 1; (1<<j) <= len; j++){
7         for(int i = 0; i + (1<<j) - 1 < len; i++)
8             st[i][j] = min(st[i][j-1],st[i+(1<<(j-1))][j-1]);
9     }
10 }
11
12 //求[l,r]之间的最小值
13 int query(int l,int r){
14     int k = (int)(log((double)(r - l + 1)) / log(2.0));
15     return min(st[l][k],st[r-(1<<k)+1][k]);
16 }

```

4.8 树状数组

```

1 int c[N];
2
3 inline int lowbit(int x) { return x&(-x); }
4
5 inline void update(int x, int c){
6     for(int i = x; i <= k; i += lowbit(i)) t[i] += c;
7 }
8
9 inline int ask(int x){
10     int tp = 0;
11     for(int i = x; i; i -= lowbit(i)) tp += t[i];
12     return tp;
13 }

```

4.9 线段树

4.9.1 动态开点与区间修改 lazy

```

1 #include <cstdio>
2 #include <algorithm>
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 const int N = 1e5+100;
5 using namespace std;
6
7 int ls[2*N],rs[2*N],maxn[2*N],lazy[2*N],rt,sz,a[N],n; //rt: root, sz: 当前节点编号
8
9 void push_down(int now){
10     if(ls[now] == 0) ls[now] = ++sz;
11     if(rs[now] == 0) rs[now] = ++sz;
12     maxn[ls[now]] += lazy[now], maxn[rs[now]] += lazy[now];
13     lazy[ls[now]] += lazy[now], lazy[rs[now]] += lazy[now];
14     lazy[now] = 0;
15 }
16 void update(int& now, int l, int r, int lx, int rx, int c){ //区间修改
17     if(!now) now = ++sz;
18     if(lx <= l && rx >= r){
19         maxn[now] += c; lazy[now] += c;
20         return;
21     }
22     if(lazy[now] != 0) push_down(now);
23     int mid = (l+r)>>1;
24     if(lx <= mid) update(ls[now],l,mid,lx,rx,c);
25     if(rx > mid) update(rs[now],mid+1,r,lx,rx,c);
26     maxn[now] = max(maxn[ls[now]],maxn[rs[now]]);
27 }
28 int query(int& now, int l, int r){ //查询最右边第一个值大于0的点
29     if(!now) now = ++sz;
30     if(maxn[now] <= 0) return -1;
31     if(l == r) return a[l];
32     if(lazy[now] != 0) push_down(now);
33     int mid = (l+r)>>1;
34     if(maxn[rs[now]] > 0) return query(rs[now],mid+1,r);
35     else return query(ls[now],l,mid);
36 }
37
38 int main()
39 {
40     scanf("%d",&n);
41     rep(i,1,n){
42         int pos,op; scanf("%d%d",&pos,&op);
43         if(op == 1){ //push
44             int xx; scanf("%d",&xx); a[pos] = xx;
45             update(rt,1,n,1,pos,1);
46         }
47         else update(rt,1,n,1,pos,-1); //pop
48         printf("%d\n",query(rt,1,n));
49     }
50     return 0;
51 }
```

4.9.2 区间递减序列和

题意：与楼房重建题意类似，但是求的是递减序列，而且询问的是区间 $[l, r]$ 的递减序列和。

思路：只需将维护内容的 $maxn$ 改为 $minn$ 即可，然后解决一下区间查询的问题。继续使用刚才的 $calc(now, tp)$ 函数，计算节点 now 在最小值为 tp 下的贡献。依然是左右区间二分，如果左区间最小值小于 tp ，则直接计算右区间贡献，然后递归左区间。如果左区间最小值大于 tp ，则直接递归右区间。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
8 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
9 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
10 << " , " << z1 << ":" << z2 << endl;
11 typedef long long ll;
12 typedef double db;
13 const int N = 1e5+100;
14 const int M = 3*1e5+100;
15 const ll inf = 1e13;
16 const db EPS = 1e-9;
17 using namespace std;
18
19 int n,q,rt,sz,ls[M],rs[M];
20 ll a[N],sum[M],minn[M],ans;
21
22 ll calc(int &now,int l,int r,ll tp){
23     if(!now) now = ++sz;
24     if(l == r) return (minn[now]<tp?minn[now]:0ll);
25     int mid = (l+r)>>1;
26     if(minn[ls[now]] < tp) return (sum[now]-sum[ls[now]]+calc(ls[now],l,mid,tp));
27     else return calc(rs[now],mid+1,r,tp);
28 }
29
30 void update(int &now,int l,int r,int pos,ll w){
31     if(!now) now = ++sz;
32     if(l == r){
33         sum[now] = minn[now] = w;
34         return;
35     }
36     int mid = (l+r)>>1;
37     if(pos <= mid) update(ls[now],l,mid,pos,w);
38     else update(rs[now],mid+1,r,pos,w);
39     minn[now] = min(minn[rs[now]],minn[ls[now]]);
40     sum[now] = sum[ls[now]]+calc(rs[now],mid+1,r,minn[ls[now]]);
41 }
42
43 ll query(int &now,int l,int r,int pos1,int pos2,ll w){
44     if(!now) now = ++sz;
45     if(pos1 <= l && pos2 >= r){
46         ans += calc(now,l,r,w);
47         return minn[now];
48     }
49     int mid = (l+r)>>1;
50     if(pos1 <= mid) w = min(w,query(ls[now],l,mid,pos1,pos2,w));
51     if(pos2 > mid) w = min(w,query(rs[now],mid+1,r,pos1,pos2,w));
52     return w;

```

```

52 }
53
54 int main()
55 {
56     int _; scanf("%d",&_);
57     while(_--){
58         rt = sz = 0;
59         memset(ls,0,sizeof ls);
60         memset(rs,0,sizeof rs);
61         scanf("%d%d",&n,&q);
62         rep(i,1,n){
63             scanf("%lld",&a[i]);
64             update(rt,1,n,i,a[i]);
65         }
66         rep(i,1,q){
67             int l,r,p,c; scanf("%d%d%d%d",&l,&r,&p,&c);
68             ans = 0, query(rt,1,n,l,r,inf);
69             printf("%lld\n",ans);
70             if(p != 0 || c != 0) update(rt,1,n,p,c);
71         }
72     }
73     return 0;
74 }
```

4.9.3 线段树维护图连通

题意: n 个点, m 条边, 每条边有一个区间 $[l_i, r_i]$, 表示只有体重在这个范围内的人才能通过这条边, 问有多少种不同的体重可以从起点到达终点。 $(1 \leq n, m \leq 10^5, 1 \leq l_i, r_i \leq 10^9)$

思路: 这应该是一类经典题目, 即按照每条边的权重建线段树, 这次是题目给出的上下界, 下次可能是到达时间等。

然后每个节点维护该权重下有哪些边是连通的。因此每个节点均维护一个 $vector$, 存储当前权重下哪些边连通。然后线段树分治的时候, 将当前权重涉及到的边全部用并查集连起来。

每到达线段树的一个节点, 就将节点的边连通, 然后判断当前 1 是否和 n 可达, 如果可达, 则返回。否则继续二分区间, 向下继续分治。没做过这类题的话, 可能不太好理解, 建议直接看代码, 代码还是很好理解的。

这里还涉及到一个问题, 就是每条边的权重是一个区间, 如果我们直接按照闭区间往线段树中添加, 是会出错的。具体出错原因可以手跑一下下面这个数据, 大致来说就是权重若为闭合区间, 在线段树分治的过程中, 会丢失很多相邻区间之间的答案。

因此离散化时, 需要将右端点 +1, 插入一个区间时, 需要将右端点离散化后的值再 -1。分治时, 查询到一段可行区间后, 对答案的贡献应该是 $b[r+1] - b[l]$, 这样就不会忽略掉 $[b[r], b[r+1]]$ 之间的一段了。而开区间的话, 会忽略掉这一段。

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
6 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
7                                         << " , " << z1 << ":" << z2 << endl;
7 typedef long long ll;
8 typedef double db;
9 const db EPS = 1e-9;
10 const int N = 2*1e5+100;
```

```

11 using namespace std;
12
13 int n,m,tot,sz[N],fa[N];
14 struct Node{
15     int u,v;
16     ll l,r;
17 }e[N];
18 ll b[N],ans;
19 vector<int> t[2*N];
20
21 int findx(ll x) {return lower_bound(b+1,b+1+tot,x)-b;}
22 int find(int x) {return x==fa[x]?x:find(fa[x]);}
23 inline int get_id(int l,int r) {return (l|r)|(l!=r);}
24
25 void update(int l,int r,int ls,int rs,int id){
26     int now = get_id(l,r);
27     if(ls <= l && r <= rs){
28         t[now].push_back(id);
29         return;
30     }
31     int mid = (l+r)>>1;
32     if(ls <= mid) update(l,mid,ls,rs,id);
33     if(rs > mid) update(mid+1,r,ls,rs,id);
34 }
35
36 void dfs(int l,int r){
37     int now = get_id(l,r);
38     int len = t[now].size();
39     vector<int> back; back.clear();
40     rep(i,0,len-1){
41         int u = find(e[t[now][i]].u), v = find(e[t[now][i]].v);
42         if(u != v){
43             if(sz[u] > sz[v]) swap(u,v);
44             fa[u] = v; sz[v] += sz[u];
45             back.push_back(u); back.push_back(v);
46         }
47     }
48     if(find(1) == find(n)) ans += b[r+1]-b[l];
49     else if(l != r){
50         int mid = (l+r)>>1;
51         dfs(l,mid); dfs(mid+1,r);
52     }
53     len = back.size();
54     rep(i,0,len-1){
55         fa[back[i]] = back[i], sz[back[i]] = 1;
56     }
57     back.clear();
58 }
59
60 int main()
61 {
62     scanf("%d%d",&n,&m);
63     rep(i,1,n) fa[i] = i, sz[i] = 1;
64     rep(i,1,m){
65         scanf("%d%d%lld%lld",&e[i].u,&e[i].v,&e[i].l,&e[i].r);
66         e[i].r++;
67         b[++tot] = e[i].l; b[++tot] = e[i].r;
68     }
69     sort(b+1,b+1+tot);

```

```

70     tot = unique(b+1,b+1+tot)-b-1;
71     rep(i,1,m) update(1,tot,findx(e[i].l),findx(e[i].r)-1,i);
72     dfs(1,tot);
73     printf("%lld\n",ans);
74     return 0;
75 }
```

4.9.4 线段树维护树直径

题意: n 个点的一棵树, 每个点初始为 0, 支持两种操作, 第一种操作 $C\ x$, 表示将第 x 个点取反, 即 1 变 0, 0 变 1。第二种操作为 G , 表示查询两个相距最远的 0 点距离。 $(1 \leq n \leq 10^5, 1 \leq m \leq 5 * 10^5)$

思路: 这道题的做法有括号序列、动态点分治、线段树维护直径。此处只介绍线段树维护直径的做法。

首先我们求一个 dfs 序, 然后在 dfs 序上建线段树, 对于线段树的每个区间, 我们维护两个点表示在这个区间中相距最远的两个 0 点。区间合并时我们只需要取出这两个区间所维护的点, 然后对这四个点两两求距离更新答案即可。

这样做的原因在于对于两个子树来说, 每个子树都有一条属于该子树的直径, 则两个子树合并后的新直径必定是从原来两个子树中的 4 个点中选取 2 个点作为答案。

大致思路就是这样, 具体细节见代码。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 typedef long long ll;
4 const int N = 1e5+100;
5 using namespace std;
6
7 int n,m,dis[N],tot,head[N],f[N][25],dfn[N],rk[N],flag[N],T;
8 pair<int,int> sgt[4*N];
9 struct Edge{
10     int to,next;
11 }e[2*N];
12
13 inline void add(int x,int y){
14     e[++tot].to = y, e[tot].next = head[x], head[x] = tot;
15 }
16
17 void dfs(int x,int fa){
18     dis[x] = dis[fa]+1; f[x][0] = fa; dfn[x] = ++tot; rk[tot] = x;
19     for(int i = 1; (1<<i) <= dis[x]; i++)
20         f[x][i] = f[f[x][i-1]][i-1];
21     for(int i = head[x]; i; i = e[i].next){
22         int y = e[i].to;
23         if(y == fa) continue;
24         dis[y] = dis[x]+1; dfs(y,x);
25     }
26 }
27
28 inline int lca(int x,int y){
29     if(dis[x] > dis[y]) swap(x,y);
30     for(int i = T; i >= 0; i--)
31         if(dis[f[y][i]] >= dis[x]) y = f[y][i];
32     if(x == y) return x;
33     for(int i = T; i >= 0; i--)
34         if(f[x][i] != f[y][i]) x = f[x][i], y = f[y][i];
35     return f[x][0];
}
```

```

36 }
37
38 inline int dist(int x,int y){
39     if(x == 0 && y == 0) return -1;
40     if(x == 0 || y == 0) return 0;
41     return dis[x]+dis[y]-2*dis[lca(x,y)];
42 }
43
44 inline pair<int,int> pushUp(pair<int,int> x,pair<int,int> y){
45     int a[5],res = 0,cnt = 0,tmp; pair<int,int> base = make_pair(0,0); a[0] = 0;
46     if(x.first && flag[x.first]) a[++cnt] = x.first;
47     if(x.second && flag[x.second]) a[++cnt] = x.second;
48     if(y.first && flag[y.first]) a[++cnt] = y.first;
49     if(y.second && flag[y.second]) a[++cnt] = y.second;
50     rep(i,1,cnt-1)
51         rep(j,i+1,cnt)
52             if((tmp=dist(a[i],a[j])) > res)
53                 res = tmp, base = make_pair(a[i],a[j]);
54     if(res == 0) base = make_pair(a[cnt],a[cnt]);
55     return base;
56 }
57
58 inline void build(int now,int l,int r){
59     if(l == r) sgt[now] = make_pair(rk[l],rk[l]), flag[rk[l]] = 1;
60     else{
61         int mid = (l+r)>>1;
62         build(now<<1,l,mid); build(now<<1|1,mid+1,r);
63         sgt[now] = pushUp(sgt[now<<1],sgt[now<<1|1]);
64     }
65 }
66
67 inline void update(int now,int l,int r,int pos){
68     if(l == r){
69         flag[rk[l]] ^= 1;
70         if(flag[rk[l]]) sgt[now] = make_pair(rk[l],rk[l]);
71         else sgt[now] = make_pair(0,0);
72         return;
73     }
74     int mid = (l+r)>>1;
75     if(pos <= mid) update(now<<1,l,mid,pos);
76     else update(now<<1|1,mid+1,r,pos);
77     sgt[now] = pushUp(sgt[now<<1],sgt[now<<1|1]);
78 }
79
80 int main()
81 {
82     scanf("%d",&n);
83     tot = 1; T = (int)(log(n)/log(2))+1;
84     rep(i,1,n-1){
85         int a,b; scanf("%d%d",&a,&b);
86         add(a,b); add(b,a);
87     }
88     tot = 0; dfs(1,0); build(1,1,n);
89     scanf("%d",&m);
90     while(m--){
91         char op[10]; scanf("%s",op);
92         if(op[0] == 'C'){
93             int x; scanf("%d",&x);
94             update(1,1,n,dfn[x]);

```

```

95         }
96         else printf("%d\n", dist(sgt[1].first, sgt[1].second));
97     }
98     return 0;
99 }
```

4.9.5 区间不同 gcd 个数

题意：给出 n 个数字，一共 q 次查询，每次询问一个 l, r ，查询区间 $[l, r]$ 中有多少个不同的 gcd ，其中一个子区间代表一个 gcd 。 $(1 \leq n, q \leq 10^5, 1 \leq a_i \leq 10^6)$

思路：区间查询不同 gcd 的个数，这类题像一类套路问题，主要要抓住 gcd 的几个性质。

1. 固定右端点，移动左端点， gcd 的值从 a_r 不断变化，要么不变，要么至少除 2（因为 gcd 最小值为 2）。因此固定右端点之后，只会存在至多 \log 个不同的 gcd ，我们对于相同的 gcd 仅保留最靠右的位置。
2. 因此对于每个右端点，我们记录一个 $vector$ ，存储对于这个右端点的所有不同的 gcd 值。我们可以根据 $vector[i - 1]$ 来更新 $vector[i]$ 。

处理完上述操作之后，我们得到了 $n * \log n$ 个 (l, r, gcd) 三元组，然后我们将所有查询按照右端点排序。

记录一个 pos ，不断右移到查询的右端点位置，每次移动时将 $vector[pos]$ 内的信息存储到树状数组中，即对于每个三元组 (l, pos, gcd) ，如果该 gcd 未出现过，则在树状数组的 l 位置 +1，并设置 $vis[gcd] = l$ 。如果该 gcd 出现过，则比较 l 是否比 $vis[gcd]$ 更大，如果更大，则修改 gcd 在树状数组中的位置。上述操作即不断维护 gcd 最靠右的出现位置。然后对于每个查询，直接在树状数组中区间查询即可。

总结：此题最关键的在于发现区间 gcd 不断除 2 的性质，然后将查询离线利用树状数组不断维护每个 gcd 最后出现的位置即可完成。

```

1 #include <bits/stdc++.h>
2 #define mem(a,b) memset(a,b,sizeof a);
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define per(i,a,b) for(int i = a; i >= b; i--)
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 typedef long long ll;
7 typedef double db;
8 const int N = 1e6+100;
9 const db EPS = 1e-9;
10 using namespace std;
11
12 void dbg() {cout << "\n";}
13 template<typename T, typename... A> void dbg(T a, A... x) {cout << a << ' '; dbg(x...)}
14 #define logs(x...) {cout << #x << " -> "; dbg(x);}
15
16 int n,q,a[N],vis[N];
17 ll ans[N],c[N];
18 struct Node{
19     int l,r,id;
20     bool operator < (Node xx) const {
21         return r < xx.r;
22     }
23 }Q[N];
24 vector<pair<int,int> > base[N];
25
26 inline int lowbit(int x) {return x&(~x+1);}
27 inline void update(int x,ll v) {for(;x<=n;x+=lowbit(x)) c[x] += v;}
28 inline ll ask(int x){
```

```

29     ll tp = 0;
30     while(x) tp += c[x], x -= lowbit(x);
31     return tp;
32 }
33
34 int gcd(int a,int b){
35     return b == 0 ? a:gcd(b,a%b);
36 }
37
38 int main()
39 {
40     while(~scanf("%d%d",&n,&q)){
41         rep(i,1,n) scanf("%d",&a[i]);
42         rep(i,1,n) base[i].clear();
43         rep(i,0,n) c[i] = 0;
44         //更新每个点的vector
45         rep(i,1,n){
46             base[i].push_back(make_pair(i,a[i])); vis[a[i]] = 1;
47             for(auto &v:base[i-1]){
48                 int tp = gcd(v.second,a[i]);
49                 if(!vis[tp]){
50                     base[i].push_back(make_pair(v.first,tp));
51                     vis[tp] = 1;
52                 }
53             }
54             for(auto &v:base[i])
55                 vis[v.second] = 0;
56         }
57         rep(i,1,q) scanf("%d%d",&Q[i].l,&Q[i].r), Q[i].id = i;
58         sort(Q+1,Q+1+q);
59         int pos = 0;
60         rep(i,1,q){
61             while(pos <= Q[i].r){
62                 for(auto &v:base[pos]){
63                     int hp = v.first;
64                     if(hp > vis[v.second]){
65                         if(vis[v.second]) update(vis[v.second],-1);
66                         vis[v.second] = hp;
67                         update(vis[v.second],1);
68                     }
69                 }
70                 pos++;
71             }
72             ans[Q[i].id] = ask(Q[i].r)-ask(Q[i].l-1);
73         }
74         rep(i,1,q) printf("%lld\n",ans[i]);
75         memset(vis,0,sizeof vis);
76     }
77     return 0;
78 }
```

4.10 扫描线

4.10.1 面积并

只要矩形一条线一条线扫描的时候，右边的线没有被读进来，则右边的线一直在扫描面积

举个简单的例子，本题中的扫描线只是一个消除矩阵面积重复计算部分的一个方法

```

1 #include <iostream>
2 #include <cstdio>
3 #include <algorithm>
4 using namespace std;
5 const int SIZE = 300+10;
6 struct Line{
7     double x,y1,y2;
8     int flag;
9 }line[SIZE];
10
11 bool cmp(Line a,Line b)
12 {
13     return a.x<b.x;
14 }
15 struct tree{
16     int l,r;
17     double ml,mr;
18     int s; //s始终 >= 0
19     double len;
20 }t[SIZE*4];
21 int n;
22 double y[SIZE];
23 void build(int p,int l,int r)
24 {
25     t[p].l = l; t[p].r = r; t[p].ml = y[l]; t[p].mr = y[r];
26     t[p].s = 0;
27     t[p].len = 0;
28     if(t[p].l+1 == t[p].r) return;
29     int mid = (l+r)>>1;
30     build(p*2,l,mid);
31     build(p*2+1,mid,r);
32 }
33
34 void callen(int p)
35 {
36     if(t[p].s > 0)
37         t[p].len = t[p].mr-t[p].ml;
38     else if(t[p].l == (t[p].r-1)) //所以s == 0的点，最终长度会被赋成0
39         t[p].len = 0;
40     else
41         t[p].len = t[p*2].len+t[p*2+1].len;
42 }
43
44 void change(int p,Line tmp)
45 {
46     if(t[p].ml == tmp.y1 && t[p].mr == tmp.y2)
47     {
48         t[p].s += tmp.flag;
49         callen(p);
50         return;
51     }
52     if(tmp.y2 <= t[p*2].mr) change(p*2,tmp);
53     else if(tmp.y1 >= t[p*2+1].ml) change(p*2+1,tmp);
54     else
55     {
56         Line tp = tmp;
57         tp.y2 = t[p*2].mr;

```

```

58         change(p*2,tp);
59         tp = tmp;
60         tp.y1 = t[p*2+1].ml;
61         change(p*2+1,tp);
62     }
63     callen(p);
64 }
65
66 int main()
67 {
68     int cnt = 1;
69     while(scanf("%d",&n) && n!=0)
70     {
71         int num = 1;
72         double x1,x2,y1,y2;
73         for(int i = 0;i < n;i++)
74         {
75             scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);
76             line[num].x = x1; line[num].y1 = y1; line[num].y2 = y2;
77             line[num].flag = 1;
78             //保存的是线, flag == 1 表示该线在左边
79             y[num++] = y1; //将所有的y都读入数组中, 进行离散化
80             line[num].x = x2; line[num].y1 = y1; line[num].y2 = y2;
81             line[num].flag = -1;
82             //flag == -1 表示该线在右边
83             y[num++] = y2;
84         }
85         sort(line+1,line+num,cmp); //按照横坐标进行排序
86         //对纵坐标进行离散化
87         sort(y+1,y+num);
88         int cm = unique(y+1,y+num)-y-1;
89         //在y轴上建立线段树
90         build(1,1,cm);
91         change(1,line[1]);
92         double ans = 0;
93         for(int i = 2;i < num;i++)
94         {
95             ans += t[1].len*(line[i].x-line[i-1].x);
96             change(1,line[i]);
97         }
98         printf("Test case #%d\n",cnt++);
99         printf("Total explored area: %.2f\n\n",ans);
100    }
101    return 0;
102 }

```

4.10.2 面积交

只要矩形一条线一条线扫描的时候，右边的线没有被读进来，则右边的线一直在扫描面积

举个简单的例子，本题中的扫描线只是一个消除矩阵面积重复计算部分的一个方法

```

1 #include <iostream>
2 #include <cstdio>
3 #include <algorithm>
4 using namespace std;
5 const int SIZE = 300+10;

```

```

6 struct Line{
7     double x,y1,y2;
8     int flag;
9 }line[SIZE];
10
11 bool cmp(Line a,Line b)
12 {
13     return a.x<b.x;
14 }
15 struct tree{
16     int l,r;
17     double ml,mr;
18     int s; //s始终 >= 0
19     double len;
20 }t[SIZE*4];
21 int n;
22 double y[SIZE];
23 void build(int p,int l,int r)
24 {
25     t[p].l = l; t[p].r = r; t[p].ml = y[l]; t[p].mr = y[r];
26     t[p].s = 0;
27     t[p].len = 0;
28     if(t[p].l+1 == t[p].r) return;
29     int mid = (l+r)>>1;
30     build(p*2,l,mid);
31     build(p*2+1,mid,r);
32 }
33
34 void callen(int p)
35 {
36     if(t[p].s > 0)
37         t[p].len = t[p].mr-t[p].ml;
38     else if(t[p].l == (t[p].r-1)) //所以s == 0的点，最终长度会被赋成0
39         t[p].len = 0;
40     else
41         t[p].len = t[p*2].len+t[p*2+1].len;
42 }
43
44 void change(int p,Line tmp)
45 {
46     if(t[p].ml == tmp.y1 && t[p].mr == tmp.y2)
47     {
48         t[p].s += tmp.flag;
49         callen(p);
50         return;
51     }
52     if(tmp.y2 <= t[p*2].mr) change(p*2,tmp);
53     else if(tmp.y1 >= t[p*2+1].ml) change(p*2+1,tmp);
54     else
55     {
56         Line tp = tmp;
57         tp.y2 = t[p*2].mr;
58         change(p*2,tp);
59         tp = tmp;
60         tp.y1 = t[p*2+1].ml;
61         change(p*2+1,tp);
62     }
63     callen(p);
64 }

```

```

65
66 int main()
67 {
68     int cnt = 1;
69     while(scanf("%d",&n) && n!=0)
70     {
71         int num = 1;
72         double x1,x2,y1,y2;
73         for(int i = 0;i < n;i++)
74         {
75             scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);
76             line[num].x = x1; line[num].y1 = y1; line[num].y2 = y2;
77             line[num].flag = 1;
78             //保存的是线, flag == 1 表示该线在左边
79             y[num++] = y1; //将所有的y都读入数组中, 进行离散化
80             line[num].x = x2; line[num].y1 = y1; line[num].y2 = y2;
81             line[num].flag = -1;
82             //flag == -1 表示该线在右边
83             y[num++] = y2;
84         }
85         sort(line+1,line+num,cmp); //按照横坐标进行排序
86         //对纵坐标进行离散化
87         sort(y+1,y+num);
88         int cm = unique(y+1,y+num)-y-1;
89         //在y轴上建立线段树
90         build(1,1,cm);
91         change(1,line[1]);
92         double ans = 0;
93         for(int i = 2;i < num;i++)
94         {
95             ans += t[1].len*(line[i].x-line[i-1].x);
96             change(1,line[i]);
97         }
98         printf("Test case #%d\n",cnt++);
99         printf("Total explored area: %.2f\n\n",ans);
100    }
101    return 0;
102 }

```

4.10.3 周长并

题意：

给一堆矩阵，求出所有矩阵拼起来，求出矩阵并起来的总周长。

思路：

其实与面积并差不多，就是求 ans 的时候， $ans += abs(last-t[1].len)$ ，每次插入一根线段，对答案的贡献值为使得 $t[1].len$ 增加或减少的长度。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #include <cmath>
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 using namespace std;
8 const int N = 10000+100;
9

```

```

10 int n;
11 struct Line{
12     int x,y1,y2;
13     int flag;
14 }s1[N],s2[N];
15
16 bool cmp(Line a,Line b)
17 {
18     return a.x < b.x;
19 }
20
21 struct Tree{
22     int l,r,s;
23     int ml,mr,len;
24 }t[N*4];
25
26 int y[4][N],ans;
27 int num1,num2;
28
29 void build(int p,int l,int r,int idx)
30 {
31     t[p].l = l, t[p].r = r, t[p].s = 0, t[p].ml = y[idx][l], t[p].mr = y[idx][r];
32     if(l == (r-1)) return;
33     int mid = (l+r)>>1;
34     build(p*2,l,mid,idx);
35     build(p*2+1,mid,r,idx);
36 }
37
38 void calc(int p)
39 {
40     if(t[p].s >= 1) t[p].len = t[p].mr-t[p].ml;
41     else if(t[p].l == (t[p].r-1)) t[p].len = 0;
42     else{
43         t[p].len = t[p*2].len+t[p*2+1].len;
44     }
45 }
46
47 void update(int p, Line tp)
48 {
49     if(tp.y1 <= t[p].ml && t[p].mr <= tp.y2)
50     {
51         t[p].s += tp.flag;
52         calc(p);
53         return;
54     }
55     if(t[p*2].mr >= tp.y2) update(p*2,tp);
56     else if(t[p*2+1].ml <= tp.y1) update(p*2+1,tp);
57     else{
58         update(p*2,tp);
59         update(p*2+1,tp);
60     }
61     calc(p);
62 }
63
64 int main()
65 {
66     while(~scanf("%d",&n))
67     {
68         num1 = 0, num2 = 0, ans = 0;

```

```

69         rep(i,1,n)
70     {
71         int x1,y1,x2,y2;
72         scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
73         s1[+num1].x = x1, s1[num1].y1 = y1, s1[num1].y2 = y2, y[1][num1] = y1, s1[
74             num1].flag = 1;
75             s1[+num1].x = x2, s1[num1].y1 = y1, s1[num1].y2 = y2, y[1][num1] = y2, s1[
76                 num1].flag = -1;
77
78         s2[+num2].x = y1, s2[num2].y1 = x1, s2[num2].y2 = x2, y[2][num2] = x1, s2[
79             num2].flag = 1;
80             s2[+num2].x = y2, s2[num2].y1 = x1, s2[num2].y2 = x2, y[2][num2] = x2, s2[
81                 num2].flag = -1;
82         }
83
84         sort(s1+1,s1+1+num1,cmp);
85         sort(s2+1,s2+1+num2,cmp);
86         sort(y[1]+1,y[1]+1+num1);
87         sort(y[2]+1,y[2]+1+num2);
88
89         int scr1 = unique(y[1]+1,y[1]+1+num1)-y[1]-1;
90         int scr2 = unique(y[2]+1,y[2]+1+num2)-y[2]-1;
91
92         build(1,1,scr1,1);
93         rep(i,1,num1)
94     {
95         int last = t[1].len;
96         update(1,s1[i]);
97         ans += abs(last-t[1].len);
98     }
99
100    build(1,1,scr2,2);
101    rep(i,1,num2)
102    {
103        int last = t[1].len;
104        update(1,s2[i]);
105        ans += abs(last-t[1].len);
106    }
107    printf("%d\n",ans);
108 }
109 return 0;
110 }
```

4.10.4 包星星问题

题意：

给出一大堆星星的坐标，给出每个星星的亮度。然后给出一个矩形，要求用这个矩形包住的星星的亮度最大。注意：如果星星在矩形边界上，则不计算这个星星的亮度。

思路：

我们来思考一下，一个矩形的位置是不是由这个矩形右上角这个点所决定的，所以我们可以把考虑矩形的位置改为考虑右上角这个点所在的位置。

然后我们可以发现，对于一颗星星， (x,y) ，只要右上角这个点在 $(x+0.1, y+0.1) \sim (x+w-0.1, y+h-0.1)$ 这个范围内，即可包住这颗星星。此处取 0.1 的原因是星星不能在矩形边界上。

因此一个星星就可以确定一个矩形，那么本题就变成了给出一大堆矩形，每个矩形都有一个权值，问其中哪一个区域矩形值之和最大。

因此我们可以将每个矩形的左右边界抽离出来，然后就变成了区间覆盖问题。

询问在线段树维护下的这根扫描线上亮度最大的值是多少，所以线段树上只需要维护一个最大值，再加上一个 lazy 标记，然后边插入边，边更新 ans，就可以通过此题。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define rep(i,a,b) for(int i = a; i <= b; i++)
6 using namespace std;
7 const int N = 100000;
8
9 struct Line{
10     double x,y1,y2;
11     int flag;
12 }line[N];
13
14 bool cmp(Line a,Line b)
15 {
16     return a.x < b.x;
17 }
18
19 int n,w,h,num,ans;
20 double y[N];
21
22 struct Tree{
23     int l,r,lazy;
24     double ml,mr;
25     int maxn;
26 }t[N*4];
27
28 void build(int p,int l,int r)
29 {
30     t[p].l = l, t[p].r = r, t[p].ml = y[l], t[p].mr = y[r], t[p].maxn = 0, t[p].lazy = 0;
31     if(l == r) return;
32     int mid = (l+r)>>1;
33     build(p*2,l,mid);
34     build(p*2+1,mid+1,r);
35 }
36
37 void pushup(int p)
38 {
39     if(t[p].lazy != 0)
40     {
41         t[p*2].lazy += t[p].lazy;
42         t[p*2+1].lazy += t[p].lazy;
43         t[p*2].maxn += t[p].lazy;
44         t[p*2+1].maxn += t[p].lazy;
45         t[p].lazy = 0;
46     }
47 }
48
49 void change(int p, Line a)
50 {
51 // cout << a.y1 << " " << a.y2 << endl;
52     if(a.y1 <= t[p].ml && t[p].mr <= a.y2)

```

```

53     {
54     // t[p].s += a.flag;
55     t[p].lazy += a.flag;
56     t[p].maxn += a.flag;
57     return;
58   }
59   pushup(p);
60   if(t[p*2].mr >= a.y2) change(p*2,a);
61   else if(t[p*2+1].ml <= a.y1) change(p*2+1,a);
62   else{
63     change(p*2,a);
64     change(p*2+1,a);
65   }
66   t[p].maxn = max(t[p*2].maxn,t[p*2+1].maxn);
67 }
68
69 int main()
70 {
71   while(~scanf("%d%d%d",&n,&w,&h))
72   {
73     ans = 0, num = 0;
74     rep(i,1,n)
75     {
76       double x1,y1,z1;
77       scanf("%lf%lf%lf",&x1,&y1,&z1);
78       line[+num].x = x1+0.1, line[num].y1 = y1+0.1, line[num].y2 = y1+h-0.1, y[
79       num] = y1+0.1, line[num].flag = z1;
80       line[+num].x = x1+w-0.1, line[num].y1 = y1+0.1, line[num].y2 = y1+h-0.1, y
81       [num] = y1+h-0.1, line[num].flag = -z1;
82     }
83     sort(line+1,line+1+num,cmp);
84     sort(y+1,y+1+num);
85     int scr = unique(y+1,y+1+num)-y-1;
86     build(1,1,scr);
87
88     rep(i,1,num)
89     {
90       change(1,line[i]);
91       ans = max(ans,t[1].maxn);
92     }
93     printf("%d\n",ans);
94   }
95   return 0;
96 }
```

4.10.5 覆盖奇数次的面积

题意：

给出 n 个矩形，求被覆盖区域为奇数次的总面积。

思路：

扫描线有很多种写法，可以打 lazy 更新到底，也可以不打 lazy，只是单纯对目标边进行更新，然后再区间合并上去。

本题问的是被覆盖区域为奇数次的总面积。因此线段树每个节点记录被覆盖的次数，被覆盖奇数次的长度，被覆盖偶数次的长度。每次加入一条边，只对被覆盖的那个最大的区间，覆盖次数 +1，对于该区间下面的区间不再进行更新，之后也不会更新。

因此假如第一次加入的线段是 (1,4)，第二次加入的线段是 (1,2)，因此第一次 (1,4) 区间覆盖次数变为 1，第二次

(1,2) 区间覆盖次数变为 1。然后区间合并的时候， len_1 表示覆盖奇数次的长度， len_2 表示覆盖偶数次的长度，(1,2) 区间 $len_1 = 1$, $len_2 = 0$. (1,4) 区间覆盖次数为奇数，因此 $len_1 = (1,2)$ 与 (3,4) 区间被覆盖偶数次的长度，因为偶 + 奇 = 奇，而 $len_2 = (1,2)$ 与 (3,4) 区间被覆盖奇数次的长度。因此可以发现，虽然在加入直线的时候，没有将更新次数一次性更新到底，但是在区间合并的时候，会将之前覆盖的长度一并算入。

由于每次插入直线的时候，最后都会合并到整根扫描线上，因此询问整根扫描线的奇偶长度得到的答案是正确的。但是如果询问某一个区间被覆盖奇数次的长度则会得到错误答案，因为这个区间被覆盖的次数还取决于这个区间之上的区间。

```

50     // t[p].len1 = t[p].mr-t[p].ml-t[p*2].len2-t[p*2+1].len2;
51     t[p].len1 = t[p*2].len1+t[p*2+1].len1;
52     t[p].len2 = t[p*2].len2+t[p*2+1].len2;
53 }
54 }
55 }
56
57 void change(int p, Line tp)
58 {
59     if(tp.y1 <= t[p].ml && tp.y2 >= t[p].mr)
60     {
61         t[p].s += tp.flag;
62         calc(p);
63         return;
64     }
65     if(t[p*2].mr >= tp.y2) change(p*2,tp);
66     else if(t[p*2+1].ml <= tp.y1) change(p*2+1,tp);
67     else{
68         change(p*2,tp);
69         change(p*2+1,tp);
70     }
71     calc(p);
72 }
73
74 int main()
75 {
76     ans = 0, num = 0;
77     scanf("%d",&n);
78     rep(i,1,n){
79         ll x1,x2,y1,y2;
80         scanf("%lld%lld%lld%lld",&x1,&y1,&x2,&y2);
81         line[++num].x = x1, line[num].y1 = y1, line[num].y2 = y2, line[num].flag = 1;
82         y[num] = y1;
83         line[++num].x = x2, line[num].y1 = y1, line[num].y2 = y2, line[num].flag = -1;
84
85         y[num] = y2;
86     }
87     sort(line+1,line+1+num,cmp);
88     sort(y+1,y+1+num);
89     int sc = unique(y+1,y+num+1)-y-1;
90     build(1,1,sc);
91     change(1,line[1]);
92     rep(i,2,num)
93     {
94         // printf("%f\n",t[1].len2);
95         ans += t[1].len2*(line[i].x-line[i-1].x);
96         change(1,line[i]);
97     }
98     cout << ans << endl;
99     return 0;
}

```

4.11 主席树

4.11.1 静态主席树

新建多个权值线段树副本，记录只考虑 1-i 个数时，每个数出现在各个区间的个数是多少，类似于建多棵权值线段树然后第 i 棵线段树，参考第 i-1 棵线段树，优化空间空间为 $4*n*log(n)$

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define rep(i,a,b) for(int i = a; i <= b; i++)
6 using namespace std;
7 const int N = 1e5+100;
8
9 struct Tree{
10     int lc,rc; //左右节点 t数组 编号
11     int l,r; //节点的左右端点
12     int sum;
13 }t[N*20];
14 int n,m,a[N],num,b[N],root[N],tot;
15
16 int build(int l,int r)
17 {
18     int p = ++tot; // 新建一个节点，编号为p，代表当前区间[l,r]
19     t[p].l = l, t[p].r = r, t[p].sum = 0;
20     if(l == r) return p;
21     int mid = (l+r)>>1;
22     t[p].lc = build(l,mid);
23     t[p].rc = build(mid+1,r);
24     return p;
25 }
26
27 int insert(int now,int pos,int k)
28 {
29     int p = ++tot;
30     t[p] = t[now]; //建立副本
31     if(t[p].l == t[p].r){
32         t[p].sum += k; //在副本上修改
33         return p;
34     }
35     int mid = (t[p].l+t[p].r)>>1;
36     if(pos <= mid) t[p].lc = insert(t[p].lc,pos,k); //保留右儿子部分，把左儿子更新
37     else t[p].rc = insert(t[p].rc,pos,k);
38     t[p].sum = t[t[p].lc].sum + t[t[p].rc].sum;
39     return p;
40 }
41
42 int ask(int lp,int rp,int k) //lp和rp所代表的区间是相同的，他们只不过是在不同状态下的副本
43 {
44     if(t[lp].l == t[lp].r) return t[lp].l; //找到答案
45     int cnt = t[t[rp].lc].sum-t[t[lp].lc].sum; // 值在[l,mid]中的数有多少个
46     if(cnt >= k) return ask(t[lp].lc,t[rp].lc,k);
47     else return ask(t[lp].rc,t[rp].rc,k-cnt);
48 }
49
50 int main()
51 {
52     num = tot = 0;
53     scanf("%d%d",&n,&m);
54     rep(i,1,n){
55         scanf("%d",&a[i]);
56         b[++num] = a[i];
57     }
58     sort(b+1,b+1+num); //离散化
59     num = unique(b+1,b+1+num)-b-1;

```

```

60     root[0] = build(1,num); //root[0]这颗树是一棵空树，关于离散化后的值域建树
61     rep(i,1,n)
62     {
63         int x = lower_bound(b+1,b+1+num,a[i])-b; //离散化后的值
64         root[i] = insert(root[i-1],x,1); //值为x的数增加1个
65     }
66     rep(i,1,m)
67     {
68         int x,y,z;
69         scanf("%d%d%d",&x,&y,&z);
70         int ans = ask(root[x-1],root[y],z);
71         printf("%d\n",b[ans]); //从离散化后的值变回原值
72     }
73     return 0;
74     //root[i]:表示只考虑1-i这些数时候建树的情况，这颗树树根的编号
75 }
```

4.11.2 区间中不同数的个数

题意：

长度为 n 的序列， q 次询问，每次给出 l 、 r ，返回序列 $[l, r]$ 中不同数的个数。 $(1 \leq n \leq 3 * 10^4, 1 \leq q \leq 2 * 10^5)$

思路：

与之前主席树的权值线段树思路不同，此题的思路是建立 n 颗线段树，第 i 颗线段树存储区间 $[1, i]$ 的信息。其中每个节点维护 sum ，表示节点对应区间中数的个数，因此每棵线段树中只保留每个数最后出现的位置。

举个例子，序列为 5 5 5 5 5，则第 1 颗线段树中只有第一个位置 sum 为 1，然后第二颗线段树从第一颗线段树继承过来，由于 5 这个数字之前出现过，因此在第二颗线段树中令第一个位置的 sum 为 0，令第二个位置的 sum 为 1，来保存每个数字最后出现的位置。

因此查询区间 $[l, r]$ 时，就在第 r 颗线段树中查询区间 $[l, n]$ 中数的个数即可。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
8 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
9 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
10 << " , " << z1 << ":" << z2 << endl;
11 typedef long long ll;
12 typedef double db;
13 const int N = 3*1e4+100;
14 const int M = 1e6+100;
15 const db EPS = 1e-9;
16 using namespace std;
17
18 int n,a[N],q;
19 int ls[40*N],rs[40*N],sum[40*N],root[N],sz;
20 int pos[M]; //记录每个数的位置
21 void build(int &now,int l,int r){
22     if(!now) now = ++sz;
23     if(l == r){
```

```

24         sum[now] = 0; return;
25     }
26     build(ls[now],l,(l+r)>>1);
27     build(rs[now],((l+r)>>1)+1,r);
28     sum[now] = sum[ls[now]]+sum[rs[now]];
29 }
30
31 void update(int x,int &now,int l,int r,int p1,int ct){
32     if(!now) now = ++sz;
33     if(l == r){
34         sum[now] = sum[x]+ct; return;
35     }
36     int mid = (l+r)>>1;
37     if(p1 <= mid){
38         if(!rs[now]) rs[now] = rs[x]; //继承前一个备份的节点
39         if(ls[now] <= now) ls[now] = 0; //因为现在要修改ls[now]，因此要重新开空间
40         //如果ls[now] <= now，表示这个节点是之前继承的，因此要赋为0，重新开空间
41         update(ls[x],ls[now],l,mid,p1,ct);
42     }
43     else{
44         if(!ls[now]) ls[now] = ls[x];
45         if(rs[now] <= now) rs[now] = 0;
46         update(rs[x],rs[now],mid+1,r,p1,ct);
47     }
48     sum[now] = sum[ls[now]]+sum[rs[now]];
49 }
50
51 int ask(int &now,int l,int r,int p1){
52     if(!now) now = ++sz;
53     if(l >= p1) return sum[now];
54     else if(r < p1) return 0;
55     int mid = (l+r)>>1, ans = 0;
56     if(mid >= p1) ans += ask(ls[now],l,mid,p1)+sum[rs[now]];
57     else ans += ask(rs[now],mid+1,r,p1);
58     return ans;
59 }
60
61 int main()
62 {
63     scanf("%d",&n);
64     rep(i,1,n) scanf("%d",&a[i]);
65     build(root[0],1,n);
66     rep(i,1,n){
67         if(pos[a[i]]){
68             update(root[i-1],root[i],1,n,pos[a[i]],-1);
69             update(root[i-1],root[i],1,n,i,1);
70             pos[a[i]] = i;
71         }
72         else update(root[i-1],root[i],1,n,i,1), pos[a[i]] = i;
73     }
74     scanf("%d",&q);
75     rep(i,1,q){
76         int xx,yy; scanf("%d%d",&xx,&yy);
77         printf("%d\n",ask(root[yy],1,n,xx));
78     }
79     return 0;
80 }

```

4.11.3 单点修改主席树

题意：

给定一个区间，求这个区间第 k 小的数，支持单点修改。

思路：

动态主席树裸题。

我们先来回顾一下静态主席树的做法，对于数组中每一个位置都维护一棵权值线段树，该权值线段树保存的是区间 $[1, x]$ 的信息。因此我想要求区间 $[l, r]$ 之间第 k 大的时候，只需要将 $\text{root}[r]-\text{root}[l-1]$ 就是维护区间 $[l, r]$ 信息的权值线段树，因此就可以快速直接求出这个区间中第 k 大的元素是多少。

现在我们来看看单点修改的操作。

如果我现在要将 $a[pos]$ 修改为 x ，那么最暴力的做法就是对于 $\text{root}[pos] \sim \text{root}[n]$ 中的每一颗权值线段树都进行修改，即将 $a[pos]$ 这个点的值减 1，将 x 这个点的值 +1。

最暴力的做法显然是无法通过此题的，因此我们可以想到有没有一种 $\log n$ 的方法，可以只修改 $\log n$ 个节点，就可以对于每一个线段树记录修改信息，于是我们想到了树状数组。

我们来回忆一下树状数组，每一个节点记录区间 $[x-\text{lowbit}(x)+1, x]$ 的所有信息，因此当需要求 $[1, x]$ 内维护的信息的时候，只需要从节点 x 出发，每次进行 $x = \text{lowbit}(x)$ 的操作，即可求出 $[1, x]$ 内维护的所有信息。

每次对 x 节点进行修改的时候，只需要不断进行 $x += \text{lowbit}(x)$ 的操作，就可以访问到所有存储 x 节点信息的节点，因此实现了 $\log n$ 的查询。

因此本题也维护一个树状数组。

节点 x 维护的是区间 $[x-\text{lowbit}(x)+1, x]$ 的权值线段树，因此当我需要访问 $[l, r]$ 区间信息的时候，只需要将 $\text{root}[r]-\text{root}[l-1]+\text{getsum}(r)-\text{getsum}(l-1)$ 这里面维护的便是区间 $[l, r]$ 的权值线段树。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
8 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
9 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
     << " , " << z1 << ":" << z2 << endl;
10 typedef long long ll;
11 typedef double db;
12 const int N = 6*1e4+100;
13 const int M = 32*N+100;
14 const db EPS = 1e-9;
15 using namespace std;
16
17 int rt[N],root[N],ls[M],rs[M],sz,n,m,tot,h1,h2,L[N],R[N],sum[M],a[N],b[N];
18 struct Node{
19     int op,xx,yy,k; //保存修改信息
20 }t[N];
21
22 int find(ll x) {return (lower_bound(b+1,b+1+tot,x)-b);}
23 int lowbit(int x) {return x&(-x);}
24
25 void update(int pre,int &now,int l,int r,int pos,int c){

```

```

26     if(!now) now = ++sz;
27     sum[now] = sum[pre]+c;
28     if(l == r) return;
29     int mid = (l+r)>>1;
30     if(pos <= mid){rs[now] = rs[pre]; update(ls[pre],ls[now],l,mid,pos,c);}
31     else{ls[now] = ls[pre]; update(rs[pre],rs[now],mid+1,r,pos,c);}
32 }
33
34 int query(int pre,int now,int l,int r,int k){
35     int sum1 = 0, sum2 = 0;
36     if(l == r) return l;
37     rep(i,1,h1) sum1 += sum[ls[L[i]]]; //左边树状数组节点累加值
38     rep(i,1,h2) sum2 += sum[ls[R[i]]]; //右边树状数组节点累加值
39     int temp = sum[ls[now]]-sum[ls[pre]]+sum2-sum1;
40     int mid = (l+r)>>1;
41     if(temp >= k){
42         rep(i,1,h1) L[i] = ls[L[i]];
43         rep(i,1,h2) R[i] = ls[R[i]];
44         return query(ls[pre],ls[now],l,mid,k);
45     }
46     else{
47         rep(i,1,h1) L[i] = rs[L[i]];
48         rep(i,1,h2) R[i] = rs[R[i]];
49         return query(rs[pre],rs[now],mid+1,r,k-temp);
50     }
51 }
52
53 void init(){
54     scanf("%d%d",&n,&m);
55     rep(i,1,n){
56         scanf("%d",&a[i]); b[++tot] = a[i];
57     }
58     rep(i,1,m){
59         char op[10]; scanf("%s",op);
60         if(op[0] == 'Q'){
61             t[i].op = 1; scanf("%d%d%d",&t[i].xx,&t[i].yy,&t[i].k);
62         }
63         else{
64             t[i].op = 2; scanf("%d%d",&t[i].xx,&t[i].yy);
65             b[++tot] = t[i].yy;
66         }
67     }
68     sort(b+1,b+1+tot);
69     tot = unique(b+1,b+1+tot)-b-1;
70     rep(i,1,n)
71         update(rt[i-1],rt[i],1,tot,find(a[i]),1);
72 }
73
74 void solve(){
75     rep(i,1,m){
76         if(t[i].op == 1){ //查询
77             h1 = h2 = 0;
78             for(int j = t[i].xx-1; j; j -= lowbit(j)) L[++h1] = root[j]; //记录树状数组要
计算的节点
79             for(int j = t[i].yy; j; j -= lowbit(j)) R[++h2] = root[j];
80             int pos = query(rt[t[i].xx-1],rt[t[i].yy],1,tot,t[i].k);
81             printf("%d\n",b[pos]);
82         }
83         else{ //修改

```

```

84         int pos1 = find(a[t[i].xx]), pos2 = find(t[i].yy);
85         for(int j = t[i].xx; j <= n; j += lowbit(j)) update(root[j],root[j],1,tot,
86             pos1,-1);
87         for(int j = t[i].xx; j <= n; j += lowbit(j)) update(root[j],root[j],1,tot,
88             pos2,1);
89         a[t[i].xx] = t[i].yy;
90     }
91 }
92 int main()
93 {
94     int _; scanf("%d",&_);
95     while(_--){
96         init();
97         solve();
98         rep(i,0,sz) ls[i] = rs[i] = sum[i] = 0;
99         rep(i,0,n) root[i] = rt[i] = 0;
100        sz = 0, tot = 0;
101    }
102    return 0;
103 }
```

4.12 李超树

4.12.1 李超树模板题

题意：

两种操作。第一种操作查询第 T 天的最大收益，第二种操作给出一个设计方案，该方案第一天收益为 S ，之后每一天比上一天多收益 P ，注意设计方案收益不可叠加。 $(1 \leq T \leq 5 * 10^4)$

思路：

这是一道模板题，每次动态插入一条直线，单点查询最大值即可。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 typedef long long ll;
4 typedef double db;
5 const int N = 5*1e4+100;
6 const int M = 1e5+100;
7 const db EPS = 1e-12;
8 using namespace std;
9
10 struct line{
11     db k,b;
12     int l,r,flag;
13     line() {}
14     line(db a1,db a2,int l1,int r1,int f1) : k(a1),b(a2),l(l1),r(r1),flag(f1) {}
15 }sgt[2*N];
16 int n;
17
18 inline int get_id(int l,int r) {return (l+r)|(l!=r);}
19 //计算某条线段在某一个横坐标的纵坐标值
20 inline db calc(line a,int pos) {return a.k*pos+a.b;}
21
22 void modify(int l,int r,line k){
23     int now = get_id(l,r);
24     if(k.l <= l && k.r >= r){
```

```

25         if(!sgt[now].flag) sgt[now] = k, sgt[now].flag = 1; //原区间没有优势线段
26         else if(calc(k,l)-calc(sgt[now],l) > EPS && calc(k,r)-calc(sgt[now],r) > EPS)
27             sgt[now] = k; //原区间优势线段被覆盖
28         else if(calc(k,l)-calc(sgt[now],l) > EPS || calc(k,r)-calc(sgt[now],r) > EPS){
29             //有一端比原线段大
30             int mid = (l+r)>>1;
31             //中点坐标处，新线段比原来优势线段更优
32             if(calc(k,mid)-calc(sgt[now],mid) > EPS) swap(k,sgt[now]);
33             if(calc(k,l)-calc(sgt[now],l) > EPS) modify(l,mid,k);
34             else modify(mid+1,r,k);
35         }
36     }
37     else{
38         int mid = (l+r)>>1;
39         if(k.l <= mid) modify(l,mid,k); //涉及了左区间
40         if(mid < k.r) modify(mid+1,r,k); //涉及了右区间
41     }
42 }
43 db query(int l,int r,int x){
44     int now = get_id(l,r);
45     if(l == r) return calc(sgt[now],x);
46     else{
47         int mid = (l+r)>>1;
48         db ans = calc(sgt[now],x);
49         if(x <= mid) return max(ans,query(l,mid,x));
50         else return max(ans,query(mid+1,r,x));
51     }
52 }
53 int main()
54 {
55     scanf("%d",&n);
56     rep(i,1,n){
57         char op[10]; scanf("%s",op);
58         if(op[0] == 'P'){
59             db k,b; scanf("%lf%lf",&b,&k);
60             b = b-k;
61             line tmp(k,b,1,5*1e4,1);
62             modify(1,5*1e4,tmp);
63         }
64         else{
65             int x; scanf("%d",&x);
66             printf("%lld\n", (ll)floor(query(1,50000,x)/100));
67         }
68     }
69     return 0;
70 }

```

4.12.2 离散化 + 维护最大最小直线

题意：

现有 n 个机器人，第 i 个机器人初始在 a_i 位置。一共有两个操作。第一种操作为 T_i command $k_i x_i$ ，表示 T_i 时间，第 k_i 个机器人以 x_i 的速度开始移动， x_i 可以为负值。第二种操作为 T_i query，表示查询 T_i 时，离原点最远的机器人的坐标。 $(0 \leq T_i \leq 10^9)$

思路：

对于每一个机器人来说，其速度变化都是一个分段函数，只需要先离线，将时间离散化，处理出每一个机器人各速度下

的分段直线。然后在第一种操作时，动态插入一条直线，第二种操作时，单点查询最大值与最小值即可完成此题。

本题主要特点就是对时间离散化，然后在离散化后的时间上插入直线。每一个节点需要同时维护最大值和最小值直线。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 typedef long long ll;
4 const int N = 1e5+100;
5 const int M = 6*1e5+100;
6 using namespace std;
7
8 int n,m,tot,RP[N];
9 ll a[N],b[M];
10 struct Query{
11     ll t,x; //x是速度
12     int k,op; //k是第k个机器人
13 }q[M];
14 struct NP{
15     ll t,x,pos,t2;
16     NP() {}
17     NP(ll _a,ll _b,ll _c,ll _d) : t(_a),x(_b),pos(_c),t2(_d) {}
18 };
19 vector<NP> hp[N];
20 struct line{
21     ll k,b;
22     int l,r;
23     line() {}
24     line(ll _a, ll _b, int _c, int _d) : k(_a), b(_b), l(_c), r(_d) {}
25 };
26 struct Node{
27     line minn,maxn;
28     int flag;
29     Node() {flag = 0;}
30 }sgt[2*M];
31
32 inline int get_id(int l,int r) {return (l+r)|(l!=r);}
33 //计算某线段在某一个横坐标的纵坐标值
34 inline ll calc(line a,ll pos) {return a.k*b[pos]+a.b;}
35 //离散化
36 int find(ll x) {return lower_bound(b+1,b+1+tot,x)-b;}
37
38 void modify(int l,int r,line k){
39     int now = get_id(l,r);
40     if(k.l <= l && k.r >= r){
41         if(!sgt[now].flag) sgt[now].minn = sgt[now].maxn = k, sgt[now].flag = 1;
42         else{
43             line tp = k;
44             if(calc(k,l) >= calc(sgt[now].maxn,l) && calc(k,r) >= calc(sgt[now].maxn,r))
45                 sgt[now].maxn = k;
46             else if(calc(k,l) > calc(sgt[now].maxn,l) || calc(k,r) > calc(sgt[now].maxn,r)){
47                 int mid = (l+r)>>1;
48                 //让非优势线段向下更新
49                 if(calc(k,mid) > calc(sgt[now].maxn,mid)) swap(k,sgt[now].maxn);
50                 //只需要判断两条直线左右端点大小
51                 if(calc(k,l) > calc(sgt[now].maxn,l)) modify(l,mid,k);
52                 else modify(mid+1,r,k);
53             }
54         }
55     }
56 }
```

```

53         k = tp;
54         if(calc(k,l) <= calc(sgt[now].minn,l) && calc(k,r) <= calc(sgt[now].minn,r))
55     ) sgt[now].minn = k;
56     else if(calc(k,l) < calc(sgt[now].minn,l) || calc(k,r) < calc(sgt[now].minn,
57 ,r)){
58         int mid = (l+r)>>1;
59         //让非优势线段向下更新
60         if(calc(k,mid) < calc(sgt[now].minn,mid)) swap(k,sgt[now].minn);
61         //只需要判断两条直线左右端点大小
62         if(calc(k,l) < calc(sgt[now].minn,l)) modify(l,mid,k);
63         else modify(mid+1,r,k);
64     }
65 }
66 else{
67     int mid = (l+r)>>1;
68     if(k.l <= mid) modify(l,mid,k);
69     if(k.r > mid) modify(mid+1,r,k);
70 }
71 }
72
73 ll Query(int l,int r,int x){
74     int now = get_id(l,r);
75     ll ans = max(calc(sgt[now].maxn,x),-calc(sgt[now].minn,x));
76     if(l == r) return ans;
77     else{
78         int mid = (l+r)>>1;
79         if(x <= mid) return max(ans,Query(l,mid,x));
80         else return max(ans,Query(mid+1,r,x));
81         return ans;
82     }
83 }
84
85 int main()
86 {
87     scanf("%d%d",&n,&m);
88     rep(i,1,n) scanf("%lld",&a[i]);
89     rep(i,1,n) hp[i].push_back({1,0,a[i],0});
90     b[++tot] = 0;
91     rep(i,1,m){
92         ll t; scanf("%lld",&t); b[++tot] = t;
93         char op[10]; scanf("%s",op+1);
94         if(op[1] == 'c'){
95             int k; ll x; scanf("%d%lld",&k,&x);
96             q[i].t = t; q[i].x = x; q[i].k = k; q[i].op = 0;
97         }
98         else q[i].op = 1, q[i].t = t;
99     }
100    tot = unique(b+1,b+1+tot)-b-1;
101    rep(i,1,m){
102        if(q[i].op == 0){
103            q[i].t = find(q[i].t);
104            int len = hp[q[i].k].size();
105            ll pos = hp[q[i].k][len-1].x*b[q[i].t]+hp[q[i].k][len-1].pos;
106            ll now = -b[q[i].t]*q[i].x+pos;
107            hp[q[i].k][len-1].t2 = q[i].t;
108            hp[q[i].k].push_back({q[i].t,q[i].x,now,0});
109        }
}

```

```

110     else q[i].t = find(q[i].t);
111 }
112 rep(i,1,n){
113     int len = hp[i].size();
114     hp[i][len-1].t2 = tot;
115 }
116 rep(i,1,n){
117     line tmp; tmp.l = hp[i][rp[i]].t;
118     tmp.r = hp[i][rp[i]].t2;
119     tmp.k = hp[i][rp[i]].x;
120     tmp.b = hp[i][rp[i]].pos;
121     rp[i]++;
122     modify(i,tot,tmp);
123 }
124 rep(i,1,m){
125     if(q[i].op == 0){
126         int xp = q[i].k;
127         line tmp; tmp.l = hp[xp][rp[xp]].t;
128         tmp.r = hp[xp][rp[xp]].t2;
129         tmp.k = hp[xp][rp[xp]].x;
130         tmp.b = hp[xp][rp[xp]].pos;
131         rp[xp]++;
132         modify(1,tot,tmp);
133     }
134     else{
135         ll ans = Query(1,tot,q[i].t);
136         printf("%lld\n",ans);
137     }
138 }
139 return 0;
140 }
```

4.12.3 李超树 + 两个 lazy

题意：

一共 N 个商店， $1 - N$ 依次编号。购买一件纪念品的花费为纪念品的花费加上税费。初始没有纪念品，税费为 0。现有三种操作。

- 1 $u \ v \ a \ b$: 新增纪念品，编号为 x 的商店，新增 $b + (x-u)*a$ 的商品， $u \leq x \leq v$
- 2 $u \ v \ a \ b$: 税费调整操作，表示编号为 x 的商店，税费增加 $b + (x-u)*a$ ， $u \leq x \leq v$
- 3 i : 询问操作，询问顾客在编号为 i 的商店，购买一件纪念品最高的花费是多少。

思路：

第一种操作就是李超树基础操作。第二种操作可以将 b 和 a 分开维护，对于 b 的维护只需打 *lazy* 区间更新即可，对于 a 的维护，我们可以这样思考，对于编号为 x 的商店，税费增加 $b + (x - u) * a = (b - u * a) + x * a$ ，因此我们将 $b - u * a$ 统一维护，再单独维护 a 。最后每个点的税费即为 $x * sum(a) + sum(b)$ 。

此题还需注意将物品离散化，维护线段的同时维护两个 sum ，两个 *lazy*，每次对税费和物品分开求最大值，累加到答案即可。当然维护 sum 的过程也可以用树状数组完成。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 typedef long long ll;
4 const int N = 6*1e5+100;
5 const int M = 1e5+100;
6 const ll inf = -5*1e18;
7 using namespace std;
8
```

```

9 int n,m;
10 struct line{
11     ll k,b;
12     int l,r;
13 };
14 struct Node{
15     line t;
16     int flag;
17     Node() {flag = 0;}
18 }sgt[2*N];
19 int b[N],tot;
20 ll sum1[2*N],sum2[2*N],lazy1[2*N],lazy2[2*N];
21 struct query{
22     int op; int u,v,a,b;
23 }q[N];
24 inline int get_id(int l,int r) {return (l+r)|(l!=r);}
25 //计算坐标
26 inline ll calc(line t,ll pos) {return (t.k*b[pos]+t.b);}
27
28 void modify(int l,int r,line k){
29     int now = get_id(l,r);
30     if(k.l <= l && r <= k.r){
31         if(!sgt[now].flag) sgt[now].flag = 1, sgt[now].t = k;
32         else if(calc(k,l) >= calc(sgt[now].t,l) && calc(k,r) >= calc(sgt[now].t,r)) sgt
33 [now].t = k;
34         else if(calc(k,l) > calc(sgt[now].t,l) || calc(k,r) > calc(sgt[now].t,r)){
35             int mid = (l+r)>>1;
36             if(calc(k,mid) > calc(sgt[now].t,mid)) swap(k,sgt[now].t);
37             if(calc(k,l) > calc(sgt[now].t,l)) modify(l,mid,k);
38             else modify(mid+1,r,k);
39         }
40     }else{
41         int mid = (l+r)>>1;
42         if(k.l <= mid) modify(l,mid,k);
43         if(k.r > mid) modify(mid+1,r,k);
44     }
45 }
46
47 void push_down(int l,int r){
48     int now = get_id(l,r), mid = (l+r)>>1, ls = get_id(l,mid), rs = get_id(mid+1,r);
49     if(lazy1[now]){
50         sum1[ls] += lazy1[now]; sum1[rs] += lazy1[now];
51         lazy1[ls] += lazy1[now]; lazy1[rs] += lazy1[now];
52         lazy1[now] = 0;
53     }
54     if(lazy2[now]){
55         sum2[ls] += lazy2[now]; sum2[rs] += lazy2[now];
56         lazy2[ls] += lazy2[now]; lazy2[rs] += lazy2[now];
57         lazy2[now] = 0;
58     }
59 }
60
61 void update(int l,int r,int L,int R,ll a,ll b){
62     int now = get_id(l,r);
63     if(L <= l && r <= R){
64         sum1[now] += a;
65         sum2[now] += b;
66         lazy1[now] += a;

```

```

67     lazy2[now] += b;
68     return;
69 }
70 push_down(l,r);
71 int mid = (l+r)>>1;
72 if(L <= mid) update(l,mid,L,R,a,b);
73 if(R > mid) update(mid+1,r,L,R,a,b);
74 sum1[now] = sum1[get_id(l,mid)]+sum1[get_id(mid+1,r)];
75 sum2[now] = sum2[get_id(l,mid)]+sum2[get_id(mid+1,r)];
76 }
77
78 ll query1(int l,int r,int pos){
79     int now = get_id(l,r);
80     if(l == r){
81         if((sgt[now].t).l <= pos && (sgt[now].t).r >= pos) return calc(sgt[now].t,pos);
82         else return inf;
83     }
84     int mid = (l+r)>>1;
85     ll ans = inf;
86     if((sgt[now].t).l <= pos && (sgt[now].t).r >= pos) ans = calc(sgt[now].t,pos);
87     if(pos <= mid) return max(query1(l,mid,pos),ans);
88     else return max(query1(mid+1,r,pos),ans);
89 }
90
91 pair<ll,ll> query2(int l,int r,int pos){
92     int now = get_id(l,r);
93     if(l == r) return make_pair(sum1[now],sum2[now]);
94     int mid = (l+r)>>1;
95     push_down(l,r);
96     if(pos <= mid) return query2(l,mid,pos);
97     else return query2(mid+1,r,pos);
98 }
99
100 int main()
101 {
102     scanf("%d%d",&n,&m);
103     rep(i,1,m){
104         scanf("%d",&q[i].op);
105         if(q[i].op != 3){
106             scanf("%d%d%d%d",&q[i].u,&q[i].v,&q[i].a,&q[i].b);
107             b[++tot] = q[i].u; b[++tot] = q[i].v;
108         }
109         else{
110             scanf("%d",&q[i].u);
111             b[++tot] = q[i].u;
112         }
113     }
114     sort(b+1,b+1+tot);
115     tot = unique(b+1,b+1+tot)-b-1;
116     rep(i,1,m){
117         if(q[i].op != 3){
118             q[i].u = lower_bound(b+1,b+1+tot,q[i].u)-b;
119             q[i].v = lower_bound(b+1,b+1+tot,q[i].v)-b;
120         }
121         else q[i].u = lower_bound(b+1,b+1+tot,q[i].u)-b;
122     }
123     rep(i,1,m){
124         if(q[i].op == 1){
125             line tmp; tmp.l = q[i].u, tmp.r = q[i].v;

```

```

126         tmp.k = q[i].a, tmp.b = (ll)q[i].b-(ll)q[i].a*(ll)b[q[i].u];
127         modify(1,tot,tmp);
128     }
129     else if(q[i].op == 2){
130         update(1,tot,q[i].u,q[i].v,q[i].a,(ll)q[i].b-(ll)q[i].a*(ll)b[q[i].u]);
131     }
132     else{
133         ll ans1 = query1(1,tot,q[i].u);
134         pair<ll, ll> ans2 = query2(1,tot,q[i].u);
135         if(ans1 == inf) printf("NA\n");
136         else{
137             ll ans = ans1+ans2.second+ans2.first*(ll)b[q[i].u];
138             printf("%lld\n",ans);
139         }
140     }
141 }
142 return 0;
143 }
```

4.12.4 树剖 + 李超树 + 区间直线最大值

题意：

一棵 n 个点的树，两个操作。操作 1，选择两个点 s 和 t ，对于该路径上的一个点 r ，在该点上添加的数字是 $a * dis + b$ ， dis 为点 r 到点 s 的距离。操作 2，给出两个点 s 和 t ，在 s 到 t 路径上选择一个点，再在这个点上选择一个数字，要求选的数字最小。每个点都有一个初始数字，为 123456789123456789。 $(n \leq 10^5, m \leq 10^5)$

思路：

树链剖分后维护李超树。第二种操作好处理，只需在线段树每个节点上维护一个最小值，每次插入直线都动态维护即可。第一种操作需要分类讨论， $lca = lca(s, t)$ 。对于 s 到 lca 上的节点 x ，每次添加数字为 $a * (dis[s] - dis[x]) + b$ ，因此 $dis[x]$ 为横坐标， k 为 $-a$ ， b 为 $a * dis[s] + b$ 。对于 lca 到 t 上的节点 x ，每次添加数字为 $a * (dis[s] + dis[x] - 2 * dis[lca]) + b$ ，因此 $dis[x]$ 为横坐标， k 为 a ， b 为 $a * (dis[s] - 2 * dis[lca]) + b$ 。

因此插入直线时对两种情况分类讨论插入直线即可完成。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 typedef long long ll;
4 const int N = 1e5+100;
5 const ll inf = 12345678912345678911;
6 using namespace std;
7
8 int n,m,tot,head[N],fa[N],d[N],son[N],siz[N],top[N],id[N],rk[N];
9 ll dis[N];
10 struct Edge{
11     int to,next;
12     ll w;
13 }e[2*N];
14 struct line { ll k,b; int l,r;};
15 struct Node{
16     line t;
17     ll mn;
18 }sgt[2*N];
19 inline ll calc(line t, ll pos) {return t.k*pos+t.b;}
20 inline int get_id(int l, int r) { return (l+r) | (l != r); } //将叶子结点赋成偶数，其余节点赋成奇数，编号最大为2n
21
22 void add(int x,int y,ll w){
```



```

76     }
77 }
78
79 int getlca(int x,int y){
80     while(top[x] != top[y]){
81         if(d[top[x]] < d[top[y]]) swap(x,y);
82         x = fa[top[x]];
83     }
84     return (d[x]<d[y]?x:y);
85 }
86
87 void modify(int x,int y,ll a,ll b){
88     int u = x, v = y, lca = getlca(x,y);
89     while(d[top[x]] > d[lca]){
90         line tmp = {-a,a*dis[u]+b,id[top[x]],id[x]};
91         insert(1,n,tmp);
92         x = fa[top[x]];
93     }
94     while(d[top[y]] > d[lca]){
95         line tmp = {a,a*(dis[u]-2ll*dis[lca])+b,id[top[y]],id[y]};
96         insert(1,n,tmp);
97         y = fa[top[y]];
98     }
99     if(x == lca){
100         line tmp = {a,a*(dis[u]-2ll*dis[lca])+b,id[x],id[y]};
101         insert(1,n,tmp);
102     }
103     else{
104         line tmp = {-a,a*dis[u]+b,id[y],id[x]};
105         insert(1,n,tmp);
106     }
107 }
108
109 ll getmin(int l,int r,int x,int y){
110     int now = get_id(l,r);
111     if(x <= l && r <= y) return sgt[now].mn;
112     else{
113         int mid = (l+r)>>1;
114         ll ans = min(calc(sgt[now].t,dis[rk[max(l,x)]]),calc(sgt[now].t,dis[rk[min(y,r)]]));
115         if(x <= mid) ans = min(ans,getmin(l,mid,x,y));
116         if(mid < y) ans = min(ans,getmin(mid+1,r,x,y));
117         return ans;
118     }
119 }
120
121 ll query(int u,int v){
122     ll ans = inf;
123     while(top[u] != top[v]){
124         if(d[top[u]] < d[top[v]]) swap(u,v);
125         ans = min(ans,getmin(1,n,id[top[u]],id[u]));
126         u = fa[top[u]];
127     }
128     if(d[u] < d[v]) ans = min(ans,getmin(1,n,id[u],id[v]));
129     else ans = min(ans,getmin(1,n,id[v],id[u]));
130     return ans;
131 }
132
133 int main()

```

```

134 {
135     tot = 1;
136     scanf("%d%d",&n,&m);
137     rep(i,1,n-1){
138         int u,v; ll w; scanf("%d%d%lld",&u,&v,&w);
139         add(u,v,w); add(v,u,w);
140     }
141     tot = 0; dfs1(1); dfs2(1,1);
142     build(1,n);
143     rep(i,1,m){
144         int op; scanf("%d",&op);
145         if(op == 1){
146             int s,t; ll a,b; scanf("%d%d%lld%lld",&s,&t,&a,&b);
147             modify(s,t,a,b);
148         }
149         else{
150             int s,t; scanf("%d%d",&s,&t);
151             printf("%lld\n",query(s,t));
152         }
153     }
154     return 0;
155 }
```

4.13 启发式合并

4.13.1 DSU 模板

适用问题:

树上启发式合并作为树上问题三剑客之一（点分治、长链剖分），以其优雅的暴力而闻名于江湖之中。

通常来说，如果一个问题可以被划分为一个个子树进行求解的问题，而且各个子儿子对答案的贡献容易添加与删除，就可以考虑使用树上启发式合并来求解。

本文主要介绍树上启发式合并的一些习题，可以从习题中仔细感受该算法的一系列特点。

算法介绍:

树上启发式合并需要两次 dfs ，第一次 dfs 进行重链剖分，第二次 dfs 进行求解。

通常有一个全局的数组用于信息记录。 dfs 之前，需要将这个数组赋初值。 dfs 时，先递归处理轻儿子，处理完之后清空轻儿子，最后再处理重儿子，处理完之后不清空。

计算完儿子的答案之后，再递归所有轻儿子，边递归边计算答案，并将轻儿子的信息添加到全局数组中。

这个做法的时间复杂度是 $O(n \log n)$ ，因为每个节点直接继承了其子树中的重儿子，即每次只有轻儿子会被重复访问，访问完之后，轻儿子即会和重儿子进行合并，每次合并 sz 至少乘 2，因此每个点最多被重复访问 $\log n$ 次，即总时间复杂度为 $O(n \log n)$ 。

算法模板:

其实树上启发式合并并没有什么模板，只需要处理好两次 dfs 的过程，然后实现插入、删除、更新三个函数即可。

以下面第一题的代码为例，给出一个大致的模板。

```

1 int sz[N],son[N];
2
3 void dfs1(int x){
4     /* 求解重儿子 */
5     sz[x] = 1;
6     for(int i = head[x]; i; i = e[i].next){
```

```

7     int y = e[i].to;
8     dfs1(y); sz[x] += sz[y];
9     if(sz[y] > sz[son[x]]) son[x] = y;
10    }
11 }
12
13 void Delete(int x){
14     /* 删除的内容 */
15     for(int i = head[x]; i; i = e[i].next) Delete(e[i].to);
16 }
17
18 void modify(int x,int fa){
19     /* 更新的内容 */
20     for(int i = head[x]; i; i = e[i].next) modify(e[i].to,fa);
21 }
22
23 void ins(int x){
24     /* 插入的内容 */
25     for(int i = head[x]; i; i = e[i].next) ins(e[i].to);
26 }
27
28 void dfs2(int x){
29     /* 求解轻儿子并清空 */
30     for(int i = head[x]; i; i = e[i].next)
31         if(e[i].to != son[x]) dfs2(e[i].to), Delete(e[i].to);
32
33     /* 求解重儿子并保留 */
34     if(son[x]) dfs2(son[x]);
35     /* 用重儿子更新答案 */
36
37     /* 枚举轻儿子更新答案，并加入轻儿子 */
38     for(int i = head[x]; i; i = e[i].next)
39         if(e[i].to != son[x]) modify(e[i].to,x), ins(e[i].to);
40
41     /* 用所有儿子更新答案 */
42 }

```

4.13.2 例题 1

题意：

给定一棵 n 个点的树，以及一张 k 个点的图。树中每一个节点都控制图中的一条边，问：对于树中每一个节点，将其子树（包括自己）中所有节点控制的边加到图中，图中连通块个数即为这个节点的答案。

思路：

树上启发式合并裸题，但是比赛的时候没有写过启发式合并以为这样写会 $T \dots$ 就没有尝试…

先具体讲一下这题的解法，再分析一下复杂度。‘用并查集维护图中的连通关系，对于当前节点，最后求其重儿子子树所形成的并查集，并将重儿子的并查集直接复用在求父节点上。对于非重儿子节点再对其子树中的边进行添加。

非常暴力的做法… 但是为什么能够快速通过呢… 我们先考虑一下序列上的启发式合并，由于每次选择将小的合并进大的，所以每个元素的大小每次至少 $\times 2$ ，因此最多被合并 $\log n$ 次就可以达到 n 的大小，所以复杂度是 $n \log n$ 。

而对于树上的启发式合并，每次将小的子树合并进大的子树中，因此子树的大小每次至少 $\times 2$ ，因此最多 $\log n$ 次子树的大小就可以达到 n ，因此复杂度也是 $n \log n$ 。

```

1 #include <cstdio>
2 #include <iostream>

```

```

3 #include <cstring>
4 #include <vector>
5 #include <algorithm>
6 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
7 #define rep(i,a,b) for(int i = a; i <= b; i++)
8 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
9 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
10 ;  

10 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2  

11 << " , " << z1 << ":" << z2 << endl;  

11 typedef long long ll;  

12 typedef double db;  

13 const int N = 1e5+100;  

14 const int M = 1e5+100;  

15 const db EPS = 1e-9;  

16 using namespace std;  

17  

18 int sz[N],n,k,l[N],r[N],ans[N],f[N],num;  

19 vector<int> p[N];  

20  

21 int find(int x){  

22     return x == f[x] ? x : (f[x] = find(f[x]));  

23 }  

24  

25 void clear(int x){  

26     f[l[x]] = l[x], f[r[x]] = r[x];  

27     for(int u : p[x]) clear(u);  

28 }  

29  

30 void unite(int x,int y){  

31     int fx = find(x), fy = find(y);  

32     if(fx != fy) f[fx] = fy, num--;  

33 }  

34  

35 void solve(int x){  

36     unite(l[x],r[x]);  

37     for(int u : p[x]) solve(u);  

38 }  

39  

40 void dfs(int x){  

41     int y = -1;  

42     for(int u : p[x])  

43         if(y == -1 || sz[y] < sz[u]) y = u;  

44     for(int u : p[x]){  

45         if(u == y) continue;  

46         num = k, dfs(u), clear(u);  

47     }  

48     num = k;  

49     if(y != -1) dfs(y);  

50     for(int u : p[x]){  

51         if(u != y) solve(u);  

52     }  

53     unite(l[x],r[x]);  

54     ans[x] = num;  

55 }  

56  

57 int main()  

58 {  

59     scanf("%d%d",&n,&k);

```

```

60     rep(i,2,n){
61         int pp; scanf("%d",&pp);
62         p[pp].push_back(i);
63     }
64     rep(i,1,n) scanf("%d%d",&l[i],&r[i]);
65     for(int i = n; i >= 1; i--){
66         for(int y : p[i]) sz[i] += sz[y];
67         sz[i]++;
68     }
69     rep(i,0,k) f[i] = i;
70     dfs(1);
71     rep(i,1,n) printf("%d\n",ans[i]);
72     return 0;
73 }
```

4.13.3 例题 2

题意：

给定一棵有根树，每条边的权值是 $[a, v]$ 的一个字母。现对于每个树上点，求出最长的一条“回文”路径。“回文”路径的含义是将路径上所有的字母取出，可以组成一个回文串。 $(1 \leq n \leq 5 * 10^5)$

思路：

树上的这类问题，我们可以依次思考点分治、树上启发式合并、长链剖分，根据该题题意，不难识别这是一道树上启发式合并问题。

确认是树上启发式合并之后，我们需要定状态。由于“回文”路径只要求所有字母可以组成一个回文串，因此我们可以利用状压的思想给每一个字母进行赋值，然后对于每个点求一个从根节点到当前点的异或和。

令 $f[i]$ 表示对于当前点 now ，其子树中存在一个点 x ， $value[x] = i$, $f[i] = dep[x]$, x 为此中情况下深度最深的点。然后计算点 now 答案时，我们只需考虑三种情况。

1. now 为最长路径的一个端点
2. 最长路径经过 now
3. 最长路径在 now 子树中，不经过 now

三种情况在代码中有比较清晰的注释，不太清楚细节的朋友可以看看代码。总体来说，这题应该属于树上启发式合并的经典问题。

总结：

此题有几个思想比较可取。

1. 对每一个字母进行状压编码
2. 每一个点维护的权值是从根到该点的异或和
3. 分 3 种情况对答案进行了枚举

```

1 #include <bits/stdc++.h>
2 #define mem(a,b) memset(a,b,sizeof a);
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define per(i,a,b) for(int i = a; i >= b; i--)
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 typedef long long ll;
7 typedef double db;
8 const int inf = 1e8;
9 const int N = 5e5+10;
10 const db EPS = 1e-9;
11 using namespace std;
12
13 void dbg() {cout << "\n";}
14 template<typename T, typename... A> void dbg(T a, A... x) {cout << a << ' '; dbg(x...)}
15 
```

```

15 #define logs(x...) {cout << #x << " -> "; dbg(x);}
16
17 int n,a[N],tot,head[N],sz[N],son[N],f[1<<22],dep[N],ans[N]; //f[i]:表示子树中异或和为f[i]
18                                     的最大深度
19 struct Node{
20     int to,next;
21 }e[N];
22
23 void add(int x,int y){
24     e[++tot].to = y, e[tot].next = head[x], head[x] = tot;
25 }
26
27 void dfs1(int x){
28     ans[x] = -inf; sz[x] = 1;
29     for(int i = head[x]; i; i = e[i].next){
30         int y = e[i].to;
31         dep[y] = dep[x]+1; a[y] ^= a[x];
32         dfs1(y); sz[x] += sz[y];
33         if(sz[y] > sz[son[x]]) son[x] = y;
34     }
35 }
36
37 void Delete(int x){
38     f[a[x]] = -inf;
39     for(int i = head[x]; i; i = e[i].next) Delete(e[i].to);
40 }
41
42 void modify(int x,int fa){
43     ans[fa] = max(ans[fa],f[a[x]]+dep[x]-2*dep[fa]);
44     for(int i = 0; i < 22; i++)
45         ans[fa] = max(ans[fa],f[a[x]^((1<<i))+dep[x]-2*dep[fa]]);
46     for(int i = head[x]; i; i = e[i].next) modify(e[i].to,fa);
47 }
48
49 void ins(int x){
50     f[a[x]] = max(f[a[x]],dep[x]);
51     for(int i = head[x]; i; i = e[i].next) ins(e[i].to);
52 }
53
54 void dfs2(int x){
55     ans[x] = 0;
56     for(int i = head[x]; i; i = e[i].next)
57         if(e[i].to != son[x]) dfs2(e[i].to), Delete(e[i].to);
58     if(son[x]) dfs2(son[x]);
59     f[a[x]] = max(f[a[x]],dep[x]);
60     //路径经过x
61     for(int i = head[x]; i; i = e[i].next)
62         if(e[i].to != son[x]) modify(e[i].to,x), ins(e[i].to);
63     //x为路径端点
64     ans[x] = max(ans[x],f[a[x]]-dep[x]);
65     for(int i = 0; i < 22; i++)
66         ans[x] = max(ans[x],f[a[x]^((1<<i))-dep[x]]);
67     //路径不经过x
68     for(int i = head[x]; i; i = e[i].next) ans[x] = max(ans[x],ans[e[i].to]);
69 }
70
71 int main()
72 {
73     scanf("%d",&n); tot = 1;

```

```

73     rep(i,0,(1<<22)-1) f[i] = -inf; //不要忘记赋初值
74     rep(i,2,n){
75         int p; char s[10];
76         scanf("%d%s",&p,s);
77         add(p,i); a[i] = 1<<(s[0]-'a');
78     }
79     dfs1(1); dfs2(1);
80     rep(i,1,n) printf("%d%c",ans[i]," \n"[i==n]);
81     return 0;
82 }
```

4.14 Splay

4.14.1 普通 Splay

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
8 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
9 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << " , " << z1 << ":" << z2 << endl;
10 typedef long long ll;
11 typedef double db;
12 const int N = 1e6+100;
13 const int M = 1e5+100;
14 const db EPS = 1e-9;
15 using namespace std;
16
17 int f[N],cnt[N],ch[N][2],siz[N],key[N],sz,rt;
18 //f[x]-x的父节点, ch[x][0]表示x的左儿子, ch[x][1]表示x的右儿子, key[x]表示x的关键点,
19 //cnt[x]表示x节点关键字出现的次数(权值), siz[x]表示包括x的这个子树的大小, sz为整棵树的大小, rt为整
20 //棵树的根
21 //空间大小为所有插入点个数, 点被删除之后空间不能复用
22 void init(){
23     memset(f,0,sizeof f); memset(cnt,0,sizeof cnt);
24     memset(ch,0,sizeof ch); memset(siz,0,sizeof siz);
25     memset(key,0,sizeof key); sz = rt = 0;
26 }
27
28 void clear(int x) { f[x] = cnt[x] = ch[x][0] = ch[x][1] = siz[x] = key[x] = 0; }
29 bool get(int x) { return ch[f[x]][1] == x; }
30 void push_up(int x) { siz[x] = siz[ch[x][0]]+siz[ch[x][1]]+cnt[x]; } //更新节点信息, 此处
31 //为siz
32 void rotate(int x){
33     int old = f[x], oldf = f[old], which = get(x);
34     ch[old][which] = ch[x][which^1]; f[ch[old][which]] = old; //把儿子过继给爸爸, 同时处理
35 //父子两个方向上的信息
36     ch[x][which^1] = old; f[old] = x; //我给我爸爸当爹, 我爸爸管我叫爸爸
37     f[x] = oldf; //我的爷爷成了我的爸爸
38     if(oldf) ch[oldf][ch[oldf][1]==old] = x;
39     push_up(old); push_up(x); //分别维护信息
40 }
41 void splay(int x){ //将x旋为根
```

```

40     for(int fa; (fa = f[x]); rotate(x))
41         if(f[fa]) rotate((get(x) == get(fa))?fa:x); //如果祖父三代连成一线，则旋转父亲，否则
42             旋转自己
43     rt = x;
44 }
45 void insert(int x){ //x为权值
46     if(rt == 0) {
47         key[++sz] = x, rt = sz;
48         cnt[sz] = siz[sz] = 1; f[sz] = ch[sz][0] = ch[sz][1] = 0;
49         return;
50     } //空树
51     int now = rt, fa = 0;
52     while(1){
53         if(x == key[now]){ //这个数出现过
54             cnt[now]++;
55             push_up(now), push_up(fa), splay(now); return;
56         }
57         fa = now, now = ch[now][key[now]<x];
58         if(now == 0){
59             ++sz; siz[sz] = cnt[sz] = 1;
60             ch[sz][0] = ch[sz][1] = 0;
61             ch[fa][x>key[fa]] = sz; //根据加入点顺序重新编号
62             f[sz] = fa, key[sz] = x, push_up(fa), splay(sz); return;
63         }
64     }
65 }
66 int rnk(int x){ //查询x的排名
67     int now = rt, ans = 0;
68     while(now){
69         if(x < key[now]) now = ch[now][0];
70         else{
71             ans += siz[ch[now][0]];
72             if(x == key[now]) { splay(now); return (ans+1); } //x在树中节点的位置
73             ans += cnt[now], now = ch[now][1]; //到达右孩子处
74         }
75     }
76     return -1; //找不到
77 }
78 int kth(int x){ //查询排名为x的数
79     int now = rt;
80     while(1){
81         if(ch[now][0] && x <= siz[ch[now][0]]) now = ch[now][0];
82         else{
83             int tmp = siz[ch[now][0]]+cnt[now];
84             if(x <= tmp) return key[now];
85             x -= tmp, now = ch[now][1];
86         }
87     }
88 }
89 int pre(){ //进行splay后，x已经在根节点了，因此只用找左子树最大节点即可，返回节点编号
90     int now = ch[rt][0];
91     while(ch[now][1]) now = ch[now][1];
92     return now;
93 }
94 int next(){ //找右子树最小
95     int now = ch[rt][1];
96     while(ch[now][0]) now = ch[now][0];
97     return now;

```

```

98 }
99
100 void del(int x){
101     rnk(x); //把x对应节点转到了根
102     if(cnt[rt] > 1) {cnt[rt]--; push_up(rt); return;} //有多个相同的数
103     if(!ch[rt][0] && !ch[rt][1]) {clear(rt); rt = 0; return;}
104     if(!ch[rt][0] || !ch[rt][1]){ //只有一个儿子
105         int oldrt = rt; rt = ch[rt][1]+ch[rt][0]; f[rt] = 0; clear(oldrt); return;
106     }
107     int oldrt = rt, leftbig = pre();
108     splay(leftbig);
109     ch[rt][1] = ch[oldrt][1];
110     f[ch[oldrt][1]] = rt;
111     clear(oldrt);
112     push_up(rt);
113 }
114
115 int main()
116 {
117     int m; scanf("%d",&m);
118     rep(i,1,m){
119         // LOG1("i",i);
120         int op,x; scanf("%d%d",&op,&x);
121         if(op == 1) insert(x);
122         else if(op == 2) del(x);
123         else if(op == 3) printf("%d\n",rnk(x));
124         else if(op == 4) printf("%d\n",kth(x));
125         else if(op == 5){
126             insert(x);
127             printf("%d\n",key[pre()]);
128             del(x);
129         }
130         else if(op == 6){
131             insert(x);
132             printf("%d\n",key[next()]);
133             del(x);
134         }
135     }
136     return 0;
137 }
```

4.14.2 区间 Splay

```

1 #include<iostream>
2 #include<cstdio>
3 using namespace std;
4 struct{int l,r,size,dat,bit,add,rev;}a[200010];
5 int L[200010],R[200010],n,m,tot,root,i,x,y,z;
6 char str[10];
7 //size-子树大小, dat-节点真实值, bit-子树中最小值, add-子树区间加标记, rev-子树翻转标记
8
9 void spreadadd(int x,int y)
10 {
11     if(!x||!y) return;
12     a[x].add+=y; a[x].dat+=y; a[x].bit+=y;
13 }
14 //add是lazy标记
15
```

```

16 void spreadrev(int x)
17 {
18     if(!a[x].rev) return;
19     swap(a[x].l,a[x].r);
20     if(a[x].l) a[a[x].l].rev^=1; //翻转标记下传
21     if(a[x].r) a[a[x].r].rev^=1;
22 }
23
24 void spread(int x)
25 {
26     spreadadd(a[x].l,a[x].add); //标记下传
27     spreadadd(a[x].r,a[x].add);
28     spreadrev(x); //rev表示是否要翻转
29     a[x].add=a[x].rev=0;
30 }
31
32 inline void update(int x) //更换左右儿子后，更新其size及bit
33 {
34     a[x].size=a[a[x].l].size+a[a[x].r].size+1;
35     a[x].bit=min(a[x].dat,min(a[a[x].l].bit,a[a[x].r].bit));
36 }
37
38 void turnleft(int &x) //x左旋，即x的右儿子左旋到x位置
39 {
40     int y=a[x].r; a[x].r=a[y].l; a[y].l=x;
41     update(x); update(y); x=y;
42 }
43
44 void turnright(int &x) //x右旋，即x的左儿子右旋到x位置
45 {
46     int y=a[x].l; a[x].l=a[y].r; a[y].r=x;
47     update(x); update(y); x=y;
48     //先更新x，因为x是儿子
49 }
50
51 void splay(int &x,int y) //将x子树中第y小的数转为根
52 {
53     if(!x) return;
54     L[0]=R[0]=0; //L存储了y转到x之后，左子树中的各个节点
55     //R存储y转到x之后，右子树中的各个节点
56     while(1)
57     {
58         //lazy、旋转标记下传
59         spread(x),spread(a[x].l),spread(a[x].r);
60         int temp=a[a[x].l].size+1; //<=x的数的个数
61         //完成了目标
62         if(y==temp||!(y<temp&&!a[x].l)||!(y>temp&&!a[x].r)) break;
63         if(y<temp) //让y的左儿子旋转上来
64         {
65             if(a[a[x].l].l&&y<=a[a[x].l].l].size) {turnright(x); if(!a[x].l) break;}
66             R[++R[0]]=x; x=a[x].l;
67         }
68         else //让y的右儿子旋转上来
69         {
70             y-=temp;
71             int temp=a[a[x].r].l].size+1;
72             if(a[a[x].r].r&&y>temp) {y-=temp; turnleft(x); if(!a[x].r) break;}
73             L[++L[0]]=x; x=a[x].r;
74     }

```

```

75     }
76     L[++L[0]]=a[x].l; R[++R[0]]=a[x].r; //此处的x已经成为y所对应的节点
77     //将左右子树一一连接
78     for(int i=L[0]-1;i>0;i--) {a[L[i]].r=L[i+1]; update(L[i]);}
79     for(int i=R[0]-1;i>0;i--) {a[R[i]].l=R[i+1]; update(R[i]);}
80     a[x].l=L[1]; a[x].r=R[1]; update(x);
81 }
82
83 void ADD(int x,int y,int z)
84 {
85     splay(root,y+1); //把后继转到树根
86     splay(a[root].l,x-1); //把前驱转到树根左儿子
87     spreadadd(a[a[root].l].r,z); //区间+, 打lazy标记
88 }
89
90 void INSERT(int x,int y) //将y插入到a[x]之后
91 {
92     splay(root,x); //令x为根
93     a[++tot].dat=y; a[tot].r=a[root].r; a[root].r=tot; //令y为x的右儿子, x原右儿子为y右儿子
94     update(tot); update(root);
95 }
96
97 void DELETE(int x) //删除数组中第x个数
98 {
99     splay(root,x); //将数组中第x个数旋转到根
100    splay(a[root].r,1); //将子树中最小的数转到根, 则根节点的右儿子没有左儿子
101    a[a[root].r].l=a[root].l; root=a[root].r;
102    update(root);
103 }
104
105 void REVERSE(int x,int y)
106 {
107     splay(root,y+1);
108     splay(a[root].l,x-1);
109     a[a[a[root].l].r].rev^=1; //子树中所有点都左右翻转
110     //此处的操作与add处不同, 原因在于区间反转不会导致该区间维护的min发生变化, 因此可以这样操作
111     //但是正常打标记的操作应该是与ADD函数一致
112 }
113
114 void REVOLVE(int x,int y,int z) //将[x,y-len]与[y-len+1,y]互换位置, len为z%(y-x+1)
115 {
116     z%=y-x+1;
117     if(!z) return;
118     int mid=y-z; //找到右区间的第一个数
119     splay(root,mid); //将mid转到根
120     splay(a[root].l,x-1); //根左儿子的右儿子对应区间[x,y-len] (不包括根左儿子的右儿子)
121     splay(a[root].r,y-a[a[root].l].size); //根右儿子的左儿子对应区间[y-len+2,y] (不包括根右儿子的左儿子)
122     z=a[root].l;
123     a[root].l=a[z].r;
124     a[z].r=a[a[root].r].l;
125     a[a[root].r].l=0;
126     update(z); update(a[root].r); update(root);
127     splay(root,1); //空出根的左儿子
128     a[root].l=z;
129     update(root);
130 }
131
132 void MIN(int x,int y)

```

```

133 {
134     splay(root,y+1); //将根节点中第y+1小的数转为根
135     splay(a[root].l,x-1); //将根节点左儿子子树中第x-1小的数转为根节点左儿子
136     printf("%d\n",a[a[root].l].bit);
137 }
138
139 // splay(root,n+2);
140 // splay(a[root].l,0);
141 // Print(a[a[root].l].r);
142 void Print(int x){ //打印区间[1,n]的数，输出之前要先把这个区间的数旋转到一个子树中
143     spread(x); //标记下传
144     if(a[x].l) Print(a[x].l);
145     printf("%d ",a[x].dat); //中序遍历
146     if(a[x].r) Print(a[x].r);
147 }
148
149 int main()
150 {
151     cin>>n;
152     a[1].size=1; a[n+2].l=n+1; a[n+2].size=n+2;
153     a[1].dat=a[n+2].dat=a[1].bit=a[0].bit=0x3fffffff;
154     //所有数右移一格，2表示1，n+1表示n
155     for(i=2;i<=n+1;i++) //一开始构建出一根链
156     {
157         scanf("%d",&a[i].dat); //dat为真实值
158         a[i].l=i-1; a[i].size=i; //l表示左儿子，size表示子树大小
159         a[i].bit=min(a[i-1].bit,a[i].dat); //bit表示最小值
160     }
161     a[n+2].bit=a[n+1].bit;
162     root=tot=n+2; //一共n+2个节点，加了两个前驱和后续
163     cin>>m;
164     for(i=0;i<m;i++)
165     {
166         scanf("%s",&str);
167         //ADD x y z 区间[x,y]每个数+z
168         if(str[0]=='A') {scanf("%d%d%d",&x,&y,&z); ADD(++x,++y,z);}
169         //INSERT x y 在序列第x个数之后插入y
170         if(str[0]=='I') {scanf("%d%d",&x,&y); INSERT(++x,y);}
171         //DELETE x 删除序列第x个数
172         if(str[0]=='D') {scanf("%d",&x); DELETE(++x);}
173         //MIN x y 查询区间[x,y]的最小值
174         if(str[0]=='M') {scanf("%d%d",&x,&y); MIN(++x,++y);}
175         //REVERSE x y 将区间[x,y]中的数翻转，如原序列1 2 3 4 5, REVERSE 3 5, 得到1 2 5 4 3
176         if(str[0]=='R'&&str[3]=='E') {scanf("%d%d",&x,&y); REVERSE(++x,++y);}
177         //REVOLVE x y z 将区间[x,y]旋转z次，即将最后一个数移到第一个位置，如原序列1 2 3 4 5,
178         REVOLVE 2 4 2, 得到1 3 4 2 5
179         //区间左右旋转可以互相转换
180         if(str[0]=='R'&&str[3]=='O') {scanf("%d%d%d",&x,&y,&z); REVOLVE(++x,++y,z);}
181
182 }

```

4.14.3 Splay (三个 lazy 标记)

Replace a b c, 将 [a,b] 中所有括号改为 c Swap a b, 将 [a,b] 中所有字符翻转 Invert a b, 将 [a,b] 中所有'(' 变为')', ')' 变为'(' Query a b, 询问 [a,b] 之间的字符至少要改变多少位才能变成合法的括号序列

```

1 #include <iostream>
2 #include <cstdio>
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 const int N = 1e5+100;
5 using namespace std;
6 struct Node{int l,r,size,dat,add,rev,res,l1,l2,r1,r2,sum;}a[N];
7 int L[N],R[N],n,m,tot,root;
8 char str[10];
9 //size-子树大小, dat-节点真实值, add-子树区间赋值标记, rev-子树翻转标记, res-子树取反标记
10 //l1、l2-最大、小前缀和, r1、r2-最大、小后缀和
11 void spreadadd(int x,int y)
12 {
13     if(!x||!y) return;
14     a[x].add=y; a[x].dat=y;
15     a[x].rev = 0, a[x].res = 0;
16     if(y < 0) a[x].sum = -a[x].size;
17     else a[x].sum = a[x].size;
18     a[x].r1 = a[x].l1 = (y==1?a[x].sum:0);
19     a[x].r2 = a[x].l2 = (y==1?0:a[x].sum);
20 }
21 //add是lazy标记
22
23 void spreadrev(int x) //翻转
24 {
25     if(!x) return;
26     swap(a[x].l1,a[x].r1);
27     swap(a[x].l2,a[x].r2);
28     swap(a[x].l,a[x].r);
29     a[x].rev ^= 1;
30 }
31
32 void spreadres(int x) //取反
33 {
34     if(!x) return;
35     swap(a[x].l1,a[x].l2);
36     swap(a[x].r1,a[x].r2);
37     a[x].sum = -a[x].sum; a[x].dat = -a[x].dat;
38     a[x].l1 = -a[x].l1, a[x].l2 = -a[x].l2;
39     a[x].r1 = -a[x].r1, a[x].r2 = -a[x].r2;
40     a[x].res ^= 1;
41 }
42
43 void spread(int x)
44 {
45     if(a[x].rev){
46         spreadrev(a[x].l);
47         spreadrev(a[x].r);
48     }
49     if(a[x].add){
50         spreadadd(a[x].l,a[x].add); //标记下传
51         spreadadd(a[x].r,a[x].add);
52     }
53     if(a[x].res){
54         spreadres(a[x].l);
55         spreadres(a[x].r);
56     }
57     a[x].add=a[x].rev=a[x].res=0;
58 }
59

```

```

60 inline void update(int x)
61 {
62     int L = a[x].l, R = a[x].r;
63     a[x].size=a[L].size+a[R].size+1;
64     a[x].sum = a[L].sum+a[R].sum+a[x].dat;
65     a[x].l1 = max(a[L].l1,a[L].sum+a[R].l1+a[x].dat);
66     a[x].l2 = min(a[L].l2,a[L].sum+a[R].l2+a[x].dat);
67     a[x].r1 = max(a[R].r1,a[R].sum+a[L].r1+a[x].dat);
68     a[x].r2 = min(a[R].r2,a[R].sum+a[L].r2+a[x].dat);
69 }
70
71 void turnleft(int &x) //x左旋, 即x的右儿子左旋到x位置
72 {
73     int y=a[x].r; a[x].r=a[y].l; a[y].l=x;
74     update(x); update(y); x=y;
75 }
76
77 void turnright(int &x) //x右旋, 即x的左儿子右旋到x位置
78 {
79     int y=a[x].l; a[x].l=a[y].r; a[y].r=x;
80     update(x); update(y); x=y;
81     //先更新x, 因为x是儿子
82 }
83
84 void splay(int &x,int y) //将x子树中第y小的数转为根
85 {
86     if(!x) return;
87     L[0]=R[0]=0; //L存储了y转到x之后, 左子树中的各个节点
88     //R存储y转到x之后, 右子树中的各个节点
89     while(1)
90     {
91         //lazy、旋转标记下传
92         spread(x),spread(a[x].l),spread(a[x].r);
93         int temp=a[a[x].l].size+1; //<=x的数的个数
94         //完成了目标
95         if(y==temp|| (y<temp&&!a[x].l)|| (y>temp&&!a[x].r)) break;
96         if(y<temp) //让y的左儿子旋转上来
97         {
98             if(a[a[x].l].l&&y<=a[a[a[x].l].l].size) {turnright(x); if(!a[x].l) break;}
99             R[++R[0]]=x; x=a[x].l;
100        }
101        else //让y的右儿子旋转上来
102        {
103            y-=temp;
104            int temp=a[a[a[x].r].l].size+1;
105            if(a[a[x].r].r&&y>temp) {y-=temp; turnleft(x); if(!a[x].r) break;}
106            L[++L[0]]=x; x=a[x].r;
107        }
108    }
109    L[++L[0]]=a[x].l; R[++R[0]]=a[x].r; //此处的x已经成为y所对应的节点
110    //将左右子树一一连接
111    for(int i=L[0]-1;i>0;i--) {a[L[i]].r=L[i+1]; update(L[i]);}
112    for(int i=R[0]-1;i>0;i--) {a[R[i]].l=R[i+1]; update(R[i]);}
113    a[x].l=L[1]; a[x].r=R[1]; update(x);
114 }
115
116 void ADD(int x,int y,int z)
117 {
118     splay(root,y+1); //把后继转到树根

```

```

119     splay(a[root].l,x-1); //把前驱转到树根左儿子
120     spreadadd(a[a[root].l].r,z); //区间+, 打lazy标记
121 }
122
123 void SWAP(int x,int y)
124 {
125     splay(root,y+1);
126     splay(a[root].l,x-1);
127     spreadrev(a[a[root].l].r);
128 }
129
130 void Invert(int x,int y)
131 {
132     splay(root,y+1);
133     splay(a[root].l,x-1);
134     spreadres(a[a[root].l].r);
135 }
136
137 void Query(int x,int y)
138 {
139     splay(root,y+1);
140     splay(a[root].l,x-1);
141     int x1 = a[a[a[root].l].l].l2;
142     int x2 = a[a[a[root].l].r].r1;
143     x1 = -x1;
144     int ans = ((x1+1)/2)+((x2+1)/2); //左端最小前缀和/2上取整+右端最大前缀和上取整
145     //abs(左端最小前缀和)=x, 表示左端至少有x个'('未匹配
146     //abs(右端最大前缀和)=x, 表示右端至少有x个')'未匹配
147     printf("%d\n",ans);
148 }
149
150 char s[N];
151
152 int main()
153 {
154     cin>>n>>m;
155     a[1].size=1; a[n+2].l=n+1; a[n+2].size=n+2;
156     a[1].dat = a[n+2].dat; update(1);
157     scanf("%s",s+1);
158     rep(i,2,n+1) a[i].dat = (s[i-1]=='('?1:-1);
159     rep(i,2,n+2)
160     {
161         a[i].l=i-1; a[i].size=i; //l表示左儿子, siz表示子树大小
162         update(i);
163     }
164     root=tot=n+2; //一共n+2个节点, 加了两个前驱和后续
165     rep(i,1,m)
166     {
167         int x,y,z; char op[5];
168         scanf("%s",str);
169         if(str[0]=='R'){
170             scanf("%d%d",&x,&y); scanf("%s",op);
171             if(op[0] == '(') z = 1;
172             else z = -1;
173             ADD(++x,++y,z);
174         }
175         else if(str[0]=='S'){
176             scanf("%d%d",&x,&y);
177             SWAP(++x,++y);

```

```

178     }
179     else if(str[0] == 'I'){
180         scanf("%d%d",&x,&y);
181         Invert(++x,++y);
182     }
183     else if(str[0] == 'Q'){
184         scanf("%d%d",&x,&y);
185         Query(++x,++y);
186     }
187 }
188 return 0;
189 }
```

4.15 LCT

4.15.1 LCT 模板题

一、适用问题:

动态树主要用于解决操作中带有加边、删边、换根的一系列问题，即树结构发生变化的问题，理论上来说，树链剖分的问题都能用 *LCT* 进行解决。

二、LCT 函数解析

LCT 本质上是对树进行实链剖分，实链剖分的意思就是将一棵树分成多条链，链中的边称为实边，链与链之间的边则称为虚边，每条链都是一个 *splay*，在 *splay* 中进行中序遍历即可还原原来的树结构。而 *LCT* 就是不断进行虚边、实边转换的一个算法。

LCT 中一共有 *clear*、*pushUp*、*pushDown*、*update*、*rotate*、*splay*、*access*、*makeRoot*、*link*、*cut*、*find*、*split* 等函数，下面大致介绍一下每个函数的具体作用以及一些坑点，更多的是提纲挈领的作用，想要从最基础的地方开始学的话，推荐 [oiwiki](https://oi-wiki.org/ds/lct/)。

(1) 简单函数（仅操作单个 *splay* 的函数）

1. *clear(x)*: 清除一个点的信息，如父亲、左右儿子、标记、维护信息等信息。
2. *pushUp(x)*: 由左右儿子的信息更新父节点的信息，与线段树的 *pushUp()* 函数没有太大差别。
3. *pushDown(x)*: 将当前节点的标记下放到儿子节点，如加、减、翻转等标记。
4. *update(x)*: 一直递归到根节点，然后把标记信息不断下放，没有涉及任何虚实边的转换。

```

1 void update(int p){ //递归地从上到下pushDown信息
2     if(!isRoot(p)) update(f[p]);
3     pushDown(p);
4 }
```

5. *rotate(x)*: 将当前节点向上旋转一层，可以自己模拟一下。此处改变了 *splay* 的内部结构，即子节点发生了改变，因此需要进行 *pushUp*，但是仍然没有进行任何虚实边的转换。

```

1 inline void rotate(int x){ //将x向上旋转一层的操作
2     int y = f[x], z = f[y], k = Get(x);
3     if(!isRoot(y)) ch[z][ch[z][1] == y] = x;
4     ch[y][k] = ch[x][!k], f[ch[y][k]] = y;
5     ch[x][!k] = y, f[y] = x, f[x] = z;
6     pushUp(y); //要先pushUp(y)
7     pushUp(x);
8 }
```

6. *splay(x)*: 将当前点旋转到 *splay* 的根节点，*splay* 到根节点作用在于不需要在向上进行更新。比如你现在要修改 *x* 的点权，但是每个节点还要维护子树 *sum* 的信息，如果 *x* 不是其所在 *splay* 的根节点，那么修改 *x* 的点权势必影响到其祖先节点的 *sum* 信息，因此需要将 *x* 旋转为其所在 *splay* 的根后再进行单点修改。注意 *splay* 函数也没有进行实边和虚边的转换。

```

1 inline void splay(int x){ //把x旋转到当前splay的根
2     update(x); //将上面的标记完全下放
3     for(int fa; fa = f[x], !isRoot(x); rotate(x)){
4         if(!isRoot(fa)) rotate(Get(fa) == Get(x) ? fa : x);
5     }
6 }
```

以上函数都属于 *LCT* 函数中的简单函数，因为这些函数都只是在单个 *splay* 中进行操作，不涉及任何虚实边的转换。

(2) 复杂函数 (涉及多个 *splay* 的操作，进行虚实边转换)

1. *access(x)*: 将点 x 到根的路径经过的点放入同一个 *splay* 中，且这个 *splay* 中仅包含从 x 到根路径上经过的点。具体操作即是将 x 点不断转成其所在 *splay* 的根，然后再进行虚实边转换一直到根。此处 *access* 函数有返回值，返回值为最后构成的 *splay* 的根节点。

```

1 inline int access(int x){ //把从根到x的所有点放在一条实链里，返回这个splay的根
2     int p; //每次改变右儿子的值，因为整棵树是中序遍历，放入右儿子才能保证先遍历父亲再遍历儿子
3     for(p = 0; x; p = x, x = f[x]){
4         splay(x), ch[x][1] = p, pushUp(x);
5     }
6     return p;
7 }
```

2. *makeRoot(x)*: 换根操作，将点 x 变成当前树的根。具体过程为先 *access* 点 x ，然后再将点 x 旋转为其 *splay* 所在根，然后将所有节点的左右儿子翻转即可。

```

1 inline void makeRoot(int p){ //使x点成为整棵树的根
2     access(p); splay(p);
3     swap(ch[p][0], ch[p][1]); //把整条链反向
4     rev[p] ^= 1;
5 }
```

3. *split(x, y)*: 从树中拎出 $x \rightarrow y$ 的路径，返回该路径的 *splay* 根节点，可以查询路径最大值、点权和、边权和等信息。

```

1 inline int split(int x,int y){
2     makeRoot(x);
3     return access(y);
4 }
```

4. *find(x)*: 即返回点 x 所在树的根节点，不是所在 *splay* 中的根节点，用于判断两点是否连通。

```

1 inline int find(int p){ //找到x所在树的根节点编号
2     access(p), splay(p);
3     while(ls) pushDown(p), p = ls;
4     return p;
5 }
```

5. *link(x, y)*: 连接树中 x, y 两点之间的边，无边变虚边，如果题目中没有保证操作一定合法，则需要自行判断 x, y 是否已经连通。

```

1 inline void link(int x,int y){ //在x、y两点间连一条边，连接了虚边
2     if (find(x) != find(y)) makeRoot(x), f[x] = y;
3 }
```

6. *cut(x, y)*: 断开树中 x, y 两点之间的实边，两个点同时断开即可。

```

1 inline void cut(int x,int p){ //把x、y两点间边删掉，此处删除的是实边，注意实边和虚边的区别
2     makeRoot(x), access(p), splay(p);
3     if (ls == x && !rs) ls = f[x] = 0;
4 }
```

三、具体细节

(1) 单点修改

由于 LCT 中维护了多个 $splay$, 因此单点修改需要把该点修改的信息不断上传, 所以我们需要先将点 x 旋转到 $splay$ 的根或者整棵树的根, 然后再进行单点修改。

如果题中只需要维护实链信息, 则只需要旋转到 $splay$ 的根, 如果需要同时维护实链和虚链信息, 即整棵子树的信息的话, 则需要令该点为树根, 即调用 $makeRoot()$ 函数。

(2) 维护边权

由于 LCT 是不断地进行虚边、实边转换, 因此没有固定的边结构, 所以直接维护边权十分困难, 因此我们将边转成点, 边 (a, b) 成为一个点 x , $link(a, x)$ 、 $link(x, b)$ 即可。

(3) 维护子树信息

普通 LCT 只能维护具有可减性的子树信息, 比如子树大小, 子树贡献等, 而子树 max 、 min 等问题则不具有可减性, 难以维护。

维护子树信息主要在于维护实边信息和虚边信息, 而进行实虚转换的函数只有 $makeRoot()$ 、 $access()$ 、 $link()$ 三个函数, 只需要在该三个函数进行一定的修改即可, 下面习题中包含了该问题可供参考。

四、例题

题意:

n 个点一棵树, 支持四种操作。 $(1 \leq n, q \leq 10^5, 0 \leq c \leq 10^4)$

1. $+ u v c$, 将 u 到 v 的路径上的点的权值都加上 c 。
2. $- u_1 v_1 u_2 v_2$, 将树中原有的边 (u_1, v_1) 删除, 加入一条新边 (u_2, v_2) , 保证操作完之后仍然是一棵树。
3. $* u v c$, 将 u 到 v 的路径上的点的权值都乘上 c 。
4. $/ u v$, 询问 u 到 v 的路径上的点的权值和, 答案 mod 51061。

思路:

三个涉及到路径的操作, 都是先把 u 变成树根, 然后 $access(v)$, 即拉起一条 u 到 v 的路径, 使得 u 到 v 路径上的点都在一个 $splay$ 中, 然后获得这个 $splay$ 的根节点, 即可对根节点打标记完成。

删边则是令 u 为根, 拉起 u 到 v 的路径, 将 v 旋转成 $splay$ 的根, 然后儿子与父亲双向断开联系。加边则是令 u 为根, 然后使 u 的父亲变成 v 。

总结:

这题应该算是 LCT 的模板题, 涉及的操作都是基础操作, 没有太多思维上的难点。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 using namespace std;
4 #define int long long
5 const int N = 100010;
6 const int mod = 51061;
7 int n, q, u, v, c;
8 char op;
9
10 struct LCT{
11     #define ls ch[p][0]
12     #define rs ch[p][1]
13     #define Get(p) (ch[f[p]][1] == p)
14     int ch[N][2], f[N], sum[N], val[N], siz[N], rev[N], add[N], mul[N];
15
16     inline void clear(int p){ //清除这个点的信息
17         ch[p][0] = ch[p][1] = f[p] = siz[p] = val[p] = sum[p] = rev[p] = add[p] = 0;
18         mul[p] = 1;
19     }
20 }
```

```

21 inline int isRoot(int p){
22     clear(0);
23     return ch[f[p]][0] != p && ch[f[p]][1] != p;
24 }
25
26 inline void pushUp(int p){
27     clear(0);
28     siz[p] = (siz[ls] + 1 + siz[rs]) % mod;
29     sum[p] = (sum[ls] + val[p] + sum[rs]) % mod;
30 }
31
32 inline void pushDown(int p){
33     clear(0);
34     if(mul[p] != 1){ //乘法
35         if(ls){ //左儿子
36             mul[ls] = (mul[ls] * mul[p]) % mod;
37             val[ls] = (val[ls] * mul[p]) % mod;
38             sum[ls] = (sum[ls] * mul[p]) % mod;
39             add[ls] = (add[ls] * mul[p]) % mod;
40         }
41         if(rs){ //右儿子
42             mul[rs] = (mul[rs] * mul[p]) % mod;
43             val[rs] = (val[rs] * mul[p]) % mod;
44             sum[rs] = (sum[rs] * mul[p]) % mod;
45             add[rs] = (add[rs] * mul[p]) % mod;
46         }
47         mul[p] = 1;
48     }
49     if(add[p]){
50         if(ls){
51             add[ls] = (add[ls] + add[p]) % mod;
52             val[ls] = (val[ls] + add[p]) % mod;
53             sum[ls] = (sum[ls] + add[p] * siz[ls] % mod) % mod;
54         }
55         if(rs){
56             add[rs] = (add[rs] + add[p]) % mod;
57             val[rs] = (val[rs] + add[p]) % mod;
58             sum[rs] = (sum[rs] + add[p] * siz[rs] % mod) % mod;
59         }
60         add[p] = 0;
61     }
62     if(rev[p]){
63         if(ls) rev[ls] ^= 1, swap(ch[ls][0], ch[ls][1]);
64         if(rs) rev[rs] ^= 1, swap(ch[rs][0], ch[rs][1]);
65         rev[p] = 0;
66     }
67 }
68
69 void update(int p){ //递归地从上到下pushDown信息
70     //没有将实边变成虚边
71     if(!isRoot(p)) update(f[p]);
72     pushDown(p);
73 }
74
75 inline void rotate(int x){ //将x向上旋转一层的操作
76     //没有将实边变成虚边
77     int y = f[x], z = f[y], k = Get(x);
78     if(!isRoot(y)) ch[z][ch[z][1] == y] = x;
79     ch[y][k] = ch[x][!k], f[ch[y][k]] = y;

```

```

80     ch[x][!k] = y, f[y] = x, f[x] = z;
81     pushUp(y); //要先pushUp(y)
82     pushUp(x);
83 }
84
85 inline void splay(int x){ //把x旋转到当前splay的根
86     //没有将实边变成虚边
87     update(x); //将上面的标记完全下放
88     for(int fa; fa = f[x], !isRoot(x); rotate(x)){
89         if(!isRoot(fa)) rotate(Get(fa) == Get(x) ? fa : x);
90     }
91 }
92
93 inline int access(int x){ //把从根到x的所有点放在一条实链里，返回这个splay的根
94     //进行了边的虚实变换
95     int p; //每次改变右儿子的值，因为整棵树是中序遍历，放入右儿子才能保证先遍历父亲再遍历儿子
96     for(p = 0; x; p = x, x = f[x]){
97         splay(x), ch[x][1] = p, pushUp(x);
98     }
99     return p;
100 }
101
102 inline void makeRoot(int p){ //使x点成为整棵树的根
103     access(p); splay(p);
104     swap(ch[p][0],ch[p][1]); //把整条链反向
105     rev[p] ^= 1;
106 }
107
108 inline void link(int x,int y){ //在x、y两点间连一条边，连接了虚边
109     if (find(x) != find(y)) makeRoot(x), f[x] = y;
110 }
111
112 inline void cut(int x,int p){ //把x、y两点间边删掉，此处删除的是实边，注意实边和虚边的区别
113     makeRoot(x), access(p), splay(p);
114     if (ls == x && !rs) ls = f[x] = 0;
115 }
116
117 inline int find(int p){ //找到x所在树的根节点编号
118     access(p), splay(p);
119     while(ls) pushDown(p), p = ls;
120     return p;
121 }
122 //中序遍历即可还原树结构
123 void print(int p){
124     if(!p) return;
125     pushDown(p);
126     print(ls);
127     printf("%lld ",p);
128     print(rs);
129 }
130 }st;
131
132 signed main() {
133     scanf("%lld%lld", &n, &q);
134     for (int i = 1; i <= n; i++) st.val[i] = 1;
135     for (int i = 1; i < n; i++) {
136         scanf("%lld%lld", &u, &v);
137         st.link(u,v);
138     }

```

```

139     while (q--) {
140         scanf(" %c%lld%lld", &op, &u, &v);
141         if (op == '+') { //+ u v c, u到v的路径上的点权值+c
142             scanf("%lld", &c);
143             //u变成树根, 拉起v到u的链, 把v旋到splay的根
144             st.makeRoot(u); v = st.access(v);
145             st.val[v] = (st.val[v] + c) % mod;
146             st.sum[v] = (st.sum[v] + st.siz[v] * c % mod) % mod;
147             st.add[v] = (st.add[v] + c) % mod;
148         }
149         if (op == '-') { // - u1 v1 u2 v2, 删除(u1,v1), 加上(u2,v2)
150             st.cut(u, v);
151             scanf("%lld%lld", &u, &v);
152             st.link(u, v);
153         }
154         if (op == '*') { /* u v c, u到v的路径乘上c
155             scanf("%lld", &c);
156             st.makeRoot(u); v = st.access(v);
157             st.val[v] = st.val[v] * c % mod;
158             st.sum[v] = st.sum[v] * c % mod;
159             st.mul[v] = st.mul[v] * c % mod;
160         }
161         if (op == '/') { //u v, 询问u到v路径权值和
162             st.makeRoot(u); v = st.access(v);
163             printf("%lld\n", st.sum[v]);
164         }
165     }
166     return 0;
167 }
```

4.15.2 边权 LCT

题意:

n 个点一棵树, 支持两种操作。 $(1 \leq n, q \leq 10^4)$

1. CHANGE $i t_i$, 将第 i 条边的边权改为 t_i 。
2. QUERY $a b$, 查询树中点 a 到点 b 的路径中的边权最大值。

思路:

边权 LCT, 需要对于每一条边建一个节点, 即树中一共有 $2 * n - 1$ 个节点, 每个边节点与上下两个节点连边。

建边需要先确定每个节点的边权之后再进行 link, 因为点修改会对该节点的祖先节点产生影响, 需要将该点旋至 splay 端点后才能进行修改。

总结:

总结一下构建 LCT 构建的关键, 构建 LCT 需要先对各个顶点赋值然后再进行 link 操作, 若是 link 之后再赋值相当于点修改, 而点修改需要将点旋为 splay 根之后才能进行更改。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 using namespace std;
4 const int N = 20000+10;
5 int n, val[N];
6
7 struct LCT{
8     #define ls ch[p][0]
9     #define rs ch[p][1]
10    #define Get(p) (ch[f[p]][1] == p)
```

```

11 int ch[N][2], f[N], maxn[N], val[N], siz[N], rev[N];
12
13 inline void clear(int p){ //清除这个点的信息
14     ch[p][0] = ch[p][1] = f[p] = siz[p] = val[p] = maxn[p] = 0;
15 }
16
17 inline int isRoot(int p){
18     clear(0);
19     return ch[f[p]][0] != p && ch[f[p]][1] != p;
20 }
21
22 inline void pushUp(int p){
23     clear(0);
24     siz[p] = siz[ls] + 1 + siz[rs];
25     maxn[p] = max(val[p],max(maxn[ls],maxn[rs]));
26 }
27
28 inline void pushDown(int p){
29     clear(0);
30     if(rev[p]){
31         if(ls) rev[ls] ^= 1, swap(ch[ls][0],ch[ls][1]);
32         if(rs) rev[rs] ^= 1, swap(ch[rs][0],ch[rs][1]);
33         rev[p] = 0;
34     }
35 }
36
37 void update(int p){ //递归地从上到下pushDown信息
38     if(!isRoot(p)) update(f[p]);
39     pushDown(p);
40 }
41
42 inline void rotate(int x){ //将x向上旋转一层的操作
43     int y = f[x], z = f[y], k = Get(x);
44     if(!isRoot(y)) ch[z][ch[z][1] == y] = x;
45     ch[y][k] = ch[x][!k], f[ch[y][k]] = y;
46     ch[x][!k] = y, f[y] = x, f[x] = z;
47     pushUp(y); //要先pushUp(y)
48     pushUp(x);
49 }
50
51 inline void splay(int x){ //把x旋转到当前splay的根
52     update(x); //将上面的标记完全下放
53     for(int fa; fa = f[x], !isRoot(x); rotate(x)){
54         if(!isRoot(fa)) rotate(Get(fa) == Get(x) ? fa : x);
55     }
56 }
57
58 inline int access(int x){ //把从根到x的所有点放在一条实链里，返回这个splay的根
59     int p;
60     for(p = 0; x; p = x, x = f[x]){
61         splay(x), ch[x][1] = p, pushUp(x);
62     }
63     return p;
64 }
65
66 inline void makeRoot(int p){ //使x点成为整棵树的根
67     access(p); splay(p);
68     swap(ch[p][0],ch[p][1]); //把整条链反向
69     rev[p] ^= 1;

```

```

70     }
71
72     inline void link(int x,int y){ //在x、y两点间连一条边
73         makeRoot(x), f[x] = y; //dfs建树，每条边都是有效的，因此不需要判断是否有效
74     }
75 }st;
76
77 int main(){
78     int _; scanf("%d",&_);
79     while(_--){
80         scanf("%d",&n);
81         rep(i,0,2*n) st.clear(i);
82         rep(i,1,n-1){
83             int a,b,c; scanf("%d%d%d",&a,&b,&c);
84             st.val[i+n] = c;
85             st.link(a,i+n);
86             st.link(i+n,b);
87         }
88         while(1){
89             char s[20]; scanf("%s",s);
90             if(s[0] == 'D') break;
91             else if(s[0] == 'C'){
92                 int a,b; scanf("%d%d",&a,&b);
93                 st.splay(a+n); //先转为splay根节点
94                 st.val[a+n] = b;
95             }
96             else{
97                 int a,b; scanf("%d%d",&a,&b);
98                 st.makeRoot(a);
99                 b = st.access(b);
100                printf("%d\n",st.maxn[b]);
101            }
102        }
103    }
104    return 0;
105 }
```

4.15.3 LCT 最大连续和

题意:

n 个点一棵树，每个节点有一个值，支持两种操作。 $(1 \leq n, q \leq 10^5)$

1 $a\ b$, 查询树中点 a 到点 b 的路径中最大连续和

2 $a\ b\ c$, 将树中点 a 到点 b 路径中所有点的值改为 c

思路:

对每一个点维护一个 $lc[i], rc[i], maxn[i]$ 表示点 i 子树中左连续的最大值、右连续的最大值以及整棵子树中的最大连续值。

需要注意一点，交换左右儿子的时候，还需要把每个节点的 lc 和 rc 进行交换，其余细节见代码。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 using namespace std;
4 typedef long long ll;
5 const ll inf = 1e9+100;
6 const int N = 1e5+10;
7 int n,Q;
8
```

```

9 struct LCT{
10 #define ls ch[p][0]
11 #define rs ch[p][1]
12 #define Get(p) (ch[f[p]][1] == p)
13 int ch[N][2], f[N];
14 ll maxn[N], sum[N], lc[N], rc[N], val[N], siz[N], lazy[N];
15 bool rev[N];
16
17 inline void clear(int p){ //清除这个点的信息
18     ch[p][0] = ch[p][1] = f[p] = val[p] = maxn[p] = lc[p] = sum[p] = rc[p] = siz[p] =
19     0;
20 }
21 inline int isRoot(int p){
22     return ch[f[p]][0] != p && ch[f[p]][1] != p;
23 }
24
25 inline void pushUp(int p){
26     siz[p] = siz[ls] + 1 + siz[rs];
27     sum[p] = val[p] + sum[ls] + sum[rs]; //ls, rs可能为0
28     maxn[p] = max(maxn[ls], max(maxn[rs], rc[ls]+lc[rs]+val[p]));
29     lc[p] = max(lc[ls], sum[ls]+val[p]+lc[rs]);
30     rc[p] = max(rc[rs], sum[rs]+val[p]+rc[ls]);
31 }
32
33 inline void pushDown(int p){
34     if(rev[p]){
35         if(ls) rev[ls] ^= 1, swap(ch[ls][0], ch[ls][1]);
36         if(rs) rev[rs] ^= 1, swap(ch[rs][0], ch[rs][1]);
37         swap(lc[ls], rc[ls]); swap(lc[rs], rc[rs]); //交换左右儿子时还要交换左右连续最大值
38         rev[p] = 0;
39     }
40     if(lazy[p] != -inf){
41         if(ls){
42             sum[ls] = siz[ls]*lazy[p];
43             val[ls] = lazy[ls] = lazy[p];
44             lc[ls] = rc[ls] = maxn[ls] = lazy[p] > 0 ? sum[ls]:0;
45         }
46         if(rs){
47             sum[rs] = siz[rs]*lazy[p];
48             val[rs] = lazy[rs] = lazy[p];
49             lc[rs] = rc[rs] = maxn[rs] = lazy[p] > 0 ? sum[rs]:0;
50         }
51         lazy[p] = -inf;
52     }
53 }
54
55 void update(int p){ //递归地从上到下pushDown信息
56     if(!isRoot(p)) update(f[p]);
57     pushDown(p);
58 }
59
60 inline void rotate(int x){ //将x向上旋转一层的操作
61     int y = f[x], z = f[y], k = Get(x);
62     if(!isRoot(y)) ch[z][ch[z][1] == y] = x;
63     ch[y][k] = ch[x][!k], f[ch[y][k]] = y;
64     ch[x][!k] = y, f[y] = x, f[x] = z;
65     pushUp(y); //要先pushUp(y)
66     pushUp(x);

```

```

67 }
68
69 inline void splay(int x){ //把x旋转到当前splay的根
70     update(x); //将上面的标记完全下放
71     for(int fa; fa = f[x], !isRoot(x); rotate(x)){
72         if(!isRoot(fa)) rotate(Get(fa) == Get(x) ? fa : x);
73     }
74 }
75
76 inline int access(int x){ //把从根到x的所有点放在一条实链里，返回这个splay的根
77     int p;
78     for(p = 0; x; p = x, x = f[x]){
79         splay(x), ch[x][1] = p, pushUp(x);
80     }
81     return p;
82 }
83
84 inline void makeRoot(int p){ //使x点成为整棵树的根
85     access(p);
86     splay(p);
87     swap(ch[p][0],ch[p][1]); //把整条链反向
88     rev[p] ^= 1;
89 }
90
91 inline void link(int x,int y){ //在x、y两点间连一条边
92     makeRoot(x), f[x] = y; //dfs建树，每条边都是有效的，因此不需要判断是否有效
93 }
94 }st;
95
96 int main(){
97     scanf("%d",&n);
98     rep(i,1,n){
99         scanf("%lld",&st.val[i]);
100        st.siz[i] = 1; st.sum[i] = st.val[i];
101        st.lazy[i] = -inf;
102        st.lc[i] = st.rc[i] = st.maxn[i] = st.val[i] > 0 ? st.val[i]:0;
103    }
104    rep(i,1,n-1){
105        int a,b; scanf("%d%d",&a,&b);
106        st.link(a,b);
107    }
108    scanf("%d",&Q);
109    while(Q--){
110        int op; scanf("%d",&op);
111        if(op == 1){ //a->b max
112            int a,b; scanf("%d%d",&a,&b);
113            st.makeRoot(a);
114            b = st.access(b);
115            printf("%lld\n",st.maxn[b]);
116        }
117        else{ //a->b to c
118            int a,b; ll c; scanf("%d%d%lld",&a,&b,&c);
119            st.makeRoot(a);
120            b = st.access(b);
121            st.val[b] = st.lazy[b] = c;
122            st.sum[b] = st.siz[b]*c;
123            st.lc[b] = st.rc[b] = st.maxn[b] = c > 0 ? st.sum[b]:0;
124        }
125    }

```

```
126     return 0;
127 }
```

4.15.4 最小差值生成树

题意:

n 个点, m 条边的一个无向图, 求边权最大值与最小值的差值最小的生成树。 $(1 \leq n \leq 5 * 10^4, 1 \leq m \leq 2 * 10^5)$

思路:

关于这类特殊生成树问题, 一般考虑用 LCT 动态维护树结构然后更新答案。

此题也可以这样考虑。将边按边权从小到大排序, 如果 (a, b) 两点不连通, 则加上该边, 如果 (a, b) 两点连通, 则将 $a \rightarrow b$ 路径上边权最小的边去除, 然后连上当前的边。维护过程不断更新最大值与最小值的差值, 不断取 \min 即可。

因此只需要维护一个边权 LCT , 并且维护路径最小值以及最小值点的编号, 然后动态加边删边即可。还需要对在树中的边打上标记, 去除的时候删去标记, 用于查找整棵树中的最小边权。

```
1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 using namespace std;
4 const int N = 2e5+5e4+100;
5 const int M = 2e5+10;
6 int n, m, vis[N], pos = 1, num, ans = 1e9;
7 struct Edge{
8     int a,b,w;
9     bool operator < (Edge xx) const {
10         return w < xx.w;
11     }
12 }e[N];
13
14 void dbg() {cout << "\n";}
15 template<typename T, typename... A> void dbg(T a, A... x) {cout << a << ' ' ; dbg(x...)}
16 #define logs(x...) {cout << #x << " -> "; dbg(x);}
17
18 struct LCT{
19     #define ls ch[p][0]
20     #define rs ch[p][1]
21     #define Get(p) (ch[f[p]][1] == p)
22     int ch[N][2], f[N], val[N], minn[N], mpos[N], rev[N];
23
24     inline void clear(int p){ //清除这个点的信息
25         ch[p][0] = ch[p][1] = f[p] = val[p] = mpos[p] = minn[p] = 0;
26     }
27
28     inline int isRoot(int p){
29         return ch[f[p]][0] != p && ch[f[p]][1] != p;
30     }
31
32     inline void pushUp(int p){
33         minn[p] = val[p]; mpos[p] = p;
34         if(ls && minn[ls] < minn[p]) minn[p] = minn[ls], mpos[p] = mpos[ls];
35         if(rs && minn[rs] < minn[p]) minn[p] = minn[rs], mpos[p] = mpos[rs];
36     }
37
38     inline void pushDown(int p){
39         if(rev[p]){

```

```

40     if(ls) rev[ls] ^= 1, swap(ch[ls][0],ch[ls][1]);
41     if(rs) rev[rs] ^= 1, swap(ch[rs][0],ch[rs][1]);
42     rev[p] = 0;
43 }
44 }
45
46 void update(int p){ //递归地从上到下pushDown信息
47     if(!isRoot(p)) update(f[p]);
48     pushDown(p);
49 }
50
51 inline void rotate(int x){ //将x向上旋转一层的操作
52     int y = f[x], z = f[y], k = Get(x);
53     if(!isRoot(y)) ch[z][ch[z][1] == y] = x;
54     ch[y][k] = ch[x][!k], f[ch[y][k]] = y;
55     ch[x][!k] = y, f[y] = x, f[x] = z;
56     pushUp(y); //要先pushUp(y)
57     pushUp(x);
58 }
59
60 inline void splay(int x){ //把x旋转到当前splay的根
61     update(x); //将上面的标记完全下放
62     for(int fa; fa = f[x], !isRoot(x); rotate(x)){
63         if(!isRoot(fa)) rotate(Get(fa) == Get(x) ? fa : x);
64     }
65 }
66
67 inline int access(int x){ //把从根到x的所有点放在一条实链里，返回这个splay的根
68     int p = 0;
69     for(p = 0; x; p = x, x = f[x]){
70         splay(x), ch[x][1] = p, pushUp(x);
71     }
72     return p;
73 }
74
75 inline void makeRoot(int p){ //使x点成为整棵树的根
76     access(p); splay(p);
77     swap(ch[p][0],ch[p][1]); //把整条链反向
78     rev[p] ^= 1;
79 }
80
81 inline void link(int x,int y){ //在x、y两点间连一条边
82     // if (find(x) != find(y))
83     makeRoot(x), f[x] = y;
84 }
85
86 inline void cut(int x,int p){ //把x、y两点间边删掉
87     makeRoot(x), access(p), splay(p);
88     if (ls == x && !rs) ls = f[x] = 0;
89 }
90
91 inline int find(int p){ //找到x所在树的根节点编号
92     access(p), splay(p);
93     while(ls) pushDown(p), p = ls;
94     return p;
95 }
96 }st;
97
98 signed main() {

```

```

99  scanf("%d%d", &n, &m);
100 rep(i,0,n) st.val[i] = st.minn[i] = 1e5;
101 rep(i,1,m) scanf("%d%d%d",&e[i].a,&e[i].b,&e[i].w);
102 sort(e+1,e+1+m);
103 rep(i,1,m) st.val[i+n] = st.minn[i+n] = e[i].w, st.mpos[i+n] = i+n;
104 rep(i,1,m){
105     if(e[i].a == e[i].b) continue;
106     if(st.find(e[i].a) != st.find(e[i].b)){
107         st.link(e[i].a,i+n); st.link(i+n,e[i].b);
108         num++; vis[i] = 1;
109         while(!vis[pos]) pos++;
110         if(num == n-1) ans = min(ans,e[i].w-e[pos].w);
111     }
112 }else{
113     st.makeRoot(e[i].a);
114     int p1 = st.access(e[i].b);
115     p1 = st.mpos[p1];
116     st.cut(e[p1-n].a,p1); st.cut(p1,e[p1-n].b); vis[p1-n] = 0;
117     st.link(e[i].a,i+n); st.link(i+n,e[i].b); vis[i] = 1;
118     while(!vis[pos]) pos++;
119     if(num == n-1) ans = min(ans,e[i].w-e[pos].w);
120 }
121 }
122 printf("%d\n",ans);
123 return 0;
124 }
```

4.15.5 维护实边与虚边

题意:

n 个点, 一共 q 次操作。一共有两种操作类型, $A\ x\ y$ 表示连通 (x,y) , 保证操作合法, 且始终是棵森林。 $Q\ x\ y$ 表示查询去除 (x,y) 边之后, x 所在树的节点数 * y 所在树的节点数。 $(1 \leq n, q \leq 10^5)$

思路:

我们一般遇到的都是维护链上节点个数的问题, 而此题要求这颗树上的节点个数, 因此我们需要同时维护虚边和实边的信息。

我们令 $sz[x]$ 表示节点 x 子树中节点个数, $sz2[x]$ 表示节点 x 虚儿子的节点个数和。因此 $sz[x] = sz[ls] + sz[rs] + 1 + sz2[x]$, 而这也正是 $pushUp$ 函数。

因此我们只需要维护 $sz2[x]$ 即可, 然后观察哪些函数会改变 $sz2[x]$ 的值, 不难发现, 只有 $makeRoot$ 、 $access$ 、 $link$ 、 cut 会改变边的虚实关系, 其中 $makeRoot$ 主要修改在于调用了 $access$ 函数, 而 cut 只是删除实边不会修改虚边, 因此真正关键的函数即为 $link$ 和 $access$ 函数, 具体操作见代码, 不难思考。

这里主要讲解 $link$ 中修改 $sz2[y]$ 信息时为什么需要将节点 y $makeRoot(y)$, 原因在于单点修改之后, 其祖先节点维护的信息都会发生变化, 因此一般的问题需要 $splay(y)$, 因为一般问题只需要维护实链信息。然后在该问题中还维护了虚链信息, 因此需要 $makeRoot(y)$ 而不是 $splay(y)$ 。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 using namespace std;
4 typedef long long ll;
5 const int N = 100010;
6 int n, q;
7
8 struct LCT{
9     #define ls ch[p][0]
10    #define rs ch[p][1]
11    #define fa ch[p][2]
12    #define son ch[p][3]
13    #define lson ch[p][4]
14    #define rson ch[p][5]
15    #define lch ch[p][6]
16    #define rch ch[p][7]
17    #define lfa ch[p][8]
18    #define rfa ch[p][9]
19    #define lson2 ch[p][10]
20    #define rson2 ch[p][11]
21    #define lch2 ch[p][12]
22    #define rch2 ch[p][13]
23    #define lfa2 ch[p][14]
24    #define rfa2 ch[p][15]
25    #define lson3 ch[p][16]
26    #define rson3 ch[p][17]
27    #define lch3 ch[p][18]
28    #define rch3 ch[p][19]
29    #define lfa3 ch[p][20]
30    #define rfa3 ch[p][21]
31    #define lson4 ch[p][22]
32    #define rson4 ch[p][23]
33    #define lch4 ch[p][24]
34    #define rch4 ch[p][25]
35    #define lfa4 ch[p][26]
36    #define rfa4 ch[p][27]
37    #define lson5 ch[p][28]
38    #define rson5 ch[p][29]
39    #define lch5 ch[p][30]
40    #define rch5 ch[p][31]
41    #define lfa5 ch[p][32]
42    #define rfa5 ch[p][33]
43    #define lson6 ch[p][34]
44    #define rson6 ch[p][35]
45    #define lch6 ch[p][36]
46    #define rch6 ch[p][37]
47    #define lfa6 ch[p][38]
48    #define rfa6 ch[p][39]
49    #define lson7 ch[p][40]
50    #define rson7 ch[p][41]
51    #define lch7 ch[p][42]
52    #define rch7 ch[p][43]
53    #define lfa7 ch[p][44]
54    #define rfa7 ch[p][45]
55    #define lson8 ch[p][46]
56    #define rson8 ch[p][47]
57    #define lch8 ch[p][48]
58    #define rch8 ch[p][49]
59    #define lfa8 ch[p][50]
60    #define rfa8 ch[p][51]
61    #define lson9 ch[p][52]
62    #define rson9 ch[p][53]
63    #define lch9 ch[p][54]
64    #define rch9 ch[p][55]
65    #define lfa9 ch[p][56]
66    #define rfa9 ch[p][57]
67    #define lson10 ch[p][58]
68    #define rson10 ch[p][59]
69    #define lch10 ch[p][60]
70    #define rch10 ch[p][61]
71    #define lfa10 ch[p][62]
72    #define rfa10 ch[p][63]
73    #define lson11 ch[p][64]
74    #define rson11 ch[p][65]
75    #define lch11 ch[p][66]
76    #define rch11 ch[p][67]
77    #define lfa11 ch[p][68]
78    #define rfa11 ch[p][69]
79    #define lson12 ch[p][70]
80    #define rson12 ch[p][71]
81    #define lch12 ch[p][72]
82    #define rch12 ch[p][73]
83    #define lfa12 ch[p][74]
84    #define rfa12 ch[p][75]
85    #define lson13 ch[p][76]
86    #define rson13 ch[p][77]
87    #define lch13 ch[p][78]
88    #define rch13 ch[p][79]
89    #define lfa13 ch[p][80]
90    #define rfa13 ch[p][81]
91    #define lson14 ch[p][82]
92    #define rson14 ch[p][83]
93    #define lch14 ch[p][84]
94    #define rch14 ch[p][85]
95    #define lfa14 ch[p][86]
96    #define rfa14 ch[p][87]
97    #define lson15 ch[p][88]
98    #define rson15 ch[p][89]
99    #define lch15 ch[p][90]
100   #define rch15 ch[p][91]
101   #define lfa15 ch[p][92]
102   #define rfa15 ch[p][93]
103   #define lson16 ch[p][94]
104   #define rson16 ch[p][95]
105   #define lch16 ch[p][96]
106   #define rch16 ch[p][97]
107   #define lfa16 ch[p][98]
108   #define rfa16 ch[p][99]
109   #define lson17 ch[p][100]
110   #define rson17 ch[p][101]
111   #define lch17 ch[p][102]
112   #define rch17 ch[p][103]
113   #define lfa17 ch[p][104]
114   #define rfa17 ch[p][105]
115   #define lson18 ch[p][106]
116   #define rson18 ch[p][107]
117   #define lch18 ch[p][108]
118   #define rch18 ch[p][109]
119   #define lfa18 ch[p][110]
120   #define rfa18 ch[p][111]
121   #define lson19 ch[p][112]
122   #define rson19 ch[p][113]
123   #define lch19 ch[p][114]
124   #define rch19 ch[p][115]
125   #define lfa19 ch[p][116]
126   #define rfa19 ch[p][117]
127   #define lson20 ch[p][118]
128   #define rson20 ch[p][119]
129   #define lch20 ch[p][120]
130   #define rch20 ch[p][121]
131   #define lfa20 ch[p][122]
132   #define rfa20 ch[p][123]
133   #define lson21 ch[p][124]
134   #define rson21 ch[p][125]
135   #define lch21 ch[p][126]
136   #define rch21 ch[p][127]
137   #define lfa21 ch[p][128]
138   #define rfa21 ch[p][129]
139   #define lson22 ch[p][130]
140   #define rson22 ch[p][131]
141   #define lch22 ch[p][132]
142   #define rch22 ch[p][133]
143   #define lfa22 ch[p][134]
144   #define rfa22 ch[p][135]
145   #define lson23 ch[p][136]
146   #define rson23 ch[p][137]
147   #define lch23 ch[p][138]
148   #define rch23 ch[p][139]
149   #define lfa23 ch[p][140]
150   #define rfa23 ch[p][141]
151   #define lson24 ch[p][142]
152   #define rson24 ch[p][143]
153   #define lch24 ch[p][144]
154   #define rch24 ch[p][145]
155   #define lfa24 ch[p][146]
156   #define rfa24 ch[p][147]
157   #define lson25 ch[p][148]
158   #define rson25 ch[p][149]
159   #define lch25 ch[p][150]
160   #define rch25 ch[p][151]
161   #define lfa25 ch[p][152]
162   #define rfa25 ch[p][153]
163   #define lson26 ch[p][154]
164   #define rson26 ch[p][155]
165   #define lch26 ch[p][156]
166   #define rch26 ch[p][157]
167   #define lfa26 ch[p][158]
168   #define rfa26 ch[p][159]
169   #define lson27 ch[p][160]
170   #define rson27 ch[p][161]
171   #define lch27 ch[p][162]
172   #define rch27 ch[p][163]
173   #define lfa27 ch[p][164]
174   #define rfa27 ch[p][165]
175   #define lson28 ch[p][166]
176   #define rson28 ch[p][167]
177   #define lch28 ch[p][168]
178   #define rch28 ch[p][169]
179   #define lfa28 ch[p][170]
180   #define rfa28 ch[p][171]
181   #define lson29 ch[p][172]
182   #define rson29 ch[p][173]
183   #define lch29 ch[p][174]
184   #define rch29 ch[p][175]
185   #define lfa29 ch[p][176]
186   #define rfa29 ch[p][177]
187   #define lson30 ch[p][178]
188   #define rson30 ch[p][179]
189   #define lch30 ch[p][180]
190   #define rch30 ch[p][181]
191   #define lfa30 ch[p][182]
192   #define rfa30 ch[p][183]
193   #define lson31 ch[p][184]
194   #define rson31 ch[p][185]
195   #define lch31 ch[p][186]
196   #define rch31 ch[p][187]
197   #define lfa31 ch[p][188]
198   #define rfa31 ch[p][189]
199   #define lson32 ch[p][190]
200   #define rson32 ch[p][191]
201   #define lch32 ch[p][192]
202   #define rch32 ch[p][193]
203   #define lfa32 ch[p][194]
204   #define rfa32 ch[p][195]
205   #define lson33 ch[p][196]
206   #define rson33 ch[p][197]
207   #define lch33 ch[p][198]
208   #define rch33 ch[p][199]
209   #define lfa33 ch[p][200]
210   #define rfa33 ch[p][201]
211   #define lson34 ch[p][202]
212   #define rson34 ch[p][203]
213   #define lch34 ch[p][204]
214   #define rch34 ch[p][205]
215   #define lfa34 ch[p][206]
216   #define rfa34 ch[p][207]
217   #define lson35 ch[p][208]
218   #define rson35 ch[p][209]
219   #define lch35 ch[p][210]
220   #define rch35 ch[p][211]
221   #define lfa35 ch[p][212]
222   #define rfa35 ch[p][213]
223   #define lson36 ch[p][214]
224   #define rson36 ch[p][215]
225   #define lch36 ch[p][216]
226   #define rch36 ch[p][217]
227   #define lfa36 ch[p][218]
228   #define rfa36 ch[p][219]
229   #define lson37 ch[p][220]
230   #define rson37 ch[p][221]
231   #define lch37 ch[p][222]
232   #define rch37 ch[p][223]
233   #define lfa37 ch[p][224]
234   #define rfa37 ch[p][225]
235   #define lson38 ch[p][226]
236   #define rson38 ch[p][227]
237   #define lch38 ch[p][228]
238   #define rch38 ch[p][229]
239   #define lfa38 ch[p][230]
240   #define rfa38 ch[p][231]
241   #define lson39 ch[p][232]
242   #define rson39 ch[p][233]
243   #define lch39 ch[p][234]
244   #define rch39 ch[p][235]
245   #define lfa39 ch[p][236]
246   #define rfa39 ch[p][237]
247   #define lson40 ch[p][238]
248   #define rson40 ch[p][239]
249   #define lch40 ch[p][240]
250   #define rch40 ch[p][241]
251   #define lfa40 ch[p][242]
252   #define rfa40 ch[p][243]
253   #define lson41 ch[p][244]
254   #define rson41 ch[p][245]
255   #define lch41 ch[p][246]
256   #define rch41 ch[p][247]
257   #define lfa41 ch[p][248]
258   #define rfa41 ch[p][249]
259   #define lson42 ch[p][250]
260   #define rson42 ch[p][251]
261   #define lch42 ch[p][252]
262   #define rch42 ch[p][253]
263   #define lfa42 ch[p][254]
264   #define rfa42 ch[p][255]
265   #define lson43 ch[p][256]
266   #define rson43 ch[p][257]
267   #define lch43 ch[p][258]
268   #define rch43 ch[p][259]
269   #define lfa43 ch[p][260]
270   #define rfa43 ch[p][261]
271   #define lson44 ch[p][262]
272   #define rson44 ch[p][263]
273   #define lch44 ch[p][264]
274   #define rch44 ch[p][265]
275   #define lfa44 ch[p][266]
276   #define rfa44 ch[p][267]
277   #define lson45 ch[p][268]
278   #define rson45 ch[p][269]
279   #define lch45 ch[p][270]
280   #define rch45 ch[p][271]
281   #define lfa45 ch[p][272]
282   #define rfa45 ch[p][273]
283   #define lson46 ch[p][274]
284   #define rson46 ch[p][275]
285   #define lch46 ch[p][276]
286   #define rch46 ch[p][277]
287   #define lfa46 ch[p][278]
288   #define rfa46 ch[p][279]
289   #define lson47 ch[p][280]
290   #define rson47 ch[p][281]
291   #define lch47 ch[p][282]
292   #define rch47 ch[p][283]
293   #define lfa47 ch[p][284]
294   #define rfa47 ch[p][285]
295   #define lson48 ch[p][286]
296   #define rson48 ch[p][287]
297   #define lch48 ch[p][288]
298   #define rch48 ch[p][289]
299   #define lfa48 ch[p][290]
300   #define rfa48 ch[p][291]
301   #define lson49 ch[p][292]
302   #define rson49 ch[p][293]
303   #define lch49 ch[p][294]
304   #define rch49 ch[p][295]
305   #define lfa49 ch[p][296]
306   #define rfa49 ch[p][297]
307   #define lson50 ch[p][298]
308   #define rson50 ch[p][299]
309   #define lch50 ch[p][300]
310   #define rch50 ch[p][301]
311   #define lfa50 ch[p][302]
312   #define rfa50 ch[p][303]
313   #define lson51 ch[p][304]
314   #define rson51 ch[p][305]
315   #define lch51 ch[p][306]
316   #define rch51 ch[p][307]
317   #define lfa51 ch[p][308]
318   #define rfa51 ch[p][309]
319   #define lson52 ch[p][310]
320   #define rson52 ch[p][311]
321   #define lch52 ch[p][312]
322   #define rch52 ch[p][313]
323   #define lfa52 ch[p][314]
324   #define rfa52 ch[p][315]
325   #define lson53 ch[p][316]
326   #define rson53 ch[p][317]
327   #define lch53 ch[p][318]
328   #define rch53 ch[p][319]
329   #define lfa53 ch[p][320]
330   #define rfa53 ch[p][321]
331   #define lson54 ch[p][322]
332   #define rson54 ch[p][323]
333   #define lch54 ch[p][324]
334   #define rch54 ch[p][325]
335   #define lfa54 ch[p][326]
336   #define rfa54 ch[p][327]
337   #define lson55 ch[p][328]
338   #define rson55 ch[p][329]
339   #define lch55 ch[p][330]
340   #define rch55 ch[p][331]
341   #define lfa55 ch[p][332]
342   #define rfa55 ch[p][333]
343   #define lson56 ch[p][334]
344   #define rson56 ch[p][335]
345   #define lch56 ch[p][336]
346   #define rch56 ch[p][337]
347   #define lfa56 ch[p][338]
348   #define rfa56 ch[p][339]
349   #define lson57 ch[p][340]
350   #define rson57 ch[p][341]
351   #define lch57 ch[p][342]
352   #define rch57 ch[p][343]
353   #define lfa57 ch[p][344]
354   #define rfa57 ch[p][345]
355   #define lson58 ch[p][346]
356   #define rson58 ch[p][347]
357   #define lch58 ch[p][348]
358   #define rch58 ch[p][349]
359   #define lfa58 ch[p][350]
360   #define rfa58 ch[p][351]
361   #define lson59 ch[p][352]
362   #define rson59 ch[p][353]
363   #define lch59 ch[p][354]
364   #define rch59 ch[p][355]
365   #define lfa59 ch[p][356]
366   #define rfa59 ch[p][357]
367   #define lson60 ch[p][358]
368   #define rson60 ch[p][359]
369   #define lch60 ch[p][360]
370   #define rch60 ch[p][361]
371   #define lfa60 ch[p][362]
372   #define rfa60 ch[p][363]
373   #define lson61 ch[p][364]
374   #define rson61 ch[p][365]
375   #define lch61 ch[p][366]
376   #define rch61 ch[p][367]
377   #define lfa61 ch[p][368]
378   #define rfa61 ch[p][369]
379   #define lson62 ch[p][370]
380   #define rson62 ch[p][371]
381   #define lch62 ch[p][372]
382   #define rch62 ch[p][373]
383   #define lfa62 ch[p][374]
384   #define rfa62 ch[p][375]
385   #define lson63 ch[p][376]
386   #define rson63 ch[p][377]
387   #define lch63 ch[p][378]
388   #define rch63 ch[p][379]
389   #define lfa63 ch[p][380]
390   #define rfa63 ch[p][381]
391   #define lson64 ch[p][382]
392   #define rson64 ch[p][383]
393   #define lch64 ch[p][384]
394   #define rch64 ch[p][385]
395   #define lfa64 ch[p][386]
396   #define rfa64 ch[p][387]
397   #define lson65 ch[p][388]
398   #define rson65 ch[p][389]
399   #define lch65 ch[p][390]
400   #define rch65 ch[p][391]
401   #define lfa65 ch[p][392]
402   #define rfa65 ch[p][393]
403   #define lson66 ch[p][394]
404   #define rson66 ch[p][395]
405   #define lch66 ch[p][396]
406   #define rch66 ch[p][397]
407   #define lfa66 ch[p][398]
408   #define rfa66 ch[p][399]
409   #define lson67 ch[p][400]
410   #define rson67 ch[p][401]
411   #define lch67 ch[p][402]
412   #define rch67 ch[p][403]
413   #define lfa67 ch[p][404]
414   #define rfa67 ch[p][405]
415   #define lson68 ch[p][406]
416   #define rson68 ch[p][407]
417   #define lch68 ch[p][408]
418   #define rch68 ch[p][409]
419   #define lfa68 ch[p][410]
420   #define rfa68 ch[p][411]
421   #define lson69 ch[p][412]
422   #define rson69 ch[p][413]
423   #define lch69 ch[p][414]
424   #define rch69 ch[p][415]
425   #define lfa69 ch[p][416]
426   #define rfa69 ch[p][417]
427   #define lson70 ch[p][418]
428   #define rson70 ch[p][419]
429   #define lch70 ch[p][420]
430   #define rch70 ch[p][421]
431   #define lfa70 ch[p][422]
432   #define rfa70 ch[p][423]
433   #define lson71 ch[p][424]
434   #define rson71 ch[p][425]
435   #define lch71 ch[p][426]
436   #define rch71 ch[p][427]
437   #define lfa71 ch[p][428]
438   #define rfa71 ch[p][429]
439   #define lson72 ch[p][430]
440   #define rson72 ch[p][431]
441   #define lch72 ch[p][432]
442   #define rch72 ch[p][433]
443   #define lfa72 ch[p][434]
444   #define rfa72 ch[p][435]
445   #define lson73 ch[p][436]
446   #define rson73 ch[p][437]
447   #define lch73 ch[p][438]
448   #define rch73 ch[p][439]
449   #define lfa73 ch[p][440]
450   #define rfa73 ch[p][441]
451   #define lson74 ch[p][442]
452   #define rson74 ch[p][443]
453   #define lch74 ch[p][444]
454   #define rch74 ch[p][445]
455   #define lfa74 ch[p][446]
456   #define rfa74 ch[p][447]
457   #define lson75 ch[p][448]
458   #define rson75 ch[p][449]
459   #define lch75 ch[p][450]
460   #define rch75 ch[p][451]
461   #define lfa75 ch[p][452]
462   #define rfa75 ch[p][453]
463   #define lson76 ch[p][454]
464   #define rson76 ch[p][455]
465   #define lch76 ch[p][456]
466   #define rch76 ch[p][457]
467   #define lfa76 ch[p][458]
468   #define rfa76 ch[p][459]
469   #define lson77 ch[p][460]
470   #define rson77 ch[p][461]
471   #define lch77 ch[p][462]
472   #define rch77 ch[p][463]
473   #define lfa77 ch[p][464]
474   #define rfa77 ch[p][465]
475   #define lson78 ch[p][466]
476   #define rson78 ch[p][467]
477   #define lch78 ch[p][468]
478   #define rch78 ch[p][469]
479   #define lfa78 ch[p][470]
480   #define rfa78 ch[p][471]
481   #define lson79 ch[p][472]
482   #define rson79 ch[p][473]
483   #define lch79 ch[p][474]
484   #define rch79 ch[p][475]
485   #define lfa79 ch[p][476]
486   #define rfa79 ch[p][477]
487   #define lson80 ch[p][478]
488   #define rson80 ch[p][479]
489   #define lch80 ch[p][480]
490   #define rch80 ch[p][481]
491   #define lfa80 ch[p][482]
492   #define rfa80 ch[p][483]
493   #define lson81 ch[p][484]
494   #define rson81 ch[p][485]
495   #define lch81 ch[p][486]
496   #define rch81 ch[p][487]
497   #define lfa81 ch[p][488]
498   #define rfa81 ch[p][489]
499   #define lson82 ch[p][490]
500   #define rson82 ch[p][491]
501   #define lch82 ch[p][492]
502   #define rch82 ch[p][493]
503   #define lfa82 ch[p][494]
504   #define rfa82 ch[p][495]
505   #define lson83 ch[p][496]
506   #define rson83 ch[p][497]
507   #define lch83 ch[p][498]
508   #define rch83 ch[p][499]
509   #define lfa83 ch[p][500]
510   #define rfa83 ch[p][501]
511   #define lson84 ch[p][502]
512   #define rson84 ch[p][503]
513   #define lch84 ch[p][504]
514   #define rch84 ch[p][505]
515   #define lfa84 ch[p][506]
516   #define rfa84 ch[p][507]
517   #define lson85 ch[p][508]
518   #define rson85 ch[p][509]
519   #define lch85 ch[p][510]
520   #define rch85 ch[p][511]
521   #define lfa85 ch[p][512]
522   #define rfa85 ch[p][513]
523   #define lson86 ch[p][514]
524   #define rson86 ch[p][515]
525   #define lch86 ch[p][516]
526   #define rch86 ch[p][517]
527   #define lfa86 ch[p][518]
528   #define rfa86 ch[p][519]
529   #define lson87 ch[p][520]
530   #define rson87 ch[p][521]
531   #define lch87 ch[p][522]
532   #define rch87 ch[p][523]
533   #define lfa87 ch[p][524]
534   #define rfa87 ch[p][525]
535   #define lson88 ch[p][526]
536   #define rson88 ch[p][527]
537   #define lch88 ch[p][528]
538   #define rch88 ch[p][529]
539   #define lfa88 ch[p][530]
540   #define rfa88 ch[p][531]
541   #define lson89 ch[p][532]
542   #define rson89 ch[p][533]
543   #define lch89 ch[p][534]
544   #define rch89 ch[p][535]
545   #define lfa89 ch[p][536]
546   #define rfa89 ch[p][537]
547   #define lson90 ch[p][538]
548   #define rson90 ch[p][539]
549   #define lch90 ch[p][540]
550   #define rch90 ch[p][541]
551   #define lfa90 ch[p][542]
552   #define rfa90 ch[p][543]
553   #define lson91 ch[p][544]
554   #define rson91 ch[p][545]
555   #define lch91 ch[p][546]
556   #define rch91 ch[p][547]
557   #define lfa91 ch[p][548]
558   #define rfa91 ch[p][549]
559   #define lson92 ch[p][550]
560   #define rson92 ch[p][551]
561   #define lch92 ch[p][552]
562   #define rch92 ch[p][553]
563   #define lfa92 ch[p][554]
564   #define rfa92 ch[p][555]
565   #define lson93 ch[p][556]
566   #define rson93 ch[p][557]
567   #define lch93 ch[p][558]
568   #define rch93 ch[p][559]
569   #define lfa93 ch[p][560]
570   #define rfa93 ch[p][561]
571   #define lson94 ch[p][562]
572   #define rson94 ch[p][563]
573   #define lch94 ch[p][564]
574   #define rch94 ch[p][565]
575   #define lfa94 ch[p][566]
576   #define rfa94 ch[p][567]
577   #define lson95 ch[p][568]
578   #define rson95 ch[p][569]
579   #define lch95 ch[p][570]
580   #define rch95 ch[p][571]
581   #define lfa95 ch[p][572]
582   #define rfa95 ch[p][573]
583   #define lson96 ch[p][574]
584   #define rson96 ch[p][575]
585   #define lch96 ch[p][576]
586   #define rch96 ch[p][577]
587   #define lfa96 ch[p][578]
588   #define rfa96 ch[p][579]
589   #define lson97 ch[p][580]
590   #define rson97 ch[p][581]
591   #define lch97 ch[p][582]
592   #define rch97 ch[p][583]
593   #define lfa97 ch[p][584]
594   #define rfa97 ch[p][585]
595   #define lson98 ch[p][586]
596   #define rson98 ch[p][587]
597   #define lch98 ch[p][588]
598   #define rch98 ch[p][589]
599   #define lfa98 ch[p][590]
600   #define rfa98 ch[p][591]
601   #define lson99 ch[p][592]
602   #define rson99 ch[p][593]
603   #define lch99 ch[p][594]
604   #define rch99 ch[p][595]
605   #define lfa99 ch[p][596]
606   #define rfa99 ch[p][597]
607   #define lson100 ch[p][598]
608   #define rson100 ch[p][599]
609   #define lch100 ch[p][6
```

```

10 #define rs ch[p][1]
11 #define Get(p) (ch[f[p]][1] == p)
12 int ch[N][2], f[N], siz[N], siz2[N], rev[N];
13
14 inline void clear(int p){ //清除这个点的信息
15     ch[p][0] = ch[p][1] = f[p] = siz[p] = siz2[p] = rev[p] = 0;
16 }
17
18 inline int isRoot(int p){
19     clear(0);
20     return ch[f[p]][0] != p && ch[f[p]][1] != p;
21 }
22
23 inline void pushUp(int p){
24     clear(0);
25     siz[p] = siz[ls] + 1 + siz[rs] + siz2[p];
26 }
27
28 inline void pushDown(int p){
29     clear(0);
30     if(rev[p]){
31         if(ls) rev[ls] ^= 1, swap(ch[ls][0], ch[ls][1]);
32         if(rs) rev[rs] ^= 1, swap(ch[rs][0], ch[rs][1]);
33         rev[p] = 0;
34     }
35 }
36
37 void update(int p){ //递归地从上到下pushDown信息
38     if(!isRoot(p)) update(f[p]);
39     pushDown(p);
40 }
41
42 inline void rotate(int x){ //将x向上旋转一层的操作
43     int y = f[x], z = f[y], k = Get(x);
44     if(!isRoot(y)) ch[z][ch[z][1] == y] = x;
45     ch[y][k] = ch[x][!k], f[ch[y][k]] = y;
46     ch[x][!k] = y, f[y] = x, f[x] = z;
47     pushUp(y); //要先pushUp(y)
48     pushUp(x);
49 }
50
51 inline void splay(int x){ //把x旋转到当前splay的根
52     update(x); //将上面的标记完全下放
53     for(int fa; fa = f[x], !isRoot(x); rotate(x)){
54         if(!isRoot(fa)) rotate(Get(fa) == Get(x) ? fa : x);
55     }
56 }
57
58 inline int access(int x){ //把从根到x的所有点放在一条实链里，返回这个splay的根
59     int p; //每次改变右儿子的值，因为整棵树是中序遍历，放入右儿子才能保证先遍历父亲再遍历儿子
60     for(p = 0; x; p = x, x = f[x]){
61         splay(x), siz2[x] += siz[ch[x][1]] - siz[p], ch[x][1] = p, pushUp(x);
62     }
63     return p;
64 }
65
66 inline void makeRoot(int p){ //使x点成为整棵树的根
67     access(p); splay(p);
68     swap(ch[p][0], ch[p][1]); //把整条链反向

```

```

69     rev[p] ^= 1;
70 }
71
72 inline void link(int x,int y){ //在x、y两点间连一条边
73     //makeRoot(x)的作用是使得x无父亲
74     //makeRoot(y)的作用是使得y无父亲，因此可以修改y的信息，不用去更新y的祖先
75     makeRoot(x), makeRoot(y), f[x] = y, siz2[y] += siz[x]; pushUp(y);
76 }
77
78 inline void cut(int x,int p){ //把x、y两点间边删掉，此处删除的是实边，注意实边和虚边的区别
79     makeRoot(x), access(p), splay(p);
80     if (ls == x && !rs) ls = f[x] = 0;
81 }
82
83 inline int find(int p){ //找到x所在树的根节点编号
84     access(p), splay(p);
85     while(ls) pushDown(p), p = ls;
86     return p;
87 }
88 }st;
89
90 signed main() {
91     scanf("%d%d", &n, &q);
92     rep(i,1,n) st.siz[i] = 1;
93     while (q--) {
94         char op[10]; int x,y; scanf("%s%d%d",op,&x,&y);
95         if(op[0] == 'A') st.link(x,y);
96         else{
97             st.cut(x,y);
98             st.makeRoot(x); st.splay(x);
99             int a1 = st.siz[x];
100            st.makeRoot(y); st.splay(y);
101            int a2 = st.siz[y];
102            st.link(x,y);
103            printf("%lld\n", (ll)a1*(ll)a2);
104        }
105    }
106    return 0;
107 }
```

4.15.6 TopTree

题意:

支持 12 种操作，包括链 \max 、 \min 、 \sum ，子树 \max 、 \min 、 \sum ，换根，换边，子树和链的修改与加值。 $(1 \leq n, m \leq 10^5)$

思路:

TopTree 典型例题，主要思路是对于每个点维护了一个 *splay*，详情看 [claris 的题解]

```

1 /*
2 TopTree即为可以维护子树信息的lct升级版
3 */
4 #include<cstdio>
5 #define N 200010
6 const int inf=~0U>>1;
7 inline void swap(int&a,int&b){int c=a;a=b;b=c;}
8 inline int max(int a,int b){return a>b?a:b;}
9 inline int min(int a,int b){return a<b?a:b;}
```

```

10 inline void read(int&a){
11     char c;bool f=0;a=0;
12     while(!((((c=getchar())>='0')&&(c<='9'))||(c=='-')));
13     if(c!=='-')a=c-'0';else f=1;
14     while(((c=getchar())>='0')&&(c<='9'))(a*=10)+=c-'0';
15     if(f)a=-a;
16 }
17 struct tag{
18     int a,b;//ax+b
19     tag(){a=1,b=0;}
20     tag(int x,int y){a=x,b=y;}
21     inline bool ex(){return a!=1||b;}
22     inline tag operator+(const tag&x){return tag(a*x.a,b*x.a+x.b);}
23 };
24 inline int atag(int x,tag y){return x*y.a+y.b;}
25 struct data{
26     int sum,minv,maxv,size;
27     data(){sum=size=0,minv=inf,maxv=-inf;};
28     data(int x){sum=minv=maxv=x,size=1;};
29     data(int a,int b,int c,int d){sum=a,minv=b,maxv=c,size=d;};
30     inline data operator+(const data&x){return data(sum+x.sum,min(minv,x.minv),max(maxv,x
        .maxv),size+x.size);}
31 };
32 inline data operator+(const data&a,const tag&b){return a.size?data(a.sum*b.a+a.size*b.b
        ,atag(a.minv,b),atag(a.maxv,b),a.size):a;}
33 //son:0-1: 重链儿子, 2-3: AAA树儿子
34 int f[N],son[N][4],a[N],tot,rt,rub,ru[N];bool rev[N],in[N];
35 int val[N];
36 data csum[N],tsum[N],asum[N];
37 tag ctag[N],ttag[N];
38 inline bool isroot(int x,int t){
39     if(t) return !f[x]||!in[f[x]]||!in[x];
40     return !f[x]||(son[f[x]][0]!=x&&son[f[x]][1]!=x)||in[f[x]]||in[x];
41 }
42 inline void rev1(int x){
43     if(!x) return;
44     swap(son[x][0],son[x][1]);rev[x]^=1;
45 }
46 inline void tagchain(int x,tag p){
47     if(!x) return;
48     csum[x]=csum[x]+p;
49     asum[x]=csum[x]+tsum[x];
50     val[x]=atag(val[x],p);
51     ctag[x]=ctag[x]+p;
52 }
53 inline void tagtree(int x,tag p,bool t){
54     if(!x) return;
55     tsum[x]=tsum[x]+p;
56     ttag[x]=ttag[x]+p;
57     if(!in[x]&&t)tagchain(x,p);else asum[x]=csum[x]+tsum[x];
58 }
59 inline void pb(int x){
60     if(!x) return;
61     if(rev[x])rev1(son[x][0]),rev1(son[x][1]),rev[x]=0;
62     if(!in[x]&&ctag[x].ex())tagchain(son[x][0],ctag[x]),tagchain(son[x][1],ctag[x]),ctag[
        x]=tag();
63     if(ttag[x].ex()){
64         tagtree(son[x][0],ttag[x],0),tagtree(son[x][1],ttag[x],0);
65         tagtree(son[x][2],ttag[x],1),tagtree(son[x][3],ttag[x],1);

```

```

66     ttag[x]=tag();
67 }
68 }
69 inline void up(int x){
70     tsum[x]=data();
71     for(int i=0;i<2;i++)if(son[x][i])tsum[x]=tsum[x]+tsum[son[x][i]];
72     for(int i=2;i<4;i++)if(son[x][i])tsum[x]=tsum[x]+asum[son[x][i]];
73     if(in[x]){
74         csum[x]=data();
75         asum[x]=tsum[x];
76     }else{
77         csum[x]=data(val[x]);
78         for(int i=0;i<2;i++)if(son[x][i])csum[x]=csum[x]+csum[son[x][i]];
79         asum[x]=csum[x]+tsum[x];
80     }
81 }
82 inline int child(int x,int t){pb(son[x][t]);return son[x][t];}
83 inline void rotate(int x,int t){
84     int y=f[x],w=(son[y][t+1]==x)+t;
85     son[y][w]=son[x][w^1];
86     if(son[x][w^1])f[son[x][w^1]]=y;
87     if(f[y])for(int z=f[y],i=0;i<4;i++)if(son[z][i]==y)son[z][i]=x;
88     f[x]=f[y];f[y]=x;son[x][w^1]=y;up(y);
89 }
90 inline void splay(int x,int t=0){
91     int s=1,i=x,y;a[1]=i;
92     while(!isroot(i,t))a[++s]=i=f[i];
93     while(s)pb(a[s--]);
94     while(!isroot(x,t)){
95         y=f[x];
96         if(!isroot(y,t)){if((son[f[y]][t]==y)^son[y][t]==x)rotate(x,t);else rotate(y,t);}
97         rotate(x,t);
98     }
99     up(x);
100 }
101 inline int newnode(){
102     int x=rub?ru[rub--]:++tot;
103     son[x][2]=son[x][3]=0;in[x]=1;
104     return x;
105 }
106 inline void setson(int x,int t,int y){son[x][t]=y;f[y]=x;}
107 inline int pos(int x){for(int i=0;i<4;i++)if(son[f[x]][i]==x)return i;return 4;}
108 inline void add(int x,int y){//从x连出一条虚边到y
109     if(!y)return;
110     pb(x);
111     for(int i=2;i<4;i++)if(!son[x][i]){
112         setson(x,i,y);
113         return;
114     }
115     while(son[x][2]&&in[son[x][2]])x=child(x,2);
116     int z=newnode();
117     setson(z,2,son[x][2]);
118     setson(z,3,y);
119     setson(x,2,z);
120     splay(z,2);
121 }
122 inline void del(int x){//将x与其虚边上的父亲断开
123     if(!x)return;
124     splay(x);

```

```

125 if(!f[x])return;
126 int y=f[x];
127 if(in[y]){
128     int s=1,i=y,z=f[y];a[1]=i;
129     while(!isroot(i,2))a[++s]=i=f[i];
130     while(s)pb(a[s--]);
131     if(z){
132         setson(z,pos(y),child(y,pos(x)^1));
133         splay(z,2);
134     }
135     ru[++rub]=y;
136 }else{
137     son[y][pos(x)]=0;
138     splay(y);
139 }
140 f[x]=0;
141 }
142 inline int fa(int x){//通过虚边的父亲
143     splay(x);
144     if(!f[x])return 0;
145     if(!in[f[x]])return f[x];
146     int t=f[x];
147     splay(t,2);
148     return f[t];
149 }
150 inline int access(int x){
151     int y=0;
152     for(;x;y=x,x=fa(x)){
153         splay(x);
154         del(y);
155         add(x,son[x][1]);
156         setson(x,1,y);
157         up(x);
158     }
159     return y;
160 }
161 inline int lca(int x,int y){
162     access(x);
163     return access(y);
164 }
165 inline int root(int x){
166     access(x);
167     splay(x);
168     while(son[x][0])x=son[x][0];
169     return x;
170 }
171 inline void makeroott(int x){
172     access(x);
173     splay(x);
174     rev1(x);
175 }
176 inline void link(int x,int y){
177     makeroott(x);
178     add(y,x);
179     access(x);
180 }
181 inline void cut(int x){
182     access(x);
183     splay(x);

```

```

184     f[son[x][0]]=0;
185     son[x][0]=0;
186     up(x);
187 }
188 inline void changechain(int x,int y,tag p){
189     makeroot(x);
190     access(y);
191     splay(y);
192     tagchain(y,p);
193 }
194 inline data askchain(int x,int y){
195     makeroot(x);
196     access(y);
197     splay(y);
198     return csum[y];
199 }
200 inline void changetree(int x,tag p){
201     access(x);
202     splay(x);
203     val[x]=atag(val[x],p);
204     for(int i=2;i<4;i++)if(son[x][i])tagtree(son[x][i],p,1);
205     up(x);
206     splay(x);
207 }
208 inline data asktree(int x){
209     access(x);
210     splay(x);
211     data t=data(val[x]);
212     for(int i=2;i<4;i++)if(son[x][i])t=t+asum[son[x][i]];
213     return t;
214 }
215 int n,m,x,y,z,k,i,ed[N][2];
216 int main(){
217     read(n);read(m);
218     tot=n;
219     for(i=1;i<n;i++)read(ed[i][0]),read(ed[i][1]); //连边
220     for(i=1;i<=n;i++)read(val[i]),up(i); //先赋点权，再连边
221     for(i=1;i<n;i++)link(ed[i][0],ed[i][1]); //每个点的权值
222     read(rt); //给出根
223     makeroot(rt);
224     while(m--){
225         read(k);
226         if(k==1){//换根，x变成根
227             read(rt);
228             makeroot(rt);
229         }
230         if(k==9){//x的父亲变成y, x父亲换成y
231             read(x),read(y);
232             if(lca(x,y)==x)continue;
233             cut(x);
234             link(y,x);
235             makeroot(rt);
236         }
237         if(k==0){//子树赋值，以x为根的子树点权值改为y
238             read(x),read(y);
239             changetree(x,tag(0,y));
240         }
241         if(k==5){//子树加，x为根子树点权值加上y
242             read(x),read(y);

```

```

243     changetree(x,tag(1,y));
244 }
245 if(k==3){//子树最小值, x为根子树中点权值求min
246     read(x);
247     printf("%d\n",asktree(x).minv);
248 }
249 if(k==4){//子树最大值, x为根子树中点权值求max
250     read(x);
251     printf("%d\n",asktree(x).maxv);
252 }
253 if(k==11){//子树和, x为根子树中点权sum
254     read(x);
255     printf("%d\n",asktree(x).sum);
256 }
257 if(k==2){//链赋值, x-y路径上点权值改为z
258     read(x),read(y),read(z);
259     changechain(x,y,tag(0,z));
260     makeroot(rt);
261 }
262 if(k==6){//链加, x-y路径上点权值加上z
263     read(x),read(y),read(z);
264     changechain(x,y,tag(1,z));
265     makeroot(rt);
266 }
267 if(k==7){//链最小值, x-y路径上点权值求min
268     read(x),read(y);
269     printf("%d\n",askchain(x,y).minv);
270     makeroot(rt);
271 }
272 if(k==8){//链最大值, x-y路径上点权值求max
273     read(x),read(y);
274     printf("%d\n",askchain(x,y).maxv);
275     makeroot(rt);
276 }
277 if(k==10){//链和, x-y路径上点权值求sum
278     read(x),read(y);
279     printf("%d\n",askchain(x,y).sum);
280     makeroot(rt);
281 }
282 }
283 return 0;
284 }
```

4.16 树链剖分

4.16.1 点剖模板

一颗 N 个节点的树，每个节点上都有初始权值。现在有四种操作：操作 1 —— 1 $x\ y\ z$ ，表示将 x 到 y 节点最短路径上所有节点的值加 z 操作 2 —— 2 $x\ y$ ，表示求 x 到 y 节点最短路径上所有节点值之和操作 3 —— 3 $x\ z$ ，表示将以 x 为根节点的子树内所有节点值加 z 操作 4 —— 4 x ，表示求以 x 为根节点的子树内所有节点值之和

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
```

```

8 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
9 ;
10 typedef long long ll;
11 typedef double db;
12 #define int long long
13 const db EPS = 1e-9;
14 const int N = 1e5+100;
15 using namespace std;
16
17 struct Edge { int next,to;} e[2*N];
18 struct Node { int l,r,ls,rs,sum,lazy;} t[2*N];
19 int n,m,root,rt,mod,val[N],head[N],tot,fa[N],d[N],son[N],size[N],top[N],id[N],rk[N];
20 //top[x]: x节点所在链的顶端节点, id[x]: 节点dfs序, rk[x]: dfs序对应的节点
21 //val[x]: 每个点初始权值, fa[x]: 每个点父节点, d[x]: 节点深度, size[x]: 节点子树大小
22 //rt: 线段树根节点编号
23 void init(){
24     memset(head,0,sizeof head);
25     tot = 1, size[0] = 0;
26 }
27 void add(int x, int y){
28     e[++tot].next = head[x], e[tot].to = y, head[x] = tot;
29 }
30
31 void dfs1(int x){ //求出每个点的子树大小、深度、重儿子
32     size[x] = 1, d[x] = d[fa[x]]+1, son[x] = 0;
33     for(int v,i = head[x]; i; i = e[i].next)
34         if((v = e[i].to)!=fa[x]){
35             fa[v] = x, dfs1(v), size[x] += size[v];
36             if(size[son[x]] < size[v])
37                 son[x] = v;
38         }
39 }
40
41 void dfs2(int x, int tp){ //求出每个节点的dfs序, dfs序对应的节点, 以及每个点所在链的顶端节点
42     top[x] = tp, id[x] = ++tot, rk[tot] = x;
43     if(son[x]) dfs2(son[x],tp);
44     for(int v,i = head[x]; i; i = e[i].next)
45         if((v = e[i].to)!=fa[x] && v!=son[x]) dfs2(v,v);
46 }
47
48 inline void pushup(int x){ //基础的线段树向上区间合并
49     t[x].sum = (t[t[x].ls].sum+t[t[x].rs].sum)%mod; //此题需要将sum和对mod取模
50 }
51
52 void build(int l, int r, int x){ //基础建树, 动态开点
53     t[x].l = l, t[x].r = r, t[x].lazy = 0;
54     if(l == r){
55         t[x].sum = val[rk[l]]; return;
56     }
57     int mid = (l+r)>>1;
58     t[x].ls = ++tot, t[x].rs = ++tot;
59     build(l,mid,t[x].ls), build(mid+1,r,t[x].rs), pushup(x);
60 }
61
62 inline int len(int x) { return t[x].r-t[x].l+1; }
63
64 inline void pushdown(int x){ //基础的线段树标记下放
65     if(t[x].lazy && t[x].l != t[x].r){

```

```

66         int ls = t[x].ls, rs = t[x].rs, lz = t[x].lazy;
67         (t[ls].lazy+=lz) %= mod, (t[rs].lazy+=lz) %= mod;
68         (t[ls].sum+=lz*len(ls)) %= mod, (t[rs].sum+=lz*len(rs)) %= mod;
69         t[x].lazy = 0;
70     }
71 }
72
73 void update(int l, int r, int x, int c){ //基础的线段树更新
74     if(t[x].l >= l && t[x].r <= r){
75         (t[x].lazy += c) %= mod, (t[x].sum += len(x)*c) %= mod;
76         return;
77     }
78     pushdown(x);
79     int mid = (t[x].l+t[x].r)>>1;
80     if(mid >= l) update(l,r,t[x].ls,c);
81     if(mid < r) update(l,r,t[x].rs,c);
82     pushup(x);
83 }
84
85 int query(int l, int r, int x){ //基础的线段树查询
86     if(t[x].l >= l && t[x].r <= r) return t[x].sum;
87     pushdown(x);
88     int mid = (t[x].l+t[x].r)>>1, tp = 0;
89     if(mid >= l) tp += query(l,r,t[x].ls);
90     if(mid < r) tp += query(l,r,t[x].rs);
91     return tp%mod;
92 }
93
94 inline int sum(int x, int y){ //将区间分为多条链，对于每条链直接查询
95     int ret = 0;
96     while(top[x] != top[y]){
97         if(d[top[x]] < d[top[y]]) swap(x,y); //让x与y到达同一条链
98         (ret += query(id[top[x]],id[x],rt)) %= mod;
99         x = fa[top[x]];
100    }
101    if(id[x] > id[y]) swap(x,y);
102    return (ret+query(id[x],id[y],rt))%mod;
103 }
104
105 inline void updates(int x, int y, int c){ //区间加z，将区间分为多条链
106     while(top[x] != top[y]){
107         if(d[top[x]] < d[top[y]]) swap(x,y);
108         update(id[top[x]],id[x],rt,c); //对于每条链直接修改
109         x = fa[top[x]];
110    }
111    if(id[x] > id[y]) swap(x,y);
112    update(id[x],id[y],rt,c);
113 }
114
115 signed main()
116 {
117     scanf("%lld%lld%lld%lld", &n, &m, &root, &mod);
118     rep(i,1,n) scanf("%lld", &val[i]);
119     init();
120     rep(i,1,n-1){
121         int x,y; scanf("%lld%lld", &x, &y);
122         add(x,y), add(y,x);
123     }
124     tot = 0, dfs1(root), dfs2(root, root);

```

```

125     tot = 0, build(1,n,rt = ++tot);
126     rep(i,1,m){
127         int op,x,y,k; scanf("%lld",&op);
128         if(op == 1){
129             scanf("%lld%lld%lld",&x,&y,&k);
130             updates(x,y,k);
131         }
132         else if(op == 2){
133             scanf("%lld%lld",&x,&y);
134             printf("%lld\n",sum(x,y));
135         }
136         else if(op == 3){
137             scanf("%lld%lld",&x,&y);
138             update(id[x],id[x]+size[x]-1,rt,y);
139         }
140         else{
141             scanf("%lld",&x);
142             printf("%lld\n",query(id[x],id[x]+size[x]-1,rt));
143         }
144     }
145     return 0;
146 }
```

4.16.2 树上路径颜色段数量

题意：

一颗 n 个节点的树，两个操作：
将 $a \rightarrow b$ 路径上的点都染成颜色 c
查询 $a \rightarrow b$ 路径上的颜色段数量

思路：

很明显是一个树剖问题，树剖的基础实现就不多说了，我们来考虑一下线段树需要维护什么。

首先求的是路径上不同颜色段数量，因此肯定需要维护一个 cnt ，表示路径上不同颜色段的数量。然后我们来看如何实现区间合并，主要观察的就是左区间的右端点和右区间的左端点是否一样，如果一样，合并的 cnt 需要减 1。但是如何去快速查询左右节点颜色呢，因此可以想到再维护区间左右节点颜色即可完成本题。

树剖成多个树链时需要注意链条交界处的颜色是否一致。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
8 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
9
10 typedef long long ll;
11 typedef double db;
12 #define int long long
13 const db EPS = 1e-9;
14 const int N = 1e5+100;
15 using namespace std;
16
17 struct Edge { int next,to;} e[2*N];
18 struct Node { int l,r,ls,rs,cnt,lazy,lc,rc;} t[2*N]; //cnt: 不同颜色个数, lc: 左端颜色, rc
19 : 右端颜色
```

```

18 int n,m,root,rt,mod,val[N],head[N],tot,fa[N],d[N],son[N],size[N],top[N],id[N],rk[N];
19 //top[x]: x节点所在链的顶端节点, id[x]: 节点dfs序, rk[x]: dfs序对应的节点
20 //val[x]: 每个点初始权值, fa[x]: 每个点父节点, d[x]: 节点深度, size[x]: 节点子树大小
21 //rt: 线段树根节点编号
22 void init(){
23     memset(head,0,sizeof head);
24     tot = 1, size[0] = 0;
25 }
26
27 void add(int x, int y){
28     e[++tot].next = head[x], e[tot].to = y, head[x] = tot;
29 }
30
31 void dfs1(int x){ //求出每个点的子树大小、深度、重儿子
32     size[x] = 1, d[x] = d[fa[x]]+1, son[x] = 0;
33     for(int v,i = head[x]; i; i = e[i].next)
34         if((v = e[i].to)!=fa[x]){
35             fa[v] = x, dfs1(v), size[x] += size[v];
36             if(size[son[x]] < size[v])
37                 son[x] = v;
38         }
39 }
40
41 void dfs2(int x, int tp){ //求出每个节点的dfs序, dfs序对应的节点, 以及每个点所在链的顶端节点
42     top[x] = tp, id[x] = ++tot, rk[tot] = x;
43     if(son[x]) dfs2(son[x],tp);
44     for(int v,i = head[x]; i; i = e[i].next)
45         if((v = e[i].to)!=fa[x] && v!=son[x]) dfs2(v,v);
46 }
47
48 inline void pushup(int x){ //基础的线段树向上区间合并
49     t[x].lc = t[t[x].ls].lc, t[x].rc = t[t[x].rs].rc, t[x].cnt = t[t[x].ls].cnt + t[t[x].rs].cnt;
50     if(t[t[x].ls].rc == t[t[x].rs].lc) t[x].cnt--;
51 }
52
53 void build(int l, int r, int x){ //基础建树, 动态开点
54     t[x].l = l, t[x].r = r, t[x].lazy = -1;
55     if(l == r){
56         t[x].lc = t[x].rc = val[rk[l]], t[x].cnt = 1; return;
57     }
58     int mid = (l+r)>>1;
59     t[x].ls = ++tot, t[x].rs = ++tot;
60     build(l,mid,t[x].ls), build(mid+1,r,t[x].rs), pushup(x);
61 }
62
63 inline void pushdown(int x){ //基础的线段树标记下放
64     if(t[x].lazy!=-1 && t[x].l != t[x].r){
65         int ls = t[x].ls, rs = t[x].rs, lz = t[x].lazy;
66         t[ls].lazy = lz, t[rs].lazy = lz;
67         t[ls].lc = t[ls].rc = lz, t[rs].lc = t[rs].rc = lz;
68         t[ls].cnt = t[rs].cnt = 1;
69         t[x].lazy = -1;
70     }
71 }
72
73 void update(int l, int r, int x, int c){ //基础的线段树更新
74     if(t[x].l >= l && t[x].r <= r){
75         t[x].lazy = c, t[x].cnt = 1, t[x].lc = t[x].rc = c;

```

```

76         return;
77     }
78     pushdown(x);
79     int mid = (t[x].l+t[x].r)>>1;
80     if(mid >= l) update(l,r,t[x].ls,c);
81     if(mid < r) update(l,r,t[x].rs,c);
82     pushup(x);
83 }
84
85 int query(int l, int r, int x){ //基础的线段树查询
86 // LOG2("l",l,"r",r);
87 // LOG2("t[x].l",t[x].l,"t[x].r",t[x].r);
88     if(t[x].l >= l && t[x].r <= r) return t[x].cnt;
89     pushdown(x);
90     int mid = (t[x].l+t[x].r)>>1, tp = 0;
91     if(mid >= l && mid >= r) tp += query(l,r,t[x].ls);
92     else if(mid < l && mid < r) tp += query(l,r,t[x].rs);
93     else if(mid >= l && mid < r){
94         tp += query(l,r,t[x].ls);
95         tp += query(l,r,t[x].rs);
96         if(t[t[x].ls].rc == t[t[x].rs].lc) tp--;
97     }
98     return tp;
99 }
100
101 int query_color(int pos, int x){ //查询单点颜色
102     if(t[x].l == pos) return t[x].lc;
103     if(t[x].r == pos) return t[x].rc;
104     if(pos >= t[x].l && pos <= t[x].r && t[x].cnt == 1) return t[x].lc;
105     int mid = (t[x].l+t[x].r)>>1;
106     if(pos <= mid) return query_color(pos,t[x].ls);
107     else return query_color(pos,t[x].rs);
108 }
109
110 inline int sum(int x, int y){ //将区间分为多条链，对于每条链直接查询
111     int ret = 0;
112     while(top[x] != top[y]){ //让x与y到达同一条链
113         if(d[top[x]] < d[top[y]]) swap(x,y); //找到更深的点
114         ret += query(id[top[x]],id[x],rt);
115         // LOG1("ret",ret);
116         if(query_color(id[top[x]],rt) == query_color(id[fa[top[x]]],rt)) ret--;
117         x = fa[top[x]];
118     }
119     if(id[x] > id[y]) swap(x,y);
120     return (ret+query(id[x],id[y],rt));
121 }
122
123 inline void updates(int x, int y, int c){ //区间加z，将区间分为多条链
124     while(top[x] != top[y]){
125         if(d[top[x]] < d[top[y]]) swap(x,y);
126         update(id[top[x]],id[x],rt,c); //对于每条链直接修改
127         x = fa[top[x]];
128     }
129     if(id[x] > id[y]) swap(x,y);
130     update(id[x],id[y],rt,c);
131 }
132
133 signed main()
134 {

```

```

135     scanf("%lld%lld",&n,&m);
136     rep(i,1,n) scanf("%lld",&val[i]);
137     init();
138     rep(i,1,n-1){
139         int x,y; scanf("%lld%lld",&x,&y);
140         add(x,y), add(y,x);
141     }
142     scanf("%lld",&m);
143     root = 1;
144     tot = 0, dfs1(root), dfs2(root, root);
145     tot = 0, build(1,n,rt = ++tot);
146     rep(i,1,m){
147         char op[10]; scanf("%s",op);
148         int x,y,z;
149         if(op[0] == 'Q'){
150             scanf("%lld%lld",&x,&y);
151             printf("%lld\n",sum(x,y));
152         }
153         else{
154             scanf("%lld%lld%lld",&x,&y,&z);
155             updates(x,y,z);
156         }
157     }
158     return 0;
159 }
```

4.16.3 动态开点树剖

题意：

现在有 n 个城市，构成了一棵树。每个城市都有自己信仰的宗教，以及城市评级。现在一共有四种操作：

某个城市改信 c 教

某个城市的评级调整为 w

$x -> y$ 路径上所有与 x 信仰相同的城市的评级之和

$x -> y$ 路径上所有与 x 信仰相同的城市的评级最大值

$$(N, Q \leq 10^5, C \leq 10^5)$$

思路：

需要维护的操作只是单点修改和区间最值与最大值，维护的操作都不难。主要困难的地方在于最值和最大值都只在信仰相同的城市之间统计，因此我们需要对每个信仰都建一颗线段树。

但是由于空间的限制，对每个宗教都建一颗完整的线段树是不可能的，因此我们需要动态开点的操作。采用动态开点的原因是本题初始最多只有 10^5 个点，操作最多也只有 10^5 次，因此有效的点最多只有 $2 * 10^5$ 个，所以我们只需要维护这些有效点即可。

我们再来仔细讲一下动态开点的原理。如下图所示，现在只有点 A 是有效点，因此我们只需给 $root -> A$ 路径上的点分配空间，不需要给其他的点分配空间，因此就达到了简化空间的目的。所以动态开一个点的空间费用最多是 $\log n$ ，我们只需要给每棵树记一个根节点，以及每个节点对应的左右儿子编号即可。因此我们也不需要之前建树的 $build$ 函数了，用 $update$ 函数动态插入每个点即可。

回到这道题来，解决的思路就很简单了。给每个宗教建一颗线段树，维护每颗线段树的根节点编号，然后对于每个线段树进行查询和修改即可，一个点从信仰 a 变为信仰 b ，只需在 a 线段树中将这个点赋为 0，然后在 b 线段树将这个点再赋值即可。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
```

```

4 #include <algorithm>
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
8 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
9 ;
10 typedef long long ll;
11 typedef double db;
12 const db EPS = 1e-9;
13 using namespace std;
14 const int N = 2*1e5+1000;
15
16 int ls[20*N], rs[20*N], val[20*N], maxn[20*N], sz, rt[N];
17 int n,m,head[N],tot,son[N],size[N],d[N],id[N],rk[N],fa[N],top[N],ans1,ans2;
18 int re[N],lev[N];
19 struct Edge{
20     int to,next;
21 }e[N];
22
23 void init() { tot = 1, memset(head,0,sizeof head); }
24 void add(int x, int y) { e[++tot].to = y, e[tot].next = head[x], head[x] = tot; }
25 void dfs1(int x){ //求出每个点的子树大小、深度、重儿子
26     size[x] = 1, d[x] = d[fa[x]]+1, son[x] = 0;
27     for(int v,i = head[x]; i; i = e[i].next)
28         if((v = e[i].to)!=fa[x]){
29             fa[v] = x, dfs1(v), size[x] += size[v];
30             if(size[son[x]] < size[v])
31                 son[x] = v;
32         }
33 }
34 void dfs2(int x, int tp){ //求出每个节点的dfs序，dfs序对应的节点，以及每个点所在链的顶端节点
35     top[x] = tp, id[x] = ++tot, rk[tot] = x;
36     if(son[x]) dfs2(son[x],tp);
37     for(int v,i = head[x]; i; i = e[i].next)
38         if((v = e[i].to)!=fa[x] && v!=son[x]) dfs2(v,v);
39 }
40 void update(int& now, int l, int r, int x, int c){ //单点修改
41     if(!now) now = ++sz;
42     if(l == r){
43         val[now] = c, maxn[now] = c;
44         return;
45     }
46     int mid = (l+r)>>1;
47     if(x <= mid) update(ls[now],l,mid,x,c);
48     if(x > mid) update(rs[now],mid+1,r,x,c);
49     val[now] = val[ls[now]]+val[rs[now]];
50     maxn[now] = max(maxn[ls[now]],maxn[rs[now]]);
51 }
52 void query(int now, int l, int r, int xx, int yy){
53     if(l >= xx && r <= yy){
54         ans1 += val[now], ans2 = max(ans2,maxn[now]);
55         return;
56     }
57     int mid = (l+r)>>1;
58     if(xx <= mid) query(ls[now],l,mid,xx,yy);
59     if(yy > mid) query(rs[now],mid+1,r,xx,yy);
60 }
61 inline void updates(int x, int y){ //区间加z，将区间分为多条链
62     int tp = re[x];

```

```

62     while(top[x] != top[y]){
63         if(d[top[x]] < d[top[y]]) swap(x,y);
64         query(rt[tp],1,n,id[top[x]],id[x]); //对于每条链直接修改
65         x = fa[top[x]];
66     }
67     if(id[x] > id[y]) swap(x,y);
68     query(rt[tp],1,n,id[x],id[y]);
69 }
70 int main()
71 {
72     scanf("%d%d",&n,&m);
73     rep(i,1,n) scanf("%d%d",&lev[i],&re[i]);
74     init();
75     rep(i,1,n-1){
76         int xx,yy; scanf("%d%d",&xx,&yy);
77         add(xx,yy), add(yy,xx);
78     }
79     tot = sz = 0, dfs1(1), dfs2(1,1);
80     rep(i,1,n) update(rt[re[i]],1,n,id[i],lev[i]);
81     rep(i,1,m){
82         char s[10]; int x,y;
83         scanf("%s",s);
84         if(s[1] == 'C'){
85             scanf("%d%d",&x,&y);
86             update(rt[re[x]],1,n,id[x],0);
87             update(rt[y],1,n,id[x],lev[x]);
88             re[x] = y;
89         }
90         else if(s[1] == 'W'){
91             scanf("%d%d",&x,&y);
92             update(rt[re[x]],1,n,id[x],y);
93             lev[x] = y;
94         }
95         else{
96             scanf("%d%d",&x,&y);
97             ans1 = ans2 = 0;
98             updates(x,y);
99             if(s[1] == 'S') printf("%d\n",ans1);
100            else printf("%d\n",ans2);
101        }
102    }
103    return 0;
104 }

```

4.16.4 树剖换根

题意：

n 个点的树，每个顶点都有一个值，需要进行三种操作：

将树根修改为 x

将 $x -> y$ 路径上所有点的值修改为 v

询问在当前树根状态下，以节点 x 为根的子树中的最小值

需要注意，树根修改了之后，节点 x 对应的子树也就变化了。

思路：

我们来分类讨论，枚举一下情况。我们需要求节点 x 在当前树根下的子树中的最小值。

如果当前树根就是 x ，那很明显答案就是整棵树的节点最小值。

如果当前树根在 x 的子树中，比如 A_1 ，那么我们可以发现答案就是去除 A_1 在 x 中的这一段子树部分，剩余部分中找最小值即可。

如果当前树根在 x 子树之外，比如 A_2 ，那么答案就是在 x 原来对应的子树中找一个最小值即可。

所以这题就变成了对于 x 与当前树根求一个 LCA 的问题，然后只有在第二种情况中才需要进行特殊处理，即在 x 的子树找到对应 A_1 的那个儿子即可。当然在树剖中，利用 top 数组就可以完成 LCA 能够完成的功能，见如下代码。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
8 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
9 ;
10 typedef long long ll;
11 typedef double db;
12 #define int long long
13 const db EPS = 1e-9;
14 const int N = 1e5+100;
15 using namespace std;
16
17 struct Edge { int next,to;} e[2*N];
18 struct Node { int l,r,ls,rs,lazy,minn;} t[2*N];
19 int n,m,root,rt,mod,val[N],head[N],tot,fa[N],d[N],son[N],size[N],top[N],id[N],rk[N];
20 //top[x]: x节点所在链的顶端节点, id[x]: 节点dfs序, rk[x]: dfs序对应的节点
21 //val[x]: 每个点初始权值, fa[x]: 每个点父节点, d[x]: 节点深度, size[x]: 节点子树大小
22 //rt: 线段树根节点编号
23 void init() { memset(head,0,sizeof head); tot = 1, size[0] = 0; }
24 void add(int x, int y) { e[++tot].next = head[x], e[tot].to = y, head[x] = tot; }
25 void dfs1(int x){ //求出每个点的子树大小、深度、重儿子
26     size[x] = 1, d[x] = d[fa[x]]+1, son[x] = 0;
27     for(int v,i = head[x]; i; i = e[i].next)
28         if((v = e[i].to)!=fa[x]){
29             fa[v] = x, dfs1(v), size[x] += size[v];
30             if(size[son[x]] < size[v])
31                 son[x] = v;
32         }
33 }
34 void dfs2(int x, int tp){ //求出每个节点的dfs序, dfs序对应的节点, 以及每个点所在链的顶端节点
35     top[x] = tp, id[x] = ++tot, rk[tot] = x;
36     if(son[x]) dfs2(son[x],tp);
37     for(int v,i = head[x]; i; i = e[i].next)
38         if((v = e[i].to)!=fa[x] && v!=son[x]) dfs2(v,v);
39 }
40 inline void pushup(int x) { t[x].minn = min(t[t[x].ls].minn,t[t[x].rs].minn); }
41 void build(int l, int r, int x){ //基础建树, 动态开点
42     t[x].l = l, t[x].r = r, t[x].lazy = 0;
43     if(l == r){
44         t[x].minn = val[rk[l]]; return;
45     }
46     int mid = (l+r)>>1;
47     t[x].ls = ++tot, t[x].rs = ++tot;
48     build(l,mid,t[x].ls), build(mid+1,r,t[x].rs), pushup(x);
49 }
50 inline int len(int x) { return t[x].r-t[x].l+1; }
51 inline void pushdown(int x){ //基础的线段树标记下放
52     if(t[x].lazy && t[x].l != t[x].r){
53         int ls = t[x].ls, rs = t[x].rs, lz = t[x].lazy;
54 }
```

```

53         (t[ls].lazy=lz), (t[rs].lazy=lz);
54         t[ls].minn = lz, t[rs].minn = lz;
55         t[x].lazy = 0;
56     }
57 }
58 void update(int l, int r, int x, int c){ //基础的线段树更新
59     if(t[x].l >= l && t[x].r <= r){
60         t[x].lazy = c, t[x].minn = c;
61         return;
62     }
63     pushdown(x);
64     int mid = (t[x].l+t[x].r)>>1;
65     if(mid >= l) update(l,r,t[x].ls,c);
66     if(mid < r) update(l,r,t[x].rs,c);
67     pushup(x);
68 }
69 int query(int l, int r, int x){ //基础的线段树查询
70     if(t[x].l >= l && t[x].r <= r) return t[x].minn;
71     pushdown(x);
72     int mid = (t[x].l+t[x].r)>>1, tp = 1e15;
73     if(mid >= l) tp = min(tp,query(l,r,t[x].ls));
74     if(mid < r) tp = min(tp,query(l,r,t[x].rs));
75     return tp;
76 }
77 inline void updates(int x, int y, int c){ //区间加z, 将区间分为多条链
78     while(top[x] != top[y]){
79         if(d[top[x]] < d[top[y]]) swap(x,y);
80         update(id[top[x]],id[x],rt,c); //对于每条链直接修改
81         x = fa[top[x]];
82     }
83     if(id[x] > id[y]) swap(x,y);
84     update(id[x],id[y],rt,c);
85 }
86 int find(int x, int y){ // top数组求LCA
87     int base = x;
88     while(top[x] != top[y]){
89         if(fa[top[y]] == base) return top[y]; //返回子树中存在当前树根的子儿子
90         if(d[top[x]] < d[top[y]]) y = fa[top[y]];
91         else x = fa[top[x]];
92     }
93     if(d[x] > d[y]) x = y;
94     if(x == base) return son[x]; //返回子树中存在当前树根的子儿子
95     else return 0; //当前树根在x子树之外
96 }
97 signed main()
98 {
99     scanf("%lld%lld",&n,&m);
100    init();
101    rep(i,1,n-1){
102        int x,y; scanf("%lld%lld",&x,&y);
103        add(x,y), add(y,x);
104    }
105    rep(i,1,n) scanf("%lld",&val[i]);
106    scanf("%lld",&root);
107    tot = 0, dfs1(root), dfs2(root, root);
108    tot = 0, build(1,n,rt = ++tot);
109    rep(i,1,m){
110        int op,x,y,z;
111        scanf("%lld",&op);

```

```

112     if(op == 1){
113         scanf("%lld",&x);
114         root = x;
115     }
116     else if(op == 2){
117         scanf("%lld%lld%lld",&x,&y,&z);
118         updates(x,y,z);
119     }
120     else{
121         scanf("%lld",&x); int thp;
122         if(root == x){
123             printf("%lld\n",query(1,n,rt));
124         }
125         else if((thp = find(x,root)) != -1){
126             printf("%lld\n",min(query(1,id[thp]-1,rt),query(id[thp]+size[thp],n,rt)
127 )));
128             }
129         }
130     }
131     return 0;
132 }
```

4.16.5 边剖 + 主席树

题意：

给定一棵 n 个节点的树，每条边都有一个权值， m 次查询，每次询问树上两点路径上边权小于 k 的边有多少条？
 $(1 \leq n, m \leq 10^5)$

思路：

比较裸的题目，可以离线操作，然后将询问值从小到大进行排序，然后每次单点修改，将比当前询问小的边加入树中。

此处需要注意是边权树剖，因此将每条边的权值压到深度更深的节点上，然后树剖路径查询时，最后两点处于同一条链时，将深度最浅的点改为该点的儿子即可。

比赛时没有考虑到将询问值排序，然后依次修改边权，因此采用了树上主席树的做法。先将所有的边都插入树中，然后直接查询区间中小于 k 的节点有多少个即可。

```

1 #include <csdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
8 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
9 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
10    << " , " << z1 << ":" << z2 << endl;
11 typedef long long ll;
12 typedef double db;
13 const db EPS = 1e-9;
14 const int N = 1e5+100;
15 using namespace std;
16 struct Edge { int next,to,w;} e[2*N];
17 struct Node { int l,r,ls,rs,sum;} t[20*N];
```

```

18 int n,m,rt,mod,val[N],head[N],tot,fa[N],d[N],son[N],size[N],top[N],id[N],rk[N],root[N],
19 //top[x]: x节点所在链的顶端节点, id[x]: 节点dfs序, rk[x]: dfs序对应的节点
20 //val[x]: 每个点初始权值, fa[x]: 每个点父节点, d[x]: 节点深度, size[x]: 节点子树大小
21 //rt: 线段树根节点编号
22 void init(){
23     memset(head,0,sizeof head);
24     tot = 1, size[0] = 0;
25 }
26
27 void add(int x, int y, int z){
28     e[++tot].next = head[x], e[tot].to = y, head[x] = tot, e[tot].w = z;
29 }
30
31 void dfs1(int x){ //求出每个点的子树大小、深度、重儿子
32     size[x] = 1, d[x] = d[fa[x]]+1, son[x] = 0;
33     for(int v,i = head[x]; i; i = e[i].next)
34         if((v = e[i].to)!=fa[x]){
35             fa[v] = x, dfs1(v), size[x] += size[v];
36             if(size[son[x]] < size[v])
37                 son[x] = v;
38         }
39 }
40
41 void dfs2(int x, int tp){ //求出每个节点的dfs序, dfs序对应的节点, 以及每个点所在链的顶端节点
42     top[x] = tp, id[x] = ++tot, rk[tot] = x;
43     if(son[x]) dfs2(son[x],tp);
44     for(int v,i = head[x]; i; i = e[i].next)
45         if((v = e[i].to)!=fa[x] && v!=son[x]) dfs2(v,v);
46 }
47
48 inline void pushup(int x){ //基础的线段树向上区间合并
49     t[x].sum = t[t[x].ls].sum+t[t[x].rs].sum;
50 }
51
52 int build(int l,int r) //主席树建树部分
53 {
54     int p = ++tot; // 新建一个节点, 编号为p, 代表当前区间[l,r]
55     t[p].l = l, t[p].r = r, t[p].sum = 0;
56     if(l == r) return p;
57     int mid = (l+r)>>1;
58     t[p].ls = build(l,mid);
59     t[p].rs = build(mid+1,r);
60     return p;
61 }
62
63 inline int len(int x) { return t[x].r-t[x].l+1; }
64
65 int ask(int lp,int rp,int k){ //主席树查询[lp,rp]比k小的有多少个
66     if(t[lp].l == t[lp].r) return (t[rp].sum-t[lp].sum);
67     int mid = (t[lp].l+t[lp].r)>>1;
68     int tp = 0;
69     if(mid < k){
70         tp += (t[t[rp].ls].sum-t[t[lp].ls].sum);
71         tp += ask(t[lp].rs,t[rp].rs,k);
72     }
73     else tp += ask(t[lp].ls,t[rp].ls,k);
74     return tp;
75 }

```

```

76
77 inline int solve(int x,int y,int k){
78     int ret = 0;
79     while(top[x] != top[y]){
        //让x与y到达同一条链
80         if(d[top[x]] < d[top[y]]) swap(x,y); //找到更深的点
81         int xx = id[top[x]]-1, yy = id[x];
82         int hp = ask(root[xx],root[yy],k);
83         ret += hp;
84         x = fa[top[x]];
85     }
86     if(x == y) return ret;
87     if(id[x] > id[y]) swap(x,y);
88     int xx = id[son[x]]-1, yy = id[y]; //边权树链剖分，最后两点同链之后，取最高点的儿子来计算
89     // -1是因为主席树要获取-1时候的副本进行sum减运算
90     int hp = ask(root[xx],root[yy],k);
91     return (ret+hp);
92 }
93
94 int insert(int now,int pos,int k) //主席树插入一个新的值
95 {
96     int p = ++tot;
97     t[p] = t[now]; //建立副本
98     if(t[p].l == t[p].r){
99         t[p].sum += k; //在副本上修改
100        return p;
101    }
102    int mid = (t[p].l+t[p].r)>>1;
103    if(pos <= mid) t[p].ls = insert(t[p].ls,pos,k); //保留右儿子部分，把左儿子更新
104    else t[p].rs = insert(t[p].rs,pos,k);
105    t[p].sum = t[t[p].ls].sum + t[t[p].rs].sum;
106    return p;
107 }
108
109 int ed[2*N][3];
110 int b[2*N],tb;
111
112 signed main()
113 {
114     scanf("%d%d",&n,&m);
115     init();
116     rep(i,1,n-1){
117         int x,y,z; scanf("%d %d %d",&x,&y,&z);
118         add(x,y,z), add(y,x,z);
119         ed[i][0] = x, ed[i][1] = y, ed[i][2] = z;
120     }
121     rot = 1;
122     tot = 0, dfs1(rot), dfs2(rot, rot);
123     rep(i,1,n-1){
124         if(d[ed[i][0]] > d[ed[i][1]]) swap(ed[i][0],ed[i][1]);
125         val[ed[i][1]] = ed[i][2];
126         b[++tb] = ed[i][2];
127     }
128     sort(b+1,b+1+tb);
129     tb = unique(b+1,b+1+tb)-b-1;
130     tot = 0;
131     root[0] = build(1,tb);
132     rep(i,1,n){
133         int x = lower_bound(b+1,b+1+tb,val[rk[i]])-b;
134         root[i] = insert(root[i-1],x,1);

```

```

135     }
136     rep(i,1,m){
137         int u,v,k; scanf("%d%d%d",&u,&v,&k);
138         int pos = upper_bound(b+1,b+1+tb,k)-b;
139         pos--;
140         if(pos == 0) printf("0\n");
141         else printf("%d\n",solve(u,v,pos));
142     }
143     return 0;
144 }
```

4.17 CDQ

4.17.1 陌上开花 (三维偏序)

描述:

有 n 朵花, 每朵花有三个属性: 花形 (s)、颜色 (c)、气味 (m), 用三个整数表示。

现在要对每朵花评级, 一朵花的级别是它拥有的美丽能超过的花的数量。

定义一朵花 A 比另一朵花 B 要美丽, 当且仅 $S_a >= S_b, C_a >= C_b, M_a >= M_b$ 。

显然, 两朵花可能有同样的属性。需要统计出评出每个等级的花的数量。

输入:

第一行为 N, K ($1 \leq N \leq 100,000, 1 \leq K \leq 200,000$), 分别表示花的数量和最大属性值。

以下 N 行, 每行三个整数 s_i, c_i, m_i ($1 \leq s_i, c_i, m_i \leq K$), 表示第 i 朵花的属性

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
8 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
9 ;
10 typedef long long ll;
11 typedef double db;
12 const int N = 2*1e5+100;
13 const int M = 1e5+100;
14 const db EPS = 1e-9;
15 using namespace std;
16
17 int n,k,ans[N],tot,t[N];
18 struct Node{
19     int a,b,c,s,ans;
20     bool operator == (Node xx) const {
21         return ((a == xx.a) && (b == xx.b) && (c == xx.c));
22     }
23     bool operator < (Node xx) const {
24         if(b != xx.b) return b < xx.b;
25         else return c < xx.c;
26     }
27 }p[N];
28
29 bool cmp(Node x,Node y){
30     if(x.a != y.a) return x.a < y.a;
31     else if(x.b != y.b) return x.b < y.b;
32     else return x.c < y.c;
33 }
```

```

33
34 inline int lowbit(int x) { return x&(-x); }
35
36 inline void update(int x, int c){
37     for(int i = x; i <= k; i += lowbit(i)) t[i] += c;
38 }
39
40 inline int ask(int x){
41     int tp = 0;
42     for(int i = x; i; i -= lowbit(i)) tp += t[i];
43     return tp;
44 }
45
46 void CDQ(int l, int r)
47 {
48     if(l == r) return;
49     int mid = (l+r)>>1;
50     CDQ(l,mid); CDQ(mid+1,r);
51     sort(p+l,p+mid+1); sort(p+mid+1,p+r+1); //将第二维排序
52     int i = l, j = mid+1;
53     while(j <= r){
54         while(i <= mid && p[i].b <= p[j].b){
55             update(p[i].c,p[i].s);
56             i++;
57         }
58         p[j].ans += ask(p[j].c);
59         j++;
60     }
61     rep(kk,l,i-1) update(p[kk].c,-p[kk].s); //消除之前的影响，此处是 i，不是 mid
62 }
63
64 int main()
65 {
66     scanf("%d%d",&n,&k);
67     rep(i,1,n) scanf("%d%d%d",&p[i].a,&p[i].b,&p[i].c);
68     //离散化
69     sort(p+1,p+1+n,cmp);
70     int cnt = 0;
71     rep(i,1,n){
72         cnt++;
73         if(p[i]==p[i+1]) continue;
74         p[++tot] = p[i], p[tot].s = cnt, cnt = 0, p[tot].ans = 0;
75     }
76     CDQ(1,tot);
77     rep(i,1,tot)
78         ans[p[i].ans+p[i].s-1] += p[i].s;
79     rep(i,0,n-1) printf("%d\n",ans[i]);
80     return 0;
81 }

```

4.17.2 动态逆序对

描述：

对于序列 A，它的逆序对数定义为满足 $i < j$ ，且 $A_i > A_j$ 的数对 (i,j) 的个数。给 1 到 n 的一个排列，按照某种顺序依次删除 m 个元素，你的任务是在每次删除一个元素之前统计整个序列的逆序对数。

输入：

输入第一行包含两个整数 n 和 m，即初始元素的个数和删除的元素个数。

以下 n 行每行包含一个 1 到 n 之间的正整数，即初始排列。

以下 m 行每行一个正整数，依次为每次删除的元素。

N<=100000 M<=50000

输出：

输出包含 m 行，依次为删除每个元素之前，逆序对的个数。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
8 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
9 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
10 << " , " << z1 << ":" << z2 << endl;
11 typedef long long ll;
12 typedef double db;
13 const int N = 1e5+100;
14 const int M = 5*1e4+100;
15 const db EPS = 1e-9;
16 using namespace std;
17
18 int n,m,tot,a[N],vis[N],b[N],mp[N];
19 ll t[N],ans[N];
20 struct Node{
21     int x,y,f; //插在x处, val = y, 贡献对象为f
22     bool operator < (Node xx) const {
23         return x < xx.x;
24     }
25 }p[N];
26 inline int lowbit(int x) { return x&(-x); }
27 inline void update(int x, int cc) { while(x <= n) t[x] += cc, x += lowbit(x); }
28 inline ll ask(int x) { ll tp = 0; while(x) tp += t[x], x -= lowbit(x); return tp; }
29
30 bool cmp(Node xx, Node yy){
31     return xx.x > yy.x;
32 }
33
34 void solve(int l, int r){
35     if(l == r) return;
36     int mid = (l+r)>>1;
37     solve(l,mid); solve(mid+1,r);
38     sort(p+l,p+mid+1); sort(p+mid+1,p+r+1);
39     int i = l, j = mid+1;
40     while(j <= r){
41         while(p[i].x <= p[j].x && i <= mid){
42             update(p[i].y,1); i++;
43         }
44         int tp = ask(n)-ask(p[j].y);
45         ans[p[j].f] += tp; j++;
46     }
47     rep(kk,l,i-1) update(p[kk].y,-1);
48
49     sort(p+l,p+mid+1,cmp); sort(p+mid+1,p+r+1,cmp);
50     i = l, j = mid+1;

```

```

51     while(j <= r){
52         while(p[i].x >= p[j].x && i <= mid){
53             update(p[i].y,1); i++;
54         }
55         int tp = ask(p[j].y);
56         ans[p[j].f] += tp; j++;
57     }
58     rep(kk,l,i-1) update(p[kk].y,-1);
59 }
60
61 int main()
62 {
63     scanf("%d%d",&n,&m);
64     rep(i,1,n){
65         scanf("%d",&a[i]);
66         mp[a[i]] = i;
67     }
68     rep(i,1,m){
69         scanf("%d",&b[i]);
70         vis[mp[b[i]]] = 1;
71     }
72     rep(i,1,n)
73         if(!vis[i]) p[++tot] = {i,a[i],m};
74     for(int i = m; i; i--) {
75         p[++tot] = {mp[b[i]],b[i],i};
76     }
77     solve(1,tot);
78     for(int i = m-1; i; i--) ans[i] += ans[i+1];
79     rep(i,1,m) printf("%lld\n",ans[i]);
80     return 0;
81 }

```

4.17.3 矩阵前缀和

描述:

维护一个 $W \times W$ 的矩阵，初始值均为 S。每次操作可以增加某格子的权值，或询问某子矩阵的总权值。修改操作数 $M \leq 160000$ ，询问数 $Q \leq 10000$ ， $W \leq 2000000$ 。

输入:

第一行两个整数, S, W ；其中 S 为矩阵初始值； W 为矩阵大小

接下来每行为一下三种输入之一（不包含引号）：

”1 x y a”，你需要把 (x,y) （第 x 行第 y 列）的格子权值增加 a

”2 x1 y1 x2 y2”，你需要求出以左下角为 $(x1,y1)$ ，右上角为 $(x2,y2)$ 的矩阵内所有格子的权值和，并输出

”3”，表示输入结束

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
8 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
;
```

```

9 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
    << " , " << z1 << ":" << z2 << endl;
10 typedef long long ll;
11 typedef double db;
12 const int N = 160000+4*1e4+100;
13 const int M = 1e4+100;
14 const db EPS = 1e-9;
15 using namespace std;
16
17 int T,n,ans[M],tot,num,t[2000010]; //tot-操作个数 num-查询个数
18 struct Node{
19     int x,y,op,val,f; //x坐标 y坐标 op:1-修改 op:2-查询 val:答案贡献(1/-1) f:对应答案编号
20     bool operator < (Node xx) const {
21         if(x != xx.x) return x < xx.x;
22         else return op < xx.op;
23     }
24 }p[N];
25
26 inline int lowbit(int x) { return x&(-x); }
27 inline void update(int x, int val) { while(x <= n) t[x] += val, x += lowbit(x); }
28 inline int ask(int x){
29     int tp = 0; while(x >= 1) tp += t[x], x -= lowbit(x);
30     return tp;
31 }
32
33 void solve(int l, int r){
34     if(l == r) return;
35     int mid = (l+r)>>1;
36     solve(l,mid); solve(mid+1,r);
37     sort(p+l,p+mid+1); sort(p+mid+1,p+r+1);
38     int i = l, j = mid+1;
39     while(j <= r){
40         if(p[j].op == 1){
41             j++; continue;
42         }
43         while(p[i].x <= p[j].x && i <= mid){
44             if(p[i].op == 1) update(p[i].y,p[i].val);
45             i++;
46         }
47         ans[p[j].f] += p[j].val*ask(p[j].y);
48         j++;
49     }
50     rep(kk,l,i-1){
51         if(p[kk].op == 1) update(p[kk].y,-p[kk].val);
52     }
53 }
54
55 int main()
56 {
57     scanf("%d%d",&T,&n);
58     int op,xx,yy,aa,zz;
59     while(scanf("%d",&op)){
60         if(op == 3) break;
61         if(op == 1){
62             scanf("%d%d%d",&xx,&yy,&aa);
63             p[++tot] = {xx,yy,1,aa,0};
64         }
65         else{
66             scanf("%d%d%d%d",&xx,&yy,&aa,&zz);

```

```

67     ++num;
68     p[++tot] = {aa,zz,2,1,num};
69     p[++tot] = {xx-1,yy-1,2,1,num};
70     p[++tot] = {xx-1,zz,2,-1,num};
71     p[++tot] = {aa,yy-1,2,-1,num};
72 }
73 }
74 solve(1,tot);
75 rep(i,1,num) printf("%d\n",ans[i]);
76 return 0;
77 }
78
79 /*
80 0 4
81 1 2 3 3
82 2 1 1 3 3
83 1 2 2 2
84 2 2 2 3 4
85 3
86 */

```

4.18 莫队

4.18.1 普通莫队

莫队概述:

莫队本质上就是用分块去优化暴力的离线算法，将总复杂度降到 $O(n\sqrt{n})$ 的位置。说白了，就是分块 + 暴力。

我们先讲‘暴力的部分’。比如一个长度为 n 的序列， m 次查询，每次查询询问区间 $[l, r]$ 之间的众数。对于这个问题，暴力求的话就是直接用桶记录每个数出现的次数，然后遍历区间 $[l, r]$ ，直接统计答案即可。这个暴力过程和莫队暴力过程没有任何区别，然后问题就变成了如何用分块来优化这个暴力呢？

在‘分块的部分’，该算法将整个序列按照 \sqrt{n} 大小进行分块，共分成 \sqrt{n} 块，然后对于所有的询问，先按照左端点所在的块编号进行排序，如果块编号相同，再按照右端点排序。询问排序完之后，就直接暴力求解即可。代码的话看一下下面习题就可以掌握了。

最后就是‘时间复杂度’的问题了。如何证明这个算法的时间复杂度呢？我们对每一个块分开进行考虑，假设有 b_i 次操作在第 i 个块中，则在这个块中，右端点一定递增，因此右端点最多移动 n 次，而左端点每次最多移动 \sqrt{n} ，一共最多移动 $b_i * \sqrt{n}$ 次，每次端点移动的时间复杂度为 $O(1)$ ，因此移动的总次数为 $\sum_{i=1}^{\sqrt{n}} (b_i * \sqrt{n} + n) = m * \sqrt{n} + n * \sqrt{n}$ ，因此总复杂度为 $O(n\sqrt{n} + m\sqrt{n})$ 。

题意:

n 双颜色不同袜子， m 次询问，每次询问给出 $[L, R]$ 区间，询问在 $[L, R]$ 区间中随机抽出两双颜色相同的袜子的概率，输出最简分数形式 (A/B) 。 $(1 \leq n, m \leq 50000)$

思路:

普通莫队算法的复杂度是 $O(N\sqrt{N})$ ，实现关键点就在于能否在区间左右端点移动时， $O(1)$ 的更新答案。

我们观察这道题目，可以发现区间 $[L, R]$ 取出两双颜色相同袜子的概率 = $\frac{\frac{1}{2} * \sum_{i=L}^R num[i]}{C(R-L+1, 2)}$ ， $num[i]$ 表示在区间 $[L, R]$ 中有多少双与 i 颜色相同的袜子，乘以 $\frac{1}{2}$ 的原因在于每一对颜色相同的袜子被计算了两遍。

分析到这里，就可以发现这是一道普通莫队的裸题，我们添加与删除时只需加上或减去当前与该点颜色相同的袜子数，这样同时可以避免重复计算。

```
1 #include <bits/stdc++.h>
```

```

2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 typedef long long ll;
4 const int N = 2*1e5+100;
5 using namespace std;
6
7 int a[N],pos[N],n,m,L,R;
8 ll ans[N][2],flag[N],Ans;
9 struct Node{
10     int l,r,id;
11     bool operator < (Node xx) const{
12         if(pos[l] == pos[xx.l]) return r < xx.r;
13         else return pos[l] < pos[xx.l];
14     }
15 }Q[N];
16
17 ll gcd(ll a,ll b) {return b == 0 ? a:gcd(b,a%b);}
18
19 void add(int x){
20     Ans += flag[a[x]];
21     flag[a[x]]++;
22 }
23
24 void del(int x){
25     flag[a[x]]--;
26     Ans -= flag[a[x]];
27 }
28
29 int main()
30 {
31     L = 1, R = 0;
32     scanf("%d%d",&n,&m);
33     int sz = sqrt(n);
34     rep(i,1,n){
35         scanf("%d",&a[i]);
36         pos[i] = i/sz;
37     }
38     rep(i,1,m){
39         scanf("%d%d",&Q[i].l,&Q[i].r);
40         Q[i].id = i;
41     }
42     sort(Q+1,Q+1+m);
43     rep(i,1,m){
44         while(L < Q[i].l) del(L),L++;
45
46         while(L > Q[i].l) L--, add(L);
47
48         while(R < Q[i].r) R++, add(R);
49
50         while(R > Q[i].r) del(R), R--;
51
52         ll len = Q[i].r-Q[i].l+1;
53         ll tp = len*(len-1ll)/(ll)2;
54         ll g = gcd(Ans,tp);
55         ans[Q[i].id][0] = Ans/g;
56         ans[Q[i].id][1] = tp/g;
57     }
58     rep(i,1,m) printf("%lld/%lld\n",ans[i][0],ans[i][1]);
59     return 0;
60 }

```

4.18.2 莫队求组合数前缀和

题意:

q 组询问，每次给出一个 n 和 m ，求 $\sum_{i=0}^{i=m} C_n^i$ 。 $(1 \leq n, m, q \leq 2 * 10^5)$

思路:

令 $F(n, m) = \sum_{i=1}^m C_n^i$ ，思考 $F(n, m)$ 与 $F(n, m+1)$ 和 $F(n+1, m)$ 之间的关系。

$F(n, m+1) = F(n, m) + C_n^{m+1}$, $F(n+1, m) = 2 * F(n, m) - C_n^m$ 。预处理出阶乘和逆元之后，即可 $O(1)$ 进行端点移动。

总结:

这其实是一道广义莫队问题，所谓广义莫队问题就是题目中并没有明确指明查询区间 $[l, r]$ 的答案，而是将所查询的问题转化为 $F(n, m)$ 的形式，然后实现 $F(n, m)$ 与 $F(n+1, m)$ 以及 $F(n, m+1)$ 之间的 $O(1)$ 转移，只要求出之间转移的公式就可以直接 $O(n * \sqrt{n})$ 离线求出最终答案。

```
1 return 0;
```

4.18.3 带修改莫队

带修改莫队概述:

带修改的莫队仍然是利用分块对查询和修改排序，尽可能地减少运行时间。

假设我们按照 k 大小进行分块，则一共有 $\frac{n}{k}$ 个块，然后对于每个操作，一共有三个参数，分别是 l 、 r 、 id ，表示区间左右端点和操作时间，我们先按照左端点的块号进行排序，再按照右端点的块号进行排序，最后按照操作时间进行排序。

莫队暴力时也需要维护三个值， L 、 R 、 T 表示当前控制的左右区间以及操作时间。‘对于每个查询，需要将 L 、 R 、 T 移动到指定位置再进行计算，因此可以将带修改莫队理解为三维莫队。

接下来估算复杂度，假设 m 次查询中，一共有 a 次查询， b 次修改。因此当确定左右端点块号时，即查询即按照时间排序时， T 最多移动 b 次，因此 T 的移动一共有 $\frac{n}{k} * \frac{n}{k} * b$ 次。而每次查询，区间左右端点最多移动 $2 * k$ 次，因此 l 、 r 最多移动 $a * 2 * k$ 次，因此总时间复杂度为 $O(b * \frac{n^2}{k^2} + 2 * a * k)$ 。我们可以求导求这个函数的最小值，可以发现最后的答案会在 $k = n^{\frac{2}{3}}$ 处取到最优解，因此整个算法的复杂度也就达到了 $O(n^{\frac{5}{3}})$ 处。

题意:

长度为 n 的初始序列，共有 m 次操作，操作 1 给出一个 l, r ，令 c_i 为 i 在 $[l, r]$ 中出现的次数，询问 $Mex(c_0, c_1, \dots, c_{10^9})$ 。操作 2 则将 a_p 改成 x 。 $(1 \leq n, m \leq 10^5)$

思路:

这个问题唯一的操作难点在于 mex 函数的求取，其实我们可以像求取 SG 函数的 mex 一样，直接暴力求取即可。然后其余部分就是常规的带修改莫队的操作了。

```
1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <cmath>
5 #include <algorithm>
6 #define __ ios::sync_with_stdio(0); cin.tie(0); cout.tie(0)
7 #define rep(i,a,b) for(int i = a; i <= b; i++)
8 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
9 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << ", " << y1 << ":" << y2 << endl
10 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << ", " << y1 << ":" << y2
    << ", " << z1 << ":" << z2 << endl;
11 typedef long long ll;
```

```

12  typedef double db;
13  const int N = 2*1e5+100;
14  const int M = 1e5+100;
15  const db EPS = 1e-9;
16  using namespace std;
17
18  int n,qq,a[N],b[N],tot,Qnum,Cnum,pos[N],ans[N],L,R,T,flag[N],vis[N];
19  struct Query{
20      int l,r,id,t;
21      bool operator < (Query xx) const {
22          if(pos[l] != pos[xx.l]) return pos[l] < pos[xx.l];
23          else if(pos[r] != pos[xx.r]) return pos[r] < pos[xx.r];
24          else return t < xx.t;
25      }
26  }q[M];
27  struct Change{
28      int pos,val;
29  }C[M];
30
31  int find(int x){
32      return lower_bound(b+1,b+1+tot,x)-b;
33  }
34
35  void add(int x){
36      if(flag[a[x]]!=0) vis[flag[a[x]]]--;
37      flag[a[x]]++; vis[flag[a[x]]]++;
38  }
39
40  void del(int x){
41      vis[flag[a[x]]]--; flag[a[x]]--;
42      if(flag[a[x]] != 0) vis[flag[a[x]]]++;
43  }
44
45  void Work(int x,int i){
46      if(C[x].pos >= q[i].l && C[x].pos <= q[i].r){
47          vis[flag[a[C[x].pos]]]--; flag[a[C[x].pos]]--;
48          if(flag[a[C[x].pos]] != 0) vis[flag[a[C[x].pos]]]++;
49          if(flag[C[x].val] != 0) vis[flag[C[x].val]]--;
50          flag[C[x].val]++; vis[flag[C[x].val]]++;
51      }
52      swap(a[C[x].pos],C[x].val);
53  }
54
55  int solve(){
56      rep(i,0,n)
57          if(!vis[i]) return i;
58  }
59
60  int main()
61  {
62      scanf("%d%d",&n,&qq);
63      rep(i,1,n){
64          scanf("%d",&a[i]);
65          b[++tot] = a[i];
66      }
67      rep(i,1,qq){
68          int op,l,r; scanf("%d%d%d",&op,&l,&r);
69          if(op == 1) Qnum++, q[Qnum] = {l,r,Qnum,Cnum};
70          else C[++Cnum] = {l,r}, b[++tot] = r;

```

```

71     }
72     sort(b+1,b+1+tot);
73     tot = unique(b+1,b+1+tot)-b-1;
74     int sz = pow(n,0.6666666666666666);
75     rep(i,1,n) pos[i] = i/sz;
76     sort(q+1,q+1+Qnum);
77     L = 1, R = 0, T = 0;
78     vis[0] = 1;
79     rep(i,1,n) a[i] = find(a[i]);
80     rep(i,1,Cnum) C[i].val = find(C[i].val);
81     rep(i,1,Qnum){
82         while(L < q[i].l) del(L++);
83         while(L > q[i].l) add(--L);
84         while(R < q[i].r) add(++R);
85         while(R > q[i].r) del(R--);
86         while(T < q[i].t) Work(++T,i);
87         while(T > q[i].t) Work(T--,i);
88         ans[q[i].id] = solve();
89     }
90     rep(i,1,Qnum) printf("%d\n",ans[i]);
91     return 0;
92 }
```

4.18.4 树上带修改莫队

树上带修莫队概述:

树上莫队问题仍然是通过分块进行解决，但是分块的序列发生了变化。这个序列需要满足，给出两点就能在序列上找出这两点之间的路径。

我们考虑常见的树上序列， dfs 序，但是很明显 dfs 序不满足这个条件，其中会有很多无效的节点。因此我们引出欧拉序来解决这个问题，欧拉序和 dfs 序的区别是， dfs 序只在遍历到这个节点时才会将这个节点加入序列，而欧拉序还会在回溯到这个节点时将节点加入序列。

因此在欧拉序中，每个点会有一个第一次到达的点和第二次到达的点，我们分别记为 $fir[i]$ 与 $las[i]$ 。对于树上两点 x, y ($fir[x] < fir[y]$)， u 为 x, y 两点的 lca ，若 $x = u$ ，则在 $[fir[x], fir[y]]$ 这段区间中，只有 x 到 y 路径上的点只出现一次。若 $x \neq u$ ，则在 $[las[x], fir[y]]$ 这段区间中只有 x 到 y 路径上的点只出现一次，而且不包含 $lca(x, y)$ 这个点。因此我们在树上莫队问题中，需要记录每个点出现的次数，第一次出现则加贡献，第二次出现则减贡献，且若 $x \neq u$ ，还需加上 lca 的贡献。

解决完树上莫队的序列问题，就可以转化成普通莫队进行计算了。不带修改则块大小为 $\sqrt{2n}$ ，带修改则块大小为 $(2n)^{\frac{2}{3}}$ ，其中 $2n$ 为欧拉序长度。

题意:

n 个点的一棵树，每个点上都有一个糖果，糖果的种类不同，第 i 类糖果的贡献为 $V[i]$ ，第 j 次吃第 i 类糖果对答案的贡献为 $V[i] * W[j]$ 。现有 q 次操作，每次可以将第 x 个点上的糖果类型改为 y ，也可以查询从 x 点到 y 点的答案。 $(1 \leq n, q \leq 10^5)$

思路:

莫队问题只需要关注加入节点和删除节点对答案的影响，因此只需要统计每一类糖果在路径中出现的次数即可完成节点增删时对答案的影响。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <cmath>
5 #include <algorithm>
6 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
```

```

7 #define rep(i,a,b) for(int i = a; i <= b; i++)
8 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
9 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
10 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
11 << " , " << z1 << ":" << z2 << endl;
12 typedef long long ll;
13 typedef double db;
14 const int N = 1e5+100;
15 const int M = 1e6+100;
16 const db EPS = 1e-9;
17 using namespace std;
18
19 int n,m,k,V[N],W[N],head[N],tot,C[N],qnum,cnum,f[N][25],t,d[N],Euler[2*N],ncnt,fir[N],
20 las[N],pos[2*N],L,R,T,flag[N],vis[N];
21 //pos-分块位置, fir-欧拉序第一次, las-欧拉序第二次, Euler-欧拉序数组、ncnt-欧拉序数组长度
22 //vis-这个树上节点出现了几次, flag-这个糖果种类
23 ll ans[N],now;
24 struct Edge{
25     int to,next;
26 }e[2*N];
27 struct Query{
28     int l,r,id,lca,t; //l, r, id-查询顺序, lca-两点lca, t-之前有几次修改
29     bool operator < (Query xx) const {
30         if(pos[l] != pos[xx.l]) return pos[l] < pos[xx.l];
31         else if(pos[r] != pos[xx.r]) return pos[r] < pos[xx.r];
32         else return t < xx.t;
33     }
34 }q[N];
35 struct Change{
36     int pos, val;
37 }ch[N];
38
39 void add(int x,int y){
40     e[++tot].to = y, e[tot].next = head[x], head[x] = tot;
41 }
42
43 void dfs(int u,int fa)
44 {
45     Euler[++ncnt] = u; fir[u] = ncnt;
46     d[u]=d[fa]+1; f[u][0]=fa;
47     for(int i=1;(1<<i)<=d[u];i++)
48         f[u][i]=f[f[u][i-1]][i-1];
49     for(int i=head[u]; i; i=e[i].next){
50         int v=e[i].to;
51         if(v!=fa) dfs(v,u);
52     }
53     Euler[++ncnt] = u; las[u] = ncnt;
54 }
55
56 int LCA(int x,int y)
57 {
58     if(d[x] > d[y]) swap(x,y);
59     for(int i = t; i >= 0; i--)
60         if(d[f[y][i]] >= d[x]) y = f[y][i]; //往上追溯, 直至y和x位于同一深度
61     if(x == y) return x; //如果已经找到了, 就返回x
62     for(int i = t; i >= 0; i--)
63         if(f[x][i] != f[y][i]) x = f[x][i], y = f[y][i]; //x和y同时往上走, 一直到x和y恰好为
64 lca的子节点

```

```

62     return f[x][0]; //x和y共同的根节点就是lca
63 }
64
65 void Add(int pos){
66     flag[C[pos]]++;
67     now += (ll)W[flag[C[pos]]]*(ll)V[C[pos]];
68 }
69
70 void Del(int pos){
71     now -= (ll)W[flag[C[pos]]]*(ll)V[C[pos]];
72     flag[C[pos]]--;
73 }
74
75 void add_del(int pos){ //增加和减少取决于这个点被遍历了几次
76     vis[pos] ? Del(pos) : Add(pos);
77     vis[pos] ^= 1;
78 }
79
80 void work(int x){
81     if(vis[ch[x].pos]){//修改点为有效点
82         add_del(ch[x].pos); //减掉
83         swap(C[ch[x].pos], ch[x].val);
84         add_del(ch[x].pos); //加上
85     }
86     else swap(C[ch[x].pos], ch[x].val);
87 }
88
89 int main()
90 {
91     scanf("%d%d%d", &n, &m, &k);
92     rep(i, 1, m) scanf("%d", &V[i]);
93     rep(i, 1, n) scanf("%d", &W[i]);
94     rep(i, 1, n-1){
95         int xx, yy; scanf("%d%d", &xx, &yy);
96         add(xx, yy); add(yy, xx);
97     }
98     rep(i, 1, n) scanf("%d", &C[i]);
99     t = (int)(log(n)/log(2))+1;
100    dfs(1, 0);
101    int sz = pow(ncnt, 2.0/3.0);
102    for(int i = 0; i <= ncnt; i++) pos[i] = i/sz;
103    rep(i, 1, k){
104        int op, x, y; scanf("%d%d%d", &op, &x, &y);
105        if(op){
106            int lca = LCA(x, y);
107            q[++qnum].t = cnum; q[qnum].id = qnum;
108            if(fir[x] > fir[y]) swap(x, y);
109            if(x == lca) q[qnum].l = fir[x], q[qnum].r = fir[y], q[qnum].lca = 0;
110            else q[qnum].l = las[x], q[qnum].r = fir[y], q[qnum].lca = lca;
111        }
112        else ch[++cnum] = {x, y};
113    }
114    sort(q+1, q+1+qnum);
115    L = 1, R = 0, T = 0;
116    rep(i, 1, qnum){
117        while(L < q[i].l){
118            add_del(Euler[L]); L++;
119        }
120        while(L > q[i].l){

```

```

121         L--; add_del(Euler[L]);
122     }
123     while(R < q[i].r){
124         R++; add_del(Euler[R]);
125     }
126     while(R > q[i].r){
127         add_del(Euler[R]); R--;
128     }
129     while(T < q[i].t){
130         ++T; work(T);
131     }
132     while(T > q[i].t){
133         work(T); --T;
134     }
135     if(q[i].lca) add_del(q[i].lca); //lca不在欧拉序列区间中
136     ans[q[i].id] = now;
137     if(q[i].lca) add_del(q[i].lca); //恢复这个区间的状态
138 }
139 rep(i,1,qnum) printf("%lld\n",ans[i]);
140 return 0;
141 }
```

4.18.5 回滚莫队（增加）

回滚莫队概述：

回滚莫队的关键点在于只能增加或删除节点，我们以求取 \max 为例。求取 \max 时，增加节点时可以顺便更新答案，但是删除节点时就非常不好维护，因此我们需要设计一个只需要增加节点的莫队算法。

1. 首先还是老套路，按照 \sqrt{n} 进行分块，并确定每一块的左右边界，分别为 $xl[i]$ $xr[i]$ 。
2. 然后我们按照左端点所在块编号为第一关键字，右端点大小为第二关键字，对所有查询进行排序。
3. 接下来对于所有左右端点在同一块中的查询，我们直接暴力求取答案，复杂度为 $O(\sqrt{n})$ 。
4. 对于左端点所在块相同的查询，其右端点不断递增，因此右端点最多移动 $O(n)$ ，总共 \sqrt{n} 个块，右端点复杂度为 $O(n\sqrt{n})$ 。
5. 接下来考虑左端点的移动，我们对于所有左端点所在块相同的查询，每一个查询结束之后都要把左端点移动到 $xr[i]+1$ 的位置，即左端点所在块的右端点 +1 的位置，这样可以保证每次查询都是不断增加节点的，因此不会影响最终答案。每个查询，左端点最多移动距离为 $O(\sqrt{n})$ ，因此左端点移动的复杂度为 $O(m\sqrt{n})$ 。所以综合左右端点的移动，该算法的复杂度为 $O(m\sqrt{n} + n\sqrt{n})$ ，考虑到 m 的范围通常与 n 一致，因此最终复杂度为 $O(n\sqrt{n})$ 。

上述过程就是回滚莫队的求取过程，习题中分别给出了增加节点和减少节点的回滚莫队算法，其它具体实现细节可以查看代码。

题意：

长度为 n 的序列，每个数的大小为 x_i 。一共 q 次查询，每次给出一个区间 l, r ，询问区间 $[l, r]$ 中每个数贡献的最大值，一个数的贡献为 $x_i * cnt[x_i]$ ，即数大小 * 该数出现次数。 $(1 \leq n, q, \leq 10^5, 1 \leq x_i \leq 10^9)$

思路：

首先把序列离散化，然后用一个桶记录每一个数字出现的次数。

接下来就是回滚莫队的基本操作了，求出每块的左右端点，然后对查询排序。每次查询时判断左右端点是否在同一个快内，如果在就暴力求，如果不在就增加节点扩充区间。每个查询结束后，要将左端点再移动到该块的右边界 +1 位置，具体的实现细节见代码。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 typedef long long ll;
4 const int N = 2e5+100;
5 using namespace std;
6
```

```

7 int n,m,sz,pos[N],a[N],b[N],tot,val[N],xl[N],xr[N],cnt[N],L,R,_cnt[N],lastblock;
8 //Maxn - 左右端点控制的最大值, temp - 临时最大值
9 //cnt - 左右端点移动时计数, _cnt - 左右端点同块时的计数
10 ll ans[N],Maxn,temp;
11 struct Node{
12     int l,r,id;
13     bool operator < (Node xx) const {
14         if(pos[l] == pos[xx.l]) return r < xx.r;
15         else return pos[l] < pos[xx.l];
16     }
17 }q[N];
18
19 void init(){
20     scanf("%d%d",&n,&m); sz = sqrt(n);
21     rep(i,1,n) {scanf("%d",&a[i]); b[++tot] = a[i];}
22     rep(i,1,m) {scanf("%d%d",&q[i].l,&q[i].r); q[i].id = i;}
23     sort(b+1,b+1+tot); tot = unique(b+1,b+1+tot)-b-1;
24     rep(i,1,n) val[i] = lower_bound(b+1,b+1+tot,a[i])-b;
25     rep(i,1,n){
26         pos[i] = i/sz;
27         xl[pos[i]] = (xl[pos[i]] == 0 || xl[pos[i]] > i) ? i : xl[pos[i]];
28         xr[pos[i]] = (xr[pos[i]] < i) ? i : xr[pos[i]];
29     }
30     sort(q+1,q+1+m);
31 }
32
33 inline ll add(int x){
34     return (++cnt[val[x]])*(ll)b[val[x]];
35 }
36
37 inline void del(int x) {cnt[val[x]]--;}
38
39 void solve(){
40     L = 1, R = 0, lastblock = -1;
41     rep(i,1,m){
42         if(pos[q[i].l] == pos[q[i].r]){
43             ll temp = 0;
44             rep(j,q[i].l,q[i].r) temp = max(temp,(++_cnt[val[j]])*(ll)b[val[j]]);
45             rep(j,q[i].l,q[i].r) _cnt[val[j]]--;
46             ans[q[i].id] = temp;
47         }
48         else{
49             if(lastblock != pos[q[i].l]){
50                 while(L < xr[pos[q[i].l]]+1) del(L), L++;
51                 while(R > L-1) del(R), R--;
52                 Maxn = 0; lastblock = pos[q[i].l];
53             }
54             //Maxn为右半部分的最大值, 不包含左端点所在块的情况
55             while(R < q[i].r) R++, Maxn = max(Maxn,add(R));
56             temp = Maxn;
57             //temp从Maxn继承而来, 表示整个区间的最大值
58             while(L > q[i].l) L--, temp = max(temp,add(L));
59             while(L < xr[pos[q[i].l]]+1) del(L), L++;
60             ans[q[i].id] = temp;
61         }
62     }
63 }
64
65 int main()

```

```

66 {
67     init();
68     solve();
69     rep(i,1,m) printf("%lld\n",ans[i]);
70     return 0;
71 }

```

4.18.6 回滚莫队（减少）

题意：

长度为 n 的序列，每个数的大小为 a_i 。一共 m 次查询，每次给出一个区间 l, r ，询问区间 $[l, r]$ 中数的 mex ，其中一个区间的 mex 指该区间内最小没有出现过的自然数。 $(1 \leq n, q \leq 2 * 10^5, 0 \leq a_i \leq 10^9)$

思路：

由于是求 mex ，而数字总数为 $2e5$ ，因此不需要对数字进行离散化。然后我们来分析这个问题的关键点，即增删节点的特性。

不难发现，对于这个问题来说，删除节点可以 $O(1)$ 的更新答案，但是增加节点后答案的变化难以确定，因此考虑采用删除节点形式的回滚莫队来解决这个问题。

删除节点的回滚莫队，就是区间长度不断缩小的情况。因此我们需要对每个查询的左端点所在块编号进行升序，对每个查询的右端点进行降序，这样可以保证右端点是不断递减的。

然后对于左右端点在同一个块中的情况，我们依然是暴力求取答案。而对于不在同一块中的情况，我们需要每次查询结束后都将左端点移动到查询左端点所在的块的左边界上，这样才能保证区间长度在不断缩小。

除了上述这些回滚莫队的共性点之外，我们还需要关注一些特性点。对于这个问题，我们需要在最开始将左右边界分别设置为 1 和 n ，这样的目的是保证区间长度是不断递减的。然后求取答案时，我们需要维护两部分答案，一部分是区间 $[l, r]$ 中完全包含左端点所在块的部分的答案，另一部分即为当前查询的结果。

保存第一部分答案的目的在于增加节点是不能 $O(1)$ 维护答案的，因此左端点递增之后答案就会变化而且不能恢复，所以如果不保存第一部分的答案是不能直接继承到下一个查询的，具体细节看代码就能够理解。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 const int N = 2e5+100;
4 using namespace std;
5
6 int n,m,a[N],sz,pos[N],xl[N],xr[N],cnt[N],ans[N],_cnt[N],lastblock,L,R;
7 struct Node{
8     int l,r,id;
9     bool operator < (Node xx) const {
10         if(pos[l] == pos[xx.l]) return r > xx.r;
11         else return pos[l] < pos[xx.l];
12     }
13 }q[N];
14
15 void init(){
16     scanf("%d%d",&n,&m);
17     rep(i,1,n) scanf("%d",&a[i]);
18     rep(i,1,m) scanf("%d%d",&q[i].l,&q[i].r), q[i].id = i;
19     rep(i,1,n)
20         if(a[i] > 2e5) a[i] = 2e5+1;
21     int sz = sqrt(n);
22     rep(i,1,n){
23         pos[i] = i/sz; //点i所在块
24         xl[pos[i]] = xl[pos[i]] == 0 ? i : xl[pos[i]]; //pos[i]块的左端点
25         xr[pos[i]] = xr[pos[i]] < i ? i : xr[pos[i]]; //pos[i]块的右端点

```

```

26     }
27     sort(q+1,q+1+m);
28 }
29
30 inline void add(int x){
31     cnt[a[x]]++;
32 }
33
34 inline void del(int x,int& hp){
35     cnt[a[x]]--;
36     if(cnt[a[x]] == 0 && a[x] < hp) hp = a[x];
37 }
38
39 void solve(){
40     L = 1, R = n, lastblock = -1;
41     rep(i,1,n) cnt[a[i]]++;
42     int minn = 0;
43     while(cnt[minn]) minn++;
44     int base_min = minn;
45     rep(i,1,m){
46         if(pos[q[i].l] == pos[q[i].r]){
47             rep(j,q[i].l,q[i].r) _cnt[a[j]]++;
48             int now = 0;
49             while(_cnt[now]) now++;
50             rep(j,q[i].l,q[i].r) _cnt[a[j]]--;
51             ans[q[i].id] = now;
52         }
53         else{
54             if(lastblock != pos[q[i].l]){
55                 //每一次进入新的块时，右端点都是直接到n的，因此区间只有左端点在递增，可以不断O(1)维护答案
56                 while(R < n) R++, add(R);
57                 while(L < xl[pos[q[i].l]]) del(L,base_min), L++;
58                 minn = base_min; lastblock = pos[q[i].l];
59             }
56             //minn为包含左端点整个块的答案，用于继承到后续查询
57             while(R > q[i].r) del(R,minn), R--;
58             //temp为查询的答案
59             int temp = minn;
60             while(L < q[i].l) del(L,temp), L++;
61             while(L > xl[pos[q[i].l]]) L--, add(L);
62             ans[q[i].id] = temp;
63         }
64     }
65 }
66
67 }
68 }
69 }
70
71 int main()
72 {
73     init();
74     solve();
75     rep(i,1,m) printf("%d\n",ans[i]);
76     return 0;
77 }

```

4.19 虚树

题意：给定一棵树，敌人在 1 号节点，每次询问给出 k 个节点，每条边都有边权，问删掉树中边使得 1 号节点不能到达 k 个节点中任意一个节点需要的最小代价

输入：第一行一个整数 n，代表岛屿数量。

接下来 n-1 行，每行三个整数 u,v,w，代表 u 号岛屿和 v 号岛屿由一条代价为 c 的桥梁直接相连，保证 $1 \leq u, v \leq n$ 且 $1 \leq c \leq 100000$ 。

第 n+1 行，一个整数 m，代表敌方机器能使用的次数。

接下来 m 行，每行一个整数 ki，代表第 i 次后，有 ki 个岛屿资源丰富，接下来 k 个整数 h1,h2,⋯hk，表示资源丰富岛屿的编号。每次输出最小代价

思路：

首先考虑最暴力的 dp，设 $f[x]$ 表示处理完以 x 为根的子树的最小花费

转移有两种情况

1. 断开自己与父亲的联系，代价为从根到该节点的最小值
2. 将子树内的节点全都处理掉的代价

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <queue>
5 #include <cmath>
6 #include <algorithm>
7 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
8 #define rep(i,a,b) for(int i = a; i <= b; i++)
9 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
10 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
11 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
12 //idq —— 存取有效节点， top —— stk栈顶， vis —— 表示这个节点是否是有效节点
13 ll me[N];
14 struct Node{
15     int to,next; ll w;
16 }e1[M],e2[M];
17
18 void add1(int x,int y,ll cc){
19     e1[++tot1].to = y, e1[tot1].next = head1[x], e1[tot1].w = cc, head1[x] = tot1;
20 }
21 void add2(int x,int y,ll cc){
22     e2[++tot2].to = y, e2[tot2].next = head2[x], e2[tot2].w = cc, head2[x] = tot2;
23 }
24 bool cmp(int a,int b){return dfn[a] < dfn[b];}
25
26 void bfs() //LCA部分
27 {
28     queue<int> q;
29     while(q.size()) q.pop();
30     q.push(1); d[1] = 1; //把1当做树根
31     while(q.size())
32     {
33         int x = q.front(); q.pop();
34         for(int i = head1[x]; i != -1; i = e1[i].next)
35         {
36             int y = e1[i].to;
37             if(d[y] == 0)
38             {
39                 d[y] = d[x] + 1;
40                 q.push(y);
41             }
42         }
43     }
44 }
```

```

42         for(int i = head1[x]; i ;i = e1[i].next){
43             int y = e1[i].to;
44             if(d[y]) continue;
45             d[y] = d[x]+1;
46             f[y][0] = x; //y走2^0步到达x
47             for(int j = 1; j <= t;j++)
48                 f[y][j] = f[f[y][j-1]][j-1];
49             q.push(y);
50         }
51     }
52 }
53
54 int LCA(int x,int y) //LCA部分
55 {
56     if(d[x] > d[y]) swap(x,y);
57     for(int i = t; i >= 0; i--)
58         if(d[f[y][i]] >= d[x]) y = f[y][i]; //往上追溯，直至y和x位于同一深度
59     if(x == y) return x; //如果已经找到了，就返回x
60     for(int i = t; i >= 0; i--)
61         if(f[x][i] != f[y][i]) x = f[x][i], y = f[y][i]; //x和y同时往上走，一直到x和y恰好为
62         lca的子节点
63     return f[x][0]; //x和y共同的根节点就是lca
64 }
65 void dfs(int x,int fa){
66     dfn[x] = ++idx;
67     for(int i = head1[x]; i; i = e1[i].next){
68         int y = e1[i].to;
69         if(y == fa) continue;
70         if(x == 1) me[y] = e1[i].w;
71         else me[y] = min(me[x],e1[i].w);
72         dfs(y,x);
73     }
74 }
75
76 void insert(int u){ //构成虚树
77     if(top == 1) {stk[++top] = u; return;}
78     int lca = LCA(u,stk[top]); //与栈顶比较
79     if(lca == stk[top]) {stk[++top] = u; return;}
80     while(top > 1 && dfn[lca] <= dfn[stk[top-1]]){
81         add2(stk[top-1],stk[top],0); --top;
82     }
83     if(lca != stk[top]){
84         add2(lca,stk[top],0); stk[top] = lca;
85     }
86     stk[++top] = u;
87 }
88
89 ll DP(int x){ //使当前节点与子树中非有效节点不连通的最小代价
90     ll cost = 0;
91     for(int i = head2[x]; i; i = e2[i].next){
92         int y = e2[i].to;
93         cost += min(me[y],DP(y));
94     }
95     head2[x] = 0;
96     if(vis[x]){
97         vis[x] = 0;
98         return me[x];
99     }

```

```

100     else return cost;
101 }
102
103 int main()
104 {
105     scanf("%d",&n);
106     t = (int)(log(n)/log(2))+1;
107     rep(i,1,n-1){
108         int xx,yy; ll cc;
109         scanf("%d%d%lld",&xx,&yy,&cc);
110         add1(xx,yy,cc); add1(yy,xx,cc);
111     }
112     bfs(); dfs(1,-1);
113     scanf("%d",&m);
114     rep(i,1,m){
115         tot2 = 1;
116         int sz; scanf("%d",&sz);
117         rep(j,1,sz){
118             scanf("%d",&idq[j]); vis[idq[j]] = 1;
119         }
120         sort(1+idq,1+idq+sz,cmp);
121         top = 0; stk[++top] = 1; //加入树根
122         rep(j,1,sz) insert(idq[j]); //构建虚树
123         while(top > 1){
124             add2(stk[top-1],stk[top],0); top--;
125         }
126         // LOG1("i",i);
127         cout << DP(1) << endl;
128     }
129     return 0;
130 }
```

4.20 十字链表

题意：给出一个 $n * m$ 的矩阵， q 次操作，每次操作在矩阵中指定两个不重叠且不接触的小矩阵，将两个矩阵中的对应元素互换。在 q 次操作之后，输出最后的结果矩阵。 $(2 \leq n, m \leq 10^3, 1 \leq q \leq 10^4)$

思路：此题是子矩阵交换，我们可以先考虑一维状态下的子序列交换，区间 $[a, b]$ 与 $[c, d]$ 交换，如下图所示。每个节点存储右指针，只需将节点 $[a-1]$ 与 $[c-1]$ 以及 $[b]$ 与 $[d]$ 交换右指针即可。此处需注意，如果两个区间重叠或相互接触，则会发生错误，可以自行手动模拟一下。

知道了一维情况下的操作，我们可以考虑二维情况下如何操作。首先每个节点需要维护向右与向下的指针，然后我们来考虑下图情况，两个子矩形相互交换元素。我们可以发现决定这个矩形具体所在位置只是 A1、A3 区域的向下指针与 A2、A4 区域的向右指针。

因此我们只需将 A1、A3 与 B1、B3 的向下指针进行交换，A2、A4 与 B2、B4 的向右指针进行交换即可完成题目要求。注意如果两个矩形有可能相交或接触，则此算法不可行。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
8 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
;
```

```

9 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
    << " , " << z1 << ":" << z2 << endl;
10 typedef long long ll;
11 typedef double db;
12 const int N = 2*1e6+100;
13 const int M = 1e5+100;
14 const db EPS = 1e-9;
15 using namespace std;
16
17 int n,m,q;
18 struct Node{
19     int r,d,v;
20 }t[N];
21
22 int Pos(int x,int y){
23     x++, y++;
24     return (x-1)*(m+1)+y;
25 }
26
27 int main()
28 {
29     scanf("%d%d%d",&n,&m,&q);
30     rep(i,1,n)
31         rep(j,1,m) scanf("%d",&t[Pos(i,j)].v);
32     //一共n+1行, m+1列, 从(0,0)开始编号
33     rep(i,0,n)
34         rep(j,0,m){
35             t[Pos(i,j)].r = Pos(i,j+1);
36             t[Pos(i,j)].d = Pos(i+1,j);
37         }
38     rep(i,1,q){
39         int a,b,c,d,h,w;
40         scanf("%d%d%d%d%d%d",&a,&b,&c,&d,&h,&w);
41         int x = 1,y = 1,u,v;
42         rep(j,1,a-1) x = t[x].d;
43         rep(j,1,b-1) x = t[x].r;
44         rep(j,1,c-1) y = t[y].d;
45         rep(j,1,d-1) y = t[y].r;
46         u = x, v = y; //先往下再往右, 左下周长部分
47         rep(j,1,h){
48             u = t[u].d, v = t[v].d;
49             swap(t[u].r,t[v].r);
50         }
51         rep(j,1,w){
52             u = t[u].r, v = t[v].r;
53             swap(t[u].d,t[v].d);
54         }
55         u = x, v = y; //先往右再往下, 右上周长部分
56         rep(j,1,w){
57             u = t[u].r, v = t[v].r;
58             swap(t[u].d,t[v].d);
59         }
60         rep(j,1,h){
61             u = t[u].d, v = t[v].d;
62             swap(t[u].r,t[v].r);
63         }
64     }
65     int x = 1;
66     x = t[x].d, x = t[x].r;

```

```

67     rep(i,1,n){
68         int y = x;
69         rep(j,1,m){
70             printf("%d",t[y].v);
71             y = t[y].r;
72             if(j == m) printf("\n");
73             else printf(" ");
74         }
75         x = t[x].d;
76     }
77     return 0;
78 }
```

4.21 柯朵莉树

柯朵莉树应用范围：

一般适用于有区间赋值操作的题，且不能出现大量的修改操作，否则会时间爆炸

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
6 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
7 #define IT set<Node>::iterator
8 typedef long long ll;
9 typedef double db;
10 const int N = 1e5+100;
11 const db EPS = 1e-9;
12 using namespace std;
13
14 struct Node{
15     //节点含义是区间为[l,r]，区间中每个点大小均为v
16     int l,r;
17     mutable ll v; //否则无法直接修改set中的值
18     Node(int L,int R = -1, ll V = 0):l(L),r(R),v(V) {}
19     bool operator < (const Node& o) const {
20         return l < o.l;
21     }
22 };
23 set<Node> s;
24
25 ll pow_mod(ll a,ll b,ll mod){
26     ll base = a%mod, ans = 1;
27     while(b){
28         if(b&1) ans = (ans*base)%mod;
29         base = (base*base)%mod;
30         b >>= 1;
31     }
32     return ans;
33 }
34
35 //分裂区间，返回区间左端点为pos的节点
36 IT split(int pos){
37     IT it = s.lower_bound(Node(pos));
```

```

38     if(it != s.end() && it->l == pos) return it;
39     --it;
40     int L = it->l, R = it->r;
41     ll V = it->v;
42     s.erase(it);
43     s.insert(Node(L,pos-1,V));
44     //pair<iterator,bool> insert (const value_type& val)
45     return s.insert(Node(pos,R,V)).first;
46 }
47
48 //推平操作, 合并set, 令区间[l,r]中所有数等于val
49 void assign_val(int l,int r,ll val){
50     IT itl = split(l), itr = split(r+1);
51     s.erase(itl,itr);
52     s.insert(Node(l,r,val));
53 }
54
55 //区间[l,r]中每个数加val
56 void add(int l,int r,ll val){
57     IT itl = split(l), itr = split(r+1);
58     for(; itl != itr; ++itl)
59         itl->v += val;
60 }
61
62 //区间[l,r]中第k大
63 ll ranks(int l,int r,int k){
64     vector<pair<ll,int> > vp; vp.clear();
65     IT itl = split(l), itr = split(r+1);
66     for(; itl != itr; ++itl)
67         vp.push_back(pair<ll,int>(itl->v,itl->r - itl->l + 1));
68     sort(vp.begin(),vp.end());
69     for(vector<pair<ll,int> >::iterator it = vp.begin(); it != vp.end(); it++){
70         k -= it->second;
71         if(k <= 0) return it->first;
72     }
73 }
74
75 //区间[l,r]x次方和
76 ll sum(int l,int r,int ex,ll mod){
77     IT itl = split(l), itr = split(r+1);
78     ll res = 0;
79     for(; itl != itr; itl++)
80         res = (res+(ll)(itl->r - itl->l + 1ll) * pow_mod(itl->v,(ll)ex,(ll)mod))%mod;
81     return res;
82 }
83
84 int n,m;
85 ll seed,a[N],vmax;
86
87 ll rd(){
88     ll ret = seed;
89     seed = (seed*7ll+13ll)%((ll)1000000007);
90     return ret;
91 }
92
93 int main()
94 {
95     scanf("%d%d%lld%lld",&n,&m,&seed,&vmax);
96     rep(i,1,n){

```

```

97     a[i] = (rd()%vmax)+1;
98     s.insert(Node(i,i,a[i]));
99 }
100    s.insert(Node(n+1,n+1,0)); //要在末尾添加一个节点
101    rep(i,1,m){
102        int op = int(rd()%4)+1;
103        int l = int(rd()%n)+1;
104        int r = int(rd()%n)+1;
105        if(l > r) swap(l,r);
106        int x,y;
107        if(op == 3) x = int(rd()%(r-l+1))+1;
108        else x = int(rd()%vmax)+1;
109        if(op == 4)
110            y = int(rd()%vmax)+1;
111        //上方均为此题读入
112
113        if(op == 1) add(l,r,(ll)x); //区间[l,r]加上x
114        else if(op == 2) assign_val(l,r,(ll)x); //区间[l,r]所有数改成x
115        else if(op == 3) printf("%lld\n",ranks(l,r,x)); //区间[l,r]中第k大数
116        else printf("%lld\n",sum(l,r,x,y)); //区间[l,r]每个数字的x次方和%mod
117    }
118    return 0;
119 }

```

4.22 AC 自动机

4.22.1 AC 自动机模板

一、适用问题:

AC 自动机主要适用于多个模式串下的匹配问题，常嵌套 dp 进行考察或者利用其 fail 树的性质进行出题。但是这些问题都有一个相同的特点，就是一定会题目给出或者人为建出多个字符串进行匹配，此点可以用于辨别 AC 自动机问题。

二、AC 自动机算法解析

基本步骤

1. 构造一颗 Trie 树，作为 AC 自动机的搜索数据结构。
2. 构造 fail 指针，使当前字符失配时跳转到具有最长公共前后缀的字符继续匹配。如同 kmp 算法一样，AC 自动机在匹配时如果当前字符匹配失败，那么利用 fail 指针进行跳转。由此可知如果跳转，跳转后的串的前缀，必为跳转前的模式串的后缀并且跳转的新位置的深度（匹配字符个数）一定小于跳之前的节点。因此我们可以利用 bfs 在 Trie 上面进行 fail 指针的构造。

fail 指针的构造

【定义】 fail 指针指向最长的可匹配后缀，fail 指向点 p，表示从根节点到点 p 是当前匹配位置的最长后缀。

【构建方法】 按深度 bfs 整颗字典树，得到当前节点的 fail 指针需要查看其父亲的 fail 指针是否有自己这个儿子，如果没有，需要继续跳 fail 指针。

常数优化

最后我们在 AC 自动机上跑字符串的时候，我们在失配时，通过 fail 求出匹配点，用该点更新失配节点对应儿子，优化常数。

举例】 假如 $next[x][i] == 0$ ，表示节点 x 没有 i 这个节点，因此我们会不断跳 x 的 fail 指针，直到找到一个节点 u，使得 $son[u][i] != 0$ ，然后我们顺便设置 $son[x][i] = son[u][i]$ ，优化常数。

注意点

给出 n 个模式串以及 1 个匹配串，询问每个模式串在匹配串中出现的次数。在 AC 自动机上直接进行匹配时被匹配到的每个节点的 fail 节点也都被匹配到了，不能忘记计算其 fail 节点被匹配的次数。

```

1 struct Trie{
2     int next[500010][26], fail[500010], end[500010]; //fail[i]是指从root到节点i这一段字母的最
3     long 后缀节点
4     int root,L; //L相当于tot
5     int newnode()
6     {
7         for(int i = 0;i < 26;i++)
8             next[L][i] = -1; //将root节点的26个子节点都指向-1
9         end[L++] = 0; //每一个子节点初始化都不是单词结尾
10        return L-1;
11    }
12    void init()
13    {
14        L = 0;
15        root = newnode(); //此处返回root = 0
16    }
17    void insert(char buf[])
18    {
19        int len = strlen(buf);
20        int now = root;
21        for(int i = 0;i < len;i++)
22        {
23            if(next[now][buf[i]-'a'] == -1)
24                next[now][buf[i]-'a'] = newnode(); //不能在原有字典树中匹配的新字母则新建一个
25                节点
26            now = next[now][buf[i]-'a'];
27        }
28        end[now]++; //now这个节点是一个单词的结尾
29    }
30    void build()
31    {
32        queue<int>Q;
33        fail[root] = root;
34        for(int i = 0;i < 26;i++)
35        {
36            if(next[root][i] == -1)
37                next[root][i] = root; //将root的未被访问的子节点指回root
38            else
39            {
40                fail[next[root][i]] = root; //root子节点的失配指针指向root
41                Q.push(next[root][i]); //队列中加入新节点
42            }
43        }
44        while( !Q.empty() )
45        {
46            int now = Q.front();
47            Q.pop();
48            for(int i = 0;i < 26;i++)
49            {
50                if(next[now][i] == -1)
51                    next[now][i] = next[fail[now]][i]; //now的第i个节点未被访问，则将now的第
52                i个节点指向now的fail节点的第i个节点
53                else
54                {
55                    fail[next[now][i]]=next[fail[now]][i];
56                    Q.push(next[now][i]);
57                }
58            }
59        }
60        int query(char buf[])
61    }

```

```

57     {
58         int len = strlen(buf);
59         int now = root;
60         int res = 0;
61         for(int i = 0;i < len;i++)
62         {
63             now = next[now][buf[i]-'a'];
64             int temp = now;
65             while( temp != root )
66             {
67                 res += end[temp]; //访问到了这里说明，从root到tmp都已经匹配成功
68                 end[temp] = 0; //加上这句话则同一模式串在匹配串中只会匹配一次，而不会多次匹配
69                 temp = fail[temp]; //循环遍历从root到tmp这串字符串的所有后缀
70             }
71         }
72         return res;
73     }
74     void debug()
75     {
76         for(int i = 0;i < L;i++)
77         {
78             printf("id = %3d,fail = %3d,end = %3d,chi = [",i,fail[i],end[i]);
79             for(int j = 0;j < 26;j++)
80                 printf("%2d",next[i][j]);
81             printf("]\n");
82         }
83     }
84 }ac;

```

4.22.2 AC 自动机 +DP1

题意:

用 n 个基本字符组成一个长度为 m 的字符串，要求字符串中不能出现给定的 p 个非法串中任何一个，输出方案总数。
 $(1 \leq n, m \leq 50, 0 \leq p \leq 10)$

思路:

数据范围比较小，因此不难往 dp 上进行思考。又因为有多个非法串，考虑在 AC 自动机建的整个 $Trie$ 图上进行 dp 。

我们定义状态为 $f[i][j]$ ，表示长度为 i ，最后一个字符在 AC 自动机的第 j 个节点上，则枚举 j 的所有子节点，设 $now = next[j][k]$ ，即 now 为 j 的第 k 个子节点，则 $f[i][now] = f[i][now] + f[i-1][j]$ ，当且仅当 now 和 j 不为非法节点。

因此我们继续定义非法节点，一个点为非法节点，即该点所代表的字符串中出现了完整的非法串，很明显一个非法串的末尾节点是非法节点，并且若 $fail[now]$ 是非法节点，则 now 也为非法节点，因为 $fail[now]$ 节点所代表的字符串为 now 节点字符串的后缀。

除此之外，此题还有两个坑点。

1. 没有取模，因此需要大整数。
2. 字符的 ASCII 码范围在 -128 128 之间， re 了一小时...

总结:

在 AC 自动机上进行 dp ，就是以 AC 自动机上的节点作为状态进行转移，本质上与普通 dp 没有差别。

```

1 #include <cstdio>
2 #include <algorithm>
3 #include <iostream>
4 #include <cstring>
5 #include <queue>

```

```

6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 typedef long long ll;
8 using namespace std;
9
10 char buf[60],base = 0;
11 int n,m,p,mp[301];
12
13 struct Trie{
14     int next[2510][60],fail[2510],end[2510]; //fail[i]是指从root到节点i这一段字母的最长后缀节
15     点
16     int root,L; //L相当于tot
17     int newnode()
18     {
19         for(int i = 0;i < 51;i++)
20             next[L][i] = -1; //将root节点的26个子节点都指向-1
21         end[L++] = 0; //每一个子节点初始化都不是单词结尾
22         return L-1;
23     }
24     void init()
25     {
26         L = 0;
27         root = newnode(); //此处返回root = 0
28     }
29     void insert(char buf[])
30     {
31         int len = strlen(buf);
32         int now = root;
33         for(int i = 0;i < len;i++)
34         {
35             int pos = mp[buf[i]-base+150];
36             if(next[now][pos] == -1)
37                 next[now][pos] = newnode(); //不能在原有字典树中匹配的新字母则新建一个节点
38             now = next[now][pos];
39         }
40         end[now] = 1; //now这个节点是一个单词的结尾
41     }
42     void build()
43     {
44         queue<int>Q;
45         fail[root] = root;
46         for(int i = 0;i < 51;i++)
47         {
48             if(next[root][i] == -1)
49                 next[root][i] = root; //将root的未被访问的子节点指向root
50             else
51             {
52                 fail[next[root][i]] = root; //root子节点的失配指针指向root
53                 Q.push(next[root][i]); //队列中加入新节点
54             }
55         }
56         while( !Q.empty() )
57         {
58             int now = Q.front();
59             if(end[fail[now]]) end[now] = 1; //判断该节点是否非法
60             Q.pop();
61             for(int i = 0;i < 51;i++)
62                 if(next[now][i] == -1)
63                     next[now][i] = next[fail[now]][i]; //now的第i个节点未被访问，则将now的第
64         i个节点指向now的fail节点的第i个节点

```

```

63             else
64             {
65                 fail[next[now][i]]=next[fail[now]][i];
66                 Q.push(next[now][i]);
67             }
68         }
69     }
70 }ac;
71
72 struct BigInteger{
73     ll A[50];
74     enum{MOD = 10000000};
75     BigInteger(){memset(A, 0, sizeof(A)); A[0]=1;}
76     void set(ll x){memset(A, 0, sizeof(A)); A[0]=1; A[1]=x;}
77     void print(){
78         printf("%lld", A[A[0]]);
79         for (ll i=A[0]-1; i>0; i--){
80             if (A[i]==0){printf("000000"); continue;}
81             for (ll k=10; k*A[i]<MOD; k*=10) printf("0");
82             printf("%lld", A[i]);
83         }
84         printf("\n");
85     }
86     ll& operator [] (int p) {return A[p];}
87     const ll& operator [] (int p) const {return A[p];}
88     BigInteger operator + (const BigInteger& B){
89         BigInteger C;
90         C[0]=max(A[0], B[0]);
91         for (ll i=1; i<=C[0]; i++)
92             C[i]+=A[i]+B[i], C[i+1]+=C[i]/MOD, C[i]%=MOD;
93         if (C[C[0]+1] > 0) C[0]++;
94         return C;
95     }
96     BigInteger operator * (const BigInteger& B){
97         BigInteger C;
98         C[0]=A[0]*B[0];
99         for (ll i=1; i<=A[0]; i++)
100            for (ll j=1; j<=B[0]; j++){
101                C[i+j-1]+=A[i]*B[j], C[i+j]+=C[i+j-1]/MOD, C[i+j-1]%=MOD;
102            }
103        if (C[C[0]] == 0) C[0]--;
104        return C;
105    }
106 }f[2][2510];
107
108 int main()
109 {
110     scanf("%d%d%d", &n, &m, &p);
111     scanf("%s", buf);
112     rep(i,0,n-1) mp[buf[i]-base+150] = i;
113     ac.init();
114     rep(i,1,p){
115         scanf("%s", buf);
116         ac.insert(buf);
117     }
118     ac.build();
119     f[0][0].set(1);
120     rep(i,1,m){
121         rep(j,0,ac.L-1) f[i%2][j].set(0);

```

```

122     rep(j,0,ac.L-1)
123         rep(k,0,n-1){
124             int now = ac.next[j][k];
125             if(ac.end[now] == 0 && ac.end[j] == 0) f[i%2][now] = f[i%2][now] + f[(i
126 -1)%2][j];
127         }
128     BigInteger ans; ans.set(0);
129     rep(i,0,ac.L-1) ans = ans+f[m%2][i];
130     ans.print();
131     return 0;
132 }
```

4.22.3 AC 自动机 + 最短路

题意:

给定 n 和 m , 表示一共有 n 个点, 每个点都有其所对应的坐标, m 条非法路径。先要从 1 号点走到 n 号点, 但路径中不能出现 m 条非法路径中任意一条, 求最短距离。 $(1 \leq n \leq 50, 1 \leq m \leq 100)$

思路:

AC 自动机上跑最短路, 思路比较明显。 AC 自动机上的每一个节点都代表一个状态, 且不能通过任何非法节点。

需要在 AC 自动机的根节点的所有儿子上都建立新节点, 且如果一个节点的 $fail$ 节点为非法节点, 则该节点也为非法节点。

建出 $Trie$ 图后, 直接在图上跑 $dijkstra$ 最短路即可。

小坑点: 坐标之差会爆 int ... (找了半小时 bug)

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 const int N = 100+10;
4 typedef double db;
5 const db inf = 1e15;
6 using namespace std;
7
8 void dbg() {cout << "\n";}
9 template<typename T, typename... A> void dbg(T a, A... x) {cout << a << ' '; dbg(x...)}
10 #define logs(x...) {cout << #x << " -> "; dbg(x);}
11
12 int n,m,base[100],vis[25100];
13 db X[N],Y[N],dis[25100];
14 struct Node{
15     db ans; int a,b;
16     bool operator < (Node xx) const {
17         return ans > xx.ans;
18     }
19 };
20 priority_queue<Node> q;
21
22 struct Trie{
23     int next[25100][60],fail[25100],end[25100]; //fail[i]是指从root到节点i这一段字母的最长后
24     //缀节点
25     int root,L; //L相当于tot
26     int newnode()
27     {
```

```

27     for(int i = 0;i < 51;i++)
28         next[L][i] = -1; //将root节点的26个子节点都指向-1
29     end[L++] = 0; //每一个子节点初始化都不是单词结尾
30     return L-1;
31 }
32 void init()
33 {
34     L = 0;
35     root = newnode(); //此处返回root = 0
36 }
37 void insert(int len)
38 {
39     int now = root;
40     for(int i = 0;i < len;i++)
41     {
42         int pos = base[i];
43         if(next[now][pos] == -1)
44             next[now][pos] = newnode(); //不能在原有字典树中匹配的新字母则新建一个节点
45         now = next[now][pos];
46     }
47     end[now] = 1; //now这个节点是一个单词的结尾
48 }
49 void build()
50 {
51     queue<int> Q;
52     fail[root] = root;
53     for(int i = 0;i < 51;i++)
54     {
55         if(next[root][i] == -1)
56             next[root][i] = root; //将root的未被访问的子节点指回root
57         else
58         {
59             fail[next[root][i]] = root; //root子节点的失配指针指向root
60             Q.push(next[root][i]); //队列中加入新节点
61         }
62     }
63     while( !Q.empty() )
64     {
65         int now = Q.front();
66         if(end[fail[now]]) end[now] = 1; //判断该节点是否非法
67         Q.pop();
68         for(int i = 0;i < 51;i++)
69             if(next[now][i] == -1)
70                 next[now][i] = next[fail[now]][i]; //now的第i个节点未被访问，则将now的第
71                 //i个节点指向now的fail节点的第i个节点
72             else
73             {
74                 fail[next[now][i]]=next[fail[now]][i];
75                 Q.push(next[now][i]);
76             }
77     }
78 }ac;
79
80 db dist(int i,int j){
81     //坐标之差会爆int...
82     db tmp = (X[i]-X[j])*(X[i]-X[j])+(Y[i]-Y[j])*(Y[i]-Y[j]);
83     tmp = sqrt(tmp);
84     return tmp;

```

```

85 }
86
87 void dijkstra(){
88     while(q.size()) q.pop();
89     rep(i,0,50)
90         if(ac.next[ac.root][i] == 0){
91             int p = ac.newnode();
92             ac.next[ac.root][i] = p;
93             rep(j,0,50) ac.next[p][j] = 0;
94         }
95     rep(i,0,ac.L) dis[i] = inf, vis[i] = 0;
96     q.push({0,ac.next[ac.root][1],1}); dis[ac.next[ac.root][1]] = 0;
97     db ans = inf;
98     while(q.size()){
99         int now = q.top().a, id = q.top().b; q.pop();
100        if(vis[now]) continue;
101        vis[now] = 1;
102        if(id == n) ans = min(ans,dis[now]);
103        rep(i,id+1,n){
104            db tp = dis[now]+dist(id,i);
105            int y1 = now, y2 = ac.next[now][i];
106            while(y2 == 0){
107                y1 = ac.fail[y1];
108                y2 = ac.next[y1][i];
109            }
110            ac.next[now][i] = y2;
111            if(!vis[y2] && !ac.end[y2] && dis[y2] > tp){
112                dis[y2] = tp;
113                q.push({dis[y2],y2,i});
114            }
115        }
116    }
117    if(ans == inf) printf("Can not be reached!\n");
118    else printf("%.2f\n",ans);
119 }
120
121 int main(){
122     while(~scanf("%d%d",&n,&m)){
123         if(n == 0 && m == 0) break;
124         rep(i,1,n) scanf("%lf%lf",&X[i],&Y[i]);
125         ac.init();
126         rep(i,1,m){
127             int k; scanf("%d",&k);
128             rep(j,0,k-1) scanf("%d",&base[j]);
129             ac.insert(k);
130         }
131         ac.build();
132         dijkstra();
133     }
134 }
```

4.22.4 AC 自动机 +DP2

题意:

给定 n 和 m , 表示要组成一个长度为 n 的字符串, 再给出 m 个字符串, 每个串都有一个能量值, 现让你组成一个串, 使得你组成的串的总能量值最大。若能量值相同输出长度小的, 长度相同则输出字典序小的。 $(0 \leq n \leq 50, 0 \leq m \leq 100)$

思路:

涉及多个串的问题，一般都采用 AC 自动机进行解决，我们以 AC 自动机中每一个节点为状态，进行状态转移。定义 $f[i][j]$ 表示长度为 i 的串，当前处于 AC 自动机上的 j 节点的最大能量值，定义 $s[i][j]$ 为 $f[i][j]$ 状态下所组成的串，然后直接进行 dp 转移即可。

注意，此问题中每个节点的贡献值不单只是其自身串的贡献，还有其所有 $fail$ 节点贡献的累加值。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 const int inf = 1e8;
4 using namespace std;
5
6 struct Trie{
7     int next[500010][26], fail[500010], end[500010]; //fail[i]是指从root到节点i这一段字母的最
8     long long L; //L相当于tot
9     int newnode()
10    {
11        for(int i = 0; i < 26; i++)
12            next[L][i] = -1; //将root节点的26个子节点都指向-1
13        end[L] = 0; //每一个子节点初始化都不是单词结尾
14        return L-1;
15    }
16    void init()
17    {
18        L = 0;
19        root = newnode(); //此处返回root = 0
20    }
21    void insert(char buf[], int tp)
22    {
23        int len = strlen(buf);
24        int now = root;
25        for(int i = 0; i < len; i++){
26            if(next[now][buf[i]-'a'] == -1)
27                next[now][buf[i]-'a'] = newnode(); //不能在原有字典树中匹配的新字母则新建一个
28                now = next[now][buf[i]-'a'];
29            }
30            end[now] = tp; //now这个节点是一个单词的结尾
31        }
32        void build()
33        {
34            queue<int> Q;
35            fail[root] = root;
36            for(int i = 0; i < 26; i++)
37            {
38                if(next[root][i] == -1)
39                    next[root][i] = root; //将root的未被访问的子节点指回root
40                else
41                {
42                    fail[next[root][i]] = root; //root子节点的失配指针指向root
43                    Q.push(next[root][i]); //队列中加入新节点
44                }
45            }
46            while( !Q.empty() ) //按长度进行bfs
47            {
48                int now = Q.front();
49                end[now] += end[fail[now]]; //继承前面节点的value
50                Q.pop();
51                for(int i = 0; i < 26; i++)

```

```

52         if(next[now][i] == -1)
53             next[now][i] = next[fail[now]][i]; //now的第i个节点未被访问，则将now的第
54             i个节点指向now的fail节点的第i个节点
55         else
56             {
57                 fail[next[now][i]] = next[fail[now]][i];
58                 Q.push(next[now][i]);
59             }
60     }
61     int query(char buf[])
62     {
63         int len = strlen(buf);
64         int now = root;
65         int res = 0;
66         for(int i = 0; i < len; i++)
67         {
68             now = next[now][buf[i] - 'a'];
69             int temp = now;
70             while( temp != root )
71             {
72                 res += end[temp]; //访问到了这里说明，从root到tmp都已经匹配成功
73                 end[temp] = 0; //加上这句话则同一模式串在匹配串中只会匹配一次，而不会多次匹配
74                 temp = fail[temp]; //循环遍历从root到tmp这串字符串的所有后缀
75             }
76         }
77         return res;
78     }
79 }ac;
80
81 char buf[211][111];
82 int n, m, f[55][1200]; //pre记录从哪个节点来以及是啥字符
83 string s[55][1200];
84
85 void solve(){
86     rep(i, 0, n)
87         rep(j, 0, ac.L) f[i][j] = -inf, s[i][j] = "\0";
88     f[0][0] = 0;
89     rep(i, 1, n)
90         rep(j, 0, ac.L-1){
91             rep(k, 0, 25){
92                 int now = ac.next[j][k];
93                 int tp = f[i-1][j] + ac.end[now];
94                 string hp = s[i-1][j] + (char)(k + 'a');
95                 if(f[i][now] < tp || (f[i][now] == tp && hp < s[i][now])){
96                     f[i][now] = tp;
97                     s[i][now] = hp;
98                 }
99             }
100        }
101        int ans = 0;
102        rep(i, 1, n)
103            rep(j, 0, ac.L-1) ans = max(ans, f[i][j]);
104        if(ans == 0) printf("\n");
105        else{
106            string str = "\0"; int len = 100;
107            rep(i, 1, n){
108                rep(j, 0, ac.L-1)
109                    if(f[i][j] == ans){

```

```

110         if(i < len || (i == len && s[i][j] < str))
111             len = i, str = s[i][j];
112     }
113 }
114 cout << str << endl;
115 }
116 }
117
118 int main()
119 {
120     int _; scanf("%d",&_);
121     while(_--)
122     {
123         scanf("%d%d",&n,&m);
124         ac.init();
125         rep(i,1,m) scanf("%s",buf[i]);
126         rep(i,1,m){
127             int tp; scanf("%d",&tp);
128             ac.insert(buf[i],tp);
129         }
130         ac.build();
131         solve();
132     }
133     return 0;
134 }
```

4.22.5 AC 自动机 + 状压 DP

题意:

给出 L 和 N , 表示密码的长度和观察到的子串个数。再给出 N 个子串, 询问长度为 L 的密码包含这 N 个子串的所有可能方案。如果方案总数小于 42, 则输出所有方案。 $(1 \leq L \leq 25, 1 \leq N \leq 10)$

思路:

N 范围比较小, 因此考虑状压 dp , $f[i][S][j]$ 表示长度为 i , 包含的子串状态为 S , 当前在 AC 自动机的第 j 个节点上, 所有可能的方案数, 直接转移即可求取答案。

但是此题还需要输出方案, 因此我们对于每个 $f[i][S][j]$ 开一个 $vector$ 存储所有可能情况, 但是需要剪枝, 不然会 mle , 不能让任何一个 $vector$ 中的方案数大于 42, 如此即可完成此题。

```

1 #include <bits/stdc++.h>
2 typedef long long ll;
3 using namespace std;
4
5 struct Trie{
6     int next[200][26], fail[200], end[200]; //fail[i]是指从root到节点i这一段字母的最长后缀节点
7     int root,L; //L相当于tot
8     int newnode()
9     {
10         for(int i = 0;i < 26;i++)
11             next[L][i] = -1; //将root节点的26个子节点都指向-1
12         end[L++] = 0; //每一个子节点初始化都不是单词结尾
13         return L-1;
14     }
15     void init()
16     {
17         L = 0;
18         root = newnode(); //此处返回root = 0
```

```

19     }
20     void insert(char buf[], int flag)
21     {
22         int len = strlen(buf);
23         int now = root;
24         for(int i = 0; i < len; i++)
25         {
26             if(next[now][buf[i] - 'a'] == -1)
27                 next[now][buf[i] - 'a'] = newnode(); //不能在原有字典树中匹配的新字母则新建一个
28             now = next[now][buf[i] - 'a'];
29         }
30         end[now] = flag; //now这个节点是一个单词的结尾
31     }
32     void build()
33     {
34         queue<int> Q;
35         fail[root] = root;
36         for(int i = 0; i < 26; i++)
37         {
38             if(next[root][i] == -1)
39                 next[root][i] = root; //将root的未被访问的子节点指向root
40             else
41             {
42                 fail[next[root][i]] = root; //root子节点的失配指针指向root
43                 Q.push(next[root][i]); //队列中加入新节点
44             }
45         }
46         while( !Q.empty() )
47         {
48             int now = Q.front();
49             end[now] += end[fail[now]];
50             Q.pop();
51             for(int i = 0; i < 26; i++)
52                 if(next[now][i] == -1)
53                     next[now][i] = next[fail[now]][i]; //now的第i个节点未被访问，则将now的第
54             i个节点指向now的fail节点的第i个节点
55             else
56             {
57                 fail[next[now][i]] = next[fail[now]][i];
58                 Q.push(next[now][i]);
59             }
60         }
61     }ac;
62
63     char buf[110];
64     int n, m;
65     ll dp[27][1050][80];
66     vector<string> base[27][1050][80];
67
68     void solve(){
69         memset(dp, 0, sizeof dp);
70         dp[0][0][0] = 1;
71         for(int i = 1; i <= n; i++)
72             for(int S = 0; S <= (1 << m) - 1; S++)
73                 for(int j = 0; j <= ac.L - 1; j++)
74                     for(int k = 0; k <= 25; k++){
75                         int now = ac.next[j][k];

```

```

76                     dp[i][Slac.end[now]][now] += dp[i-1][S][j];
77                 }
78             ll ans = 0;
79             for(int j = 0; j <= ac.L-1; j++) ans += dp[n][(1<<m)-1][j];
80             if(ans <= 42){
81                 for(int i = 0; i <= n; i++)
82                     for(int S = 0; S <= (1<<m)-1; S++)
83                         for(int j = 0; j <= ac.L-1; j++){
84                             dp[i][S][j] = 0; base[i][S][j].clear();
85                         }
86             dp[0][0][0] = 1;
87             for(int i = 1; i <= n; i++)
88                 for(int S = 0; S <= (1<<m)-1; S++)
89                     for(int j = 0; j <= ac.L-1; j++){
90                         if(dp[i-1][S][j] == 0) continue;
91                         for(int k = 0; k <= 25; k++){
92                             int now = ac.next[j][k];
93                             dp[i][Slac.end[now]][now] += dp[i-1][S][j];
94                             if((int)base[i][Slac.end[now]][now].size()+(int)base[i-1][S][j]
95 ].size() > ans) continue;
96                             if(dp[i-1][S][j] != 0 && (int)base[i-1][S][j].size() == 0){
97                                 string tp = "\0"; tp += (char)('a'+k);
98                                 base[i][Slac.end[now]][now].push_back(tp);
99                             }
100                         else{
101                             for(auto &v:base[i-1][S][j]){
102                                 string tp = v+(char)('a'+k);
103                                 base[i][Slac.end[now]][now].push_back(tp);
104                             }
105                         }
106                     }
107             vector<string> fin; fin.clear();
108             for(int j = 0; j < ac.L; j++){
109                 for(auto &v:base[n][(1<<m)-1][j])
110                     fin.push_back(v);
111             }
112             printf("%lld\n",ans);
113             sort(fin.begin(),fin.end());
114             for(auto &v:fin)
115                 cout << v << endl;
116         }
117     else printf("%lld\n",ans);
118 }
119
120 int main()
121 {
122     scanf("%d%d",&n,&m);
123     ac.init();
124     for(int i = 0;i < m;i++){
125         scanf("%s",buf);
126         ac.insert(buf,1<<i);
127     }
128     ac.build();
129     solve();
130     return 0;
131 }
```

4.22.6 AC 自动机 + 缩点

题意:

给定 m , 表示构成字符串只能使用前 m 个字符, 再给定一个包含 n 个禁止串的集合。然后询问是否能够构成一个无限长的串, 满足以下三个条件。

1. 该串中只使用了前 m 个字符
2. 该串中不包含任何禁止串
3. 该串不能在任何位置开始出现循环节

如果可以构造则输出 *Yes*, 否则输出 *No*。 $(1 \leq n \leq 100, 1 \leq m \leq 26)$

思路:

先不考虑循环节的问题, 如果单纯考察能否构造一个只使用了前 m 个字符, 不包含禁止串且长度无限的串, 这个问题就是一个经典问题。

我们将禁止串插入 *AC* 自动机中, 所有的 *next* 出边组成一个有向图, 我们将所有非法节点从图中去除, 得到一个合法的有向图, 我们可以在这个合法的图上任意走。因此仅当这个图中出现了环, 我们才可以构建出一个无限长的串, 否则不行。

在上面的基础上, 我们考虑如何构建一个不出现循环节的长度无限的串, 无限长则说明有环, 那如果这是个简单环, 则在不断绕行的过程中一定会出现循环节。而如果这个环中有好几个环, 则可以在左边的环绕一圈, 右边的再绕两圈, 因此不会出现循环节。

因此此题只需要构建 *AC* 自动机之后, 去除无效节点, 然后对剩下的有向图跑强连通分量, 对于每个强连通分量查看是否有多个环即可。一个简单环中最多只有 n 条边, 如果大于 n 条边, 则非简单环, 因此只需判定一个环中的边数量即可。

```

1 #include <bits/stdc++.h>
2 const int N = 1e5+10;
3 typedef long long ll;
4 using namespace std;
5
6 char buf[N];
7 int n,m, pos[N], tot, head[N], dfn[N], top, sz[N], vis[N]; //top记录dfs序
8 vector<pair<int,int>> v[N];
9 ll c[N], ans[N];
10 struct Node{
11     int to,next;
12 }e[N];
13 void add(int x,int y){
14     e[++tot].to = y, e[tot].next = head[x], head[x] = tot;
15 }
16
17 struct Trie{
18     int next[N][26], fail[N], end[N], fa[N]; //fail[i]是指从root到节点i这一段字母的最长后缀节点
19     int root,L; //L相当于tot
20     int newnode(){
21         for(int i = 0;i < 26;i++)
22             next[L][i] = -1; //将root节点的26个子节点都指向-1
23         fa[L] = 0; end[L++] = 0; //每一个子节点初始化都不是单词结尾
24         return L-1;
25     }
26     void init()
27     {
28         L = 0;
29         root = newnode(); //此处返回root = 0
30     }
31     void insert(char buf[])
32 
```

```

33     {
34         int len = strlen(buf), cnt = 0;
35         int now = root;
36         for(int i = 0;i < len;i++)
37         {
38             if(buf[i] == 'B') now = fa[now];
39             else if(buf[i] == 'P') pos[++cnt] = now;
40             else{
41                 if(next[now][buf[i]-'a'] == -1){
42                     next[now][buf[i]-'a'] = newnode(); //不能在原有字典树中匹配的新字母则新建
43                     fa[next[now][buf[i]-'a']] = now;
44                 }
45                 now = next[now][buf[i]-'a'];
46             }
47         }
48     }
49     void build()
50     {
51         queue<int>Q;
52         fail[root] = root;
53         for(int i = 0;i < 26;i++)
54         {
55             if(next[root][i] == -1)
56                 next[root][i] = root; //将root的未被访问的子节点指回root
57             else
58             {
59                 fail[next[root][i]] = root; //root子节点的失配指针指向root
60                 Q.push(next[root][i]); //队列中加入新节点
61             }
62         }
63         while( !Q.empty() )
64         {
65             int now = Q.front();
66             add(fail[now],now);
67             Q.pop();
68             for(int i = 0;i < 26;i++)
69                 if(next[now][i] == -1)
70                     next[now][i] = next[fail[now]][i]; //now的第i个节点未被访问，则将now的第
71                     i个节点指向now的fail节点的第i个节点
72                 else
73                 {
74                     fail[next[now][i]]=next[fail[now]][i];
75                     Q.push(next[now][i]);
76                 }
77         }
78     }ac;
79
80     inline int lowbit(int x) {return x&(~x+1);}
81     inline void update(int x,ll v) {for(;x<=ac.L;x+=lowbit(x)) c[x] += v;}
82     inline ll ask(int x){
83         ll tp = 0;
84         while(x) tp += c[x], x -= lowbit(x);
85         return tp;
86     }
87
88     void dfs(int x){
89         dfn[x] = ++top; sz[x] = 1;

```

```

90     for(int i = head[x]; i; i = e[i].next){
91         int y = e[i].to; dfs(y);
92         sz[x] += sz[y];
93     }
94 }
95
96 void solve(){
97     int len = strlen(buf), now = ac.root;
98     int cnt = 0;
99     for(int i = 0; i < len; i++){
100         if(buf[i] == 'B'){
101             update(dfn[now], -1);
102             now = ac.fa[now];
103             vis[now] = 0;
104             continue;
105         }
106         else if(buf[i] == 'P'){
107             cnt++;
108             for(auto &hp:v[cnt]){
109                 int x = hp.first, y = cnt, id = hp.second;
110                 ll mp = ask(dfn[pos[x]]+sz[pos[x]]-1)-ask(dfn[pos[x]]-1);
111                 ans[id] = mp;
112             }
113             continue;
114         }
115         else now = ac.next[now][buf[i]-'a'];
116         if(!vis[now]){
117             vis[now] = 1;
118             update(dfn[now], 1);
119         }
120         else{
121             vis[now] = 0;
122             update(dfn[now], -1);
123         }
124     }
125     for(int i = 1; i <= m; i++) printf("%lld\n",ans[i]);
126 }
127
128 int main()
129 {
130     scanf("%s",buf);
131     scanf("%d",&m);
132     for(int i = 1; i <= m; i++){
133         int l,r; scanf("%d%d",&l,&r);
134         v[r].push_back(make_pair(l,i));
135     }
136     ac.init();
137     ac.insert(buf);
138     tot = 1;
139     ac.build();
140     for(int i = 0; i < ac.L; i++)
141         if(!dfn[i]) dfs(i);
142     solve();
143     return 0;
144 }
```

4.22.7 阿狸打字机

题意:

给出一个长串。其中'B' 表示最后一个字母被删除，'P' 表示将当前字符打印出来。一共 m 次询问，每次询问给出一个 (x, y) ，表示查询第 x 次打印的字符串在第 y 次打印的字符串中出现了多少次。 $(1 \leq n, m \leq 10^5)$

思路:

首先先将长串加入到 AC 自动机中，然后记录每个节点的父节点，遇到'B' 就往上走，遇到 P 就标记第 x 次打印对应的 是 AC 自动机上的哪个节点。

构造完 AC 自动机后，问题就变成了如何查询其中一个节点在另一个节点中出现的次数。我们首先考虑暴力匹配，其实就是要取出字符串 y ，然后在 AC 自动机中跑，所有经过的节点都标记一下，然后查询有多少个被标记的节点中包含 x 。即在 y 跑完之后，倒着将标记压到 $fail$ 节点上，然后查询 x 节点的标记次数即可。

在这个暴力过程中，我们可以发现一个节点在另一个节点中出现的次数，取决于其所有 $fail$ 子孙被标记的次数，因此我们将查询离线，按照右端点排序。然后根据 AC 自动机的 $fail$ 指针建立 $fail$ 树，求出 dfs 序，用树状数组维护每个节点子树中被标记的次数。

总结:

此题最大的收获是彻底加深了对于 $fail$ 指针的理解，一个节点在另一个字符串中出现的次数，取决于该节点所有 $fail$ 子树中被标记的次数之和。

```

1 #include <bits/stdc++.h>
2 const int N = 1e5+10;
3 typedef long long ll;
4 using namespace std;
5
6 char buf[N];
7 int n,m, pos[N], tot, head[N], dfn[N], top, sz[N], vis[N]; //top记录dfs序
8 vector<pair<int,int>> v[N];
9 ll c[N], ans[N];
10 struct Node{
11     int to,next;
12 }e[N];
13 void add(int x,int y){
14     e[++tot].to = y, e[tot].next = head[x], head[x] = tot;
15 }
16
17 struct Trie{
18     int next[N][26], fail[N], end[N], fa[N]; //fail[i]是指从root到节点i这一段字母的最长后缀节点
19     int root,L; //L相当于tot
20     int newnode(){
21     {
22         for(int i = 0;i < 26;i++)
23             next[L][i] = -1; //将root节点的26个子节点都指向-1
24         fa[L] = 0; end[L++] = 0; //每一个子节点初始化都不是单词结尾
25         return L-1;
26     }
27     void init()
28     {
29         L = 0;
30         root = newnode(); //此处返回root = 0
31     }
32     void insert(char buf[])
33     {
34         int len = strlen(buf), cnt = 0;
35         int now = root;
36         for(int i = 0;i < len;i++)
37         {
38             if(buf[i] == 'B') now = fa[now];
39             else if(buf[i] == 'P') pos[+cnt] = now;

```

```

40         else{
41             if(next[now][buf[i]-'a'] == -1){
42                 next[now][buf[i]-'a'] = newnode(); //不能在原有字典树中匹配的新字母则新建
43                 fa[next[now][buf[i]-'a']] = now;
44             }
45             now = next[now][buf[i]-'a'];
46         }
47     }
48 }
49 void build()
50 {
51     queue<int>Q;
52     fail[root] = root;
53     for(int i = 0;i < 26;i++)
54     {
55         if(next[root][i] == -1)
56             next[root][i] = root; //将root的未被访问的子节点指向root
57         else
58         {
59             fail[next[root][i]] = root; //root子节点的失配指针指向root
60             Q.push(next[root][i]); //队列中加入新节点
61         }
62     }
63     while( !Q.empty() )
64     {
65         int now = Q.front();
66         add(fail[now],now);
67         Q.pop();
68         for(int i = 0;i < 26;i++)
69             if(next[now][i] == -1)
70                 next[now][i] = next[fail[now]][i]; //now的第i个节点未被访问，则将now的第
i个节点指向now的fail节点的第i个节点
71             else
72             {
73                 fail[next[now][i]]=next[fail[now]][i];
74                 Q.push(next[now][i]);
75             }
76     }
77 }
78 }ac;
79
80 inline int lowbit(int x) {return x&(~x+1);}
81 inline void update(int x,ll v) {for(;x<=ac.L;x+=lowbit(x)) c[x] += v;}
82 inline ll ask(int x){
83     ll tp = 0;
84     while(x) tp += c[x], x -= lowbit(x);
85     return tp;
86 }
87
88 void dfs(int x){
89     dfn[x] = ++top; sz[x] = 1;
90     for(int i = head[x]; i; i = e[i].next){
91         int y = e[i].to; dfs(y);
92         sz[x] += sz[y];
93     }
94 }
95
96 void solve(){

```

```

97     int len = strlen(buf), now = ac.root;
98     int cnt = 0;
99     for(int i = 0; i < len; i++){
100         if(buf[i] == 'B'){
101             update(dfn[now], -1);
102             now = ac.fa[now];
103             vis[now] = 0;
104             continue;
105         }
106         else if(buf[i] == 'P'){
107             cnt++;
108             for(auto &hp:v[cnt]){
109                 int x = hp.first, y = cnt, id = hp.second;
110                 ll mp = ask(dfn[pos[x]]+sz[pos[x]]-1)-ask(dfn[pos[x]]-1);
111                 ans[id] = mp;
112             }
113             continue;
114         }
115         else now = ac.next[now][buf[i]-'a'];
116         if(!vis[now]){
117             vis[now] = 1;
118             update(dfn[now], 1);
119         }
120         else{
121             vis[now] = 0;
122             update(dfn[now], -1);
123         }
124     }
125     for(int i = 1; i <= m; i++) printf("%lld\n", ans[i]);
126 }
127
128 int main()
129 {
130     scanf("%s", buf);
131     scanf("%d", &m);
132     for(int i = 1; i <= m; i++){
133         int l, r; scanf("%d%d", &l, &r);
134         v[r].push_back(make_pair(l, i));
135     }
136     ac.init();
137     ac.insert(buf);
138     tot = 1;
139     ac.build();
140     for(int i = 0; i < ac.L; i++)
141         if(!dfn[i]) dfs(i);
142     solve();
143     return 0;
144 }
```

4.23 回文串问题

4.23.1 Manacher 模板

一、适用问题:

Manacher 算法主要解决的是给出一个字符串， $O(n)$ 复杂度下求出以字符串中任意一个节点为中心所能扩展的最大距离。

二、Manacher 算法解析

(1) 扩充字符串

为了统一奇偶字符串，算法首先在每两个字符（包括头尾）之间加没出现的字符（如 *），这样所有字符串长度就都是奇数了，简化了问题。

例如: $abcde \rightarrow *a * b * c * d * e *$

(2) 具体 dp 过程

记录 p_i 表示 i 能向两边推（包括 i ）的最大距离，如果能求出 p 数组，那每一个回文串就都确定了。

我们假设 $p[1:i]$ 已经求好了，现在要求 $p[i]$ 。假设之前能达到的最右边为 R ，对应的中点为 pos , j 是 i 的对称点。

1. 第一种情况， $i + p[j] < R$, 即 $p[i] = p[j]$ 。
2. 第二种情况， $i + p[j] \geq R$, 先设 $p[i] = R - i$ ，然后再继续增加 $p[i]$ ，并令 $pos = i$ ，更新 R 。

由于 R 一定是递增的，因此时间复杂度为 $O(n)$ ，可以发现一个串本质不同的回文串最多有 n 个，因此只有 R 增加的时候才会产生本质不同的回文串。

(3) 算法特点

1. Manacher 充分利用了回文的性质，从而达到线性时间。
2. Manacher 右端点递增过程中产生了所有的本质不同回文串，即一个串中本质不同回文串最多只有 n 个。
3. Manacher 算法的核心在于求出每一个节点往两边推的最远距离，所有涉及该算法的问题也都是在这个功能上做文章。

```

1 void Manacher(char s1[]){
2     int len, tot, R = 0, pos = 0;
3     //对字符串加'#'号
4     len = strlen(s1+1);
5     s2[0] = '$';
6     s2[len]= '#';
7     for(int i = 1; i <= len; i++){
8         s2[++ln] = s1[i], s2[++ln] = '#';
9     }
//求p[i]数组
10    for(int i = 1; i <= ln; i++){
11        if(R >= i) p[i] = min(p[pos*2-i], R-i+1);
12        if(p[i] + i > R){
13            for(; s2[R+1] == s2[i*2-R-1] && R+1 <= ln && i*2-R-1 <= ln; R++);
14            pos = i;
15            p[i] = R-i+1;
16        }
17    }
18 }
```

4.23.2 回文自动机模板

一、适用问题:

回文自动机其实可以当作回文串问题的大杀器，主要可以解决以下问题：

1. 本质不同回文串个数
2. 每个本质不同回文串出现的次数
3. 以某一个节点为结尾的回文串个数
4. ...

而且都是在 $O(n)$ 的复杂度内解决了这些问题。但是要注意，回文自动机并不能完全替代马拉车算法，因为马拉车针对的是以一个节点为中心所能扩展的最远的距离，而回文自动机更多的是处理以一个节点为结尾的回文串数量。

二、回文自动机算法解析

(1) 基础思想

与 AC 自动机一样，回文自动机的构造也使用了 Trie 树的结构。

不同点在于回文自动机有两个根，一个是偶数长度回文串的根，一个是奇数长度回文串的根，简称为奇根和偶根。自动机中每个节点表示一个回文串，且都有一个 *fail* 指针，指向当前节点所表示的回文串的最长回文后缀（不包括其本身）。

(2) 具体构建过程

首先将偶根的 *fail* 指针指向奇根，奇根的 *fail* 指针指向偶根，然后令奇根的长度为 -1，偶根长度为 1。

在构建过程中，始终要记录一个 *last* 指针，表示上一次插入之后所位于的回文自动机节点。然后判断插入当前节点之后是否能够形成一个新的回文串，如果不能则跳 *fail* 指针，整体逻辑与 AC 自动机的构建没有太大差别。

fail 指针表示当前节点所代表的回文串的最长后缀回文串，此处 *fail* 指针的定义与 AC 自动机有一定区别，但本质目的相同，都是为了尽可能地保存之前的匹配结果。

而 *fail* 指针的构建是根据当前节点的父节点的 *fail* 节点构建而来，与 AC 自动机的 *fail* 构建没有太大差别。

大致上就是这样一个构建过程，具体细节可以查看下述模板，也可以查看 *oiwiki* 的图解构建。

(3) 统计每个本质不同回文串的出现次数

与 AC 自动机一样，每个节点所代表字符串出现的次数要加上其所有 *fail* 子孙出现的次数，因此需要倒序遍历所有节点将节点出现次数累加到其 *fail* 节点上。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 #define per(i,a,b) for(int i = a; i >= b; i--)
4 const int N = 1e5+10;
5 using namespace std;
6
7 struct PAM{
8     #define KIND 26
9     int n,last,tot;
10    int len[N],trie[N][KIND],fail[N],cnt[N],S[N],num[N];
11    //len[i]: 节点i所代表的回文串长度, fail[i]: 当前回文串的最长回文后缀(不包括自身)
12    //cnt[i]: 节点i所代表的回文串的个数, S[i]: 第i次添加的字符, num[i]: 以第i个字符为结尾的回文串
13    //个数
14    //last: 上一个字符构成最长回文串的位置, 方便下一个字符的插入
15    //tot: 总结点个数 = 本质不同的回文串的个数+2, n: 插入字符的个数
16    int newnode(int l){
17        rep(i,0,KIND-1) trie[tot][i] = 0;
18        cnt[tot] = 0, len[tot] = l, num[tot] = 0;
19        return tot++;
20    }
21    inline void init(){
22        tot = n = last = 0, newnode(0), newnode(-1);
23        S[0] = -1, fail[0] = 1;
24    }
25    int get_fail(int x){ //获取fail指针
26        while(S[n-len[x]-1] != S[n]) x = fail[x];
27        return x;
28    }
29    inline int insert(int c){ //插入字符
30        c -= 'a';
31        S[++n] = c;
32        int cur = get_fail(last);
33        //在节点cur前的字符与当前字符相同, 即构成一个回文串
34        if(!trie[cur][c]){ //这个回文串没有出现过
35            trie[cur][c] = tot;
36            tot++;
37            num[tot] = 1;
38            cnt[tot] = 1;
39            len[tot] = len[cur]+1;
40            S[tot] = c;
41            fail[tot] = cur;
42        } else {
43            num[trie[cur][c]]++;
44            cnt[trie[cur][c]]++;
45            len[trie[cur][c]] = len[cur]+1;
46            S[trie[cur][c]] = c;
47            fail[tot] = trie[cur][c];
48        }
49    }
50}
```

```

34         int now = newnode(len[cur]+2);
35         fail[now] = trie[get_fail(fail[cur])][c];
36         trie[cur][c] = now;
37         num[now] = num[fail[now]]+1; //更新以当前字符为结尾的回文串的个数
38     }
39     last = trie[cur][c];
40     cnt[last]++; //更新当前回文串的个数
41     return num[last]; //返回以当前字符结尾的回文串的个数
42 }
43 void count(){ //统计每个本质不同回文串的个数
44     per(i,tot-1,0) cnt[fail[i]] += cnt[i];
45 }
46 }pam;

```

4.23.3 模板题

题意:

n 个字符, 找出其中所有的奇长度回文串, 按长度降序排列, 求出前 k 个回文串长度的乘积。 $(1 \leq n \leq 10^6, 1 \leq k \leq 10^{12})$

思路:

模板题。直接对于所有本质不同回文串求出其个数, 然后用 *pair* 存到 *vector* 中, 再利用快速幂计算答案即可。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 #define per(i,a,b) for(int i = a; i >= b; i--)
4 typedef long long ll;
5 const int N = 1e6+1000;
6 const ll mod = 19930726;
7 using namespace std;
8
9 struct PAM{
10     #define KIND 26
11     int n,last,tot;
12     int len[N],trie[N][KIND],fail[N],S[N],num[N];
13     ll cnt[N];
14     //len[i]: 节点i所代表的回文串长度, fail[i]: 当前回文串的最长回文后缀(不包括自身)
15     //cnt[i]: 节点i所代表的回文串的个数, S[i]: 第i次添加的字符, num[i]: 以第i个字符为结尾的回文串
16     //个数
17     //last: 上一个字符构成最长回文串的位置, 方便下一个字符的插入
18     //tot: 总结点个数 = 本质不同的回文串的个数+2, n: 插入字符的个数
19     int newnode(int l){
20         rep(i,0,KIND-1) trie[tot][i] = 0;
21         cnt[tot] = 0, len[tot] = l, num[tot] = 0;
22         return tot++;
23     }
24     inline void init(){
25         tot = n = last = 0, newnode(0), newnode(-1);
26         S[0] = -1, fail[0] = 1;
27     }
28     int get_fail(int x){ //获取fail指针
29         while(S[n-len[x]-1] != S[n]) x = fail[x];
30         return x;
31     }
32     inline int insert(int c){ //插入字符
33         c -= 'a';
34         S[++n] = c;
35         int cur = get_fail(last);

```

```

35     //在节点cur前的字符与当前字符相同，即构成一个回文串
36     if(!trie[cur][c]){ //这个回文串没有出现过
37         int now = newnode(len[cur]+2);
38         fail[now] = trie[get_fail(fail[cur])][c];
39         trie[cur][c] = now;
40         num[now] = num[fail[now]]+1; //更新以当前字符为结尾的回文串的个数
41     }
42     last = trie[cur][c];
43     cnt[last]++; //更新当前回文串的个数
44     return num[last]; //返回以当前字符结尾的回文串的个数
45 }
46 void count(){ //统计每个本质不同回文串的个数
47     per(i,tot-1,0) cnt[fail[i]] += cnt[i];
48 }
49 }pam;
50
51 int n;
52 ll k;
53 char buf[N];
54 vector<pair<int,ll> > v;
55
56 ll pow_mod(ll a,ll b){
57     ll ans = 1, base = a;
58     while(b){
59         if(b&1) ans = (ans*base)%mod;
60         base = (base*base)%mod;
61         b /= 2ll;
62     }
63     return ans;
64 }
65
66 int main(){
67     scanf("%d%lld",&n,&k);
68     scanf("%s",buf);
69     pam.init();
70     rep(i,0,n-1) pam.insert(buf[i]);
71     pam.count();
72     ll num = 0;
73     rep(i,0,pam.tot-1)
74         if(pam.len[i] > 0 && pam.len[i]%2 == 1){
75             v.push_back(make_pair(pam.len[i],pam.cnt[i]));
76             num += pam.cnt[i];
77         }
78     sort(v.begin(),v.end());
79     if(num < k) printf("-1\n");
80     else{
81         ll ans = 1;
82         int pos = v.size()-1;
83         while(k > 0){
84             if(k >= v[pos].second){
85                 ans = (ans*pow_mod(v[pos].first,v[pos].second))%mod;
86                 k -= v[pos].second;
87             }
88             else{
89                 ans = (ans*pow_mod(v[pos].first,k))%mod;
90                 k = 0;
91             }
92             pos--;
93         }
}

```

```

94         printf("%lld\n", ans);
95     }
96     return 0;
97 }
```

4.23.4 Manacher 例题

题意:

给定一个长度为 n 的字符串，询问该字符串中长度最长的双倍回文串的长度。双倍回文串指 AA^TAA^T ，如 $abbaabba$ 即是双倍回文串，而 $ababa$ 则不是。 $(1 \leq n \leq 5 * 10^5)$

思路:

由于只需要求最长回文串，并不要求输出个数，因此我们直接在 *Manacher* 的 dp 过程中求出答案即可。

首先明确一点，*Manacher* 右端点递增过程中，涉及到了所有的本质不同回文串，因此我们在每次右端点递增时，令当前节点为双倍回文串的中心节点，然后查询左半部分是否也为一个回文串，如果是则更新答案。

变型:

当然此题也可以进行延伸，可以将查询内容改成有多少个双倍回文串。如果询问有多少个双倍回文串的话，我的想法是用回文自动机 + *Manacher* 算法进行解决。

首先 *Manacher* 求出每一个节点为中心所能扩展的最远距离。然后在根据当前字符串建立回文自动机，并且在构建时对于每个节点记录构成该节点的末尾坐标 pos ，之后再遍历回文自动机上的所有节点，利用 *Manacher* 计算的 p 数组来验证该节点所代表的回文串是否是双倍回文串即可。

```

1 #include <bits/stdc++.h>
2 const int N = 5e5+100;
3 using namespace std;
4
5 int n,tot,p[2*N],R,pos,ans;
6 char s[N],s2[2*N];
7
8 void manacher(){
9     s2[0] = '$';
10    s2[tot = 1] = '#';
11    for(int i = 1; i <= n; i++)
12        s2[++tot] = s[i], s2[++tot] = '#';
13    for(int i = 1; i <= tot; i++){
14        if(R >= i) p[i] = min(p[pos*2-i], R-i+1);
15        if(p[i] + i > R){
16            for(; s2[R+1] == s2[i*2-R-1]; R++){
17                int y = (3*i-R-1)/2;
18                if(s2[i] == '#' && s2[R+1] == '#' && (R+1-i)%4 == 0 && p[y] >= i-y+1)
19                    ans = max(ans, 2*(i-y));
20            }
21            pos = i;
22            p[i] = R-i+1;
23        }
24    }
25    printf("%d\n",ans);
26 }
27 int main(){
28     scanf("%d",&n); scanf("%s",s+1);
29     manacher();
30 }
```

4.23.5 Manacher+ 回文自动机

题意:

给定一个长度为 n 的字符串，先可以选择该字符串中的一个节点，将其替换成 & 符号，问替换之后该字符串中本质不同回文串个数。 $(1 \leq n \leq 5 * 10^5)$

思路:

一个比较明显的考虑方式就是枚举替换的位置，然后计算该位置改变后增加的回文串数量与减少的回文串数量。

首先我们考虑增加的回文串数量，显然就是以该节点为中心所形成的回文串总数，因此我们直接用马拉车的 p 数组进行求解即可。

然后再来考虑减少的回文串数量。我们对于回文自动机中所有本质不同回文串的末尾出现位置记录一个 mx 和 mi ，表示最近出现的位置和最早出现的位置，记该回文串长度为 len 。则如果 $mx - len + 1 \leq mi$ ，则修改位置出现在 $[mx - len + 1, mi]$ 中时，本质不同回文串数量就会减 1，因此我们维护一个差分数组， $sum[mx - len + 1]$ 加 1， $sum[mi]$ 减 1，然后对于每个自动机上的节点都这样处理一遍即可。

最后再枚举替换点统计答案。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 #define per(i,a,b) for(int i = a; i >= b; i--)
4 const int N = 1e5+10;
5 using namespace std;
6
7 char buf[N], s2[2*N];
8 int sum[N], p[2*N];
9
10 void dbg() {cout << "\n";}
11 template<typename T, typename... A> void dbg(T a, A... x) {cout << a << ' ' ; dbg(x...)}
12 #define logs(x...) {cout << #x << " -> " ; dbg(x);}
13
14 struct PAM{
15     #define KIND 26
16     int n, last, tot;
17     int len[N], trie[N][KIND], fail[N], cnt[N], S[N], num[N], mx[N], mi[N];
18     //len[i]: 节点i所代表的回文串长度, fail[i]: 当前回文串的最长回文后缀(不包括自身)
19     //cnt[i]: 节点i所代表的回文串的个数, S[i]: 第i次添加的字符, num[i]: 以第i个字符为结尾的回文串
20     //个数
21     //last: 上一个字符构成最长回文串的位置, 方便下一个字符的插入
22     //tot: 总结点个数 = 本质不同的回文串的个数+2, n: 插入字符的个数
23     int newnode(int l){
24         rep(i,0,KIND-1) trie[tot][i] = 0;
25         cnt[tot] = 0, len[tot] = l, num[tot] = 0, mx[tot] = 0, mi[tot] = 2*1e5;
26         return tot++;
27     }
28     inline void init(){
29         tot = n = last = 0, newnode(0), newnode(-1);
30         S[0] = -1, fail[0] = 1;
31     }
32     int get_fail(int x){ //获取fail指针
33         while(S[n-len[x]-1] != S[n]) x = fail[x];
34         return x;
35     }
36     inline int insert(int c,int pos){ //插入字符
37         c -= 'a';
38         S[++n] = c;

```

```

38     int cur = get_fail(last);
39     //在节点cur前的字符与当前字符相同，即构成一个回文串
40     if(!trie[cur][c]){ //这个回文串没有出现过
41         int now = newnode(len[cur]+2);
42         fail[now] = trie[get_fail(fail[cur])][c];
43         trie[cur][c] = now;
44         num[now] = num[fail[now]]+1; //更新以当前字符为结尾的回文串的个数
45     }
46     last = trie[cur][c];
47     cnt[last]++; //更新当前回文串的个数
48     mx[last] = max(mx[last],pos);
49     mi[last] = min(mi[last],pos); //更新每个回文串的最早/晚出现位置
50     return num[last]; //返回以当前字符结尾的回文串的个数
51 }
52 void count(){ //统计每个本质不同回文串的个数
53     per(i,tot-1,2){
54         cnt[fail[i]] += cnt[i];
55         mx[fail[i]] = max(mx[fail[i]],mx[i]);
56         mi[fail[i]] = min(mi[fail[i]],mi[i]);
57         if(mi[i] >= mx[i]-len[i]+1){
58             sum[mx[i]-len[i]+1] += 1;
59             sum[mi[i]+1] += -1;
60         }
61     }
62 }
63 }pam;
64
65 void manacher(){
66     int len = strlen(buf+1), tot = 1, R = 0, pos = 0;
67     s2[0] = '$'; s2[tot = 1] = '#';
68     for(int i = 1; i <= len; i++){
69         s2[++tot] = buf[i], s2[++tot] = '#';
70     for(int i = 1; i <= tot; i++){
71         if(R >= i) p[i] = min(p[pos*2-i],R-i+1);
72         if(p[i] + i > R){
73             for(; s2[R+1] == s2[i*2-R-1] && R+1 <= tot && i*2-R-1 <= tot; R++);
74             pos = i;
75             p[i] = R-i+1;
76         }
77     }
78 }
79
80 int main(){
81     int _; scanf("%d",&_);
82     while(_--){
83         int ans = 0, ct = 0;
84         scanf("%s",buf+1);
85         pam.init();
86         int len = strlen(buf+1);
87         rep(i,0,len) sum[i] = 0;
88         rep(i,1,len) pam.insert(buf[i],i);
89         pam.count();
90         manacher();
91         rep(i,1,len){
92             sum[i] += sum[i-1];
93             int to = pam.tot-2-sum[i]+((p[i*2]-2)/2+1);
94             if(ans < pam.tot-2-sum[i]+((p[i*2]-2)/2+1)){
95                 ans = pam.tot-2-sum[i]+((p[i*2]-2)/2+1);
96                 ct = 1;

```

```

97         }
98     else if(ans == pam.tot-2-sum[i]+((p[i*2]-2)/2+1)) ct++;
99   }
100  printf("%d %d\n",ans,ct);
101 }
102 return 0;
103 }
```

4.23.6 两颗回文自动机

题意：

给定两个字符串，询问满足条件的四元组个数， (x, y, u, v) 满足条件当且仅当 $s[x\dots y] = h[u\dots v]$ ，且 $s[x\dots y]$ 为回文串。 $(1 \leq n \leq 10^5)$

思路：

建两颗回文自动机，然后从奇根和偶根分别遍历，同时遍历两棵树，如果到达相同位置则计算对答案的贡献，也算是比较裸的一道题目。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 #define per(i,a,b) for(int i = a; i >= b; i--)
4 const int N = 4*1e5+10;
5 typedef long long ll;
6 using namespace std;
7
8 char buf1[N],buf2[N];
9 ll ans = 0;
10
11 struct PAM{
12     #define KIND 26
13     int n,last,tot;
14     int len[N],trie[N][KIND],fail[N],S[N],num[N];
15     ll cnt[N];
16     //len[i]: 节点i所代表的回文串长度, fail[i]: 当前回文串的最长回文后缀(不包括自身)
17     //cnt[i]: 节点i所代表的回文串的个数, S[i]: 第i次添加的字符, num[i]: 以第i个字符为结尾的回文串
18     //个数
19     //last: 上一个字符构成最长回文串的位置, 方便下一个字符的插入
20     //tot: 总结点个数 = 本质不同的回文串的个数+2, n: 插入字符的个数
21     int newnode(int l){
22         rep(i,0,KIND-1) trie[tot][i] = 0;
23         cnt[tot] = 0, len[tot] = l, num[tot] = 0;
24         return tot++;
25     }
26     inline void init(){
27         tot = n = last = 0, newnode(0), newnode(-1);
28         S[0] = -1, fail[0] = 1;
29     }
30     int get_fail(int x){ //获取fail指针
31         while(S[n-len[x]-1] != S[n]) x = fail[x];
32         return x;
33     }
34     inline int insert(int c){ //插入字符
35         c -= 'a';
36         S[++n] = c;
37         int cur = get_fail(last);
38         //在节点cur前的字符与当前字符相同, 即构成一个回文串
39         if(!trie[cur][c]){ //这个回文串没有出现过
40             trie[cur][c] = tot;
41             tot++;
42             num[tot] = 1;
43             cnt[tot] = 1;
44             len[tot] = len[cur] + 1;
45         } else {
46             num[trie[cur][c]]++;
47         }
48     }
49 };
50
51 PAM pam;
52
53 int main(){
54     string s1,s2;
55     cin >> s1 >> s2;
56     int n1 = s1.size(), n2 = s2.size();
57     for(int i = 0; i < n1; i++) {
58         for(int j = 0; j < n2; j++) {
59             if(s1[i] == s2[j]) {
60                 pam.insert(s1[i]);
61             }
62         }
63     }
64     cout << ans << endl;
65 }
```

```

39     int now = newnode(len[cur]+2);
40     fail[now] = trie[get_fail(fail[cur])][c];
41     trie[cur][c] = now;
42     num[now] = num[fail[now]]+1; //更新以当前字符为结尾的回文串的个数
43   }
44   last = trie[cur][c];
45   cnt[last]++; //更新当前回文串的个数
46   return num[last]; //返回以当前字符结尾的回文串的个数
47 }
48 void count(){ //统计每个本质不同回文串的个数
49   per(i,tot-1,2) cnt[fail[i]] += cnt[i];
50 }
51 }pam[2];
52
53 void dfs(int t1,int t2){
54   rep(i,0,KIND-1){
55     int y1 = pam[0].trie[t1][i], y2 = pam[1].trie[t2][i];
56     if(y1 && y2){
57       dfs(y1,y2);
58       ans = ans+pam[0].cnt[y1]*pam[1].cnt[y2];
59     }
60   }
61 }
62
63 int main(){
64   int _; scanf("%d",&_);
65   rep(Ca,1,_){
66     ans = 0;
67     pam[0].init(); pam[1].init();
68     scanf("%s",buf1); scanf("%s",buf2);
69     int len1 = strlen(buf1), len2 = strlen(buf2);
70     rep(i,0,len1-1) pam[0].insert(buf1[i]);
71     rep(i,0,len2-1) pam[1].insert(buf2[i]);
72     pam[0].count(); pam[1].count();
73     dfs(0,0); dfs(1,1);
74     printf("Case #%-d: %lld\n",Ca,ans);
75   }
76   return 0;
77 }
```

4.23.7 马拉车 + 字典树

题意:

给出 n 个字符串，求解在其中 $n * n$ 个拼接中，回文串的数量。 $(\sum \text{len}(i) \leq 2 * 10^6)$

思路:

考虑马拉车 + 字典树解决该问题。

两个串拼在一起是回文串，即 s 与 h 拼接之后为回文串，主要有两种情况。

1. $|s| \leq |h|$ ，则将 s 与 h 的反串进行匹配， h 中剩下的未匹配部分为回文串。
2. $|h| \leq |s|$ ，则将 s 与 h 的反串进行匹配， s 中剩下的未匹配部分为回文串。

分类完之后即可用马拉车算法求出回文串范围，然后用字典树进行反串匹配。

```

1 #include <iostream>
2 #include <cstdio>
3 #include <algorithm>
```

```

4 #include <cstring>
5 #include <vector>
6 #define mem(a,b) memset(a,b,sizeof a);
7 #define rep(i,a,b) for(int i = a; i <= b; i++)
8 #define per(i,a,b) for(int i = a; i >= b; i--)
9 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
10 typedef long long ll;
11 typedef double db;
12 const int N = 2e6+1000;
13 const db EPS = 1e-9;
14 using namespace std;
15
16 int n,trie[N][26],tot,p[2*N],flag[N],L[N],sum1[N],sum2[N];
17 char s[N],s1[N],s2[2*N];
18 ll ans = 0;
19
20 void init(char base[],int len,int op = 0){
21     //manacher
22     int ln = 1, R = 0, pos = 0;
23     rep(i,1,len) s1[i] = base[i-1];
24     s2[0] = '$'; s2[ln = 1] = '#';
25     for(int i = 1; i <= len; i++){
26         s2[++ln] = s1[i], s2[++ln] = '#';
27         for(int i = 1; i <= ln; i++){
28             if(R >= i) p[i] = min(p[pos*2-i],R-i+1);
29             if(p[i]+i > R){
30                 for(; s2[R+1] == s2[i*2-R-1] && R+1 <= ln && i*2-R-1 <= ln; R++);
31                 pos = i;
32                 p[i] = R-i+1;
33             }
34         }
35     //预处理
36     if(op){
37         for(int i = 1; i <= len; i++){
38             int tp = i+(len-i)/2;
39             flag[i] = 0;
40             tp *= 2;
41             if((len-i) % 2 == 1) tp += 2;
42             else tp++;
43             if(p[tp] >= ln-tp+1) flag[i] = 1;
44         }
45         return;
46     }
47     for(int i = 1; i <= len; i++){
48         int tp = i-(i-1)/2;
49         flag[i] = 0;
50         tp *= 2;
51         if((i-1)%2 == 1) tp -= 2;
52         else tp--;
53         if(i != 1 && p[tp] >= tp) flag[i] = 1;
54     }
55     //插入字典树
56     int now = 0;
57     for(int i = len; i >= 1; i--){
58         if(!trie[now][s1[i]-'a']){
59             trie[now][s1[i]-'a'] = ++tot, sum1[tot] = sum2[tot] = 0;
60             rep(j,0,25) trie[tot][j] = 0;
61         }
62         now = trie[now][s1[i]-'a'];
}

```

```

63         if(flag[i]) sum2[now]++; //后半部分回文
64         if(i == 1) sum1[now]++; //末尾
65     }
66 }
67
68 void solve(char base[],int len){
69     init(base,len,1);
70     rep(i,1,len) s1[i] = base[i-1];
71     int now = 0;
72     for(int i = 1; i <= len; i++){
73         if(!trie[now][s1[i]-'a']) return;
74         now = trie[now][s1[i]-'a'];
75         if(i == len) ans += sum2[now]+sum1[now];
76         else if(flag[i]) ans += sum1[now];
77     }
78 }
79
80 int main()
81 {
82     while(~scanf("%d",&n)){
83         tot = 0; ans = 0;
84         rep(i,0,25) trie[0][i] = 0;
85         int l = 0;
86         rep(i,1,n){
87             int tp; scanf("%d%s",&tp,s+l);
88             init(s+l,tp);
89             L[i] = l; L[i+1] = l+tp;
90             l += tp;
91         }
92         ans = 0;
93         rep(i,1,n) solve(s+L[i],L[i+1]-L[i]);
94         printf("%lld\n",ans);
95     }
96     return 0;
97 }
```

4.24 后缀数组

4.24.1 后缀数组模板

一、适用问题:

后缀数组的题目非常灵活多变，主要涉及字符串所有后缀的字典序比较以及最长公共前缀。

本文主要介绍后缀数组的一些经典应用，虽然是经典应用，但是其思想应该属于后缀数组类问题的本质思想。

二、算法介绍:

求解后缀数组的算法主要有倍增法、DC3 算法，具体的算法实现此处就略过了，想要具体了解的话可以自行 *google* 搜索。

下文给出的后缀数组模板是 DC3 算法，时间复杂度为 $O(n)$ 。

(1) 后缀数组中各大数组介绍

1. $suffix[i]$: 以 i 为起始位置的后缀
2. $sa[i]$: 排名第 i 的后缀的起始位置
3. $rak[i]$: 表示 $suffix[i]$ 的排名
4. $height[i]$: $suffix(sa[i-1])$ 和 $suffix(sa[i])$ 的最长公共前缀
5. $h[i] = height[rak[i]]$, $h[i] \geq h[i-1] - 1$

有了上述数组，我们便可以求出字符串任意两个后缀的最长公共前缀。

$\text{suffix}[i]$ 和 $\text{suffix}[j]$ 之间的最长公共前缀 = $\min(\text{height}[\text{rak}[i] + 1], \dots, \text{height}[\text{rak}[j]])$

后缀数组所有的题目都是在这几个数组上做文章，具体的题目类型分类见下文例题。

```

1 #include <bits/stdc++.h>
2 const int N = 1e6+10;
3 using namespace std;
4
5 int height[N],sa[N],rak[N],len;
6 /*
7     suffix[i]: 以 i 为起始位置的后缀
8     sa[i]: 排名第 i 的后缀的起始位置
9     rak[i]: 表示 suffix[i] 的排名
10    height[i]: suffix(sa[i-1]) 和 suffix(sa[i]) 的最长公共前缀
11        . h[i] = height[rak[i]], h[i] >= h[i-1]-1
12        . suffix[i] 和 suffix[j] 之间的最长公共前缀 = min(height[rak[i]+1]...height[rak[j]])
13 */
14
15 bool cmp(int *y,int a,int b,int k)
16 {
17     int a1 = y[a],b1 = y[b];
18     int a2 = a + k >= len ? -1 : y[a + k];
19     int b2 = b + k >= len ? -1 : y[b + k];
20     return a1 == b1 && a2 == b2;
21 }
22
23 int t1[N],t2[N],cc[N];
24
25 void get_sa(char s[])
26 {
27     int *x = t1,*y = t2,m = 200;
28     for(int i = 0;i < m;i++) cc[i] = 0;
29     for(int i = 0;i < len;i++) ++ cc[x[i]] = s[i];
30     for(int i = 1;i < m;i++) cc[i] += cc[i - 1];
31     for(int i = len - 1;~i;i--) sa[-- cc[x[i]]] = i;
32     for(int k = 1;k < len;k <= 1)
33     {
34         int p = 0;
35         for(int i = len - k;i < len;i++) y[p++] = i;
36         for(int i = 0;i < len;i++) if(sa[i] >= k) y[p++] = sa[i] - k;
37         for(int i = 0;i < m;i++) cc[i] = 0;
38         for(int i = 0;i < len;i++) ++ cc[x[y[i]]];
39         for(int i = 1;i < m;i++) cc[i] += cc[i - 1];
40         for(int i = len - 1;~i;i--) sa[-- cc[x[y[i]]]] = y[i];
41         swap(x,y); m = 1; x[sa[0]] = 0;
42
43         for(int i = 1;i < len;i++)
44             x[sa[i]] = cmp(y,sa[i - 1],sa[i],k) ? m - 1 : m++;
45         if(m >= len) break;
46     }
47 }
48
49 void get_height(char s[])
50 {
51     for(int i = 0;i < len;i++) rak[sa[i]] = i;
52     int h = 0;
53     height[0] = 0;
54     for(int i = 0;i < len;i++)

```

```

55     {
56         if(!rak[i]) continue;
57         int j = sa[rak[i] - 1];
58         if(h) h--;
59         while(s[i + h] == s[j + h]) h++;
60         height[rak[i]] = h;
61     }
62 }
63
64 char s[N];
65
66 int main(){
67     scanf("%s",s);
68     len = strlen(s);
69     get_sa(s);
70     get_height(s);
71     return 0;
72 }
```

4.24.2 单字符串常见问题

一、最长公共前缀

题意: 给定一个字符串, 询问某两个后缀的最长公共前缀。

思路: 模板题。对字符串求出后缀数组后, 用线段树或 *st* 表进行 *rmq* 即可。

二、可重叠最长重复子串

题意: 给定一个字符串, 求最长重复子串, 这两个子串可以重叠。

思路: 模板题。即求字符串中任意两个后缀最长公共前缀的最大值, 即输出 *height* 数组中的最大值即可。

三、不可重叠最长重复子串

题意: 给定一个字符串, 求最长不可重复子串, 这两个子串不可以重叠。

思路: 这个题与上面那个题最大的区别在于, 此题找到的两个子串不可以重叠。因此我们在此处引入后缀数组中一个重要的思想, 二分 + *height* 分组。

我们二分一个答案 *w*, 然后根据这个答案对 *height* 数组进行分组, 即每一组中的 *height* 数组值大于 *w*。然后记录一个 *maxx* 和 *minn* 分别表示该组中所有后缀起始位置的最大值和最小值, 然后判断两个位置之间的差是否大于等于 *w*, 如果大于等于则 *w* 可行。

四、可重叠的 *k* 次最长重复子串

题意: 给定一个字符串, 求至少出现 *k* 次的最长重复子串, 这 *k* 个子串可以重叠。

思路: 利用上面那个题的 == 二分 + *height* 分组 == 思想可以轻松解决此题。即分完组之后判断该组内的字符串数是否大于 *w* 即可。

五、不相同的子串的个数

题意: 给定一个字符串, 求不相同的子串个数。

思路: 求出该字符串的后缀数组, 然后遍历 *height* 数组, 排名为 *i* 的后缀的起始位置为 *sa[i]*, 如果可以相同的话, 对答案的贡献为 $n - sa[i] + 1$ 。但是现在要求不相同的子串个数, 因此要减去最长的相同前缀, 即贡献为 $n - sa[i] + 1 - height[i]$ 。

六、连续重复子串 [整个串]

题意: 给定一个字符串 L , 已知这个字符串是由某个字符串 S 重复 R 次得到的, 求 R 的最大值。

思路: 设 LEN 为字符串 L 的长度。枚举字符串 S 的长度 len , 保证 len 能够被 LEN 整除。然后再判断 $suffix[0]$ 与 $suffix[len]$ 的最长公共前缀长度是否为 $LEN - len$, 如果是则表示该长度符合答案。

这里求后缀最长公共前缀时不需要 rmq , 由于 $suffix[0]$ 是固定的, 因此只需要求出 $height[i]$ 数组到 $height[rak[0]]$ 的最小值即可, 总时间复杂度为 $O(n)$ 。

七、重复次数最多的连续重复子串 [子串]

题意: 给定一个字符串 L , 在该字符串的所有子串中找到一个重复度最多的子串, 重复度相同则输出字典序最小。重复度即上题定义的重复次数, 例如” $ababab$ ” 重复次数为 3, ” $abcd$ ” 重复次数为 1。

思路: 我们枚举 len , 去求解长度为 len 下最大的重复度 R , 即有无一个子串由 R 个长度为 len 的子串连续拼接而成。

因此我们将字符串 L 分段, $0 \ len \ 2 * len \ 3 * len \ ... \ k * len$, 对于 $suffix[i * len]$ 与 $suffix[(i + 1) * len]$ 求出最长公共前缀 x , 再另 $pos = i * len - (len - x \% len)$, 求 $suffix[pos]$ 与 $suffix[pos + len]$ 的最长公共前缀, 然后更新答案。

这样操作的原因其实是枚举所有的以长度 len 进行重复的子串, 这也是对 len 取模数的原因。可能这样说还是有些难以理解, 推荐自己静下来想一想, 或者看看代码。

最后分析下时间复杂度, $\frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \dots + \frac{n}{n} = O(n \log n)$ 。

```

1 #include <iostream>
2 #include <cstdio>
3 #include <algorithm>
4 #include <cstring>
5 #include <cmath>
6 #define mem(a,b) memset(a,b,sizeof a);
7 #define rep(i,a,b) for(int i = a; i <= b; i++)
8 #define per(i,a,b) for(int i = a; i >= b; i--)
9 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
10 typedef long long ll;
11 typedef double db;
12 const int N = 1e5+10;
13 const db EPS = 1e-9;
14 using namespace std;
15
16 void dbg() {cout << "\n";}
17 template<typename T, typename... A> void dbg(T a, A... x) {cout << a << ' '; dbg(x...)}
18 #define logs(x...) {cout << #x << " -> "; dbg(x);}
19
20 int n,height[N],sa[N],rak[N],len; //所有数组都从0开始计数
21 /*
22     suffix[i]: 以i为起始位置的后缀
23     sa[i]: 排名第i的后缀的起始位置
24     rak[i]: 表示suffix[i]的排名
25     height[i]: suffix(sa[i-1])和suffix(sa[i])的最长公共前缀
26         . h[i] = height[rak[i]], h[i] >= h[i-1]-1
27         . suffix[i]和suffix[j]之前的最长公共前缀 = min(height[rak[i]+1]...height[rak[j]])
28 */
29
30 bool cmp(int *y,int a,int b,int k)
31 {
32     int a1 = y[a],b1 = y[b];
33     int a2 = a + k >= len ? -1 : y[a + k];
34     int b2 = b + k >= len ? -1 : y[b + k];
35     return a1 == b1 && a2 == b2;

```

```

36 }
37
38 int t1[N],t2[N],cc[N];
39
40 void get_sa(char s[])
41 {
42     int *x = t1,*y = t2,m = 200;
43     len = strlen(s);
44     for(int i = 0;i < m;i++) cc[i] = 0;
45     for(int i = 0;i < len;i++) ++ cc[x[i]] = s[i];
46     for(int i = 1;i < m;i++) cc[i] += cc[i - 1];
47     for(int i = len - 1;~i;i--) sa[-- cc[x[i]]] = i;
48     for(int k = 1;k < len;k <= 1)
49     {
50         int p = 0;
51         for(int i = len - k;i < len;i++) y[p++] = i;
52         for(int i = 0;i < len;i++) if(sa[i] >= k) y[p++] = sa[i] - k;
53         for(int i = 0;i < m;i++) cc[i] = 0;
54         for(int i = 0;i < len;i++) ++ cc[x[y[i]]];
55         for(int i = 1;i < m;i++) cc[i] += cc[i - 1];
56         for(int i = len - 1;~i;i--) sa[-- cc[x[y[i]]]] = y[i];
57         swap(x,y); m = 1; x[sa[0]] = 0;
58
59         for(int i = 1;i < len;i++)
60             x[sa[i]] = cmp(y,sa[i - 1],sa[i],k) ? m - 1 : m++;
61         if(m >= len) break;
62     }
63 }
64
65 void get_height(char s[])
66 {
67     len = strlen(s);
68     for(int i = 0;i < len;i++) rak[sa[i]] = i;
69     int h = 0;
70     height[0] = 0;
71     for(int i = 0;i < len;i++)
72     {
73         if(!rak[i]) continue;
74         int j = sa[rak[i] - 1];
75         if(h) h--;
76         while(s[i + h] == s[j + h]) h++;
77         height[rak[i]] = h;
78     }
79 }
80
81 char s[N];
82
83 int st[N][20];
84
85 void init(){
86     for(int i = 0; i < len; i++) st[i][0] = height[i];
87     for(int j = 1; (1<<j) <= len; j++){
88         for(int i = 0; i + (1<<j) - 1 < len; i++)
89             st[i][j] = min(st[i][j-1],st[i+(1<<(j-1))][j-1]);
90     }
91 }
92
93 int query(int l,int r){
94     int k = (int)(log((double)(r - l + 1)) / log(2.0));

```

```

95     return min(st[l][k], st[r-(1<<k)+1][k]);
96 }
97
98 int main(){
99     int _ = 0;
100    while(~scanf("%s", s)){
101        if(s[0] == '#') break;
102        len = strlen(s);
103        get_sa(s);
104        get_height(s);
105        init();
106        int ans = 0, LEN = 0;
107        rep(L,1,len){
108            for(int j = L; j < len; j += L){
109                int p1 = rak[j];
110                int p2 = rak[j-L];
111                if(p1 > p2) swap(p1,p2);
112                int thp = query(p1+1,p2);
113                if((thp/L)+1 > ans)
114                    ans = (thp/L)+1, LEN = L;
115
116                thp = L-thp%L;
117                if(j-L-thp < 0) continue;
118                p1 = rak[j-thp], p2 = rak[j-L-thp];
119                if(p1 > p2) swap(p1,p2);
120                thp = query(p1+1,p2);
121                if((thp/L)+1 > ans)
122                    ans = (thp/L)+1, LEN = L;
123            }
124        }
125        int pos = -1;
126        rep(i,0,len-1-LEN){
127            int p1 = rak[i], p2 = rak[i+LEN];
128            if(p1 > p2) swap(p1,p2);
129            int thp = query(p1+1,p2);
130            if((thp/LEN)+1 == ans){
131                if(pos == -1) pos = i;
132                else if(rak[i] < rak[pos]) pos = i;
133            }
134        }
135        s[pos+LEN*ans] = '\0';
136        printf("Case %d: %s\n", ++_, s+pos);
137    }
138    return 0;
139 }

```

4.24.3 最长公共子串（双串）

题意:

给定两个字符串 A 和 B , 求最长公共子串。

思路:

这里介绍一下后缀数组的另一个套路, 即涉及两个或多个串时, 往往会将两个串用一个没出现的符号 (比如'\$'), 将两个串连接在一起。

比如此题即可将两个串拼接在一起, 求出后缀数组之后, 枚举 $height$ 数组, 判断 $height$ 数组中相邻的两个后缀是否来自不同的串, 如果是则更新答案。

```

1 #include <iostream>
2 #include <cstdio>
3 #include <algorithm>
4 #include <cstring>
5 #include <cmath>
6 #define mem(a,b) memset(a,b,sizeof a);
7 #define rep(i,a,b) for(int i = a; i <= b; i++)
8 #define per(i,a,b) for(int i = a; i >= b; i--)
9 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
10 typedef long long ll;
11 typedef double db;
12 const int N = 2*1e5+10;
13 const db EPS = 1e-9;
14 using namespace std;
15
16 void dbg() {cout << "\n";}
17 template<typename T, typename... A> void dbg(T a, A... x) {cout << a << ' ' ; dbg(x...)}
18 #define logs(x...) {cout << #x << " -> " ; dbg(x);}
19
20 int n,height[N],sa[N],rak[N],len; //所有数组都从0开始计数
21 /*
22     suffix[i]: 以i为起始位置的后缀
23     sa[i]: 排名第i的后缀的起始位置
24     rak[i]: 表示suffix[i]的排名
25     height[i]: suffix(sa[i-1])和suffix(sa[i])的最长公共前缀
26         . h[i] = height[rak[i]], h[i] >= h[i-1]-1
27         . suffix[i]和suffix[j]之前的最长公共前缀 = min(height[rak[i]+1]...height[rak[j]])
28 */
29
30 bool cmp(int *y,int a,int b,int k)
31 {
32     int a1 = y[a],b1 = y[b];
33     int a2 = a + k >= len ? -1 : y[a + k];
34     int b2 = b + k >= len ? -1 : y[b + k];
35     return a1 == b1 && a2 == b2;
36 }
37
38 int t1[N],t2[N],cc[N];
39
40 void get_sa(char s[])
41 {
42     int *x = t1,*y = t2,m = 200;
43     len = strlen(s);
44     for(int i = 0;i < m;i++) cc[i] = 0;
45     for(int i = 0;i < len;i++) ++ cc[x[i]] = s[i];
46     for(int i = 1;i < m;i++) cc[i] += cc[i - 1];
47     for(int i = len - 1;~i;i--) sa[-- cc[x[i]]] = i;
48     for(int k = 1;k < len;k <= 1)
49     {
50         int p = 0;
51         for(int i = len - k;i < len;i++) y[p ++] = i;
52         for(int i = 0;i < len;i++) if(sa[i] >= k) y[p ++] = sa[i] - k;
53         for(int i = 0;i < m;i++) cc[i] = 0;
54         for(int i = 0;i < len;i++) ++ cc[x[y[i]]];
55         for(int i = 1;i < m;i++) cc[i] += cc[i - 1];
56         for(int i = len - 1;~i;i--) sa[-- cc[x[y[i]]]] = y[i];
57         swap(x,y); m = 1; x[sa[0]] = 0;
58 }

```

```

59         for(int i = 1;i < len;i++)
60             x[sa[i]] = cmp(y,sa[i - 1],sa[i],k) ? m - 1 : m++;
61         if(m >= len) break;
62     }
63 }
64
65 void get_height(char s[])
66 {
67     len = strlen(s);
68     for(int i = 0;i < len;i++) rak[sa[i]] = i;
69     int h = 0;
70     height[0] = 0;
71     for(int i = 0;i < len;i++)
72     {
73         if(!rak[i]) continue;
74         int j = sa[rak[i] - 1];
75         if(h) h--;
76         while(s[i + h] == s[j + h]) h++;
77         height[rak[i]] = h;
78     }
79 }
80
81 char s1[N],s2[N],s[N];
82
83 int main(){
84     scanf("%s",s1);
85     scanf("%s",s2);
86     int len1 = strlen(s1), len2 = strlen(s2);
87     rep(i,0,len1-1) s[i] = s1[i];
88     s[len1] = '$';
89     rep(i,0,len2-1) s[i+len1+1] = s2[i];
90     len = strlen(s);
91     get_sa(s);
92     get_height(s);
93 //#[0,len1-1] [len1+1,len1+len2]
94     int ans = 0;
95     rep(i,1,len1+len2){
96         int p1 = sa[i], p2 = sa[i-1];
97         if(p1 <= len1-1 && p2 >= len1+1) ans = max(ans,height[i]);
98         else if(p2 <= len1-1 && p1 >= len1+1) ans = max(ans,height[i]);
99     }
100    printf("%d\n",ans);
101    return 0;
102 }
103 int n;
104 ll k;
105 char buf[N];
106 vector<pair<int,ll> > v;
107
108 ll pow_mod(ll a,ll b){
109     ll ans = 1, base = a;
110     while(b){
111         if(b&1) ans = (ans*base)%mod;
112         base = (base*base)%mod;
113         b /= 2ll;
114     }
115     return ans;
116 }
117

```

```

118 int main(){
119     scanf("%d%lld",&n,&k);
120     scanf("%s",buf);
121     pam.init();
122     rep(i,0,n-1) pam.insert(buf[i]);
123     pam.count();
124     ll num = 0;
125     rep(i,0,pam.tot-1)
126         if(pam.len[i] > 0 && pam.len[i]%2 == 1){
127             v.push_back(make_pair(pam.len[i],pam.cnt[i]));
128             num += pam.cnt[i];
129         }
130     sort(v.begin(),v.end());
131     if(num < k) printf("-1\n");
132     else{
133         ll ans = 1;
134         int pos = v.size()-1;
135         while(k > 0){
136             if(k >= v[pos].second){
137                 ans = (ans*pow_mod(v[pos].first,v[pos].second))%mod;
138                 k -= v[pos].second;
139             }
140             else{
141                 ans = (ans*pow_mod(v[pos].first,k))%mod;
142                 k = 0;
143             }
144             pos--;
145         }
146         printf("%lld\n",ans);
147     }
148     return 0;
149 }
```

4.24.4 长度不小于 k 的公共子串个数

题意:

给定两个字符串 A 和 B , 求长度不小于 k 的公共子串的个数, 即询问有多少个三元组 (i, j, len) 表示 $A[i] A[i + len - 1]$ 与 $B[j] B[j + len - 1]$ 相同, 且 $len \geq k$ 。

思路:

首先将两个串拼接起来, 求出后缀数组之后, 按 k 对 $height$ 数组进行分组。对于两个最长公共前缀为 h 的后缀, 其对答案的贡献为 $h - k + 1$, 因此我们分组之后, 先考虑 A 对 B 的贡献, 维护一个单调栈, 存储信息为 $pair(int, int)$ 表示 $height$ 的值与个数, 每当有更小的值进入时就不断向前合并, 主要是 $second$ 的信息合并, 并且记录一个全局的 $total = \sum first * second$, 用于计算贡献。

之后再计算一遍 B 对 A 的贡献即可结束此题。

```

1 #include <iostream>
2 #include <cstdio>
3 #include <algorithm>
4 #include <cstring>
5 #include <cmath>
6 #define mem(a,b) memset(a,b,sizeof a);
7 #define rep(i,a,b) for(int i = a; i <= b; i++)
8 #define per(i,a,b) for(int i = a; i >= b; i--)
9 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
10 typedef long long ll;
```

```

11  typedef double db;
12  const int N = 2*1e5+10;
13  const db EPS = 1e-9;
14  using namespace std;
15
16  void dbg() {cout << "\n";}
17  template<typename T, typename... A> void dbg(T a, A... x) {cout << a << ' ' ; dbg(x...)}
18  #define logs(x...) {cout << #x << " -> "; dbg(x);}
19
20  int n,k,height[N],sa[N],rak[N],len; //所有数组都从0开始计数
21  /*
22      suffix[i]: 以i为起始位置的后缀
23      sa[i]: 排名第i的后缀的起始位置
24      rak[i]: 表示suffix[i]的排名
25      height[i]: suffix(sa[i-1])和suffix(sa[i])的最长公共前缀
26          . h[i] = height[rak[i]], h[i] >= h[i-1]-1
27          . suffix[i]和suffix[j]之前的最长公共前缀 = min(height[rak[i]+1]...height[rak[j]])
28  */
29
30  bool cmp(int *y,int a,int b,int k)
31  {
32      int a1 = y[a],b1 = y[b];
33      int a2 = a + k >= len ? -1 : y[a + k];
34      int b2 = b + k >= len ? -1 : y[b + k];
35      return a1 == b1 && a2 == b2;
36  }
37
38  int t1[N],t2[N],cc[N];
39
40  void get_sa(char s[])
41  {
42      int *x = t1,*y = t2,m = 200;
43      len = strlen(s);
44      for(int i = 0;i < m;i++) cc[i] = 0;
45      for(int i = 0;i < len;i++) ++ cc[x[i]] = s[i];
46      for(int i = 1;i < m;i++) cc[i] += cc[i - 1];
47      for(int i = len - 1;~i;i--) sa[-- cc[x[i]]] = i;
48      for(int k = 1;k < len;k <= 1)
49      {
50          int p = 0;
51          for(int i = len - k;i < len;i++) y[p++] = i;
52          for(int i = 0;i < len;i++) if(sa[i] >= k) y[p++] = sa[i] - k;
53          for(int i = 0;i < m;i++) cc[i] = 0;
54          for(int i = 0;i < len;i++) ++ cc[x[y[i]]];
55          for(int i = 1;i < m;i++) cc[i] += cc[i - 1];
56          for(int i = len - 1;~i;i--) sa[-- cc[x[y[i]]]] = y[i];
57          swap(x,y); m = 1; x[sa[0]] = 0;
58
59          for(int i = 1;i < len;i++)
60              x[sa[i]] = cmp(y,sa[i - 1],sa[i],k) ? m - 1 : m++;
61          if(m >= len) break;
62      }
63  }
64
65  void get_height(char s[])
66  {
67      len = strlen(s);
68      for(int i = 0;i < len;i++) rak[sa[i]] = i;

```

```

69     int h = 0;
70     height[0] = 0;
71     for(int i = 0;i < len;i++)
72     {
73         if(!rak[i]) continue;
74         int j = sa[rak[i] - 1];
75         if(h) h--;
76         while(s[i + h] == s[j + h]) h++;
77         height[rak[i]] = h;
78     }
79 }
80
81 char s1[N],s2[N],s[N];
82
83 int tot;
84 pair<int,int> st[N];
85
86 int main(){
87     while(~scanf("%d",&k)){
88         if(k == 0) break;
89         scanf("%s",s1);
90         scanf("%s",s2);
91         int len1 = strlen(s1), len2 = strlen(s2);
92         rep(i,0,len1-1) s[i] = s1[i];
93         s[len1] = '$';
94         rep(i,0,len2-1) s[i+len1+1] = s2[i];
95         len = strlen(s);
96         get_sa(s);
97         get_height(s);
98
99         //A:[0,len1-1], B:[len1+1,len1+len2]
100        ll ans = 0, total = 0;
101        tot = -1;
102        // rep(i,0,len1+len2) logs(i,sa[i],height[i]);
103        rep(i,1,len1+len2){
104            if(height[i] < k) {tot = -1; total = 0; continue;}
105            else{
106                // logs(i,sa[i]);
107                int cnt = 0;
108                if(sa[i-1] <= len1-1) cnt = 1;
109                while(tot >= 0 && st[tot].first >= height[i]){
110                    total -= (ll)(st[tot].first-k+1)*(ll)st[tot].second;
111                    cnt += st[tot].second;
112                    tot--;
113                }
114                if(cnt != 0){
115                    st[++tot] = make_pair(height[i],cnt);
116                    total += (ll)(height[i]-k+1)*(ll)cnt;
117                }
118                if(sa[i] >= len1+1) ans += total;
119            }
120        }
121        tot = -1; total = 0;
122        rep(i,1,len1+len2){
123            if(height[i] < k) {tot = -1; total = 0; continue;}
124            else{
125                int cnt = 0;
126                if(sa[i-1] >= len1+1) cnt = 1;
127                while(tot >= 0 && st[tot].first >= height[i]){

```

```

128         total -= (ll)(st[tot].first-k+1)*(ll)st[tot].second;
129         cnt += st[tot].second;
130         tot--;
131     }
132     if(cnt != 0){
133         st[++tot] = make_pair(height[i],cnt);
134         total += (ll)(height[i]-k+1)*(ll)cnt;
135     }
136     // logs(i,sa[i],height[i],total);
137     if(sa[i] <= len1-1) ans += total;
138 }
139 }
140 printf("%lld\n",ans);
141 rep(i,0,len1+len2) s[i] = '\0';
142 }
143 return 0;
144 }
```

4.24.5 n 字符串问题

一、至少在 k 个字符串中出现的最长子串

题意：给定 n 个字符串，求出现在不少于 k 个字符串中的最长子串。

思路：首先将 n 个字符串拼接在一起，再二分答案进行 $height$ 分组，然后查看每一组中出现了多少个不同的字符串，如果大于等于 k 则符合题意。

二、每个字符串至少出现两次且不重叠的最长子串

题意：给定 n 个字符串，求在每个字符串中至少出现两次且不重叠的最长子串。

思路：与上题没有太大的差别，还是先将 n 个字符串拼接在一起，再二分答案进行 $height$ 分组，记录每一组中出现位置的最大值和最小值，然后查看有多少组的最大值和最小值之差符合条件，如果大于等于 k 则符合题意。

```
1 return 0;
```

4.25 整体二分

4.25.1 无修改第 K 大

一、适用问题：

整体二分，即对所有的查询进行一个整体的二分答案，需要数据结构题满足以下性质。

1. 询问的答案具有可二分性
2. 修改对判定答案的贡献相对独立，修改之间互不影响效果
3. 修改如果对判定答案有贡献，则贡献为一确定的与判定标准无关的值
4. 贡献满足交换律、结合律，具有可加性
5. 题目允许离线操作

(来自《浅谈数据结构题的几个非经典解法》)

上面的性质看上去复杂，其实只要满足询问答案具有可二分性，且题目允许离线操作，就可以考虑一下是否可以利用整体二分算法进行求解。

二、整体二分算法解析

上面的文字可能有些过于理论，我们现在用浅显一点的方式来理解这个算法。

假设你现在有 q 次查询，查询区间第 k 大的值。首先考虑如果只有 1 个查询，是否可以直接二分解决。

显然是可以的，我们只需要定位到具体区间，数一下小于等于当前二分值的数个数是否大于等于 k 即可。于是问题就变成了如何从单次二分演变到整体二分。

我们首先维护一个操作序列，即每个点的赋值和查询，共 $n + q$ 个操作。然后实现一个 $solve(l, r, L, R)$ 函数，表示当前的操作序列在 $[L, R]$ 范围内，而该操作序列中所有的查询操作的答案都在 $[l, r]$ 中。

于是我们二分一个值 $mid = (l + r)/2$ ，然后将 $[L, R]$ 中所有的赋值操作中数值小于等于 mid 的数加入到对应位置，比如 $a[x] \leq mid$ ，则 $sum[x] = sum[x] + 1$ ，对于所有的查询操作，判断其查询区间 $[x, y]$ 的值是否大于等于 k ，如果是则将其递归到 $solve(l, mid)$ 中，否则递归到 $solve(mid + 1, r)$ 中，具体内容看一下下面的例题就可以理解。

最后分析一下时间复杂度，最多分了 $\log n$ 层，每一层的时间复杂度为 $O(n\log n)$ ，因此总时间复杂度为 $O(n\log^2 n)$ 。

最后附上《浅谈数据结构题的几个非经典解法》中对该算法的理论概述。

询问的答案可二分且修改对判定标准的贡献相对独立，且贡献的值与判定标准无关。因此如果我们已经计算过某些修改对询问的贡献，那么这个贡献永远不会改变，我们没有必要当判定标准改变时再次计算这部分修改的贡献，只要记录下当前的总贡献，再进一步二分时，直接加上新的贡献即可。

三、例题解析

题意：

无修改的区间第 k 大数问题。 $(1 \leq n \leq 10^5, 1 \leq m \leq 5000)$

思路：

主席树模板题，但此处我们要用整体二分的方法来解决此题。

首先我们将所有赋值操作和查询操作都放到一个数组中，形成了此题的操作序列。然后就是代码中的核心关键点 $solve(l, r, L, R)$ 函数，该函数表示区间 $[L, R]$ 中的操作序列中的查询操作的答案一定在 $[l, r]$ 范围内。

因此问题就变成了如何将 $[L, R]$ 中的序列进行分组，再递归到 $solve(l, mid)$ 和 $solve(mid + 1, r)$ 中。我们只需遍历 $[L, R]$ 中的所有操作，如果是赋值操作，则判断数值 x 是否大于 mid ，如果小于等于 mid ，则将该操作丢到 q_1 数组中，并在树状数组的 x 位置加 1；否则将操作丢到 q_2 数组中。

如果是查询操作，则在树状数组中查询区间 $[q[i].x, q[i].y]$ 的值 tmp ，如果 $k \leq tmp$ ，则将该操作丢到 q_1 中；否则将 $k = k - tmp$ ，然后丢到 q_2 中。

可能说起来比较复杂，但是代码比较清晰，推荐直接对代码进行理解。

```

1 #include <iostream>
2 #include <algorithm>
3 #include <cstdio>
4 #define mem(a,b) memset(a,b,sizeof a);
5 #define rep(i,a,b) for(int i = a; i <= b; i++)
6 #define per(i,a,b) for(int i = a; i >= b; i--)
7 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
8 typedef long long ll;
9 typedef double db;
10 const int N = 1e5+100;
11 const int inf = 1e9+10;
12 const db EPS = 1e-9;
13 using namespace std;
14
15 void dbg() {cout << "\n";}
16 template<typename T, typename... A> void dbg(T a, A... x) {cout << a << ' ' ; dbg(x...)}
17 #define logs(x...) {cout << #x << " -> "; dbg(x);}
18
19 int n,m,ans[N],c[N];
20 struct Node{int x,y,k,id;}q[2*N],q1[2*N],q2[2*N];

```

```

21 inline int lowbit(int x) {return x&(~x+1);}
22 inline void update(int x,int v) {for(; x<=n; x+=lowbit(x)) c[x] += v;}
23 inline int ask(int x){
24     int res = 0;
25     while(x) res += c[x], x -= lowbit(x);
26     return res;
27 }
28
29 void solve(int l,int r,int L,int R){
30     if(l > r || L > R) return;
31     if(l == r){
32         rep(i,L,R) if(q[i].id) ans[q[i].id] = l;
33         return;
34     }
35     int cnt1 = 0, cnt2 = 0, mid = (l+r)>>1;
36     rep(i,L,R){
37         if(q[i].id){ //查询
38             int tmp = ask(q[i].y)-ask(q[i].x-1);
39             if(q[i].k <= tmp) q1[++cnt1] = q[i];
40             else q[i].k -= tmp, q2[++cnt2] = q[i];
41         }
42         else{ //赋值
43             if(q[i].x <= mid) update(q[i].y,1), q1[++cnt1] = q[i];
44             else q2[++cnt2] = q[i];
45         }
46     }
47     rep(i,1,cnt1) if(!q1[i].id) update(q1[i].y,-1);
48     rep(i,1,cnt1) q[L+i-1] = q1[i];
49     rep(i,1,cnt2) q[L+cnt1+i-1] = q2[i];
50     solve(l,mid,L,L+cnt1-1); solve(mid+1,r,L+cnt1,R);
51 }
52
53 int main()
54 {
55     scanf("%d%d",&n,&m);
56     rep(i,1,n) {scanf("%d",&q[i].x); q[i].id = 0; q[i].y = i;}
57     rep(i,1,m) {scanf("%d%d%d",&q[i+n].x,&q[i+n].y,&q[i+n].k); q[i+n].id = i;}
58     solve(-inf,inf,1,n+m);
59     rep(i,1,m) printf("%d\n",ans[i]);
60     return 0;
61 }

```

4.25.2 带修改第 K 大

题意:

带修改的区间第 k 大数问题。 $(1 \leq n \leq 5 * 10^4, 1 \leq m \leq 10^4)$

思路:

带修改第 k 大问题，如果要用主席树来解决的话，则需要再加上一层树状数组来维护修改信息，即用树套树解决该问题。

但是如果用整体二分来处理这个问题的话，难度则会瞬间骤降。其实此题与上题唯一的区别就是这题多了一个修改操作，而修改操作无非就是删除原来的数，加上新的数。

因此对于 $a[x] = y$ 的修改操作，我们将其拆成两部分，第一部分为删除 $a[x]$ ，然后令 $a[x] = y$ ，第二部分是加上 $a[x]$ ，具体细节可以参考下面的代码实现。

```
1 #include <iostream>
```

```

2 #include <algorithm>
3 #include <cstdio>
4 #include <cstring>
5 #define mem(a,b) memset(a,b,sizeof a);
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 #define per(i,a,b) for(int i = a; i >= b; i--)
8 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
9 typedef long long ll;
10 typedef double db;
11 const int N = 1e5+100;
12 const int inf = 1e9+10;
13 const db EPS = 1e-9;
14 using namespace std;
15
16 void dbg() {cout << "\n";}
17 template<typename T, typename... A> void dbg(T a, A... x) {cout << a << ' ' ; dbg(x...)}
18 #define logs(x...) {cout << #x << " -> " ; dbg(x);}
19
20 int n,m,ans[N],c[N],a[N];
21 struct Node{int x,y,k,id;}q[2*N],q1[2*N],q2[2*N];
22 inline int lowbit(int x) {return x&(~x+1);}
23 inline void update(int x,int v) {for(; x<=n; x+=lowbit(x)) c[x] += v;}
24 inline int ask(int x){
25     int res = 0;
26     while(x) res += c[x], x -= lowbit(x);
27     return res;
28 }
29
30 void solve(int l,int r,int L,int R){
31     if(l > r || L > R) return;
32     if(l == r){
33         rep(i,L,R) if(q[i].k) ans[q[i].id] = l;
34         return;
35     }
36     int cnt1 = 0, cnt2 = 0, mid = (l+r)>>1;
37     rep(i,L,R){
38         if(q[i].k){ //查询
39             int tmp = ask(q[i].y)-ask(q[i].x-1);
40             if(q[i].k <= tmp) q1[++cnt1] = q[i];
41             else q[i].k -= tmp, q2[++cnt2] = q[i];
42         }
43         else{ //赋值
44             if(q[i].x <= mid) update(q[i].id,q[i].y), q1[++cnt1] = q[i];
45             else q2[++cnt2] = q[i];
46         }
47     }
48     rep(i,1,cnt1) if(!q1[i].k) update(q1[i].id,-q1[i].y);
49     rep(i,1,cnt1) q[L+i-1] = q1[i];
50     rep(i,1,cnt2) q[L+cnt1+i-1] = q2[i];
51     solve(l,mid,L+cnt1-1); solve(mid+1,r,L+cnt1,R);
52 }
53
54 int main()
55 {
56     int _; scanf("%d",&_);
57     while(_--){
58         scanf("%d%d",&n,&m);
59         int cnt = 0, tot = 0;

```

```

60     memset(c, 0, sizeof c);
61     rep(i, 1, n) {
62         scanf("%d", &a[i]);
63         q[++cnt] = {a[i], 1, 0, i};
64     }
65     rep(i, 1, m) {
66         char op[5]; scanf("%s", op);
67         int x, y, k;
68         if(op[0] == 'Q') {
69             scanf("%d%d%d", &x, &y, &k);
70             q[++cnt] = {x, y, k, ++tot};
71         }
72     else{
73         scanf("%d%d", &x, &y);
74         q[++cnt] = {a[x], -1, 0, x};
75         a[x] = y;
76         q[++cnt] = {a[x], 1, 0, x};
77     }
78     solve(-inf, inf, 1, cnt);
79     rep(i, 1, tot) printf("%d\n", ans[i]);
80 }
81 return 0;
82 }
83 }
```

4.25.3 线段树 + 整体二分

题意:

n 个位置, m 个操作。操作有两种, 1 $a b c$ 表示在第 a 个位置到第 b 个位置, 每个位置加入一个数 c ; 2 $a b c$ 表示询问从第 a 个位置到第 b 个位置, 第 c 大的数是多少。 $(1 \leq n, m \leq 5 * 10^4)$

思路:

其实和上面第二个问题没有太大的差别, 只不过上一个问题问题是单点修改, 而这题变成了区间修改。因此我们用线段树维护一下整体二分即可解决。

```

1 #include <iostream>
2 #include <algorithm>
3 #include <cstdio>
4 #include <cstring>
5 #define mem(a,b) memset(a,b,sizeof a);
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 #define per(i,a,b) for(int i = a; i >= b; i--)
8 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
9 typedef long long ll;
10 typedef double db;
11 const int N = 1e5+100;
12 const int inf = 1e9+10;
13 const db EPS = 1e-9;
14 using namespace std;
15
16 void dbg() {cout << "\n";}
17 template<typename T, typename... A> void dbg(T a, A... x) {cout << a << ' ' ; dbg(x...)}
18 #define logs(x...) {cout << #x << " -> "; dbg(x);}
19
20 int n,m;
21 ll ans[N],sum[2*N],lazy[2*N];
```

```

22 struct Node{ll x,y,k,id;}q[2*N],q1[2*N],q2[2*N];
23
24 inline int get_id(int l,int r) {return (l+r)|(l!=r);}
25 inline void pushDown(int l,int r){
26     int mid = (l+r)>>1, now = get_id(l,r), ls = get_id(l,mid), rs = get_id(mid+1,r);
27     sum[ls] += lazy[now]*(ll)(mid-l+1); sum[rs] += lazy[now]*(ll)(r-mid);
28     lazy[ls] += lazy[now]; lazy[rs] += lazy[now];
29     lazy[now] = 0;
30 }
31 inline void update(int l,int r,int L,int R,int v){
32     int now = get_id(l,r);
33     if(L <= l && r <= R){
34         sum[now] += (ll)v*(ll)(r-l+1);
35         lazy[now] += v;
36         return;
37     }
38     if(lazy[now]) pushDown(l,r);
39     int mid = (l+r)>>1;
40     if(L <= mid) update(l,mid,L,R,v);
41     if(R > mid) update(mid+1,r,L,R,v);
42     sum[now] = sum[get_id(l,mid)]+sum[get_id(mid+1,r)];
43 }
44 inline ll query(int l,int r,int L,int R){
45     int now = get_id(l,r);
46     if(L <= l && r <= R) return sum[now];
47     if(lazy[now]) pushDown(l,r);
48     int mid = (l+r)>>1;
49     ll thp = 0;
50     if(L <= mid) thp += query(l,mid,L,R);
51     if(R > mid) thp += query(mid+1,r,L,R);
52     return thp;
53 }
54
55 void solve(int l,int r,int L,int R){
56     if(l > r || L > R) return;
57     if(l == r){
58         rep(i,L,R) if(q[i].id) ans[q[i].id] = l;
59         return;
60     }
61     int cnt1 = 0, cnt2 = 0, mid = (l+r)>>1;
62     rep(i,L,R){
63         if(q[i].id){ //查询
64             ll tmp = query(1,n,q[i].x,q[i].y);
65             if(q[i].k <= tmp) q2[++cnt2] = q[i];
66             else q[i].k -= tmp, q1[++cnt1] = q[i];
67         }
68         else{ //赋值
69             //由于右区间的起始点为mid+1, 因此此处为 >= mid+1
70             if(q[i].k >= mid+1) update(1,n,q[i].x,q[i].y,1), q2[++cnt2] = q[i];
71             else q1[++cnt1] = q[i];
72         }
73     }
74     rep(i,1,cnt2) if(!q2[i].id) update(1,n,q2[i].x,q2[i].y,-1);
75     rep(i,1,cnt1) q[L+i-1] = q1[i];
76     rep(i,1,cnt2) q[L+cnt1+i-1] = q2[i];
77     solve(l,mid,L+cnt1-1); solve(mid+1,r,L+cnt1,R);
78 }
79
80 int main()

```

```

81 {
82     scanf("%d%d",&n,&m);
83     int tot = 0;
84     rep(i,1,m){
85         ll op,x,y,k;
86         scanf("%lld%lld%lld%lld",&op,&x,&y,&k);
87         if(op == 1) q[i] = {x,y,k,0};
88         else q[i] = {x,y,k,++tot};
89     }
90     solve(-inf,inf,1,m);
91     rep(i,1,tot) printf("%lld\n",ans[i]);
92     return 0;
93 }
```

4.25.4 并查集 + 整体二分

题意:

一个 n 个点, m 条边的图, 第 i 条边连接 a_i 和 b_i , 保证图是连通的。

现在有 q 次询问, 每次询问给出一个三元组 $x \ y \ z$, 表示询问从 $x \ y$ 两个点出发, 一共扩展 z 个不同的点 (包括起始点), 求所经过的边中最大编号的最小值。 $(3 \leq n \leq 10^5, 1 \leq q \leq 10^5)$

思路:

此题较之上面三题, 没有那么套路, 因此我们先从只有一个询问开始找思路。

首先考虑能不能把图变成树, 因为图上问题往往都很复杂, 而变成树上问题后我们的可操作空间会大很多。继续思考不难发现, 如果我们按边的编号为权值构建一棵最小生成树, 每次询问的答案也一定会落在最小生成树上的边上。

转到树上问题之后, 我们考虑能不能二分答案然后 *check*, 如果只有一个询问的话, 显然是可以的。只需要维护一个可加边可删边的按秩合并的并查集即可。

既然单个查询可以二分, 那一定可以用整体二分的方法对所有查询进行二分。我们在二分值为 mid 时, 将所有编号小于等于 mid 的边连接起来, 然后再递归到 $[mid + 1, r]$ 区间。右区间递归结束后, 再撤销二分值为 mid 时连接的边, 然后递归 $[l, mid]$ 。

如果连接操作是 $fa[x] = y, sz[y] = sz[y] + sz[x]$, 那么撤销操作的时候, 如果只有 $fa[x] = x, sz[y] = sz[y] - sz[x]$ 是不够的, 需要从 y 开始不断向上访问, 对于访问到的每一个节点都减去 $sz[x]$, 如此才能保证撤销操作的正确性。

```

1 #include <bits/stdc++.h>
2 #define mem(a,b) memset(a,b,sizeof a);
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define per(i,a,b) for(int i = a; i >= b; i--)
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 typedef long long ll;
7 typedef double db;
8 const db EPS = 1e-9;
9 const int N = 1e6+100;
10 using namespace std;
11
12 void dbg() {cout << "\n";}
13 template<typename T, typename... A> void dbg(T a, A... x) {cout << a << ' '; dbg(x...)}
14 #define logs(x...) {cout << #x << " -> "; dbg(x);}
15
16 int n,m,Q,fa[N],sz[N],ans[N];
17 struct Node {int x,y,k,id,h1,h2;} q[N],q1[N],q2[N];
18
```

```

19 int find(int x) {return x == fa[x] ? x : find(fa[x]);}
20 int calc(int x,int y){
21     int fx = find(x), fy = find(y);
22     return fx == fy ? sz[fx] : (sz[fx] + sz[fy]);
23 }
24 pair<int,int> merge(int x,int y){
25     int fx = find(x), fy = find(y);
26     if(fx == fy) return make_pair(-1,-1);
27     if(sz[fx] < sz[fy]){
28         fa[fx] = fy, sz[fx] += sz[fx];
29         return make_pair(fx,fy);
30     }
31     else{
32         fa[fy] = fx, sz[fx] += sz[fy];
33         return make_pair(fy,fx);
34     }
35 }
36 void Delete(int x,int y) {
37     fa[x] = x;
38     while(y){
39         sz[y] -= sz[x];
40         if(y == fa[y]) break;
41         y = fa[y];
42     }
43 }
44
45 void solve(int l,int r,int L,int R){
46     if(l > r || L > R) return;
47     if(l == r){
48         rep(i,L,R) if(q[i].k) ans[q[i].id] = l;
49         return;
50     }
51     int mid = (l+r)>>1, cnt1 = 0, cnt2 = 0;
52     // logs(mid,L,R);
53     rep(i,L,R){
54         if(q[i].k){ //查询
55             int tmp = calc(q[i].x,q[i].y);
56             if(q[i].k <= tmp) q1[++cnt1] = q[i];
57             else q2[++cnt2] = q[i];
58         }
59     else{
60         if(q[i].id <= mid){
61             pair<int,int> tmp = merge(q[i].x,q[i].y);
62             q1[++cnt1] = q[i];
63             q1[cnt1].h1 = tmp.first; q1[cnt1].h2 = tmp.second;
64         }
65         else q2[++cnt2] = q[i];
66     }
67 }
68 rep(i,1,cnt1) q[L+i-1] = q1[i];
69 rep(i,1,cnt2) q[L+cnt1+i-1] = q2[i];
70 solve(mid+1,r,L+cnt1,R);
71 rep(i,1,cnt1) if(!q[L+i-1].k) Delete(q[L+i-1].h1,q[L+i-1].h2);
72 solve(l,mid,L+cnt1-1);
73 }
74
75 int main()
76 {
77     int cnt = 0;

```

```

78     scanf("%d%d",&n,&m);
79     rep(i,1,n) fa[i] = i, sz[i] = 1;
80     rep(i,1,m){
81         int x,y; scanf("%d%d",&x,&y);
82         if(merge(x,y).first == -1) continue;
83         q[++cnt] = {x,y,0,i,0,0};
84         // logs(i);
85     }
86     scanf("%d",&Q);
87     rep(i,1,Q){
88         int x,y,z; scanf("%d%d%d",&x,&y,&z);
89         q[++cnt] = {x,y,z,i,0,0};
90     }
91     rep(i,1,n) fa[i] = i, sz[i] = 1;
92     solve(0,m+1,1,cnt);
93     rep(i,1,Q) printf("%d\n",ans[i]);
94     return 0;
95 }
```

4.26 分块

题意：长度为 n 的序列，每个点都有一个颜色，一共有 C 个颜色，支持两种操作，第一种给出 $l \ r \ x$ ，将区间 $[l, r]$ 全部染成 x ，第二种给出一个 x ，询问 x 颜色一共在序列中出现了多少次。

所有操作结束后，还要查询每种颜色出现的次数，给出最多出现的次数。 $(1 \leq n, C, m \leq 10^5)$

思路：非常明显的区间推平操作，应该使用珂朵莉树进行解决，即用 set 维护每一块相同区域。

老司机树是可以的，但是并没有采用，还是使用了分块的方法，虽然 T 的飞起... 一开始的想法是给每个块记一个 map ，标记每个块中各个数字出现的次数，如果整个块被赋值了，就 $map.clear()$ 。但是 T 的不行，第 63 个点怎么也过不去...

后来采用了完全的暴力分块，即每个块打个 $lazy$ ，如果 $lazy$ 不为 0，则整个块中所有点大小均为 $lazy$ 。如果 $lazy$ 为 0，则暴力修改、查询。那么如何分析这个做法的时间复杂度呢？

一共两种操作，修改操作每次最多涉及 \sqrt{n} 个块，如果块被完全覆盖， $O(1)$ 打上标记，如果块只被部分覆盖，下放标记 $O(\sqrt{n})$ ，修改点值 $O(\sqrt{n})$ ，由于每次最多只有 2 个块被部分覆盖，最多 \sqrt{n} 个块被完全覆盖，因此修改操作总时间复杂度为 $O(\sqrt{n})$ 。

而对于查询操作，如果该块被打上了标记，则查询复杂度为 $O(1)$ ，如果没有打上标记，则直接暴力查询，时间复杂度为 $O(\sqrt{n})$ 。此题是修改与查询次数一致，而修改操作每次最多使得 2 个块标记消失。因此可以发现，我们可以故意设置数据使得此种做法 TLE，即利用修改操作使得所有块的 $lazy$ 标记消失，然后每次查询颜色时就会遍历所有块，可以将时间复杂度卡成 $O(n^2)$ 。（不过此题数据较水，此种方法也可以过，而且很快，390 ms 即可通过...）

因此正确的做法是当块没有 $lazy$ 标记时，为了避免直接暴力询问，我们对于每个块记录一个 $unordered_map$ ，查询复杂度为 $O(1)$ 。修改复杂度会变成 $O(\sqrt{n} * log(n))$ ，查询复杂度为 $O(\sqrt{n})$ ，但是避免了被卡的可能。（790 ms 可以通过）

总结：对于一些不太常见的数据结构问题，一定要尝试考虑分块，因为分块方法用途很多，而且算法核心是暴力。

而对于分块算法，一定要多多尝试，你可能觉得这样分块会被你故意构造的一组数据卡成 $O(n^2)$ ，但是出题人未必就考虑到了你的这种做法，因此一定要多尝试，不要只去尝试那些 $O(n * log n * \sqrt{n})$ 的算法。当然如果能直接想到 $O(\sqrt{n})$ 的算法则是最好的！

```

1 #include <bits/stdc++.h>
2 #define mem(a,b) memset(a,b,sizeof a);
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
```

```

4 #define per(i,a,b) for(int i = a; i >= b; i--)
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 typedef long long ll;
7 typedef double db;
8 const int N = 1e5+10;
9 const db EPS = 1e-9;
10 using namespace std;
11
12 void dbg() {cout << "\n";}
13 template<typename T, typename... A> void dbg(T a, A... x) {cout << a << ' ' ; dbg(x...)}
14 #define logs(x...) {cout << #x << " -> "; dbg(x);}
15
16 int n,m,a[N],L[N],R[N],pos[N],cnt,sz,lazy[N],num[N];
17 unordered_map<int,int> mp[1010];
18
19 void change(int l,int r,int x){
20     rep(i,pos[l],pos[r]){
21         if(L[i] >= l && R[i] <= r){
22             // mp[i].clear();
23             // mp[i][x] = R[i]-L[i]+1;
24             lazy[i] = x;
25         }
26         else{
27             if(lazy[i]){
28                 rep(j,L[i],R[i]) a[j] = lazy[i];
29                 lazy[i] = 0;
30             }
31             rep(j,max(l,L[i]),min(R[i],r)){
32                 // mp[i][a[j]]--;
33                 // mp[i][x]++;
34                 a[j] = x;
35             }
36         }
37     }
38 }
39
40 int ask(int x){
41     int ans = 0;
42     rep(i,1,cnt){
43         if(lazy[i]){
44             if(lazy[i] == x) ans += R[i]-L[i]+1;
45         }
46         else{
47             rep(j,L[i],R[i])
48                 if(a[j] == x) ans++;
49         }
50     }
51     return ans;
52 }
53
54 int main()
55 {
56     int h; scanf("%d%d%d",&n,&h,&m);
57     sz = sqrt(n);
58     cnt = n/sz;
59     if(n%sz != 0) cnt++;
60     rep(i,1,cnt){
61         L[i] = sz*(i-1)+1;

```

```
62     R[i] = min(sz*i,n);
63     mp[i][1] = R[i]-L[i]+1;
64 }
65 rep(i,1,n) pos[i] = ((i-1)/sz)+1;
66 rep(i,1,n) a[i] = 1;
67
68 while(m--){
69     ll p,x,a,b;
70     scanf("%lld%lld%lld%lld",&p,&x,&a,&b);
71     ll s = ask(p);
72     ll m1 = (a+s*s)%n;
73     ll m2 = (a+(s+b)*(s+b))%n;
74     change(min(m1,m2)+1,max(m1,m2)+1,x);
75 }
76 rep(i,1,cnt)
77     if(lazy[i]){
78         rep(j,L[i],R[i]) a[j] = lazy[i];
79         lazy[i] = 0;
80     }
81 rep(i,1,n)
82     num[a[i]]++;
83     int maxx=0;
84     rep(i,1,h)
85         maxx = max(maxx,num[i]);
86     printf("%d\n",maxx);
87     return 0;
88 }
```

5 图论

5.1 最短路

5.1.1 Dijkstra

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <string>
5 #include <queue>
6 #include <algorithm>
7 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
8 #define rep(i,a,b) for(int i = a; i <= b; i++)
9 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
10 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
11 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
12 << " , " << z1 << ":" << z2 << endl;
13 typedef long long ll;
14 typedef double db;
15 const int N = 5*1e6+100;
16 const int M = 5*1e6+100;
17 const ll inf = 5*1e16;
18 const db EPS = 1e-9;
19 using namespace std;
20
21 struct Edge{
22     int to,next;
23     ll w;
24 }e[M];
25 int head[N],tot,n,vis[N];
26 ll c[N],dis[N];
27 void add(int x,int y,ll w){
28     e[++tot].to = y, e[tot].next = head[x], e[tot].w = w, head[x] = tot;
29 }
30
31 priority_queue< pair<ll,int> > q;
32
33 ll dijkstra(int s)
34 {
35     while(q.size()) q.pop();
36     memset(vis,0,sizeof vis);
37     rep(i,0,2*n+2) dis[i] = inf;
38     dis[s] = 0;
39     q.push(make_pair(0ll,s));
40     while(q.size())
41     {
42         int x = q.top().second;
43         q.pop();
44         if(vis[x]) continue;
45         vis[x] = 1; //只有从队列中弹出时才是更新好了这个点，此时才可以更新vis，不能在加入的时候就更新vis
46         for(int i = head[x]; i; i = e[i].next)
47         {
48             int y = e[i].to;
49             ll z = e[i].w;
50             if(dis[y] > (ll)(dis[x] + z)){

```

```

51         dis[y] = dis[x]+z;
52         q.push(make_pair(-dis[y],y));
53     }
54 }
55 if(dis[n*2+2] == inf) return -1;
56 return dis[2*n+2];
57 }
58 }
59
60 int main()
61 {
62     __; cin >> n; tot = 1;
63     rep(i,1,n) cin >> c[i];
64     string b1,b2;
65     int r1,r2;
66     rep(i,1,n){
67         string s1,s2;
68         cin >> s1; s2 = s1;
69         reverse(s2.begin(),s2.end());
70         if(i == 1){
71             b1 = s1, b2 = s2;
72             r1 = i, r2 = i+n;
73             add(2*n+1,r1,0);
74             add(2*n+1,r2,c[1]);
75         }
76         else{
77             if(b1 <= s1) add(r1,i,0ll);
78             if(b1 <= s2) add(r1,i+n,c[i]);
79             if(b2 <= s1) add(r2,i,0ll);
80             if(b2 <= s2) add(r2,i+n,c[i]);
81             b1 = s1, b2 = s2;
82             r1 = i, r2 = i+n;
83         }
84     }
85     add(n,2*n+2,0);
86     add(2*n,2*n+2,0);
87     ll x1 = dijkstra(2*n+1);
88     cout << x1 << endl;
89     return 0;
90 }
```

5.1.2 Dijkstra 记录路径

```

1 #include <cstdio>
2 #include <iostream>
3 #include <stack>
4 #include <cstring>
5 using namespace std;
6 const int N = 100+10;
7 const int inf = 0x3fffffff;
8
9 int e[N][N],dis[N],path[N],vis[N];
10 int n,m;
11
12 void dijkstra(int x)
13 {
14     //初始化dis数组
15     for(int i = 1;i <= n;i++)

```

```

16     dis[i] = e[x][i];
17
18     memset(vis,0,sizeof(vis));
19     vis[x] = 1;
20     for(int i = 1;i <= n-1;i++)
21     {
22         int minn = inf;
23         int u;
24         for(int j = 1;j <= n;j++)
25         {
26             if(!vis[j] && dis[j] < minn)
27             {
28                 minn = dis[j];
29                 u = j;
30             }
31         }
32         vis[u] = 1;
33         for(int j = 1;j <= n;j++)
34         {
35             if(e[u][j] < inf)
36             {
37                 if(e[u][j]+dis[u] < dis[j])
38                 {
39                     dis[j] = e[u][j]+dis[u];
40                     path[j] = u; //记录该点是被哪一个点松弛的
41                 }
42             }
43         }
44     }
45 }
46
47 void print(int x,int y) //x为起点, y为终点
48 {
49     stack<int> q;
50     int tmp = y;
51     while(path[y] != -1)
52     {
53         q.push(y);
54         y = path[y];
55     }
56     q.push(y);
57
58     //打印路径
59     printf("%d=>%d, length:%d, path: %d ",x,tmp,dis[tmp],x);
60     while(!q.empty()) //先进后出,获得正序
61     {
62         printf("%d ",q.top());//打印堆的头节点
63         q.pop(); //将堆的头节点弹出
64     }
65     printf("\n");
66 }
67
68 int main()
69 {
70     cin>>n>>m;
71     memset(path,-1,sizeof(path));
72     //把图初始化
73     for(int i = 1;i <= n;i++)
74     {

```

```

75         for(int j = 1;j <= n;j++)
76     {
77         if(i == j) e[i][j] = 0;
78         else e[i][j] = inf;
79     }
80 }
81 //读入边
82 for(int i = 1;i <= m;i++)
83 {
84     int t1,t2,t3;
85     cin>>t1>>t2>>t3;
86     e[t1][t2] = t3;
87 }
88 dijkstra(1);
89 cout<<"n:"<<n<<endl;
90 print(1,n);
91 return 0;
92 }
```

5.1.3 分层图

题意：

给一张有向图，给一个 k，可以将图中 k 条边的边权变为 0，求从 1->n 的最短路径

解法：

由于 $k \leq 10$ ，由此考虑分层图做法

给每个点建 10 个分身， $dis[i][j]$ 表示从城市 1 到达城市 i ，一共改了 j 条边的最短路径

其余操作和正常最短路求法一致，更新的时候有两种更新方法。假设队列弹出的节点是 $dis[i][j]$ ， y 为 i 可达到的点，则用 $dis[i][j]$ 去更新 $dis[y][j]$ 和 $dis[y][j+1]$ ，相当于扩点操作，其余操作和 dij 模板一致
如此操作即可求出答案

```

1 #include <cstdio>
2 #include <iostream>
3 #include <algorithm>
4 #include <cstring>
5 #include <queue>
6 #define rep(i,a,b) for(int i = a;i <= b;i++)
7 using namespace std;
8 const int N = 1e5+100;
9 const int M = 5*1e5+10;
10 typedef long long ll;
11 const ll inf = 1e15;
12
13 struct Edge{
14     int to,next,w;
15 }e[M];
16 int n,m,k,head[N],tot;
17 ll dis[N][15];
18 int vis[N][15];
19
20 void init()
21 {
22     tot = 1;
23     rep(i,0,n) head[i] = 0;
24 }
25
26 void add(int x,int y,int z)
```

```

27 {
28     e[++tot].to = y, e[tot].next = head[x], head[x] = tot, e[tot].w = z;
29 }
30
31 struct Point{
32     ll ans;
33     int u,cnt;
34 }thp;
35
36 bool operator < (Point a,Point b)
37 {
38     return a.ans > b.ans;
39 }
40
41 priority_queue<Point> q;
42
43 void dijkstra(int s)
44 {
45     while(q.size()) q.pop();
46     thp.ans = 0, thp.u = s, thp.cnt = 0;
47     q.push(thp);
48     rep(i,1,n)
49         rep(j,0,k) vis[i][j] = 0;
50     rep(i,1,n)
51         rep(j,0,k) dis[i][j] = inf;
52     dis[s][0] = 0;
53     while(q.size())
54     {
55         thp = q.top();
56         q.pop();
57         int u = thp.u;
58         int x = thp.cnt;
59         if(vis[u][x]) continue;
60         vis[u][x] = 1;
61         for(int i = head[u]; i ; i = e[i].next)
62         {
63             int y = e[i].to;
64             if(!vis[y][x]){
65                 if(dis[y][x] > dis[u][x]+e[i].w){
66                     dis[y][x] = dis[u][x] + e[i].w;
67                     thp.u = y, thp.ans = dis[y][x], thp.cnt = x;
68                     q.push(thp);
69                 }
70             }
71             if(!vis[y][x+1] && (x < k)){
72                 if(dis[y][x+1] > dis[u][x]){
73                     dis[y][x+1] = dis[u][x];
74                     thp.u = y, thp.ans = dis[y][x+1], thp.cnt = x+1;
75                     q.push(thp);
76                 }
77             }
78         }
79     }
80 }
81
82 int main()
83 {
84     int T;
85     scanf("%d",&T);

```

```

86     while(T--)
87     {
88         init();
89         scanf("%d%d%d",&n,&m,&k);
90         rep(i,1,m)
91         {
92             int x,y,z;
93             scanf("%d%d%d",&x,&y,&z);
94             add(x,y,z);
95         }
96         dijkstra(1);
97         printf("%lld\n",dis[n][k]);
98     }
99     return 0;
100 }
```

5.1.4 spfa

```

1 //最短路 —— spfa
2 #include <cstdio>
3 #include <iostream>
4 #include <queue>
5 #include <algorithm>
6 #include <cstring>
7 using namespace std;
8
9 const int maxn = 1000+10;
10 const int maxm = 4000+10;
11 const int inf = 0xffffffff;
12
13 struct edge{
14     int u,v,w;
15 }e[maxm];
16 int book[maxn],first[maxn],next[maxm];
17 //book用来标记      first[i] 指以节点i为出发点的边的编号
18 int dis[maxn];
19 int n,m;
20 queue<int>q;          //用以保存节点的队列
21
22 //x为要求从哪一个点出发
23 void spfa(int x)
24 {
25     memset(book,0,sizeof(book));
26     for(int i = 1;i <= n;i++) dis[i] = inf;    //距离赋成无穷大
27     while(!q.empty()) q.pop();
28     q.push(x);
29     book[x] = 1;
30     dis[x] = 0;
31     while(!q.empty())
32     {
33         int k = first[q.front()];
34         while(k!=-1) //遍历图
35         {
36             if(dis[e[k].v] > dis[e[k].u]+e[k].w) //用队列中的边去松弛两端的点
37             {
38                 dis[e[k].v] = dis[e[k].u]+e[k].w; //松弛成功
39                 if(book[e[k].v] == 0) //把松弛成功的点加入队列
40                 {
```

```

41             q.push(e[k].v);
42         }
43     }
44     k = next[k]; //遍历这个点的下一条边
45   }
46   book[q.front()] = 0; //将这个点出队列
47   q.pop();
48 }
49 }
50 }
51
52 int main()
53 {
54     cin>>m>>n;
55     memset(first,-1,sizeof(first));
56     for(int i = 1;i <= m;i++)
57     {
58         scanf("%d%d%d",&e[i].u,&e[i].v,&e[i].w);
59         //双向边的保存
60         e[m+i].u = e[i].v;
61         e[m+i].v = e[i].u;
62         e[m+i].w = e[i].w;
63         next[i] = first[e[i].u]; //表示e[i].u这个点当前这条边的上一条边
64         first[e[i].u] = i; //表示e[i].u这个点现在的边
65         next[m+i] = first[e[i+m].u];
66         first[e[i+m].u] = i+m;
67     }
68     spfa(1);
69     cout<<dis[n]<<endl;
70     return 0;
71 }
```

5.1.5 spfa 判断负环

```

1 //spfa判断负环
2 #include <cstdio>
3 #include <iostream>
4 #include <queue>
5 #include <cstring>
6 using namespace std;
7 const int maxn = 105;
8 const int inf = 0x3fffffff;
9
10 bool p[maxn][maxn];
11 bool f[maxn][maxn];
12 int n, pow[maxn], book[maxn], cnt[maxn], dis[maxn];
13
14 void Floyd() //判连通
15 {
16     for(int k = 1;k <= n;k++)
17         for(int i = 1;i <= n;i++)
18             for(int j = 1;j <= n;j++)
19                 f[i][j] = f[i][j] || (f[i][k] && f[k][j]);
20 }
21
22 bool spfa(int x)
23 {
24     memset(book,0,sizeof(book));
```

```

25     memset(cnt,0,sizeof(cnt));
26     memset(dis,0,sizeof(dis));
27     queue<int>q;
28     q.push(x);
29     book[x] = 1;
30     dis[x] = 100;
31     cnt[x]++;
32     while(!q.empty())
33     {
34         x = q.front();
35         q.pop();
36         if(cnt[x] > n) continue; //如果一个点进入队列次数>n, 就不能用这个点去松弛, 否则会TLE
37         for(int i = 1;i <= n;i++)
38         {
39             if(p[x][i] && dis[i] < pow[x]+dis[x] && book[i] == 0 && (pow[x]+dis[x]) >
40             0)
41             {
42                 q.push(i);
43                 dis[i] = pow[x]+dis[x];
44                 cnt[i]++;
45                 book[i] = 1;
46                 if(cnt[i] >= n) dis[i] = inf; //判断负环
47             }
48         }
49     }
50     return dis[n] > 0;
51 }
52
53 int main()
54 {
55     while(~scanf("%d",&n) && n!=-1)
56     {
57         memset(p,0,sizeof(p));
58         memset(f,0,sizeof(f));
59         for(int i = 1;i <= n;i++)
60         {
61             int k;
62             scanf("%d%d",&pow[i],&k);
63             for(int j = 1;j <= k;j++)
64             {
65                 int tmp;
66                 scanf("%d",&tmp);
67                 p[i][tmp] = true;
68                 f[i][tmp] = true;
69             }
70         }
71         Floyd();
72         if(spf(1) && f[1][n])
73             printf("winnable\n");
74         else
75             printf("hopeless\n");
76     }
77     return 0;
78 }
```

5.1.6 最短路计数问题

题意: 一个高度为 h 的电梯, 初始位置在第一层。电梯有四个按钮。

1. 向上移动 a 格
2. 向上移动 b 格
3. 向上移动 c 格
4. 返回第一层

问 $1 \leq h \leq 10^{18}$, $1 \leq a, b, c \leq 100000$

思路: 观察数据范围可以发现, $a b c$ 数据范围比较小, 可以从此处着手。

不难发现, 如果 2 可以达到, 则 $2+a$ 、 $2+2*a$ 、 $2+3*a\dots$ 均可到达。即我们以 a 作为基底, 如果 x ($x < a$) 可以到达, 则 y ($y \% a = x$) 也可到达。

因此问题转化为对于 x ($x < a$) 来说, 最小的可达的 y 是多少? ($y \% a = x$) 既然问题变成了求最小, 那么不难想到使用最短路来解决这个问题。

我们枚举 x ($0 \leq x < a$), 令 x 与 $(x+b)\%a$ 和 $(x+c)\%a$ 连边, 边权分别为 b 、 c 。然后起点为 1 , 即可求出到达所有 x 时的最小值。

求取答案时就可以枚举 x , 然后求 $dis[x]$ h 中有多少个 $mod a = x$ 的值即可。

```

1 #include <bits/stdc++.h>
2 #define rep(i,a,b) for(int i = a; i <= b; i++)
3 typedef long long ll;
4 typedef double db;
5 const int N = 1e5+100;
6 const int M = 1e6+100;
7 const ll inf = 1e15;
8 const db EPS = 1e-9;
9 using namespace std;
10
11 ll h,a,b,c,dis[N];
12 int head[N],tot,vis[N];
13 struct Node{
14     int to,next;
15     ll w;
16 }e[M];
17
18 void init() {tot = 1;}
19
20 void add(int x, int y, ll w){
21     e[++tot].to = y, e[tot].next = head[x], head[x] = tot, e[tot].w = w;
22 }
23
24 void dijkstra(int s){
25     priority_queue<pair<ll,int> > q;
26     while(q.size()) q.pop();
27     rep(i,0,a-1) dis[i] = inf, vis[i] = 0;
28     dis[s%a] = 1;
29     q.push(make_pair(-dis[s%a],s%a));
30     while(q.size()){
31         int x = q.top().second; q.pop();
32         if(vis[x]) continue;
33         vis[x] = 1;
34         for(int i = head[x]; i; i = e[i].next){
35             int y = e[i].to;
36             if(dis[y] > dis[x] + e[i].w){
37                 dis[y] = dis[x] + e[i].w;
38                 q.push(make_pair(-dis[y],y));
39             }
40         }
41     }
42 }
43

```

```

44 int main()
45 {
46     scanf("%lld%lld%lld%lld", &h, &a, &b, &c);
47     init();
48     rep(i, 0, a-1){
49         add(i, (i+b)%a, b);
50         add(i, (i+c)%a, c);
51     }
52     dijkstra(1);
53     ll ans = 0;
54     rep(i, 0, a-1){
55         if(dis[i] == inf || dis[i] > h) continue;
56         ans += 1ll + (h-dis[i])/a;
57     }
58     printf("%lld\n", ans);
59     return 0;
60 }
```

5.2 最小生成树

```

1 //最小生成树 ——Kruscal
2 #include <cstdio>
3 #include <iostream>
4 #include <algorithm>
5 #include <cstring>
6 using namespace std;
7
8 const int maxn = 1000+10;
9 const int maxm = 4000+10;
10
11 struct edge{
12     int u,v,w;
13 }e[maxn];
14 int n,m,sum;    //sum为总共的路程
15 int p[maxn]; //记录每个点的祖先
16
17 bool cmp(edge e1,edge e2)
18 {
19     return e1.w < e2.w;
20 }
21
22 int find(int x)
23 {
24     return p[x]==x?x:p[x]=find(p[x]);
25 }
26
27 bool merge(int x,int y)
28 {
29     int t1 = find(x);
30     int t2 = find(y);
31     if(t1!=t2)
32     {
33         p[t1] = t2;
34         return true;
35     }
36     return false;
37 }
38
```

```

39 int main()
40 {
41     scanf("%d%d",&n,&m);
42     for(int i = 1;i <= m;i++)
43     {
44         scanf("%d%d%d",&e[i].u,&e[i].v,&e[i].w);
45     }
46     sort(e+1,e+m+1,cmp);
47     for(int i = 1;i <= n;i++) p[i] = i;
48
49 //Kruskal算法核心部分
50 int count = 0; //最小生成树已经有几条边了
51 sum = 0;
52 int maxx = -1;
53 int minn = 10010;
54 for(int i = 1;i <= m;i++)
55 {
56     //判断一条边的两个顶点是否已经连通，即判断是否已在同一个集合中
57     if(merge(e[i].u,e[i].v))
58     {
59         count++;
60         sum = sum+e[i].w;
61         maxx = max(maxx,e[i].w);
62         minn = min(minn,e[i].w);
63     }
64     if(count == n-1) break;
65 }
66 cout<<sum<<endl;
67 cout<<maxx-minn<<endl;
68
69 return 0;
70 }

```

5.3 最近公共祖先

5.3.1 LCA (dfs 版本)

```

1 #include <cstdio>
2 #include <iostream>
3 #include <algorithm>
4 #include <cstring>
5 #include <queue>
6 #include <cmath>
7 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
8 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
9 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
10 #define rep(i,a,b) for(int i = a;i <= b;i++)
11 using namespace std;
12 const int N = 5*1e5+100;
13
14 int f[N][25],d[N],dis[N]; //f[x][k]:表示点x向根节点走2^k步到达的点
15 //d[x]:表示点x在图中的深度 dis[x]:表示根节点到点x的距离
16 struct Edge{
17     int to,next;
18 }e[2*N]; //树的边为n-1, 此处是因为无向边, 所以正反各一条
19 int head[N],tot,t,n,m,s; //n个点, m次询问
20

```

```

21 void init()
22 {
23     tot = 1;
24     memset(head, 0, sizeof head);
25     memset(d, 0, sizeof d);
26 }
27
28 void add(int x, int y)
29 {
30     e[++tot].to = y; e[tot].next = head[x]; head[x] = tot;
31 }
32
33 void dfs(int u, int fa)
34 {
35     d[u] = d[fa] + 1; f[u][0] = fa;
36     for (int i = 1; (1 << i) <= d[u]; i++)
37         f[u][i] = f[f[u][i - 1]][i - 1];
38     for (int i = head[u]; i; i = e[i].next) {
39         int v = e[i].to;
40         if (v != fa) dfs(v, u);
41     }
42 }
43
44 int lca(int x, int y)
45 {
46     if (d[x] > d[y]) swap(x, y);
47     for (int i = t; i >= 0; i--)
48         if (d[f[y][i]] >= d[x]) y = f[y][i]; // 往上追溯，直至y和x位于同一深度
49     if (x == y) return x; // 如果已经找到了，就返回x
50     for (int i = t; i >= 0; i--)
51         if (f[x][i] != f[y][i]) x = f[x][i], y = f[y][i]; // x和y同时往上走，一直到x和y恰好为
52         lca的子节点
53     return f[x][0]; // x和y共同的根节点就是lca
54 }
55
56 int main()
57 {
58     scanf("%d%d%d", &n, &m, &s);
59     t = (int)(log(n) / log(2)) + 1;
60     init();
61     rep(i, 1, n - 1) {
62         int x, y;
63         scanf("%d%d", &x, &y);
64         add(x, y), add(y, x);
65     }
66     dfs(s, 0);
67     for (int i = 1; i <= m; i++) {
68         int x, y;
69         scanf("%d%d", &x, &y);
70         printf("%d\n", lca(x, y));
71     }
72 }
73 }
```

5.3.2 LCA (bfs 版本)

```
1 #include <cstdio>
```

```

2 #include <iostream>
3 #include <algorithm>
4 #include <cstring>
5 #include <queue>
6 #include <cmath>
7 #define rep(i,a,b) for(int i = a;i <= b;i++)
8 using namespace std;
9 const int N = 50010;
10
11 int f[N][20],d[N],dis[N]; //f[x][k]:表示点x向根节点走2^k步到达的点
12 //d[x]:表示点x在图中的深度 dis[x]:表示根节点到点x的距离
13 struct Edge{
14     int to,next,w;
15 }e[2*N]; //树的边为n-1, 此处是因为无向边, 所以正反各一条
16 int head[N],tot,t,n,m; //n个点, m次询问
17
18 void init()
19 {
20     tot = 1;
21     memset(head,0,sizeof head);
22     memset(d,0,sizeof d);
23 }
24
25 void add(int x,int y,int z)
26 {
27     e[++tot].to = y; e[tot].next = head[x]; e[tot].w = z; head[x] = tot;
28 }
29
30 void bfs()
31 {
32     queue<int> q;
33     while(q.size()) q.pop();
34     q.push(1); d[1] = 1; dis[1] = 0; //把1当做树根
35     while(q.size())
36     {
37         int x = q.front(); q.pop();
38         for(int i = head[x]; i ;i = e[i].next){
39             int y = e[i].to;
40             if(d[y]) continue;
41             d[y] = d[x]+1;
42             dis[y] = dis[x]+e[i].w; //dist[y]:从1到y的距离
43             f[y][0] = x; //y走2^0步到达x
44             for(int j = 1; j <= t;j++)
45                 f[y][j] = f[f[y][j-1]][j-1];
46             q.push(y);
47         }
48     }
49 }
50
51 /*
52     这里有一个非常容易错的点, 因为每一个点的父节点初始化为0, 所以如果你的点从0-n编号的话, 一定要注意
53     bfs的时候将0作为根节点
54     否则如果用1作为根节点的话, 你会发现1的父节点是0, 那么程序就开始出错了!
55
56     所以更好的方法是以后给点编号的时候, 统一用1-n编号, 不要用0编号, 否则极易出错
57
58     从1-n编号的话, 可以从任意起点拉起一棵树, 但是要注意题目中n的最小值
59 */

```

```

60 int lca(int x,int y)
61 {
62     if(d[x] > d[y]) swap(x,y);
63     for(int i = t; i >= 0; i--)
64         if(d[f[y][i]] >= d[x]) y = f[y][i]; //往上追溯，直至y和x位于同一深度
65     if(x == y) return x; //如果已经找到了，就返回x
66     for(int i = t; i >= 0; i--)
67         if(f[x][i] != f[y][i]) x = f[x][i], y = f[y][i]; //x和y同时往上走，一直到x和y恰好为
68         lca的子节点
69     return f[x][0]; //x和y共同的根节点就是lca
70 }
71 int main()
72 {
73     int T;
74     scanf("%d",&T);
75     while(T--)
76     {
77         scanf("%d%d",&n,&m);
78         t = (int)(log(n)/log(2))+1;
79         init();
80         rep(i,1,n-1){
81             int x,y,z;
82             scanf("%d%d%d",&x,&y,&z);
83             add(x,y,z), add(y,x,z);
84         }
85         bfs();
86         for(int i = 1;i <= m;i++)
87         {
88             int x,y;
89             scanf("%d%d",&x,&y);
90             printf("%d\n",dis[x]+dis[y]-2*dis[lca(x,y)]);
91         }
92     }
93     return 0;
94 }
```

5.4 拓扑排序

题意：

一个 $n \times n$ 的网格图，每行每列各涂一次，每行每列均有 26 种涂法，输出涂色方案。

思路：

可以发现本题有个性质，最后涂的行或列，一定只有一个元素。

因此开一个结构体记录每行每列，各个颜色的涂色情况，以及该行或列一共有几种不同的颜色。

然后类似于跑拓扑排序，将只有一种颜色的行列加入队列，再依次撤销。

将新的颜色数变为 1 的行列加入队列，即可完成本题。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <queue>
5 #include <algorithm>
6 #define rep(i,a,b) for(int i = a; i <= b; i++)
7 #define LOG1(x) cout << "x: " << x << endl;
8 #define LOG2(x,y) cout << "x: " << x << ", y: " << y << endl;
9 typedef long long ll;
```

```

10  typedef double db;
11  const db EPS = 1e-9;
12  using namespace std;
13  const int N = 3000+100;
14
15  int n;
16  char mp[N][N];
17  struct Node{
18      int total;
19      int num[30];
20  }t[2*N];
21  int ans[2*N][4],tot;
22  int vis[2*N];
23
24  void solve()
25  {
26      tot = 0;
27      queue<int> q;
28      while(q.size()) q.pop();
29      rep(i,1,n*2) {
30          if(t[i].total == 1) q.push(i);
31          else if(t[i].total == 0) ans[++tot][1] = i, ans[tot][2] = 0, vis[i] = 1;
32      }
33      while(q.size()){
34          int x = q.front();
35          q.pop();
36          if(vis[x]) continue;
37          t[x].total = 0;
38          ans[++tot][1] = x;
39          vis[x] = 1;
40          rep(i,0,27)
41              if(t[x].num[i] >= 1) ans[tot][2] = i;
42          int pp = ans[tot][2];
43          if(x <= n){
44              rep(i,n+1,n*2){
45                  if(mp[x][i-n]-'a' != pp) continue;
46                  t[i].num[pp]--;
47                  if(t[i].num[pp] == 0){
48                      t[i].total--;
49                      if(t[i].total == 1) q.push(i);
50                  }
51              }
52          }
53          else{
54              rep(i,1,n){
55                  if(mp[i][x-n]-'a' != pp) continue;
56                  t[i].num[pp]--;
57                  if(t[i].num[pp] == 0){
58                      t[i].total--;
59                      if(t[i].total == 1) q.push(i);
60                  }
61              }
62          }
63      }
64      for(int i = 2*n; i >= 1; i--)
65      {
66          int x = ans[i][1], y = ans[i][2];
67          if(x <= n) printf("h %d ",x);
68          else printf("v %d ",x-n);

```

```

69         printf("%c\n", 'a'+y);
70     }
71 }
72
73 int main()
74 {
75     scanf("%d",&n);
76     rep(i,1,n) scanf("%s",mp[i]+1);
77     rep(i,1,n)
78         rep(j,1,n){
79             if(mp[i][j] == '?') continue;
80             int x = mp[i][j]-'a';
81             t[i].num[x]++;
82             if(t[i].num[x] == 1) t[i].total++;
83             t[j+n].num[x]++;
84             if(t[j+n].num[x] == 1) t[j+n].total++;
85         }
86     solve();
87     return 0;
88 }
```

5.5 欧拉路

题意：

给出 m 条边，求出一条欧拉路，起点任意，终点任意，每条边只经过一次。要求给出的欧拉路字典序最小。 $(1 \leq m \leq 1024, 1 \leq n \leq 500)$

思路：

先总结一下‘有向图、无向图求欧拉路与欧拉回路的性质’。

无向图：有且仅有两个点度数为奇数则有欧拉路，所有点度数均为偶数则有欧拉回路。

有向图：所有点入度 = 出度则有欧拉回路。有且仅有两个点入度不等于出度，且起点出度比入度大 1，终点入度比出度大 1 则有欧拉路。

算法——Hierholzer (解决无向图、有向图、欧拉路、欧拉回路问题)

选一个点 x 为起点，存在边 $\langle x, y \rangle$ ，则删去边 $\langle x, y \rangle$ ，若为无向图还需删除 $\langle y, x \rangle$ 。若无边可走，则将 x 加入结果栈。最后输出结构栈即可。

回到此题，要求找到字典序最小的欧拉路。因此选择一个编号最小的奇数点进行递归，递归过程中优先走字典序更小的点，可以用 *multiset* 维护。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <set>
5 #include <stack>
6 #include <algorithm>
7 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
8 #define rep(i,a,b) for(int i = a; i <= b; i++)
9 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
10 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
11 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
12     << " , " << z1 << ":" << z2 << endl;
12 typedef long long ll;
13 typedef double db;
14 const int N = 500+100;
```

```

15 const int M = 1e5+100;
16 const db EPS = 1e-9;
17 using namespace std;
18
19 multiset<int> st[N];
20 stack<int> stk;
21 int m,deg[N];
22
23 void dfs(int x){
24     while(st[x].size()){
25         int y = (*st[x].begin());
26         st[x].erase(st[x].begin());
27         st[y].erase(st[y].find(x));
28         dfs(y);
29     }
30     stk.push(x);
31 }
32
33 int main()
34 {
35     rep(i,0,500) st[i].clear();
36     while(stk.size()) stk.pop();
37     scanf("%d",&m);
38     rep(i,1,m){
39         int xx,yy; scanf("%d%d",&xx,&yy);
40         st[xx].insert(yy);
41         st[yy].insert(xx);
42         deg[xx]++, deg[yy]++;
43     }
44     int x = -1;
45     rep(i,0,500) if(deg[i]%2){x = i; break;}
46     if(x == -1) x = 1;
47     dfs(x);
48     while(stk.size()){
49         printf("%d\n",stk.top());
50         stk.pop();
51     }
52     return 0;
53 }
```

5.6 双连通分量

5.6.1 边双缩点以及求割边

求边双连通分量、割边 (e-DCC)

将每个 e-DCC 看成一个节点，进行缩点

求边双连通分量——即该分量中不存在割边

边双连通图的条件——当且仅当任意一条边都包含在至少一个简单环中

求边双连通分量的原理：

先求出无向图中所有的桥，把桥都删除后，无向图会分成若干个连通块，每一个连通块就是一个“边双连通分量”

```

1 #include <cstdio>
2 #include <iostream>
3 #include <algorithm>
4 #include <cstring>
5 #define rep(i,a,b) for(int i = a;i <= b;i++)
6 using namespace std;
```

```

7 const int N = 1e5+10, M = 5*1e5+10;
8
9 struct Edge{
10     int to,next;
11 }e[2*M],ec[2*M]; //双向边, ec为缩点之后的图【森林/树】
12 int head[N],dfn[N],low[N],headc[N]; //dfn:记录该点的dfs序           low:记录该点所能达到的最小dfn
13 //hc:缩点之后记录每一个点所连接的第一条边的边序
14 bool bridge[2*M]; //记录该边是否为割边
15 int tot,n,m,num,dcc,tc; //tot记录边序, num记录dfs序, dcc记录有多少个边双连通分量
16 //tc:缩点之后记录边序
17 int c[N]; //c[x]表示节点x所属的“边双连通分量”的编号
18
19 void init()
20 {
21     num = 0; tot = 1; dcc = 0;
22     rep(i,0,n) head[i] = dfn[i] = low[i] = c[i] = 0;
23     rep(i,0,2*m+2) bridge[i] = 0;
24 }
25
26 void initcc()
27 {
28     tc = 1; //缩点之后记录边序
29     // memset(headc,0,sizeof headc); //缩点之后记录每一个点所连接的第一条边的边序
30     rep(i,0,dcc) headc[i] = 0;
31 }
32
33 void add(int x,int y)
34 {
35     e[++tot].to = y; e[tot].next = head[x]; head[x] = tot;
36 }
37
38 void addc(int x,int y) //对缩点之后的图进行加边操作
39 {
40     ec[++tc].to = y; ec[tc].next = headc[x]; headc[x] = tc;
41 }
42
43 void tarjan(int x,int in_edge)
44 {
45     dfn[x] = low[x] = ++num;
46     for(int i = head[x]; i ; i = e[i].next)
47     {
48         int y = e[i].to;
49         if(!dfn[y]){
50             tarjan(y,i); //传入边(x,y)的序号
51             low[x] = min(low[x],low[y]);
52             if(low[y] > dfn[x]) //该边连接(x,y), y点无法连接x点上面的点, 因此该边是割边
53                 bridge[i] = bridge[i^1] = true;
54         }
55         else if(i != (in_edge^1)) //y点所连接的边 不能是(x,y)边的反向边
56             low[x] = min(low[x],dfn[y]);
57     }
58 }
59
60 void dfs(int x) //用于将图划分为多个边双连通分量
61 {
62     c[x] = dcc;
63     for(int i = head[x]; i ; i = e[i].next)
64     {

```

```

65     int y = e[i].to;
66     if(c[y] || bridge[i]) continue; //如果点y已经属于别的强连通分量，或者边i是割边，则
67     continue
68     dfs(y);
69 }
70
71 int main()
72 {
73     while(~scanf("%d%d",&n,&m))
74     {
75         init();
76         rep(i,1,m){
77             int x,y;
78             scanf("%d%d",&x,&y);
79             add(x,y); add(y,x);
80         }
81         rep(i,1,n)
82             if(!dfn[i]) tarjan(i,0); //将边序号传进去
83         rep(i,1,n)
84             if(!c[i]){ //如果i点未被标记过
85                 ++dcc;
86                 dfs(i);
87             }
88         initc(); //对缩点之后的图进行初始化 这里很重要！不要忘记！！！
89         for(int i = 2; i <= tot; i += 2){ //遍历所有边 2-tot
90             int x = e[i^1].to, y = e[i].to; //记录该边连接的两个端点
91             if(c[x] == c[y]) continue; //如果连接的两个点属于同一连通分量，则continue
92             //没必要用mp记录，不会重复，可以用ct记录有多少条边
93             addc(c[x],c[y]); //用连通的分量的编号来代表整个连通分量，以此来进行缩点
94             addc(c[y],c[x]);
95         }
96         printf("缩点之后的森林，点数%d, 边数%d(可能有重边)\n",dcc,tc/2);
97     /* printf("There are %d e-DCCs.\n",dcc);
98     rep(i,1,n){
99         printf("%d belongs to DCC %d.\n",i,c[i]); //输出每个点属于的e-DCC
100    }*/
101   /* for(int i = 2; i <= tot; i+=2)
102      if(bridge[i])
103          printf("%d %d\n",e[i^1].to,e[i].to); //该割边：(x,y)*/
104 }
105 return 0;
106 }
```

5.6.2 边双缩点 LCA

题意：

给出一个无向图， n 个点， m 条边，可能有重边与自环，也可能不连通。 q 组询问，每组询问给出 3 个点， u 、 v 、 w ，问是否存在两条路径不存在公共边，并且一条路径是 $v \rightarrow u$ ，另一条路径是 $w \rightarrow u$ ，存在输出 Yes，否则输出 No。 $(1 \leq n \leq 10^5, 0 \leq m \leq 2 * 10^5, 1 \leq q \leq 10^5)$

思路：

既然是无向图上求两条互不相交的路径，比较直接的想法就是先求出边双连通分量进行缩点，然后在树上进行考虑。

求出边双连通分量之后，假如 u 、 v 、 w 三点在同一个双连通分量中，则答案必定为 Yes。若 v 、 w 在同一个双连通分量中，而 u 在另一个双连通分量中，则答案必定为 No。若 v 或 w 和 u 在同一个双连通分量中，另一个点不在其中，则答案也为 Yes。考虑完了一个和两个双连通分量的情况之后，我们来考虑三个的情况。

假如 u, v, w 分属于三个不同的双连通分量中，则需要进行分类。*Yes* 的情况只有两种，第一种情况是 v 和 w 都在 u 子树中，即 $\text{lca}(v, w) = u$ 即可，如图 (1)。第二种情况是 v, w 中有一个在 u 的子树中，另一个则不在。对于这种情况，我们先求出 $\text{lca}(v, w) = y$ ，再求出 $x_1 = \text{lca}(u, w), x_2 = \text{lca}(u, v)$ ，则 x_1 与 x_2 中一定有一个为 y ，另一个为 u ，才能输出 *Yes*，否则输出 *No*。到此，这题分类讨论就结束了。

但是这一题还需要注意一些细节，因为图可能不连通，因此需要预先判断 u, v, w 三个点是否连通，如果不连通，直接输出 *No*。还有一个细节，因为图可能是个森林，因此需要对每一个树进行 *lca* 处理。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <cmath>
5 #include <queue>
6 #include <algorithm>
7 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
8 #define rep(i,a,b) for(int i = a; i <= b; i++)
9 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
10 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
11 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
12 << " , " << z1 << ":" << z2 << endl;
13 typedef long long ll;
14 typedef double db;
15 const int N = 2*1e5+100;
16 const int M = 5*1e5+100;
17 const db EPS = 1e-9;
18 using namespace std;
19
20 struct Edge{
21     int to,next;
22 }e[M],ec[M];
23 int n,m,q,head[N],dfn[N],low[N],headc[N],dis[N],d[N],f[N][20],t;
24 bool bridge[M];
25 int tot,num,dcc,tc,DD[N];
26 int c[N];
27
28 void init(){
29     tot = tc = 1;
30     num = dcc = 0;
31     rep(i,0,n) head[i] = headc[i] = 0;
32     rep(i,0,2*m) bridge[i] = 0;
33     rep(i,0,n) c[i] = dfn[i] = low[i] = 0;
34     rep(i,0,n) dis[i] = d[i] = DD[i] = 0;
35 }
36 void add(int x,int y){
37     e[++tot].to = y; e[tot].next = head[x]; head[x] = tot;
38 }
39
40 void addc(int x,int y)
41 {
42     ec[++tc].to = y; ec[tc].next = headc[x]; headc[x] = tc;
43 }
44
45 void tarjan(int x,int in_edge)
46 {
47     dfn[x] = low[x] = ++num;
48     for(int i = head[x]; i ; i = e[i].next)
49     {
50         int y = e[i].to;

```

```

51     if(!dfn[y]){
52         tarjan(y,i); //传入边(x,y)的序号
53         low[x] = min(low[x],low[y]);
54         if(low[y] > low[x]) //该边连接(x,y), y点无法连接x点上面的点, 因此该边是割边
55             bridge[i] = bridge[i^1] = true;
56     }
57     else if(i != (in_edge^1)) //y点所连接的边 不能是(x,y)边的反向边
58         low[x] = min(low[x],dfn[y]);
59 }
60 }
61
62 void dfsD(int x,int hp){
63     DD[x] = hp;
64     for(int i = head[x]; i; i = e[i].next){
65         int y = e[i].to;
66         if(DD[y] == 0) dfsD(y,hp);
67     }
68 }
69
70 void dfs(int x) //用于将图划分为多个边双连通分量
71 {
72     c[x] = dcc;
73     for(int i = head[x]; i ; i = e[i].next)
74     {
75         int y = e[i].to;
76         if(c[y] || bridge[i]) continue; //如果点y已经属于别的强连通分量, 或者边i是割边, 则
77         continue
78         dfs(y);
79     }
80 }
81
82 void bfs(int s)
83 {
84     queue<int> q;
85     while(q.size()) q.pop();
86     q.push(s); d[s] = 1; dis[s] = 0; //把1当做树根
87     while(q.size())
88     {
89         int x = q.front(); q.pop();
90         for(int i = headc[x]; i ;i = ec[i].next){
91             int y = ec[i].to;
92             if(d[y]) continue;
93             d[y] = d[x]+1;
94             dis[y] = dis[x]+1; //dist[y]:从1到y的距离
95             f[y][0] = x; //y走2^0步到达x
96             for(int j = 1; j <= t;j++)
97                 f[y][j] = f[f[y][j-1]][j-1];
98             q.push(y);
99         }
100    }
101
102 int lca(int x,int y)
103 {
104     if(d[x] > d[y]) swap(x,y);
105     for(int i = t; i >= 0; i--)
106         if(d[f[y][i]] >= d[x]) y = f[y][i]; //往上追溯, 直至y和x位于同一深度
107     if(x == y) return x; //如果已经找到了, 就返回x
108     for(int i = t; i >= 0; i--)

```

```

109         if(f[x][i] != f[y][i]) x = f[x][i], y = f[y][i]; //x和y同时往上走，一直到x和y恰好为
110         lca的子节点
111     return f[x][0]; //x和y共同的根节点就是lca
112 }
113
114 int main()
115 {
116     int _; scanf("%d",&_);
117     while(_--){
118         scanf("%d%d%d",&n,&m,&q);
119         init();
120         rep(i,1,m){
121             int xx,yy; scanf("%d%d",&xx,&yy);
122             add(xx,yy); add(yy,xx);
123         }
124         int ctt = 0;
125         rep(i,1,n)
126             if(!DD[i]) dfsD(i,++ctt);
127         rep(i,1,n)
128             if(!dfn[i]) tarjan(i,0); //将边序号传进去
129         rep(i,1,n)
130             if(!c[i]){ //如果i点未被标记过
131                 ++dcc;
132                 dfs(i);
133             }
134         rep(i,2,tot){
135             int x = e[i^1].to, y = e[i].to; //记录该边连接的两个端点
136             if(c[x] == c[y]) continue; //如果连接的两个点属于同一连通分量，则continue
137             addc(c[x],c[y]); //用连通的分量的编号来代表整个连通分量，以此来进行缩点
138             addc(c[y],c[x]);
139         }
140         t = (int)(log(dcc+1)/log(2))+1;
141         rep(i,1,dcc)
142             if(d[i] == 0) bfs(i);
143         rep(i,1,q){
144             int u,v,w; scanf("%d%d%d",&u,&v,&w);
145             if(DD[u] == DD[v] && DD[u] == DD[w]){
146                 if(c[u] == c[v] && c[u] == c[w]) printf("Yes\n");
147                 else if(c[v] == c[w] && c[v] != c[u]) printf("No\n");
148                 else if(c[u] != c[v] && c[u] != c[w] && c[v] != c[w]){
149                     int y = lca(c[v],c[w]);
150                     if(y == c[u]) printf("Yes\n");
151                     else{
152                         int x1 = lca(c[v],c[u]);
153                         int x2 = lca(c[w],c[u]);
154                         if(x1 == y && x2 == c[u]) printf("Yes\n");
155                         else if(x2 == y && x1 == c[u]) printf("Yes\n");
156                         else printf("No\n");
157                     }
158                 }
159                 else if(c[v] == c[u] && c[w] != c[v]) printf("Yes\n");
160                 else if(c[w] == c[u] && c[w] != c[v]) printf("Yes\n");
161                 else printf("No\n");
162             }
163             else printf("No\n");
164         }
165     }
166     return 0;

```

167 }

5.6.3 点双缩点以及求割点

v-DCC 的缩点

由于一个割点可能属于多个 v-DCC，因此缩点之后，图中包含 $p+t$ 个节点， p 个割点， t 个 v-DCC，将每个 v-DCC 和每个割点都作为新图中的节点

并将每个割点与包含它的所有 v-DCC 之间连边

此处需要注意：

缩点之后，所有的点双连通分量变成一个点，该点只与割点相连，不会与其他点双连通分量相连

无向图中求割点

x 是割点当且仅当搜索树上存在 x 的一个子节点 y ，满足

$\text{dfn}[x] \leq \text{low}[y]$

即 y 点无法访问 x 点之上的点

如果 x 不是根节点，则只需要一个点 y 即可

如果 x 是根节点，则需要两个点 y_1, y_2 存在， x 才能是割点，此时如果去掉 x ，则 y_1 与 y_2 不连通

dfn : 记录该点的 dfs 序 low : 记录该点所能达到的最小 dfn 值

无向图中求点双连通分量

操作方法：

1. 当一个节点第一次被访问时，把该节点入栈。
2. 当割点判定法则中的条件 $\text{dfn}[x] \leq \text{low}[y]$ 成立时，无论 x 是否为根，都要：
 - (1) 从栈顶不断弹出节点，直至节点 y 被弹出。
 - (2) 刚才弹出的所有节点与节点 x 一起构成一个 v-DCC

割点：

若 x 为割点，则从图中删去节点 x 及所有与 x 相关联的边之后， G 分裂成了两个或两个以上不相连的子图

点双连通分量 (v-DCC) —— 分量中无割点

注意：点双连通分量与“删除割点后图中剩余的连通块”是不一样的概念

桥不属于任何 e-DCC，但是割点可能属于多个 v-DCC

无向连通图是点双连通图所需要满足的条件：【其中一个】

1. 图的顶点数不超过 2。
2. 图中任意两点都同时包含在至少一个简单环中。其中“简单环”指的是不自交的环，也就是我们通常画出的环

```

1 #include <cstdio>
2 #include <iostream>
3 #include <algorithm>
4 #include <cstring>
5 #include <vector>
6 #define rep(i,a,b) for(int i = a;i <= b;i++)
7 using namespace std;
8 const int N = 1e5+10, M = 5*1e5+10;
9
10 struct Edge{
11     int to,next;
12 }e[M*2],ec[M*2]; //双向边

```

```

13 int head[N],dfn[N],low[N],stack[N]; //dfn:记录该点的dfs序      low:记录该点所能达到的最小dfn
    值
14 //stack:手动模拟堆栈,先进先出
15 int n,m,tot,num,root,top,cnt; //top:用来记录stack的堆顶位置 cnt:记录一共有多少个点双连通分量
16 bool cut[N]; //记录该点是否为割点
17 vector<int> dcc[N]; //dcc[i]保存编号为i的v-DCC中的所有节点
18 int new_id[2*N]; //用于记录缩点之后的新图的各个点的编号
19 int headc[2*N],tc; //tc:记录新图的边的序号      headc:记录新图中每个点的第一条边
20
21 void init()
22 {
23     tot = 1; //记录第一条边
24     num = 0; //记录dfs序
25     cnt = 0; //记录点双连通分量个数
26     top = 0; //初始化堆栈
27     memset(head,0,sizeof head);
28     memset(dfn,0,sizeof dfn);
29     memset(low,0,sizeof low);
30     memset(cut,0,sizeof cut);
31     rep(i,1,n){ //n个点
32         dcc[i].clear();
33     }
34 }
35
36 void initc()
37 {
38     //cnt记录一共有多少个点双连通分量
39     num = cnt; //割点的编号从cnt+1开始, 1-cnt的点为点双连通分量的编号
40     tc = 1;
41     memset(headc,0,sizeof headc);
42 }
43
44 void add(int x,int y)
45 {
46     e[++tot].to = y; e[tot].next = head[x]; head[x] = tot;
47 }
48
49 void addc(int x,int y)
50 {
51     ec[++tc].to = y; e[tot].next = headc[x]; headc[x] = tc;
52 }
53
54 void tarjan(int x)
55 {
56     dfn[x] = low[x] = ++num;
57     stack[++top] = x; //当一个节点第一次被访问时,把该节点入栈
58     if(x == root && head[x] == 0){ //孤立点-无边连接
59         dcc[++cnt].push_back(x); //孤立点自己就是一个点双连通分量
60         return;
61     }
62     int flag = 0; //记录存在几个子节点,使得low[y]>=dfn[x],用于判断割点
63     for(int i = head[x]; i ; i = e[i].next)
64     {
65         int y = e[i].to;
66         if(!dfn[y]){
67             tarjan(y);
68             low[x] = min(low[x],low[y]);
69             if(low[y] >= dfn[x]){
70                 flag++;
71             }
72         }
73     }
74 }
75
76 void print()
77 {
78     rep(i,1,n)
79     {
80         cout << i << " : ";
81         rep(j,1,dcc[i].size())
82         {
83             cout << dcc[i][j];
84             if(j != dcc[i].size() - 1)
85                 cout << " ";
86         }
87         cout << endl;
88     }
89 }
90
91 void output()
92 {
93     rep(i,1,nc)
94     {
95         cout << tc << " : ";
96         rep(j,1,dcc[i].size())
97         {
98             cout << dcc[i][j];
99             if(j != dcc[i].size() - 1)
100                cout << " ";
101         }
102         cout << endl;
103     }
104 }
105
106 void printc()
107 {
108     rep(i,1,nc)
109     {
110         cout << tc << " : ";
111         rep(j,1,headc[i])
112         {
113             cout << j;
114             if(j != headc[i] - 1)
115                 cout << " ";
116         }
117         cout << endl;
118     }
119 }
120
121 void outputc()
122 {
123     rep(i,1,nc)
124     {
125         cout << tc << " : ";
126         rep(j,1,headc[i])
127         {
128             cout << j;
129             if(j != headc[i] - 1)
130                 cout << " ";
131         }
132         cout << endl;
133     }
134 }
135
136 void printe()
137 {
138     rep(i,1,tot)
139     {
140         cout << i << " : ";
141         rep(j,1,e[i].next)
142         {
143             cout << e[i].to;
144             if(j != e[i].next - 1)
145                 cout << " ";
146         }
147         cout << endl;
148     }
149 }
150
151 void outpute()
152 {
153     rep(i,1,tot)
154     {
155         cout << i << " : ";
156         rep(j,1,e[i].next)
157         {
158             cout << e[i].to;
159             if(j != e[i].next - 1)
160                 cout << " ";
161         }
162         cout << endl;
163     }
164 }
165
166 void printec()
167 {
168     rep(i,1,tc)
169     {
170         cout << i << " : ";
171         rep(j,1,ec[i].next)
172         {
173             cout << ec[i].to;
174             if(j != ec[i].next - 1)
175                 cout << " ";
176         }
177         cout << endl;
178     }
179 }
180
181 void outputec()
182 {
183     rep(i,1,tc)
184     {
185         cout << i << " : ";
186         rep(j,1,ec[i].next)
187         {
188             cout << ec[i].to;
189             if(j != ec[i].next - 1)
190                 cout << " ";
191         }
192         cout << endl;
193     }
194 }
195
196 void printec()
197 {
198     rep(i,1,tc)
199     {
200         cout << i << " : ";
201         rep(j,1,ec[i].next)
202         {
203             cout << ec[i].to;
204             if(j != ec[i].next - 1)
205                 cout << " ";
206         }
207         cout << endl;
208     }
209 }
210
211 void outputec()
212 {
213     rep(i,1,tc)
214     {
215         cout << i << " : ";
216         rep(j,1,ec[i].next)
217         {
218             cout << ec[i].to;
219             if(j != ec[i].next - 1)
220                 cout << " ";
221         }
222         cout << endl;
223     }
224 }
225
226 void printec()
227 {
228     rep(i,1,tc)
229     {
230         cout << i << " : ";
231         rep(j,1,ec[i].next)
232         {
233             cout << ec[i].to;
234             if(j != ec[i].next - 1)
235                 cout << " ";
236         }
237         cout << endl;
238     }
239 }
240
241 void outputec()
242 {
243     rep(i,1,tc)
244     {
245         cout << i << " : ";
246         rep(j,1,ec[i].next)
247         {
248             cout << ec[i].to;
249             if(j != ec[i].next - 1)
250                 cout << " ";
251         }
252         cout << endl;
253     }
254 }
255
256 void printec()
257 {
258     rep(i,1,tc)
259     {
260         cout << i << " : ";
261         rep(j,1,ec[i].next)
262         {
263             cout << ec[i].to;
264             if(j != ec[i].next - 1)
265                 cout << " ";
266         }
267         cout << endl;
268     }
269 }
270
271 void outputec()
272 {
273     rep(i,1,tc)
274     {
275         cout << i << " : ";
276         rep(j,1,ec[i].next)
277         {
278             cout << ec[i].to;
279             if(j != ec[i].next - 1)
280                 cout << " ";
281         }
282         cout << endl;
283     }
284 }
285
286 void printec()
287 {
288     rep(i,1,tc)
289     {
290         cout << i << " : ";
291         rep(j,1,ec[i].next)
292         {
293             cout << ec[i].to;
294             if(j != ec[i].next - 1)
295                 cout << " ";
296         }
297         cout << endl;
298     }
299 }
300
301 void outputec()
302 {
303     rep(i,1,tc)
304     {
305         cout << i << " : ";
306         rep(j,1,ec[i].next)
307         {
308             cout << ec[i].to;
309             if(j != ec[i].next - 1)
310                 cout << " ";
311         }
312         cout << endl;
313     }
314 }
315
316 void printec()
317 {
318     rep(i,1,tc)
319     {
320         cout << i << " : ";
321         rep(j,1,ec[i].next)
322         {
323             cout << ec[i].to;
324             if(j != ec[i].next - 1)
325                 cout << " ";
326         }
327         cout << endl;
328     }
329 }
330
331 void outputec()
332 {
333     rep(i,1,tc)
334     {
335         cout << i << " : ";
336         rep(j,1,ec[i].next)
337         {
338             cout << ec[i].to;
339             if(j != ec[i].next - 1)
340                 cout << " ";
341         }
342         cout << endl;
343     }
344 }
345
346 void printec()
347 {
348     rep(i,1,tc)
349     {
350         cout << i << " : ";
351         rep(j,1,ec[i].next)
352         {
353             cout << ec[i].to;
354             if(j != ec[i].next - 1)
355                 cout << " ";
356         }
357         cout << endl;
358     }
359 }
360
361 void outputec()
362 {
363     rep(i,1,tc)
364     {
365         cout << i << " : ";
366         rep(j,1,ec[i].next)
367         {
368             cout << ec[i].to;
369             if(j != ec[i].next - 1)
370                 cout << " ";
371         }
372         cout << endl;
373     }
374 }
375
376 void printec()
377 {
378     rep(i,1,tc)
379     {
380         cout << i << " : ";
381         rep(j,1,ec[i].next)
382         {
383             cout << ec[i].to;
384             if(j != ec[i].next - 1)
385                 cout << " ";
386         }
387         cout << endl;
388     }
389 }
390
391 void outputec()
392 {
393     rep(i,1,tc)
394     {
395         cout << i << " : ";
396         rep(j,1,ec[i].next)
397         {
398             cout << ec[i].to;
399             if(j != ec[i].next - 1)
400                 cout << " ";
401         }
402         cout << endl;
403     }
404 }
405
406 void printec()
407 {
408     rep(i,1,tc)
409     {
410         cout << i << " : ";
411         rep(j,1,ec[i].next)
412         {
413             cout << ec[i].to;
414             if(j != ec[i].next - 1)
415                 cout << " ";
416         }
417         cout << endl;
418     }
419 }
420
421 void outputec()
422 {
423     rep(i,1,tc)
424     {
425         cout << i << " : ";
426         rep(j,1,ec[i].next)
427         {
428             cout << ec[i].to;
429             if(j != ec[i].next - 1)
430                 cout << " ";
431         }
432         cout << endl;
433     }
434 }
435
436 void printec()
437 {
438     rep(i,1,tc)
439     {
440         cout << i << " : ";
441         rep(j,1,ec[i].next)
442         {
443             cout << ec[i].to;
444             if(j != ec[i].next - 1)
445                 cout << " ";
446         }
447         cout << endl;
448     }
449 }
450
451 void outputec()
452 {
453     rep(i,1,tc)
454     {
455         cout << i << " : ";
456         rep(j,1,ec[i].next)
457         {
458             cout << ec[i].to;
459             if(j != ec[i].next - 1)
460                 cout << " ";
461         }
462         cout << endl;
463     }
464 }
465
466 void printec()
467 {
468     rep(i,1,tc)
469     {
470         cout << i << " : ";
471         rep(j,1,ec[i].next)
472         {
473             cout << ec[i].to;
474             if(j != ec[i].next - 1)
475                 cout << " ";
476         }
477         cout << endl;
478     }
479 }
480
481 void outputec()
482 {
483     rep(i,1,tc)
484     {
485         cout << i << " : ";
486         rep(j,1,ec[i].next)
487         {
488             cout << ec[i].to;
489             if(j != ec[i].next - 1)
490                 cout << " ";
491         }
492         cout << endl;
493     }
494 }
495
496 void printec()
497 {
498     rep(i,1,tc)
499     {
500         cout << i << " : ";
501         rep(j,1,ec[i].next)
502         {
503             cout << ec[i].to;
504             if(j != ec[i].next - 1)
505                 cout << " ";
506         }
507         cout << endl;
508     }
509 }
510
511 void outputec()
512 {
513     rep(i,1,tc)
514     {
515         cout << i << " : ";
516         rep(j,1,ec[i].next)
517         {
518             cout << ec[i].to;
519             if(j != ec[i].next - 1)
520                 cout << " ";
521         }
522         cout << endl;
523     }
524 }
525
526 void printec()
527 {
528     rep(i,1,tc)
529     {
530         cout << i << " : ";
531         rep(j,1,ec[i].next)
532         {
533             cout << ec[i].to;
534             if(j != ec[i].next - 1)
535                 cout << " ";
536         }
537         cout << endl;
538     }
539 }
540
541 void outputec()
542 {
543     rep(i,1,tc)
544     {
545         cout << i << " : ";
546         rep(j,1,ec[i].next)
547         {
548             cout << ec[i].to;
549             if(j != ec[i].next - 1)
550                 cout << " ";
551         }
552         cout << endl;
553     }
554 }
555
556 void printec()
557 {
558     rep(i,1,tc)
559     {
560         cout << i << " : ";
561         rep(j,1,ec[i].next)
562         {
563             cout << ec[i].to;
564             if(j != ec[i].next - 1)
565                 cout << " ";
566         }
567         cout << endl;
568     }
569 }
570
571 void outputec()
572 {
573     rep(i,1,tc)
574     {
575         cout << i << " : ";
576         rep(j,1,ec[i].next)
577         {
578             cout << ec[i].to;
579             if(j != ec[i].next - 1)
580                 cout << " ";
581         }
582         cout << endl;
583     }
584 }
585
586 void printec()
587 {
588     rep(i,1,tc)
589     {
590         cout << i << " : ";
591         rep(j,1,ec[i].next)
592         {
593             cout << ec[i].to;
594             if(j != ec[i].next - 1)
595                 cout << " ";
596         }
597         cout << endl;
598     }
599 }
600
601 void outputec()
602 {
603     rep(i,1,tc)
604     {
605         cout << i << " : ";
606         rep(j,1,ec[i].next)
607         {
608             cout << ec[i].to;
609             if(j != ec[i].next - 1)
610                 cout << " ";
611         }
612         cout << endl;
613     }
614 }
615
616 void printec()
617 {
618     rep(i,1,tc)
619     {
620         cout << i << " : ";
621         rep(j,1,ec[i].next)
622         {
623             cout << ec[i].to;
624             if(j != ec[i].next - 1)
625                 cout << " ";
626         }
627         cout << endl;
628     }
629 }
630
631 void outputec()
632 {
633     rep(i,1,tc)
634     {
635         cout << i << " : ";
636         rep(j,1,ec[i].next)
637         {
638             cout << ec[i].to;
639             if(j != ec[i].next - 1)
640                 cout << " ";
641         }
642         cout << endl;
643     }
644 }
645
646 void printec()
647 {
648     rep(i,1,tc)
649     {
650         cout << i << " : ";
651         rep(j,1,ec[i].next)
652         {
653             cout << ec[i].to;
654             if(j != ec[i].next - 1)
655                 cout << " ";
656         }
657         cout << endl;
658     }
659 }
660
661 void outputec()
662 {
663     rep(i,1,tc)
664     {
665         cout << i << " : ";
666         rep(j,1,ec[i].next)
667         {
668             cout << ec[i].to;
669             if(j != ec[i].next - 1)
670                 cout << " ";
671         }
672         cout << endl;
673     }
674 }
675
676 void printec()
677 {
678     rep(i,1,tc)
679     {
680         cout << i << " : ";
681         rep(j,1,ec[i].next)
682         {
683             cout << ec[i].to;
684             if(j != ec[i].next - 1)
685                 cout << " ";
686         }
687         cout << endl;
688     }
689 }
690
691 void outputec()
692 {
693     rep(i,1,tc)
694     {
695         cout << i << " : ";
696         rep(j,1,ec[i].next)
697         {
698             cout << ec[i].to;
699             if(j != ec[i].next - 1)
700                 cout << " ";
701         }
702         cout << endl;
703     }
704 }
705
706 void printec()
707 {
708     rep(i,1,tc)
709     {
710         cout << i << " : ";
711         rep(j,1,ec[i].next)
712         {
713             cout << ec[i].to;
714             if(j != ec[i].next - 1)
715                 cout << " ";
716         }
717         cout << endl;
718     }
719 }
720
721 void outputec()
722 {
723     rep(i,1,tc)
724     {
725         cout << i << " : ";
726         rep(j,1,ec[i].next)
727         {
728             cout << ec[i].to;
729             if(j != ec[i].next - 1)
730                 cout << " ";
731         }
732         cout << endl;
733     }
734 }
735
736 void printec()
737 {
738     rep(i,1,tc)
739     {
740         cout << i << " : ";
741         rep(j,1,ec[i].next)
742         {
743             cout << ec[i].to;
744             if(j != ec[i].next - 1)
745                 cout << " ";
746         }
747         cout << endl;
748     }
749 }
750
751 void outputec()
752 {
753     rep(i,1,tc)
754     {
755         cout << i << " : ";
756         rep(j,1,ec[i].next)
757         {
758             cout << ec[i].to;
759             if(j != ec[i].next - 1)
760                 cout << " ";
761         }
762         cout << endl;
763     }
764 }
765
766 void printec()
767 {
768     rep(i,1,tc)
769     {
770         cout << i << " : ";
771         rep(j,1,ec[i].next)
772         {
773             cout << ec[i].to;
774             if(j != ec[i].next - 1)
775                 cout << " ";
776         }
777         cout << endl;
778     }
779 }
780
781 void outputec()
782 {
783     rep(i,1,tc)
784     {
785         cout << i << " : ";
786         rep(j,1,ec[i].next)
787         {
788             cout << ec[i].to;
789             if(j != ec[i].next - 1)
790                 cout << " ";
791         }
792         cout << endl;
793     }
794 }
795
796 void printec()
797 {
798     rep(i,1,tc)
799     {
800         cout << i << " : ";
801         rep(j,1,ec[i].next)
802         {
803             cout << ec[i].to;
804             if(j != ec[i].next - 1)
805                 cout << " ";
806         }
807         cout << endl;
808     }
809 }
810
811 void outputec()
812 {
813     rep(i,1,tc)
814     {
815         cout << i << " : ";
816         rep(j,1,ec[i].next)
817         {
818             cout << ec[i].to;
819             if(j != ec[i].next - 1)
820                 cout << " ";
821         }
822         cout << endl;
823     }
824 }
825
826 void printec()
827 {
828     rep(i,1,tc)
829     {
830         cout << i << " : ";
831         rep(j,1,ec[i].next)
832         {
833             cout << ec[i].to;
834             if(j != ec[i].next - 1)
835                 cout << " ";
836         }
837         cout << endl;
838     }
839 }
840
841 void outputec()
842 {
843     rep(i,1,tc)
844     {
845         cout << i << " : ";
846         rep(j,1,ec[i].next)
847         {
848             cout << ec[i].to;
849             if(j != ec[i].next - 1)
850                 cout << " ";
851         }
852         cout << endl;
853     }
854 }
855
856 void printec()
857 {
858     rep(i,1,tc)
859     {
860         cout << i << " : ";
861         rep(j,1,ec[i].next)
862         {
863             cout << ec[i].to;
864             if(j != ec[i].next - 1)
865                 cout << " ";
866         }
867         cout << endl;
868     }
869 }
870
871 void outputec()
872 {
873     rep(i,1,tc)
874     {
875         cout << i << " : ";
876         rep(j,1,ec[i].next)
877         {
878             cout << ec[i].to;
879             if(j != ec[i].next - 1)
880                 cout << " ";
881         }
882         cout << endl;
883     }
884 }
885
886 void printec()
887 {
888     rep(i,1,tc)
889     {
890         cout << i << " : ";
891         rep(j,1,ec[i].next)
892         {
893             cout << ec[i].to;
894             if(j != ec[i].next - 1)
895                 cout << " ";
896         }
897         cout << endl;
898     }
899 }
900
901 void outputec()
902 {
903     rep(i,1,tc)
904     {
905         cout << i << " : ";
906         rep(j,1,ec[i].next)
907         {
908             cout << ec[i].to;
909             if(j != ec[i].next - 1)
910                 cout << " ";
911         }
912         cout << endl;
913     }
914 }
915
916 void printec()
917 {
918     rep(i,1,tc)
919     {
920         cout << i << " : ";
921         rep(j,1,ec[i].next)
922         {
923             cout << ec[i].to;
924             if(j != ec[i].next - 1)
925                 cout << " ";
926         }
927         cout << endl;
928     }
929 }
930
931 void outputec()
932 {
933     rep(i,1,tc)
934     {
935         cout << i << " : ";
936         rep(j,1,ec[i].next)
937         {
938             cout << ec[i].to;
939             if(j != ec[i].next - 1)
940                 cout << " ";
941         }
942         cout << endl;
943     }
944 }
945
946 void printec()
947 {
948     rep(i,1,tc)
949     {
950         cout << i << " : ";
951         rep(j,1,ec[i].next)
952         {
953             cout << ec[i].to;
954             if(j != ec[i].next - 1)
955                 cout << " ";
956         }
957         cout << endl;
958     }
959 }
960
961 void outputec()
962 {
963     rep(i,1,tc)
964     {
965         cout << i << " : ";
966         rep(j,1,ec[i].next)
967         {
968             cout << ec[i].to;
969             if(j != ec[i].next - 1)
970                 cout << " ";
971         }
972         cout << endl;
973     }
974 }
975
976 void printec()
977 {
978     rep(i,1,tc)
979     {
980         cout << i << " : ";
981         rep(j,1,ec[i].next)
982         {
983             cout << ec[i].to;
984             if(j != ec[i].next - 1)
985                 cout << " ";
986         }
987         cout << endl;
988     }
989 }
990
991 void outputec()
992 {
993     rep(i,1,tc)
994     {
995         cout << i << " : ";
996         rep(j,1,ec[i].next)
997         {
998             cout << ec[i].to;
999             if(j != ec[i].next - 1)
1000                cout << " ";
1001        }
1002        cout << endl;
1003    }
1004 }
1005
1006 void printec()
1007 {
1008     rep(i,1,tc)
1009     {
1010         cout << i << " : ";
1011         rep(j,1,ec[i].next)
1012         {
1013             cout << ec[i].to;
1014             if(j != ec[i].next - 1)
1015                 cout << " ";
1016         }
1017         cout << endl;
1018     }
1019 }
1020
1021 void outputec()
1022 {
1023     rep(i,1,tc)
1024     {
1025         cout << i << " : ";
1026         rep(j,1,ec[i].next)
1027         {
1028             cout << ec[i].to;
1029             if(j != ec[i].next - 1)
1030                 cout << " ";
1031         }
1032         cout << endl;
1033     }
1034 }
1035
1036 void printec()
1037 {
1038     rep(i,1,tc)
1039     {
1040         cout << i << " : ";
1041         rep(j,1,ec[i].next)
1042         {
1043             cout << ec[i].to;
1044             if(j != ec[i].next - 1)
1045                 cout << " ";
1046         }
1047         cout << endl;
1048     }
1049 }
1050
1051 void outputec()
1052 {
1053     rep(i,1,tc)
1054     {
1055         cout << i << " : ";
1056         rep(j,1,ec[i].next)
1057         {
1058             cout << ec[i].to;
1059             if(j != ec[i].next - 1)
1060                 cout << " ";
1061         }
1062         cout << endl;
1063     }
1064 }
1065
1066 void printec()
1067 {
1068     rep(i,1,tc)
1069     {
1070         cout << i << " : ";
1071         rep(j,1,ec[i].next)
1072         {
1073             cout << ec[i].to;
1074             if(j != ec[i].next - 1)
1075                 cout << " ";
1076         }
1077         cout << endl;
1078     }
1079 }
1080
1081 void outputec()
1082 {
1083     rep(i,1,tc)
1084     {
1085         cout << i << " : ";
1086         rep(j,1,ec[i].next)
1087         {
1088             cout << ec[i].to;
1089             if(j != ec[i].next - 1)
1090                 cout << " ";
1091         }
1092         cout << endl;
1093     }
1094 }
1095
1096 void printec()
1097 {
1098     rep(i,1,tc)
1099     {
1100         cout << i << " : ";
1101         rep(j,1,ec[i].next)
1102         {
1103             cout << ec[i].to;
1104             if(j != ec[i].next - 1)
1105                 cout << " ";
1106         }
1107         cout << endl;
1108     }
1109 }
1110
1111 void outputec()
1112 {
1113     rep(i,1,tc)
1114     {
1115         cout << i << " : ";
1116         rep(j,1,ec[i].next)
1117         {
1118             cout << ec[i].to;
1119             if(j != ec[i].next - 1)
1120                 cout << " ";
1121         }
1122         cout << endl;
1123     }
1124 }
1125
1126 void printec()
1127 {
1128     rep(i,1,tc)
1129     {
1130         cout << i << " : ";
1131         rep(j,1,ec[i].next)
1132         {
1133             cout << ec[i].to;
1134             if(j != ec[i].next - 1)
1135                 cout << " ";
1136         }
1137         cout << endl;
1138     }
1139 }
1140
1141 void outputec()
1142 {
1143     rep(i,1,tc)
1144     {
1145         cout << i << " : ";
1146         rep(j,1,ec[i].next)
1147         {
1148             cout << ec[i].to;
1149             if(j != ec[i].next - 1)
1150                 cout << " ";
1151         }
1152         cout << endl;
1153     }
1154 }
1155
1156 void printec()
1157 {
1158     rep(i,1,tc)
1159     {
1160         cout << i << " : ";
1161         rep(j,1,ec[i].next)
1162         {
1163             cout << ec[i].to;
1164             if(j != ec[i].next - 1)
1165                 cout << " ";
1166         }
1167         cout << endl;
1168     }
1169 }
1170
1171 void outputec()
1172 {
1173     rep(i,1,tc)
1174     {
1175         cout << i << " : ";
1176         rep(j,1,ec[i].next)
1177         {
1178             cout << ec[i].to;
1179             if(j != ec[i].next - 1)
1180                 cout << " ";
1181         }
1182         cout << endl;
1183     }
1184 }
1185
1186 void printec()
1187 {
1188     rep(i,1,tc)
1189     {
1190         cout << i << " : ";
1191         rep(j,1,ec[i].next)
1192         {
1193             cout << ec[i].to;
1194             if(j != ec[i].next - 1)
1195                 cout << " ";
1196         }
1197         cout << endl;
1198     }
1199 }
1200
1201 void outputec()
1202 {
1203     rep(i,1,tc)
1204     {
1205         cout << i << " : ";
1206         rep(j,1,ec[i].next)
1207         {
1208             cout << ec[i].to;
1209             if(j != ec[i].next - 1)
1210                 cout << " ";
1211         }
1212         cout
```

```

71             if(x != root || flag > 1) cut[x] = true;      //x为割点
72             cnt++; //出现了一个新的点双连通分量
73             int z;
74             do{
75                 z = stack[top--]; //x, y以及y之后的所有点构成一个点双连通分量
76                 dcc[cnt].push_back(z);
77             }while(z != y);
78             dcc[cnt].push_back(x);
79         }
80     }
81     else low[x] = min(low[x], dfn[y]);
82 }
83 }
84
85 int main()
86 {
87     while(~scanf("%d%d", &n, &m))
88     {
89         init();
90         rep(i, 1, m){
91             int x, y;
92             scanf("%d%d", &x, &y);
93             if(x == y) continue; //判断自环
94             add(x, y); add(y, x);
95         }
96         rep(i, 1, n)
97             if(!dfn[i]) root = i, tarjan(i); //因为原图可能不连通, 因此需要对于每个点进行dfs
98         //以下是缩点操作
99         initc();
100        rep(i, 1, n)
101            if(cut[i]) new_id[i] = ++num; //cut:记录该点是否为割点, 记录割点i在新图中的编号
102    为num+1
103    //建新图, 从每个v-DCC到它包含的所有割点连边
104    rep(i, 1, cnt) //cnt表示点双连通分量个数
105        for(int j = 0; j < dcc[i].size(); j++)
106        {
107            int x = dcc[i][j];
108            if(cut[x]){//如果x为割点, 则将x与其所属的点强连通分量相连
109                addc(i, new_id[x]);
110                addc(new_id[x], i);
111            }
112        }
113        // else c[x] = i; 此处可有可无, 此处的作用为c[x]表示x所在的强连通分量的编号
114    }
115    printf("缩点之后的森林, 点数为%d, 边数为%d\n", num, tc/2); //num为新图中点数
116    printf("编号 1~%d 的为原图的v-DCC, 编号 > %d 的为原图割点\n", cnt, cnt); //cnt为点双连
117    通分量个数
118    for(int i = 2; i < tc; i+=2)
119        printf("%d %d\n", ec[i^1].to, e[i].to); //输出新图中的所有无向边
120
121    /* rep(i, 1, cnt){
122        printf("e-DCC #%d:", i);
123        for(int j = 0; j < dcc[i].size(); j++)
124            printf(" %d", dcc[i][j]); //将第i个点双连通分量中的点全部输出
125        puts("");
126    }*/
127
128    /* rep(i, 1, n)
129        if(cut[i]) printf("%d ", i); //cut[i]标记i节点是否为割点
130        puts("are cut-vertexes"); */

```

```

128     }
129     return 0;
130 }
```

5.6.4 点双例题

题意：

n 个点，m 条边

如果把第 i 个点去掉，将有多少对点不能互通

```

1 #include <cstdio>
2 #include <iostream>
3 #include <algorithm>
4 #include <cstring>
5 #define rep(i,a,b) for(int i = a;i <= b;i++)
6 using namespace std;
7 const int N = 1e5+10, M = 5*1e5+10;
8 typedef long long ll;
9
10 struct Edge{
11     int to,next;
12 }e[M*2];
13 int head[N],dfn[N],low[N],siz[N]; //dfn:记录该点的dfs序    low:记录该点所能达到的最小dfn值
14 ll ans[N];
15 bool cut[N]; //记录该点是否为割点
16 int n,m,tot,num,root;
17
18 void init()
19 {
20     tot = 1;
21     memset(head,0,sizeof head);
22     memset(cut,0,sizeof cut);
23     memset(low,0,sizeof low);
24     memset(dfn,0,sizeof dfn);
25     memset(ans,0,sizeof ans);
26     memset(siz,0,sizeof siz);
27 }
28
29 void add(int x,int y)
30 {
31     e[++tot].to = y; e[tot].next = head[x]; head[x] = tot;
32 }
33
34 void tarjan(int x)
35 {
36     dfn[x] = low[x] = ++num; //记录步长
37     siz[x] = 1; //记录该点之下有几个点
38     int flag = 0,sum = 0; //标记有几个子节点 low[y] >= dfn[x]
39     for(int i = head[x]; i ; i = e[i].next)
40     {
41         int y = e[i].to;
42         if(!dfn[y])
43         {
44             tarjan(y); //每一次被dfs，都是因为该点未被访问过
45             siz[x] += siz[y];
46             low[x] = min(low[x],low[y]);
47             if(low[y] >= dfn[x]){

```

```

48         flag++; //根节点需要满足两次这个条件 非根节点只需满足一次这个条件 即为割点
49         ans[x] += (ll)siz[y]*(n-siz[y]); //flag++说明如果去掉x，则y与外界不再连通
50         sum += siz[y];
51         if(x != 1 || flag > 1) cut[x] = true;
52     }
53 } else{
54     //siz[x] += siz[y]; //此处不能加这句话，因为y点之前已经被访问过了，说明就算去掉x，y仍然可以连通，因此不能算入
55     low[x] = min(low[x],dfn[y]); //因为该点之前已经被搜过了
56 }
57 }
58 if(cut[x])
59     ans[x] += (ll)(n-sum-1)*(sum+1)+(n-1);
60 else
61     ans[x] = 2*(n-1);
62 }
63 }
64
65 int main()
66 {
67     while(~scanf("%d%d",&n,&m))
68     {
69         init();
70         rep(i,1,m){
71             int x,y;
72             scanf("%d%d",&x,&y);
73             if(x == y) continue;
74             add(x,y); add(y,x);
75         }
76         tarjan(1); //这里不需要从每一个点进行dfs，是因为题目保证所有城镇连通
77         rep(i,1,n)
78             printf("%lld\n",ans[i]);
79     }
80     return 0;
81 }
```

5.7 强连通分量

5.7.1 强连通分量及缩点

求有向图中的强连通分量

将 SCC 中的强连通分量进行缩点

求解算法：

1. 当节点 x 第一次被访问时，把 x 入栈，初始化 $low[x] = dfn[x]$
2. 扫描从 x 出发的每条边 (x,y)
 - (1) 若 y 没被访问过，则说明 (x,y) 是树枝边，递归访问 y，从 y 回溯之后，令 $low[x] = \min(low[x], low[y])$.
 - (2) 若 y 被访问过并且 y 在栈中，即 y 被访问过，并且 y 不属于之前的强连通分量，则令 $low[x] = \min(low[x], dfn[y])$.
3. 从 x 回溯之前，判断是否有 $low[x] = dfn[x]$ 。若成立，则不断从栈中弹出节点，直至 x 出栈。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <algorithm>
4 #include <cstring>
5 #include <vector>
6 #define rep(i,a,b) for(int i = a;i <= b;i++)
7 using namespace std;
8 const int N = 1e5+10, M = 1e6+10;
```

```

9
10 struct Edge{
11     int to,next;
12 }e[M],ec[M];
13 int head[N],dfn[N],low[N]; //dfn:dfs的序号 low:该点能够到达的最小的dfn值的点
14 int stack[N],vis[N],c[N]; //stack:维护通过点x到达的所有点 vis[x]:标记点x是否在堆栈中 c[x]:
    记录点x所在的强连通分量的编号
15 vector<int> scc[N]; //scc[i]:记录第i个强连通分量中的所有节点
16 int n,m,tot,num,top,cnt; //tot记录边的序号, num记录dfs序, top记录stack堆栈的顶点, cnt记录图
    中强连通分量个数
17 int tc,headc[N];
18
19 void init()
20 {
21     tot = 1; num = 0;
22     top = 0; cnt = 0;
23     memset(head,0,sizeof head);
24     memset(c,0,sizeof c);
25     memset(vis,0,sizeof vis);
26     memset(dfn,0,sizeof dfn);
27     memset(low,0,sizeof low);
28     rep(i,1,n)
         scc[i].clear();
29 }
30
31
32 void initc()
33 {
34     tc = 1;
35     memset(headc,0,sizeof headc);
36 }
37
38 void add(int x,int y)
39 {
40     e[++tot].to = y; e[tot].next = head[x]; head[x] = tot;
41 }
42
43 void addc(int x,int y)
44 {
45     ec[++tc].to = y; ec[tc].next = headc[x]; headc[x] = tc;
46 }
47
48 void tarjan(int x)
49 {
50     dfn[x] = low[x] = ++num;
51     stack[++top] = x, vis[x] = 1; //标记点x在栈中
52     for(int i = head[x]; i ; i = e[i].next)
53     {
54         int y = e[i].to;
55         if(!dfn[y]){//如果点y没有被访问过
56             tarjan(y); //从y点一直递归下去
57             low[x] = min(low[x],low[y]);
58         }
59         else if(vis[y]) low[x] = min(low[x],dfn[y]);
60         //在求双连通分量时, 此处为dfn[y], 即y点被访问过, 则可用y的dfn值来更新x的low值
61         //但是在强连通分量中, 图是有向图, 因此vis[y]==0的点, 要么没被访问过, 要么已经出栈了, 即已经属
        于另一块强连通分量中
62         //若y点已经属于另一块强连通分量, 由于是有向图, 因此y无法与x构成环, 所以不能用y点dfn值来更新x
        的low值
63     }
}

```

```

64     if(dfn[x] == low[x])    //low[x] == dfn[x]成立, 表明栈中从x到栈顶的所有节点构成一个强连通分
65     {                      //这些节点无法与其他节点一起构成环
66         cnt++; int y;
67         do{
68             y = stack[top--], vis[y] = 0; //节点出栈
69             c[y] = cnt, scc[cnt].push_back(y); //将这些节点存入新的强连通分量中
70         }while(x != y);
71     }
72 }
73
74 int main()
75 {
76     while(~scanf("%d%d", &n, &m))
77     {
78         init();
79         rep(i, 1, m){
80             int x, y;
81             scanf("%d%d", &x, &y);
82             add(x, y);
83         }
84         rep(i, 1, n)
85             if(!dfn[i]) tarjan(i);
86             //将每个SCC缩成一个点, 对于原图中的每条有向边
87             //如果c[x]!=c[y], 则在编号为c[x]和编号为c[y]的SCC之间连边
88             //最后结果将是一张有向无环图
89         initc();
90         rep(x, 1, n){
91             for(int i = head[x]; i ; i = e[i].next){
92                 int y = e[i].to;
93                 if(c[x] == c[y]) continue;
94                 addc(c[x], c[y]);
95             }
96         }
97     }
98     return 0;
99 }
```

5.7.2 2-SAT

题意：

有一个小镇上只有一个牧师。这个小镇上有一个传说，在九月一日结婚的人会受到爱神的保佑，但是要牧师举办一个仪式。这个仪式要么在婚礼刚刚开始的时候举行，要么举行完婚礼正好结束。

现在已知有 n 场婚礼，告诉你每一场的开始和结束时间，以及举行仪式所需的时间。问牧师能否参加所有的婚礼，如果能则输出一种方案。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <algorithm>
4 #include <cstring>
5 #include <vector>
6 #define rep(i,a,b) for(int i = a;i <= b;i++)
7 using namespace std;
8 const int N = 1e5+10, M = 1e7;
9
10 struct Edge{
11     int to,next;
```

```

12 }e[M];
13 int head[N],dfn[N],low[N]; //dfn:dfs的序号 low:该点能够到达的最小的dfn值的点
14 int stack[N],vis[N],c[N]; //stack:维护通过点x到达的所有点 vis[x]:标记点x是否在堆栈中 c[x]:
    记录点x所在的强连通分量的编号
15 vector<int> scc[N]; //scc[i]:记录第i个强连通分量中的所有节点
16 int n,m,tot,num,top,cnt; //tot记录边的序号, num记录dfn序, top记录stack堆栈的顶点, cnt记录图
    中强连通分量个数
17 int s[N],t[N],d[N];
18
19 void init()
20 {
21     tot = 1; num = 0;
22     top = 0; cnt = 0;
23     memset(head,0,sizeof head);
24     memset(c,0,sizeof c);
25     memset(vis,0,sizeof vis);
26     memset(dfn,0,sizeof dfn);
27     memset(low,0,sizeof low);
28     rep(i,1,n)
29         scc[i].clear();
30 }
31
32 void add(int x,int y)
33 {
34     e[++tot].to = y; e[tot].next = head[x]; head[x] = tot;
35 }
36
37 void tarjan(int x)
38 {
39     dfn[x] = low[x] = ++num;
40     stack[++top] = x, vis[x] = 1; //标记点x在栈中
41     for(int i = head[x]; i ; i = e[i].next)
42     {
43         int y = e[i].to;
44         if(!dfn[y]){//如果点y没有被访问过
45             tarjan(y); //从y点一直递归下去
46             low[x] = min(low[x],low[y]);
47         }
48         else if(vis[y]) low[x] = min(low[x],dfn[y]);
49     }
50     if(dfn[x] == low[x]) //low[x] == dfn[x]成立, 表明栈中从x到栈顶的所有节点构成一个强连通分
        量
51     { //这些节点无法与其他节点一起构成环
52         cnt++; int y;
53         do{
54             y = stack[top--],vis[y] = 0; //节点出栈
55             c[y] = cnt, scc[cnt].push_back(y); //将这些节点存入新的强连通分量中
56         }while(x != y);
57     }
58 }
59
60 int overlap(int a1,int b1,int a2,int b2)
61 {
62     if(a1>=a2&&a1<b2 || b1>a2&&b1<=b2 || a1<=a2&&b1>=b2) return 1;
63     return 0;
64 }
65
66 void solve()
67 {

```

```

68     rep(i,1,n){
69         if(c[i] == c[i+n]){
70             printf("NO\n");
71             return;
72         }
73     }
74     printf("YES\n");
75     int val[N];
76     memset(val,0,sizeof val);
77     rep(i,1,n){
78         if(c[i] < c[i+n]) //Tarjan之后的图就已经进行了拓扑排序
79             val[i] = 0; //如果! a -> a, 则a为真
80         else val[i] = 1;
81     }
82     rep(i,1,n){
83         if(val[i] == 0){
84             printf("%02d:%02d %02d:%02d\n",s[i]/60,s[i]%60,(s[i]+d[i])/60,(s[i]+d[i])
85             %60);
86         }
87         else
88             printf("%02d:%02d %02d:%02d\n",(t[i]-d[i])/60,(t[i]-d[i])%60,t[i]/60,t[i]
89             ]%60);
90     }
91 }
92 int main()
93 {
94     while(~scanf("%d",&n))
95     {
96         init();
97         rep(i,1,n){
98             int x1,y1,x2,y2;
99             scanf("%d:%d %d:%d %d",&x1,&y1,&x2,&y2,&d[i]);
100            s[i] = x1*60+y1;
101            t[i] = x2*60+y2;
102        }
103        //每个点都有两个状态, 一个为真, 一个为假, 因此共2*n个状态
104        rep(i,1,n){
105            rep(j,i+1,n){
106                if(overlap(s[i],s[i]+d[i],s[j],s[j]+d[j]))
107                    add(i,j+n),add(j,i+n);
108                if(overlap(s[i],s[i]+d[i],t[j]-d[j],t[j]))
109                    add(i,j),add(j+n,i+n);
110                if(overlap(t[i]-d[i],t[i],s[j],s[j]+d[j]))
111                    add(i+n,j+n),add(j,i);
112                if(overlap(t[i]-d[i],t[i],t[j]-d[j],t[j]))
113                    add(i+n,j),add(j+n,i);
114            }
115            rep(i,1,2*n)
116                if(!dfn[i]) tarjan(i);
117            solve();
118        }
119    }
120 }

```

5.8 二分图匹配

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #define rep(i,a,b) for(int i = a;i <= b;i++)
5 using namespace std;
6 const int MAXN = 5010; //点数最大值
7 const int MAXM = 50010; //边数最大值
8
9 struct Edge{
10     int to,next; //next表示该边的上一条边的编号
11 }e[MAXM];
12 int head[MAXN],tot,uN; //uN表示一共有多少个点
13 //head[u]表示以u为起点的边的编号
14 int girl[MAXN],used[MAXN];
15 //girl表示每个点的匹配情况
16 //used表示改点有没有被标记过
17
18 void init()
19 {
20     tot = 0;
21     memset(head,-1,sizeof head);
22 }
23
24 void addedge(int u,int v)
25 {
26     tot++; //边的编号从1~n
27     e[tot].to = v; e[tot].next = head[u]; //next表示与这条边共点的上一条边
28     //如果next为-1，则表示该边无上一条边
29     head[u] = tot; //head[u]表示以u为起点的第一条边的编号
30 }
31
32 bool dfs(int u)
33 {
34     for(int i = head[u]; i != -1;i = e[i].next){
35         int v = e[i].to;
36         if(!used[v])
37         {
38             used[v] = 1;
39             if(!girl[v] || dfs(girl[v]))
40             {
41                 girl[v] = u;
42                 return true;
43             }
44         }
45     }
46     return false;
47 }
48
49 int solve()
50 {
51     int res = 0;
52     memset(girl,0,sizeof girl);
53     //点的编号 0 ~ uN-1
54     for(int u = 0; u < uN;u++)
55     {
56         memset(used,0,sizeof used);
57         if(dfs(u)) res++;
58     }
59     return res;

```

```

60 }
61
62 int main()
63 {
64     //读入图
65     printf("%d\n", solve());
66     return 0;
67 }

```

5.9 第 K 小问题

5.9.1 S 到 T 的第 K 短路

```

1 #include <cstdio>
2 #include <iostream>
3 #include <algorithm>
4 #include <cstring>
5 #include <queue>
6 #define rep(i,a,b) for(int i = a;i <= b;i++)
7 using namespace std;
8 const int M = 1e5+100;
9 const int N = 2000;
10 const int inf = 0x3f3f3f3f;
11
12 int n,m,s,t,k;
13 struct Edge{
14     int to,next,w;
15 }e[M],ef[M];
16 int head[N],headf[N],tot,totf;
17 int dis[N],book[N],dist[N];
18 struct Po{
19     int id; //到达哪一个点
20     int fw; //到该点已经走过的距离
21     int w; //到该点已经走过的距离+该点距离终点的距离
22 }tmp;
23
24 bool operator < (Po x,Po y) //优先队列只能重载 <
25 {
26     return x.w>y.w; //如果放在结构体内重载，需要加const
27 }
28 void init()
29 {
30     tot = 1; totf = 1;
31     memset(head,0,sizeof head);
32     memset(headf,0,sizeof headf);
33 }
34
35 void add(int x,int y,int w)
36 {
37     e[++tot].to = y; e[tot].next = head[x]; head[x] = tot; e[tot].w = w;
38 }
39
40 void addf(int x,int y,int w)
41 {
42     ef[++totf].to = y; ef[totf].next = headf[x]; headf[x] = totf; ef[totf].w = w;
43 }
44
45 void spfa(int x)
46 {

```

```

47     queue<int> q;
48     memset(book,0,sizeof book);
49     rep(i,1,n) dis[i] = inf;
50     while(!q.empty()) q.pop();
51     q.push(x);
52     book[x] = 1;
53     dis[x] = 0;
54     while(!q.empty())
55     {
56         int p = q.front();
57         for(int i = headf[p]; i ; i = ef[i].next)
58         {
59             if(dis[ef[i].to] > dis[p]+ef[i].w)
60             {
61                 dis[ef[i].to] = dis[p]+ef[i].w;
62                 if(book[ef[i].to] == 0)
63                 {
64                     q.push(ef[i].to);
65                     book[ef[i].to] = 1;
66                 }
67             }
68         }
69         book[p] = 0;
70         q.pop();
71     }
72 }
73
74 int bfs()
75 {
76     if(dis[s] == inf) return -1;      //如果不加这句话，就是wa，确保连通性
77     int val = 0;
78     priority_queue<Po> q;
79     while(!q.empty()) q.pop();
80     tmp/fw = 0; //走到id这个点目前所需要的距離
81     tmp.id = s;
82     tmp.w = tmp/fw+dis[tmp.id];
83     q.push(tmp);
84     if(t==s)k++; //这里也是一个大坑点
85     while(!q.empty())
86     {
87         Po p = q.top();
88         q.pop(); //优先队列，必须在push之前就pop，不然队列顶的那个点会被调到底下去
89         int id = p.id; //p所在的点
90         if(id == t) val++;
91         if(val == k) return p/fw; //求第k短路
92         for(int i = head[id]; i ; i = e[i].next) //加入新边
93         {
94             tmp/fw = p/fw+e[i].w;
95             tmp.id = e[i].to;
96             tmp.w = tmp/fw+dis[tmp.id]; //此处为估价函数
97             q.push(tmp);
98         }
99     }
100    if(val!=k){return -1;}
101 }
102
103 int main()
104 {
105     while(~scanf("%d%d",&n,&m))

```

```

106     {
107         init();
108         rep(i,1,m)
109         {
110             int x,y,z;
111             scanf("%d%d%d",&x,&y,&z);
112             add(x,y,z);
113             addf(y,x,z);
114         }
115         scanf("%d%d%d",&s,&t,&k);
116         spfa(t);
117         printf("%d\n",bfs());
118     }
119     return 0;
120 }
```

5.9.2 有向全图第 k 小路径

题意：给出一个 n 个点 m 条边的有向图，每条边都有一个权重。一共有 q 组询问，每次询问 k ，表示询问图中第 k 小的路径。 $(1 \leq n, m, q, k \leq 5 * 10^4)$

思路： q 组询问，我们离线下来，只用求出最大的那个 k 的答案即可。

然后一样的套路，定状态。一条路径的表示，无非是起点和终点， (u, v) ，但是如果这样定状态的话，会发现不好更新，难道更新 v 的所有出边，如果更新 v 的所有出边很容易 T ？

因此我们令 u 为 v 的前驱，状态定义为 (u, v, id, w) 表示 u 是 v 的前驱，且 v 是 u 的第 id 条出边，当前路径权重为 w 。

然后每次更新 u 的第 $id + 1$ 条出边，或者更新 v 的第一条出边即可，记得将每个点的出边排序。

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
6 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
7     << " , " << z1 << ":" << z2 << endl;
8 typedef long long ll;
9 typedef double db;
10 const db EPS = 1e-9;
11 const int N = 5*1e4+10;
12 using namespace std;
13 int n,m,qn,Q[N],maxn;
14 ll ans[N];
15 vector<pair<ll,int>> v[N];
16 struct Node{
17     int u,v,id;
18     ll w;
19     Node() {}
20     Node(int x,int y,int d,ll z):u(x),v(y),id(d),w(z) {}
21     bool operator < (Node xx) const {
22         return w > xx.w;
23     }
24 };
```

```

25 priority_queue<Node> q;
26
27 void solve(){
28     while(q.size()) q.pop();
29     int ct = 0;
30     rep(i,1,n) sort(v[i].begin(),v[i].end());
31     rep(i,1,n){
32         if(v[i].size())
33             q.push({i,v[i][0].second,0,v[i][0].first});
34     }
35 }
36 while(q.size()){
37     Node x = q.top(); q.pop();
38     ct++;
39     ans[ct] = x.w;
40     if(ct == maxn) break;
41     if(v[x.v].size()){
42         q.push({x.v,v[x.v][0].second,0,x.w+(v[x.v][0].first)});
43     }
44     if((int)v[x.u].size() > (int)(x.id+1)){
45         q.push({x.u,v[x.u][x.id+1].second,x.id+1,x.w-v[x.u][x.id].first+v[x.u][x.id+1].first});
46     }
47 }
48 rep(i,1,qn){
49     printf("%lld\n",ans[Q[i]]);
50 }
51 }
52
53 int main()
54 {
55     int _; scanf("%d",&_);
56     while(_--){
57         scanf("%d%d%d",&n,&m,&qn);
58         rep(i,0,n) v[i].clear();
59         rep(i,1,m){
60             int x,y; ll w; scanf("%d%d%lld",&x,&y,&w);
61             v[x].push_back(make_pair(w,y));
62         }
63         maxn = 0;
64         rep(i,1,qn){
65             scanf("%d",&Q[i]);
66             maxn = max(maxn,Q[i]);
67         }
68         solve();
69     }
70     return 0;
71 }

```

5.9.3 无向全图第 k 小最短路

题意: n 个点, m 条边的有权无向图, 询问图中第 k 大的最短路, 注意路径 (u, v) 被计算, 仅当这条路径是 (u, v) 之间的最短路, 保证有解。 $(2 \leq n, m \leq 2 * 10^5, 1 \leq k \leq 400)$

思路: 无向图中的问题, 主要难点在于如何才能避免算重。

我们注意到此题是两点之间的最短路, 而两点之间只会有一条最短路, 路径端点相同且长度相同算一条。因此我们将路径两点存入 map , 当且仅当路径两点第一次在一条路径中出现时, 才更新答案。

因此我们令 u 为 v 的前驱，状态定义为 (u, v, id, w) 表示 u 是 v 的前驱，且 v 是 u 的第 id 条出边，当前路径权重为 w 。

因此状态就是路径的起点和终点以及路径的权重，每次选择终点的出边进行状态更新即可。

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
6 ;
7 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
8 << " , " << z1 << ":" << z2 << endl;
9 typedef long long ll;
10 typedef double db;
11 const int N = 2*1e5+100;
12 const int M = 4*1e5+100;
13 const db EPS = 1e-9;
14 using namespace std;
15
16 int n,m,k;
17 vector<pair<ll,int> > v[N];
18 struct Node{
19     ll w; int u,v;
20     Node(ll a,int b,int c) : w(a), u(b), v(c) {}
21     bool operator < (Node xx) const {
22         return w > xx.w;
23     }
24 };
25 map<pair<int,int>,int> mp;
26 vector<Node> temp;
27 priority_queue<Node> q;
28
29 void dijsktra(){
30     int cnt = 0;
31     mp.clear();
32     while(q.size()) q.pop();
33     rep(i,1,n) q.push({0,i,i});
34     while(q.size()){
35         Node tp = q.top(); q.pop();
36         if(mp.find(make_pair(tp.u,tp.v)) != mp.end()) continue;
37         if(tp.u != tp.v && mp.find(make_pair(tp.u,tp.v)) == mp.end() && mp.find(
38             make_pair(tp.v,tp.u)) == mp.end()){
39             cnt++;
40             if(cnt == k){
41                 printf("%lld\n",tp.w);
42                 return;
43             }
44             mp[make_pair(tp.u,tp.v)] = 1;
45             int rt = 0;
46             for(pair<ll,int> &thp:v[tp.v]){
47                 rt++;
48                 if(rt+cnt > k) break;
49                 int y = thp.second;
50                 if(mp.find(make_pair(tp.u,y)) != mp.end()) continue;
51                 q.push({tp.w+thp.first, tp.u, y});
52             }
53         }
54     }
55 }
```

```

51     }
52 }
53
54 int main()
55 {
56     scanf("%d%d%d", &n, &m, &k);
57     rep(i, 1, m) {
58         int x, y; ll w; scanf("%d%d%lld", &x, &y, &w);
59         temp.push_back({w, y, x});
60     }
61     sort(temp.begin(), temp.end());
62     int siz = temp.size();
63     for(int i = siz - 1; i >= max(siz - 1 - k, 0); i--) {
64         int x = temp[i].u, y = temp[i].v, w = temp[i].w;
65         v[x].push_back(make_pair(w, y));
66         v[y].push_back(make_pair(w, x));
67     }
68     rep(i, 1, n) sort(v[i].begin(), v[i].end());
69     dijsktra();
70     return 0;
71 }
```

5.9.4 无向全图第 k 小团

题意：给出一个 n 个点的无向图，每个点都有一个权重，求出图中第 k 小的团，找不到输出 -1 。团的定义为完全图。
 $(1 \leq n \leq 100, 1 \leq k \leq 10^6)$

思路：此类的问题的思路比较统一，先定一个状态，然后再将这个状态丢入堆中，去更新其它状态。

我们定状态为团的权重，并用 $bitset$ 记录团中所有出现的点，并记录每个团中编号最大的节点。

我们更新时每次选一个编号比当前团中任意节点都大的一个节点加入团中。判断一个点是否可以加入团，只需判断这个点的邻接矩阵和记录团中点的 $bitset$ 进行与运算，如果最终结果仍然是团中点的 $bitset$ ，则证明当前点和团中所有点都有连边，因此加入这个点仍可以构成一个团。

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0); cin.tie(0); cout.tie(0)
3 #define rep(i, a, b) for(int i = a; i <= b; i++)
4 #define LOG1(x1, x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1, x2, y1, y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
6 #define LOG3(x1, x2, y1, y2, z1, z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
7 << " , " << z1 << ":" << z2 << endl;
7 typedef long long ll;
8 typedef double db;
9 const int N = 100 + 10;
10 const db EPS = 1e-9;
11 using namespace std;
12
13 char s[N];
14 int n, k;
15 ll w[N];
16 bitset<105> mp[N];
17 struct Node{
18     ll w; int id;
19     bitset<105> vis;
20     Node() {w = id = 0; vis.reset();}
```

```

21     bool operator < (Node xx) const {
22         return w > xx.w;
23     }
24 };
25
26 priority_queue<Node> q;
27
28 int main()
29 {
30     scanf("%d%d",&n,&k);
31     rep(i,1,n) scanf("%lld",&w[i]);
32     rep(i,1,n){
33         scanf("%s",s+1);
34         rep(j,1,n)
35             if(s[j] == '1') mp[i].set(j);
36     }
37     while(q.size()) q.pop();
38     Node empty; ll ans = 0;
39     q.push(empty);
40     while(q.size()){
41         Node x = q.top(); q.pop();
42         k--;
43         if(k == 0) {ans = x.w; break;}
44         rep(i,x.id+1,n)
45             if(x.vis[i] == 0 && (x.vis&mp[i]) == x.vis){
46                 Node nw = x;
47                 nw.w += w[i];
48                 nw.vis[i] = 1;
49                 nw.id = i;
50                 q.push(nw);
51             }
52     }
53     if(k != 0) printf("-1\n");
54     else printf("%lld\n",ans);
55     return 0;
56 }
```

5.10 简单图构造

题意：

给出 n 个点，再给出每个点的出度，问能否构造一个没有自环的简单图，如果可以判断构造是否唯一，输出构造方案。
 $(2 \leq n \leq 100)$

思路：

运用 *HavelHakimi* 定理，将点的度数排序，每次让度数最大的点向度数小的点连边，连完之后重新排序再次让度数最大的点向后连边，过程中不出现度数为负的点则可以构造。

然后问题就变成了如何判断是否有多种构造方式。我们只需在度数最大的点向后连边时，连最后一条边的时候查看最后一个点的度数是否与该点度数相同，如果相同则说明有多种构造方案。

```

1 #include<stdio.h>
2 #include<string.h>
3 #include<algorithm>
4 using namespace std;
5 const int N=105;
6 struct node
7 {
```

```

8     int d,id;
9 } e[N],e1[N]; //存点
10 int n,tot;
11 pair<int,int>p1[N*N],p2[N*N]; //存图的边
12 bool cmp(node a,node b) //按度从大到小排序
13 {
14     if(a.d==b.d) return a.id>b.id;
15     return a.d>b.d;
16 }
17 bool Havel_Hakimi() //判断是否可图化
18 {
19     tot=0;
20     for(int i=0;i<n-1;i++)
21     {
22         sort(e+i,e+n,cmp);
23         if(i+e[i].d>=n) return 0;
24         for(int j=i+1;j<=i+e[i].d;j++)
25         {
26             p1[tot++]=make_pair(e[i].id,e[j].id); //记录边
27             e[j].d--;
28             if(e[j].d<0) return 0;
29         }
30     }
31     if(e[n-1].d!=0) return 0;
32     return 1;
33 }
34 bool fun() //判断是否有其他成图方式
35 {
36     int flag=0;
37     int tot1=0;
38     for(int i=0;i<n-1;i++)
39     {
40         sort(e1+i,e1+n,cmp);
41         int tmp=e1[i].d+i;
42         if(tmp<n-1&&e1[tmp].d==e1[tmp+1].d&&e1[tmp].d!=0)
43         {
44             flag=1;
45             swap(e1[tmp].id,e1[tmp+1].id);
46         }
47         for(int j=i+1;j<=tmp;j++)
48         {
49             p2[tot1++]=make_pair(e1[i].id,e1[j].id);
50             e1[j].d--;
51         }
52     }
53     return flag;
54 }
55 int main()
56 {
57     while(~scanf("%d",&n))
58     {
59         int sum=0;
60         for(int i=0; i<n; i++)
61         {
62             scanf("%d",&e[i].d);
63             e1[i].id=e[i].id=i+1;
64             e1[i].d=e[i].d;
65             sum+=e[i].d;
66         }
}

```

```

67     if(sum%2) printf("IMPOSSIBLE\n"); //如果度数和为奇数一定不能成图，一条边连两个点
68
69 {
70     if(!Havel_Hakimi()) printf("IMPOSSIBLE\n"); //不可图化
71     else
72     {
73         if(fun()) //可图化且图不唯一，输出两个图的信息
74         {
75             printf("MULTIPLE\n");
76             printf("%d %d\n",n,tot);
77             for(int i=0;i<tot;i++)
78                 printf("%d%c",p1[i].first,i==tot-1?\n:' ');
79             for(int i=0;i<tot;i++)
80                 printf("%d%c",p1[i].second,i==tot-1?\n:' ');
81             printf("%d %d\n",n,tot);
82             for(int i=0;i<tot;i++)
83                 printf("%d%c",p2[i].first,i==tot-1?\n:' ');
84             for(int i=0;i<tot;i++)
85                 printf("%d%c",p2[i].second,i==tot-1?\n:' ');
86         }
87     else //图唯一
88     {
89         printf("UNIQUE\n");
90         printf("%d %d\n",n,tot);
91         for(int i=0;i<tot;i++)
92             printf("%d%c",p1[i].first,i==tot-1?\n:' ');
93         for(int i=0;i<tot;i++)
94             printf("%d%c",p1[i].second,i==tot-1?\n:' ');
95         if(tot==0) printf("\n\n"); //特判图只有一个节点没有边，要输出两个换行
96     }
97 }
98 }
99
100 return 0;
101 }
```

6 网络流

6.1 最大流

6.1.1 最小点覆盖

题意：

$N * N$ 的矩阵中有 K 个小行星，现在每行每列都有一个武器，可以消除此行或者此列中的所有小行星，问最少需要多少个武器可以将矩阵中所有小行星消除。

思路：

这是一个典型的最小点覆盖问题，可以用二分图匹配算法或者最大流算法进行解决。我们主要来讨论网络流的做法。

先来回顾一下最大流 *Dinic* 的基础性质，先在残量网络上 *BFS* 求出所有节点的层次，构造了一个分层图。然后在分层图上 *DFS* 寻找增广路，在回溯时实时更新剩余容量。当在残量网络中 S 不能到达 T 时，算法结束。

因此最大流这个算法有个很重要的性质，即 == 最大流 = 最小割 = 最大匹配 = 最小点覆盖 ==，基本覆盖了大部分最大流应用的题目。

现在我们再回过头来看这题如何建图，这题就是用最少的点覆盖所有的边，典型的最小点覆盖题目。因此我们将其转化为最大匹配，即每个点只能选择一次，最多选择多少条边。因此对于一个在 (i, j) 位置的行星，将左边第 i 个点和右边第 j 个点相连，图中所有边流量均设为 1，跑最大流即可。

最后再来一个不严谨的”最大匹配 = 最小点覆盖”的证明。即当图中选的边已经达到了最大匹配之后，一定不存在一条未选的边，边的两个端点同时都没有被选。并且由于每个点只能被选择一次，因此每条边都代表只选了一个点，而这些点的集合覆盖了所有的边。不太严谨的证明，只是有利于记忆。详细证明可以网上继续查阅题解。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #include <queue>
6 #define rep(i,a,b) for(int i = a;i <= b;i++)
7 using namespace std;
8 const int inf = 1<<29,N = 1000+10,M = 300500; //处理1e4-1e5规模的网络
9
10 struct Edge{
11     int to,next,v;
12 }e[M];
13 int n,m,s,t,k; //顶点个数 边数 源点 汇点
14 int head[N],tot,dis[N],mp[N][N];
15 queue<int> q;
16
17 void init() //千万别忘了初始化!
18 {
19     tot = 1; memset(head,0,sizeof head); //点的编号是2~n, 因为2^1 = 3, 3^1 = 2; 符合后续
        代码的操作
20 }
21
22 void add(int x,int y,int v)
23 {
24     e[++tot].to = y; e[tot].next = head[x]; e[tot].v = v; head[x] = tot;
25     e[++tot].to = x; e[tot].next = head[y]; e[tot].v = 0; head[y] = tot; //反向边与正向
        边的流量之和为v
26 }
27
28 bool bfs()
29 {

```

```

30     memset(dis,0,sizeof dis);
31     while(!q.empty()) q.pop();
32     q.push(s); dis[s] = 1;
33     while(!q.empty())
34     {
35         int x = q.front(); q.pop();
36         for(int i = head[x];i;i = e[i].next)
37         {
38             if(e[i].v && !dis[e[i].to]){
39                 q.push(e[i].to);
40
41                 dis[e[i].to] = dis[x]+1;
42                 if(e[i].to == t) return 1; //找到一条路就return
43             }
44         }
45     }
46     return 0;
47 }
48 }
49
50 int dinic(int x,int flow) //找增广路
51 {
52     if(x == t) return flow;
53     int rest = flow,k; //rest为输入的流量
54     for(int i = head[x];i && rest; i = e[i].next)
55     {
56         if(e[i].v && dis[e[i].to] == dis[x]+1){
57             k = dinic(e[i].to,min(rest,e[i].v));
58             if(!k) dis[e[i].to] = 0; //剪枝，去掉增广完毕的点
59             e[i].v -= k;
60             e[i^1].v += k; //反向边加上flow，相当于我们可以反悔从这条路流过
61             rest -= k; //k为能够被送出去的流量
62         }
63     }
64     return flow-rest; //总共被送出去了多少流量
65 }
66
67 int solve()
68 {
69     int flow = 0,maxflow = 0;
70     while(bfs())
71         while((flow = dinic(s,inf))) maxflow += flow;
72     return maxflow;
73 }
74
75 int main()
76 {
77     while(~scanf("%d%d",&n,&k))
78     {
79         init();
80         s = 1, t = 1+n+n+1;
81         rep(i,1,k){
82             int xx,yy; scanf("%d%d",&xx,&yy);
83             add(xx+1,yy+1+n,1);
84         }
85         rep(i,1,n) add(s,i+1,1), add(1+n+i,t,1);
86         printf("%d\n",solve());
87     }
88     return 0;

```

89 }

6.1.2 最大流 + 二分

题意：

给出了一个 $Y * X$ 的地图，在地图的四个边缘有门，用 ‘D’ 表示，‘X’ 表示障碍物即不能走，‘!’ 表示这个位置初始有一个人，现在地图中的所有人要逃出门外，每个 ‘!’ 这个点可以站好多人，但是出门的时候只能一个一个出，每个人移动一格的时间为 1，问最少需要多少时间，所有人可以撤出场地。如果不能撤出，输出 impossible。

思路：

因为是最少的时间，所以一开始想到的是最小割和最小费用最大流，然后发现均无法解决这个问题，因为难以解决每个时间每个门只能被经过一次这个问题。

后来发现我们可以 == 对每一个时刻的门分别建点 ==，然后使这个门的容量为 1，因此就可以满足每个时刻每个门只能出去一个人这个条件。

但是如果对门的所有时刻都建点的话，那么无疑跑出来的结果是不对的，即我们现在只能验证当前每一个时刻上限的情况下，所有人能不能出来。因此发现这个题的答案，即所需要的时间是单调的，因此可以进行二分答案。二分所有人出门的时间，然后用最大流建图来验证。

当二分的时间为 x 时，则对每个门都建 x 个点，然后每个人到达这个门所需的时间如果为 y 的话，则将这个人与这个门所有 $\geq y$ 的时刻连边。至此即可完成本题。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #include <map>
6 #include <queue>
7 #define rep(i,a,b) for(int i = a;i <= b;i++)
8 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
9 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
10 ;
11 using namespace std;
12 const int inf = 1<<29,N = 1e5+100,M = 5*1e6+100; //处理1e4-1e5规模的网络
13
14 struct Edge{
15     int to,next,v;
16 }e[M];
17 int n,m,s,t; //顶点个数 边数 源点 汇点
18 int head[N],tot,dis[N];
19 queue<int> q;
20 char mp[20][20];
21 int numdoor,mandoor[200][200],numman,hp[200],vis[300];
22 int dir[4][2] = {{1,0},{-1,0},{0,1},{0,-1}};
23
24 void bbfs(int x)
25 {
26     // printf("x:%d\n",x);
27     queue<pair<int,int> > qp;
28     while(qp.size()) qp.pop();
29     memset(vis,0,sizeof vis);
30     vis[x] = 1;
31     qp.push(make_pair(x,0));
32     while(qp.size()){
33         int xx = qp.front().first, dist = qp.front().second;
34         int yy = xx%m;
35         if(xx/m == t) return;
36         if(mp[xx/m][yy] == 'D') continue;
37         if(mp[xx/m][yy] == 'X') continue;
38         if(mp[xx/m][yy] == '!') continue;
39         if(dis[xx] <= dist) continue;
40         dis[xx] = dist;
41         for(int i = 0;i < 4;i++){
42             int nx = xx + dir[i][0];
43             int ny = yy + dir[i][1];
44             if(nx < 0 || nx >= N || ny < 0 || ny >= M) continue;
45             if(mp[nx][ny] == 'D') continue;
46             if(mp[nx][ny] == 'X') continue;
47             if(mp[nx][ny] == '!') continue;
48             if(dis[nx] >= dist) continue;
49             if(e[head[nx]].v >= m) continue;
50             e[head[nx]].v++;
51             e[e[head[nx]].next].to = nx;
52             e[e[head[nx]].next].v = 1;
53             e[e[head[nx]].next].next = head[nx];
54             head[nx] = e[head[nx]].next;
55             qp.push(make_pair(nx,dis[nx]+1));
56         }
57     }
58 }
```

```

34     qp.pop();
35     xx = xx/m+1;
36     rep(i,0,3){
37         int xxx = xx+dir[i][0], yyy = yy+dir[i][1];
38         int tp = (xxx-1)*m+yyy;
39         if(xxx < 1 || xxx > n || yyy < 1 || yyy > m || mp[xxx][yyy] == 'X')
        continue;
40         if(vis[tp]) continue;
41         if(mp[xxx][yyy] == 'D'){
42             // printf("oops!\n");
43             // LOG2("x",x,"tp",tp);
44             // LOG1("dist",dist);
45             mandoor[hp[x]][hp[tp]] = dist+1;
46             continue;
47         }
48         vis[tp] = 1;
49         qp.push(make_pair(tp,dist+1));
50     }
51 }
52 }
53
54 void init_solve()
55 {
56     memset(mandoor,0,sizeof mandoor);
57     numman = numdoor = 0;
58     memset(hp,0,sizeof hp);
59     rep(i,1,n)
60         rep(j,1,m)
61             if(mp[i][j] == '.') hp[(i-1)*m+j] = ++numman;
62             else if(mp[i][j] == 'D') hp[(i-1)*m+j] = ++numdoor;
63     rep(i,1,n)
64         rep(j,1,m)
65             if(mp[i][j] == '.'){
66                 // printf("i:%d,j:%d\n",i,j);
67                 b bfs((i-1)*m+j);
68             }
69 }
70
71 void init() //千万别忘了初始化!
72 {
73     tot = 1; memset(head,0,sizeof head); //点的编号是2~n, 因为 $2^1 = 3$ ,  $3^1 = 2$ ; 符合后续
        代码的操作
74 }
75
76 void add(int x,int y,int v)
77 {
78     e[++tot].to = y; e[tot].next = head[x]; e[tot].v = v; head[x] = tot;
79     e[++tot].to = x; e[tot].next = head[y]; e[tot].v = 0; head[y] = tot; //反向边与正向
        边的流量之和为v
80 }
81
82 bool bfs()
83 {
84     memset(dis,0,sizeof dis);
85     while(!q.empty()) q.pop();
86     q.push(s); dis[s] = 1;
87     while(!q.empty())
88     {
89         int x = q.front(); q.pop();

```

```

90         for(int i = head[x];i;i = e[i].next)
91     {
92         if(e[i].v && !dis[e[i].to]){
93             q.push(e[i].to);
94             dis[e[i].to] = dis[x]+1;
95             if(e[i].to == t) return 1; //找到一条路就return
96         }
97     }
98 }
99 return 0;
100}
101
102int dinic(int x,int flow) //找增广路
103{
104    if(x == t) return flow;
105    int rest = flow,k; //rest为输入的流量
106    for(int i = head[x];i && rest; i = e[i].next)
107    {
108        if(e[i].v && dis[e[i].to] == dis[x]+1){
109            k = dinic(e[i].to,min(rest,e[i].v));
110            if(!k) dis[e[i].to] = 0; //剪枝，去掉增广完毕的点
111            e[i].v -= k;
112            e[i^1].v += k; //反向边加上flow，相当于我们可以反悔从这条路流过
113            rest -= k; //k为能够被送出去的流量
114        }
115    }
116    return flow-rest; //总共被送出去了多少流量
117}
118
119
120int solve()
121{
122    int flow = 0,maxflow = 0;
123    while(bfs())
124        while(flow = dinic(s,inf)) maxflow += flow;
125    return maxflow;
126}
127
128void mainsolve()
129{
130    int l = 1, r = 200, ans = -1;
131    while(l <= r){
132        int mid = (l+r)>>1;
133        init();
134        s = 1, t = 1+numman+mid*numdoor+1;
135        rep(i,1,numman) add(s,i+1,1);
136        rep(i,1,numdoor)
137            rep(j,1,mid) add(1+numman+(i-1)*mid+j,t,1);
138        rep(i,1,numman)
139            rep(j,1,numdoor)
140                if(mandoor[i][j] <= mid && mandoor[i][j])
141                    rep(k,mandoor[i][j],mid) add(1+i,1+numman+(j-1)*mid+k,1);
142        if(solve() == numman) r = mid-1, ans = mid;
143        else l = mid+1;
144    }
145    if(ans == -1) printf("impossible\n");
146    else printf("%d\n",ans);
147}
148

```

```

149 int main()
150 {
151     int _; scanf("%d",&_);
152     while(_--)
153     {
154         scanf("%d%d",&n,&m);
155         rep(i,1,n) scanf("%s",mp[i]+1);
156         init_solve();
157         mainsolve();
158     }
159     return 0;
160 }
```

6.1.3 最大流路径输出

题意：

给出 n 个起重机，每个起重机有两个属性， $W[i]$ 表示这个起重机的重量， $L[i]$ 表示这个起重机能够拉起的最大重量（可以拉重物也可以拉起重机）。现在有 m 栋楼以及 m 个重物，要求给出每栋楼起重机的分配方案，使得每栋楼最后留下的起重机可以拉起对应的重物。 $(1 \leq n \leq 100, 1 \leq M \leq 100)$

思路：

比赛的时候以为所有的起重机都得用上... 然后就想到了费用流... 然后就是 *wa wa wa...* 其实这是一道比较简单的最大流问题，主要难点应该在路径输出。

我们先来考虑下如何建图。首先对于 n 个起重机进行拆点，拆点的目的是化边权为点权来限制流量。然后将源点 s 和所有 $W[i] = 0$ 的点的入点相连，流量为 1。每个拆开的点的入点和出点连一条流量为 1 的边。对于点 x 的出点与点 y 的入点， x 与 y 相连（流量为 1），当且仅当 $W[y] \leq L[x]$ 。

再将 m 个重物与汇点相连，流量为 1。最后对于每个重物，将所有 $L[i]$ 大于重物重量的点与重物相连，流量为 1。然后直接跑最大流即可。如果最大流为 m ，则输出路径，如果小于 m ，则输出 *impossible*。

最后考虑本题最关键的步骤，求出最大流之后如何输出路径。

对于 m 个点，通过反向边往回跑，仅当反向边流量不为 0，而正向边流量为 0 时才走这条边，跑到源点即停止，进行 m 遍 *dfs* 即可。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #include <queue>
6 #include <vector>
7 #define rep(i,a,b) for(int i = a;i <= b;i++)
8 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
9 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
10 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
    << " , " << z1 << ":" << z2 << endl;
11 using namespace std;
12 const int inf = 1<<29,N = 1000,M = 1e5; //处理1e4-1e5规模的网络
13
14 struct Edge{
15     int to,next,v,f;
16 }e[M];
17 int n,m,s,t; //顶点个数 边数 源点 汇点
18 int head[N],tot,dis[N],L[N],W[N],T[N];
19 queue<int> q;
```

```

20 vector<int> ans[N];
21
22 void init() //千万别忘了初始化!
23 {
24     tot = 1; memset(head,0,sizeof head); //点的编号是2~n, 因为 $2^1 = 3$ ,  $3^1 = 2$ ; 符合后续
        代码的操作
25 }
26
27 void add(int x,int y,int v)
28 {
29     e[++tot].to = y; e[tot].next = head[x]; e[tot].v = v; head[x] = tot; e[tot].f = 1;
30     e[++tot].to = x; e[tot].next = head[y]; e[tot].v = 0; head[y] = tot; e[tot].f = -1;
        //反向边与正向边的流量之和为v
31 }
32
33 bool bfs() //判断是否有增广路径
34 {
35     memset(dis,0,sizeof dis);
36     while(!q.empty()) q.pop();
37     q.push(s); dis[s] = 1;
38     while(!q.empty())
39     {
40         int x = q.front(); q.pop();
41         for(int i = head[x];i;i = e[i].next)
42         {
43             if(e[i].v && !dis[e[i].to]){
44                 q.push(e[i].to);
45                 dis[e[i].to] = dis[x]+1;
46                 if(e[i].to == t) return 1; //找到一条路就return
47             }
48         }
49     }
50     return 0;
51 }
52
53 int dinic(int x,int flow) //dfs找增广路, 找到一条路, 就把增广的流量加到答案中
54 {
55     if(x == t) return flow;
56     int rest = flow,k; //rest为输入的流量
57     for(int i = head[x];i && rest; i = e[i].next)
58     {
59         if(e[i].v && dis[e[i].to] == dis[x]+1){
60             k = dinic(e[i].to,min(rest,e[i].v));
61             if(!k) dis[e[i].to] = 0; //剪枝, 去掉增广完毕的点
62             e[i].v -= k;
63             e[i^1].v += k; //反向边加上flow, 相当于我们可以反悔从这条路流过
64             rest -= k; //k为能够被送出去的流量
65         }
66     }
67     return flow-rest; //总共被送出去了多少流量
68 }
69
70 int solve()
71 {
72     int flow = 0,maxflow = 0;
73     while(bfs())
74         while((flow = dinic(s,inf))) maxflow += flow; //while 循环内容 —— 判断是否还有
            增广路
75     return maxflow;

```

```

76 }
77
78 void dfs(int x,int flag){ //路径输出
79     if(1 < x && x <= 1+n) ans[flag].push_back(x-1);
80     if(x == s) return;
81     for(int i = head[x]; i; i = e[i].next)
82         if(e[i^1].v == 0 && e[i].to <= 1+2*n && e[i].f == -1) dfs(e[i].to,flag);
83 }
84
85 int main()
86 {
87     scanf("%d",&n); tot = 1;
88     rep(i,1,n) scanf("%d%d",&W[i],&L[i]);
89     scanf("%d",&m);
90     rep(i,1,m) scanf("%d",&T[i]);
91     s = 1, t = 2+2*n+m;
92     rep(i,1,m) add(1+2*n+i,t,1);
93     rep(i,1,n)
94         if(W[i] == 0) add(s,i+1,1);
95     rep(i,1,n) add(i+1,1+n+i,1);
96     rep(i,1,n)
97         rep(j,1,n)
98             if(i != j && L[i] >= W[j]) add(i+1+n,1+j,1);
99     rep(i,1,m)
100    rep(j,1,n)
101        if(L[j] >= T[i]) add(1+n+j,1+2*n+i,1);
102    int maxflow = solve();
103    if(maxflow != m) printf("impossible\n");
104    else{
105        rep(i,1,m)
106            dfs(1+2*n+i,i);
107        rep(i,1,m){
108            for(int j = ans[i].size()-1; j >= 0; j--){
109                printf("%d",ans[i][j]);
110                if(j == 0) printf("\n");
111                else printf(" ");
112            }
113        }
114    }
115    return 0;
116 }

```

6.1.4 最大权闭合子图

一、算法介绍

有一个有向图，每一个点都有一个权值（可以为正或负或0），选择一个权值和最大的子图，使得每个点的后继都在子图里面，这个子图就叫最大权闭合子图。

能选的子图有 $\emptyset, 4, 3, 4, 2, 4, 1, 2, 3, 4$ ，它们的权值分别为 0, -1, 5, -6, 4。
所以最大权闭合子图为 3, 4，权值为 5。

· 解法

这个问题可以转化为最小割问题，用网络流解决。

从源点 s 向每个正权点连一条容量为权值的边，每个负权点向汇点 t 连一条容量为权值的绝对值的边，有向图原来的边容量全部为无限大。

求它的最小割，割掉后，与源点 s 连通的点构成最大权闭合子图，权值为（正权值之和-最小割）。

· 如何理解

割掉一条边的含义

由于原图的边都是无穷大，那么割边一定是与源点 s 或汇点 t 相连的。

割掉 s 与 i 的边，表示不选择 i 点作为子图的点；

割掉 i 与 t 的边，表示选择 i 点为子图的点。

如果 s 与 i 有边，表示 i 存在子图中；

如果 i 与 t 有边，表示 i 不存在于子图中。

· 合法性

只有 s 与 t 不连通时，才能得到闭合子图。

如果 s 与 t 连通，则存在点 i,j，使得 s 到 i 有边，i 到 j 连通，j 到 t 有边，所以 j 一定是 i 的后继，但选择了 i，没有选择 j，不是闭合子图。

如果 s 与 t 不连通，选择了正权点 i，一定选择了 i 后继中的所有负权点。设 j 是 i 的后继中的正权点，则割掉 s 到 j 的边是没有意义的，最小割不会割掉它，则 j 一点被选中，所以 i 的所有后继都被选中，符合闭合图的定义。

· 最优性

最小割 = (不选的正权之和 + 要选的负权绝对值之和)

最大权闭合子图 = (正权之和 - 不选的正权之和 - 要选的负权绝对值之和) = 正权值和 - 最小割

因为正权值和是定值，而最小割保证值最小，所以最大权闭合子图一定最优。

二、例题

题意：一共有 m 个实验， n 个仪器，每个实验完成之后能得到 p_i 的报酬，每个仪器购买需要 v_i 的花费，每个实验需要依托某几个对应的器材才能完成。现要求给出购买仪器以及做实验的方案，使得净收益最大，给出具体方案与结果。 $(1 \leq n, m \leq 50)$

思路：以往普通费用流的问题，其最终的总流量都是固定的，因此可以依据总流量固定这一特点，对流量赋予费用来求解。

而此题的难点恰好在此，涉及了费用，但最终总流量不固定，如果总流量不固定是很难用费用流算法来建图的，因此我们来考虑最大流。

而最大流通常解决的都是总流量问题，或者有时二分一下答案用最大流来判断是否合法，基本不涉及费用的问题，由此可以判断此题非之前遇到的普通最大流问题，是根据一种特殊的建图方式用最大流来求解出了费用问题。

其实这是一道最大权闭合子图问题，应用最大流最小割定理，通过求解最大流来获得最小割。建图方式如下：

1. 最大权闭合子图问题，针对的是在一个有向图中，每个点都有权值，现要在这个有向图中找到一个闭合子图，即子图中不存在出边不在子图中的点，使该子图中点权值和最大。

2. 具体建图方式则为，有向图中的边流量赋为无穷，源点连向权值为正的点，流量为其权值；权值为负的点连向汇点，流量为其权值绝对值。最终的答案为 (正权值之和 - 最小割)

证明该定理的话，可以通过最小割只能断源点的连边或者汇点的连边来证明，即放弃正权点，或者接纳负权点。最大流结束后，仍与源点相连的点极为最大权闭合子图中的点。

如上建图即可跑出答案，最后的方案输出即 bfs 一遍找到仍与源点相连的点，此处可以依据最大流最后 bfs 的 dis 数组来判断。

总结：通过此题我们学会了一个用最大流算法解决总流量不固定时的最大收益问题。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #include <queue>
6 #define rep(i,a,b) for(int i = a;i <= b;i++)
7 using namespace std;
```

```

8 const int inf = 1<<29,N = 2000+10,M = 300500; //处理1e4-1e5规模的网络
9
10 struct Edge{
11     int to,next,v;
12 }e[M];
13 int n,m,s,t; //顶点个数 边数 源点 汇点
14 int head[N],tot,dis[N],mp[N][N];
15 queue<int> q;
16
17 void init() //千万别忘了初始化!
18 {
19     tot = 1; memset(head,0,sizeof head); //点的编号是2~n, 因为 $2^1 = 3$ ,  $3^1 = 2$ ; 符合后续
        代码的操作
20 }
21
22 void add(int x,int y,int v)
23 {
24     e[++tot].to = y; e[tot].next = head[x]; e[tot].v = v; head[x] = tot;
25     e[++tot].to = x; e[tot].next = head[y]; e[tot].v = 0; head[y] = tot; //反向边与正向
        边的流量之和为v
26 }
27
28 bool bfs()
29 {
30     memset(dis,0,sizeof dis);
31     while(!q.empty()) q.pop();
32     q.push(s); dis[s] = 1;
33     while(!q.empty())
34     {
35         int x = q.front(); q.pop();
36         for(int i = head[x];i;i = e[i].next)
37         {
38             if(e[i].v && !dis[e[i].to]){
39                 q.push(e[i].to);
40                 dis[e[i].to] = dis[x]+1;
41                 if(e[i].to == t) return 1; //找到一条路就return
42             }
43         }
44     }
45     return 0;
46 }
47
48 int dinic(int x,int flow) //找增广路
49 {
50     if(x == t) return flow;
51     int rest = flow,k; //rest为输入的流量
52     for(int i = head[x];i && rest; i = e[i].next)
53     {
54         if(e[i].v && dis[e[i].to] == dis[x]+1){
55             k = dinic(e[i].to,min(rest,e[i].v));
56             if(!k) dis[e[i].to] = 0; //剪枝, 去掉增广完毕的点
57             e[i].v -= k;
58             e[i^1].v += k; //反向边加上flow, 相当于我们可以反悔从这条路流过
59             rest -= k; //k为能够被送出去的流量
60         }
61     }
62     return flow-rest; //总共被送出去了多少流量
63 }
64

```

```

65 int solve()
66 {
67     int flow = 0, maxflow = 0;
68     while(bfs())
69         while((flow = dinic(s,inf))) maxflow += flow;
70     return maxflow;
71 }
72
73 vector<int> v1, v2;
74 string S;
75
76 int main()
77 {
78     scanf("%d%d", &m, &n); getchar();
79     init();
80     s = n+m+1, t = n+m+2;
81     int ans = 0;
82     rep(i, 1, m){
83         getline(cin, S);
84         int len = S.length(), cnt = 0, base = 0;
85         rep(j, 0, len-1){
86             if(S[j] >= '0' && S[j] <= '9'){
87                 if(base == 0) cnt++;
88                 base = base*10+S[j]-'0';
89             }
90             else{
91                 if(cnt == 1) add(s, i, base), ans += base;
92                 else add(i, m+base, inf);
93                 base = 0;
94             }
95         }
96         if(cnt == 1) add(s, i, base);
97         else add(i, m+base, inf);
98     }
99     rep(i, 1, n){
100        int v; scanf("%d", &v);
101        add(m+i, t, v);
102    }
103    ans -= solve();
104    rep(i, 1, m+n)
105        if(dis[i] != 0){
106            if(i <= m) v1.push_back(i);
107            else v2.push_back(i-m);
108        }
109    int sz1 = v1.size(), sz2 = v2.size();
110    rep(i, 0, sz1-1) printf("%d%c", v1[i], " \n"[i==sz1-1]);
111    rep(i, 0, sz2-1) printf("%d%c", v2[i], " \n"[i==sz2-1]);
112    printf("%d\n", ans);
113    return 0;
114 }

```

6.1.5 最小路径覆盖

题意：

给出一个 n 个点, m 条边的 DAG, 现要在该图中找到最少的不相交路径, 求出具体路径数。 $(1 \leq n \leq 150, 1 \leq m \leq 6000)$

思路：

最小路径覆盖问题, 如果是第一次遇到该问题, 可以先排除费用流的做法, 因为总流量不固定, 也正因为总流量不固定,

我们采取最大流算法。

将一个点拆成两个点，若存在边 (x, y) ，则将左边的 x 连向右边的 y ，最后答案为 $ans = n - maxflow$ 。接下来给出该算法的大致证明，在构建的图中，若选择了边 (x, y) ，则意味着在原图中将这两点放在了一条路径中，相当于这两点所在的路径发生了合并，因此总路径数 -1 ，即最终 $ans = n - maxflow$ 。

此题还可以扩展到最少相交路径问题（POJ-2594）。而处理最少相交路径只需要将连边条件 (x, y) 从原图中存在边 (x, y) 改成原图中 (x, y) 可达即可，即可以跨越一个点进行路径合并。

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #include <queue>
6 #define rep(i,a,b) for(int i = a;i <= b;i++)
7 using namespace std;
8 const int inf = 1<<29,N = 2000+10,M = 300500; //处理1e4-1e5规模的网络
9
10 struct Edge{
11     int to,next,v;
12 }e[M];
13 int n,m,s,t; //顶点个数 边数 源点 汇点
14 int head[N],tot,dis[N],mp[N][N];
15 queue<int> q;
16
17 void dbg() {cout << "\n";}
18 template<typename T, typename... A> void dbg(T a, A... x) {cout << a << ' '; dbg(x...)}
19 #define logs(x...) {cout << #x << " -> "; dbg(x);}
20
21 void init() //千万别忘了初始化!
22 {
23     tot = 1; memset(head,0,sizeof head); //点的编号是2~n, 因为 $2^1 = 3$ ,  $3^1 = 2$ ; 符合后续
24     代码的操作
25 }
26 void add(int x,int y,int v)
27 {
28     e[++tot].to = y; e[tot].next = head[x]; e[tot].v = v; head[x] = tot;
29     e[++tot].to = x; e[tot].next = head[y]; e[tot].v = 0; head[y] = tot; //反向边与正向
30     边的流量之和为v
31 }
32 bool bfs()
33 {
34     memset(dis,0,sizeof dis);
35     while(!q.empty()) q.pop();
36     q.push(s); dis[s] = 1;
37     while(!q.empty())
38     {
39         int x = q.front(); q.pop();
40         for(int i = head[x];i;i = e[i].next)
41         {
42             if(e[i].v && !dis[e[i].to]){
43                 q.push(e[i].to);
44                 dis[e[i].to] = dis[x]+1;
45                 if(e[i].to == t) return 1; //找到一条路就return
46             }
47         }
48     }
49 }
```

```

48     }
49     return 0;
50 }
51
52 int dinic(int x,int flow) //找增广路
53 {
54     if(x == t) return flow;
55     int rest = flow,k; //rest为输入的流量
56     for(int i = head[x];i && rest; i = e[i].next)
57     {
58         if(e[i].v && dis[e[i].to] == dis[x]+1){
59             k = dinic(e[i].to,min(rest,e[i].v));
60             if(!k) dis[e[i].to] = 0; //剪枝，去掉增广完毕的点
61             e[i].v -= k;
62             e[i^1].v += k; //反向边加上flow，相当于我们可以反悔从这条路流过
63             rest -= k; //k为能够被送出去的流量
64         }
65     }
66     return flow-rest; //总共被送出去了多少流量
67 }
68
69 int solve()
70 {
71     int flow = 0,maxflow = 0;
72     while(bfs())
73         while((flow = dinic(s,inf))) maxflow += flow;
74     return maxflow;
75 }
76
77 int to[N],deg[N];
78
79 int main()
80 {
81     scanf("%d%d",&n,&m);
82     init();
83     s = 2*n+1, t = 2*n+2;
84     int ans = 0;
85     rep(i,1,m){
86         int x,y; scanf("%d%d",&x,&y);
87         add(x,n+y,1);
88     }
89     rep(i,1,n) add(s,i,1), add(i+n,t,1);
90     ans = solve();
91     rep(i,2,tot){
92         int x = e[i^1].to, y = e[i].to;
93         if(x <= n && y <= 2*n && e[i].v == 0){
94             to[x] = y-n;
95             deg[y-n]++;
96         }
97     }
98     rep(i,1,n){
99         if(!deg[i]){
100             int pos = i;
101             while(pos){
102                 printf("%d",pos);
103                 pos = to[pos];
104                 if(pos == 0) printf("\n");
105                 else printf(" ");
106             }
107         }
108     }
109 }
```

```

107         }
108     }
109     printf("%d\n",n-ans);
110     return 0;
111 }
```

6.2 费用流

```

1 #include <cstdio>
2 #include <iostream>
3 #include <algorithm>
4 #include <cstring>
5 #include <queue>
6 #define rep(i,a,b) for(int i = a;i <= b;i++)
7 using namespace std;
8 const int N = 1100, M = 1e5+100;
9
10 struct Edge{
11     int to,next,cap,cost; //cap为该边的容量, cost为该边的花费
12 }e[M];
13 int head[N],tot,s,t; //tot记录边数, s为源点, t为汇点
14 int dis[N],incf[N],pre[N],vis[N]; //dis为到每一个点的最短距离, incf为每一个点流入的流量
15 //pre为每一个点的最短距离是由哪一条边更新而来, vis标记该点有没有被访问过
16 int n,m,maxflow,ans;
17
18 void init()
19 {
20     tot = 1; //从2开始“成对存储”, 2和3是一对, 4和5是一对
21     memset(head,0,sizeof head);
22 }
23
24 void add(int x,int y,int z,int c) //z为容量, c为花费
25 {
26     //正向边, 初始容量为z, 单位费用为c
27     e[++tot].to = y; e[tot].next = head[x]; head[x] = tot; e[tot].cap = z; e[tot].cost
28     = c;
29     //反向边, 初始容量为0, 单位费用为-c, 与正向边“成对存储”
30     e[++tot].to = x; e[tot].next = head[y]; head[y] = tot; e[tot].cap = 0; e[tot].cost
31     = -c;
32 }
33
34 bool spfa()
35 {
36     //用最短路找增广路
37     queue<int> q;
38     //建议不要用memset赋值, 太容易出错了, 当dis设置为long long时
39     //tmp还是一个int, 就特别容易出错
40     fill(dis,dis+N,1e8);
41     //memset(dis,0x3f,sizeof dis); //memset是按字节赋值, 一个字节8位, 一个16进制位表示4个二进制
42     //位, 因此0x3f为一个字节
43     //此处dis赋值为inf, 寻找最短路(最小费用流)
44     //如果dis赋值为0xcf, 则为寻找最长路(最大费用流)
45     //0xcf为11001111, 因此按位赋值之后为负数
46     memset(vis,0,sizeof vis); //每个点都没被访问过
47     q.push(s); dis[s] = 0; vis[s] = 1;
48     incf[s] = 1<<30; //incf为每一个点流入的流量
49     while(q.size())
50     {
51         int u = q.front(); q.pop();
52         for(int i = head[u]; i != -1; i = e[i].next)
53         {
54             int v = e[i].to;
55             if(dis[v] > dis[u] + e[i].cost && e[i].cap > 0)
56             {
57                 dis[v] = dis[u] + e[i].cost;
58                 pre[v] = i;
59                 if(vis[v] == 0)
60                 {
61                     q.push(v);
62                     vis[v] = 1;
63                 }
64             }
65         }
66     }
67     if(dis[t] == 1e8)
68     {
69         maxflow = 0;
70         return false;
71     }
72     else
73     {
74         maxflow = incf[s];
75         int cur = t;
76         while(cur != s)
77         {
78             int i = pre[cur];
79             int v = e[i].to;
80             e[i].cap -= 1;
81             e[i^1].cap += 1;
82             cur = v;
83         }
84     }
85     return true;
86 }
```

```

48     int x = q.front(); vis[x] = 0; q.pop();
49     for(int i = head[x]; i ; i = e[i].next) //通过与x相连的各边遍历与x相连的点
50     {
51         if(!e[i].cap) continue; //该边剩余流量为0, 不在残量网络中, 无法通过
52         int y = e[i].to;
53         if(dis[y] > dis[x] + e[i].cost) //如果此处为 <, 则为最长路算法【最大费用流】
54         {
55             dis[y] = dis[x]+e[i].cost;
56             incf[y] = min(incf[x],e[i].cap);
57             pre[y] = i; //记录y点的最短路是由哪一条边更新而来的
58             if(!vis[y]) vis[y] = 1, q.push(y);
59         }
60     }
61 }
62 // int tmp = 0x3f3f3f3f;
63 if(dis[t] == 1e8) return false; //汇点不可达, 已求出最大流
64 return true;
65 }
66
67 void update()
68 {
69     int x = t;
70     while(x!=s)
71     {
72         int i = pre[x];
73         e[i].cap -= incf[t];
74         e[i^1].cap += incf[t];
75         x = e[i^1].to; //相反边的终点, 即为x的上一个点
76     }
77     maxflow += incf[t];
78     ans += dis[t]*incf[t]; //dis[t]为这条最短路上的总花费, incf为这条最短路上的流
79 }
80
81 void solve()
82 {
83     ans = 0; maxflow = 0;
84     while(spfa()) update();
85     printf("%d\n",ans);
86 }
87
88 int main()
89 {
90     while(~scanf("%d%d",&n,&m))
91     {
92         //建图过程
93         s = 1; t = 2+n;
94         init();
95         add(s,2,0);
96         add(n+1,t,2,0);
97         rep(i,1,m){
98             int a,b,c;
99             scanf("%d%d%d",&a,&b,&c);
100            add(a+1,b+1,1,c); add(b+1,a+1,1,c);
101        }
102        solve();
103    }
104    return 0;
105 }
106

```

```
107 /*  
108 常见错误：  
109 1.没有init();  
110 2.spfa()函数部分dis是long long型的，用memset赋值，再用int型的tmp比较...  
111 3.建议spfa()函数部分的dis用fill来进行赋值【不易出错】  
112 */
```

7 计算几何

7.1 计算几何板子

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <vector>
5 #include <cmath>
6 #include <algorithm>
7 #define rep(i,a,b) for(int i = a; i <= b; i++)
8 #define LOG1(x) cout << "x: " << x << endl;
9 #define LOG2(x,y) cout << "x: " << x << ", y: " << y << endl;
10#define pi acos(-1.0)
11#define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y)) //向量(p1,p2)
    与(p1,p3)叉乘
12#define crossOp(p1,p2,p3) sign(cross(p1,p2,p3)) //判断正负, 顺时针为负, 为0则代表三点共线
13using namespace std;
14
15//实数比较
16typedef double db;
17const db EPS = 1e-9;
18inline int sign(db a) {return a < -EPS ? -1 : a > EPS;} //返回-1表示a < 0, 1表示a > 0, 0
    表示a = 0
19inline int cmp(db a, db b) {return sign(a-b); } //返回-1表示a < b, 1表示a > b, 0表示 a==b
20
21//点类
22struct Point {
23    db x,y;
24    Point() {}
25    Point(db _x, db _y) : x(_x), y(_y) {}
26    Point operator+(Point p) { return {x+p.x, y+p.y}; }
27    Point operator-(Point p) { return {x-p.x, y-p.y}; }
28    Point operator*(db d) { return {x*d, y*d}; }
29    Point operator/(db d) { return {x/d, y/d}; }
30    Point rotleft() { return Point(-y,x); } //逆时针旋转90度
31    Point rotright() { return Point(y,-x); } //顺时针旋转90度
32    db dot(Point p) { return x*p.x+y*p.y; } //点积
33    db det(Point p) { return x*p.y-y*p.x; } //叉积
34    Point rot(db an) { return {x*cos(an)-y*sin(an),x*sin(an)+y*cos(an)}; } //旋转
35    db abs() { return sqrt(abs2()); }
36    db abs2() { return x*x+y*y; }
37    db disto(Point p) { return (*this-p).abs(); }
38    //此时在x负半轴上的点, 排序结果是最小的。如果去掉sign(x)>=0, 则排序结果是最大的
39    int quad() const { return sign(y) == 1 || (sign(y) == 0 && sign(x) >= 0); } //判断该
        点是否在x轴上方或x轴上
40
41    bool operator<(Point p) const {
42        int c = cmp(x, p.x);
43        if (c) return c == -1; //先判断x大小
44        return cmp(y, p.y) == -1; //再判断y大小
45    }
46    bool operator==(Point p) const {
47        return cmp(x, p.x) == 0 && cmp(y, p.y) == 0;
48    }
49    bool operator!=(Point p) const{
50        return (cmp(x, p.x) || cmp(y,p.y));
51    }
52};

```

```

53
54 db area(vector<Point> ps){ //凸包面积
55     db ret = 0; rep(i,0,ps.size()-1) ret += ps[i].det(ps[(i+1)%ps.size()]);
56     return ret/2;
57 }
58
59 db perimeter(vector<Point> ps){ //凸包周长
60     db ret = 0; rep(i,0,ps.size()-1) ret += ps[i].disTo(ps[(i+1)%ps.size()]);
61     return ret;
62 }
63
64 db dot(Point A, Point B, Point C){ //三点点积
65     return (B-A).dot(C-A);
66 }
67
68 db rad(Point p1, Point p2){ //返回两个向量的夹角, 范围为-2*pi~2*pi, 顺时针转为负数, 逆时针转为正
69     //数
70     return atan2l(p1.det(p2),p1.dot(p2));
71     //取绝对值就是旋转角度, 否则会有正负区别, 返回幅度制
72 }
73 /*
74 坐标系变换 —— 选中点集中的两点, 两点构成的向量为x轴
75 Point p1 = {AA[2].x-AA[1].x,AA[2].y-AA[1].y}; //选定两点作为向量
76 rep(i,1,n){
77     Point p2 = {AA[i].x-AA[1].x,AA[i].y-AA[1].y};
78     db ang = rad(p1,p2); //返回两个向量角度
79     db len = p2.abs();
80     Point tmp = {len*cos(ang),len*sin(ang)}; //得到新坐标系下的点坐标
81     mp[tmp] = 1; //可以用map存变更坐标轴之后的点
82 }
83 */
84
85 //求凸包
86 vector<Point> convexHull(vector<Point> ps) {
87     int n = ps.size(); if(n <= 1) return ps;
88     sort(ps.begin(),ps.end());
89     vector<Point> qs(n*2); int k = 0;
90     for(int i = 0; i < n; qs[k++] = ps[i++])
91         while(k > 1 && cross0p(qs[k-2],qs[k-1],ps[i]) <= 0) --k; //把 <= 改成 <, 即可
92         将凸包边上的点也包括在凸包中, 不稳定凸包问题
93     for(int i = n-2, t = k; i >= 0; qs[k++] = ps[i--])
94         while(k > t && cross0p(qs[k-2],qs[k-1],ps[i]) <= 0) --k;
95     qs.resize(k-1);
96     return qs;
97 }
98
99 //最小矩形覆盖
100 db minRectangleCover(vector<Point> ps){
101     //凸包点集顺序按逆时针
102     int n = ps.size();
103     if(n < 3) return 0.0;
104     ps.push_back(ps[0]);
105     db ans = -1;
106     int r = 1, p = 1, q;
107     for(int i = 0; i < n; i++){
108         //求出离边 ps[i]-ps[i+1] 最远的点 r
109         while(sign(cross(ps[i],ps[i+1],ps[r+1])-cross(ps[i],ps[i+1],ps[r])) >= 0) //叉积
110             最大即为到点r到ps[i+1]-ps[i]这条边的距离最大

```

```

109         r = (r+1)%n;
110        //卡出 ps[i]-ps[i+1] 方向上正向 n 最远的点 p
111        while(sign(dot(ps[i],ps[i+1],ps[p+1])-dot(ps[i],ps[i+1],ps[p])) >= 0) //正向最
112        远即为点积最大
113        p = (p+1)%n;
114        if(i == 0) q = p;
115        //卡出 ps[i]-ps[i+1] 方向上负向最远的点 q
116        while(sign(dot(ps[i],ps[i+1],ps[q+1]) - dot(ps[i],ps[i+1],ps[q])) <= 0) //负向最
117        大即为点积最小
118        q = (q+1)%n;
119        db d = ps[i].disTo(ps[i+1]); //线段长度
120        d = d*d;
121        //叉积求出高，点积求出底边
122        db tmp = cross(ps[i],ps[i+1],ps[r]) *
123            (dot(ps[i],ps[i+1],ps[p])-dot(ps[i],ps[i+1],ps[q]))/d;
124        if(ans < 0 || ans > tmp) ans = tmp;
125    }
126    return ans;
127 }
128
129 bool cmp1(Point a, Point b){ //按照角度排序，第四象限-第三象限-第二象限-第一象限
130     if(a.quad() != b.quad()) return a.quad() < b.quad();
131     else return sign(a.det(b)) > 0;
132 }
133
134 bool cmp2(Point a, Point b){ //第二种极角排序方式，用角度直接排
135     return a.ang < b.ang; //tp[j].ang = atan2(am[j].y-am[i].y,am[j].x-am[i].x)，返回幅
136     度制，范围是-pi~pi
137 }
138
139 struct Line {
140     Point s,e;
141     Line() {}
142     Line(Point _s, Point _e) : s(_s), e(_e) {}
143     bool operator == (Line v) { return (s == v.s) && (e == v.e); }
144     // 根据一个点和倾斜角 angle 确定直线，0 <= angle < pi
145     Line(Point p, double angle) {
146         s = p;
147         if(sign(angle-pi/2) == 0) e = (s+Point(0,1));
148         else e = (s+Point(1,tan(angle)));
149     }
150     // ax+by+c = 0
151     Line(db a, db b, db c){
152         if(!sign(a)) s = Point(0,-c/b), e = Point(1,-c/b);
153         else if(!sign(b)) s = Point(-c/a,0), e = Point(-c/a,1);
154         else s = Point(0,-c/b), e = Point(1,(-c-a)/b);
155     }
156     void adjust() { if(e<s) swap(s,e); }
157     db length() { return s.disTo(e); } //求线段长度
158     db angle() { //返回直线倾斜角 0 <= angle < pi, 弧度制
159         db k = atan2(e.y-s.y,e.x-s.x);
160         if(sign(k) < 0) k += pi;
161         if(sign(k-pi) == 0) k -= pi;
162         return k;
163     }
164     Point crosspoint(Line v){ //求两直线交点
165         db a1 = (v.e-v.s).det(s-v.s);
166         db a2 = (v.e-v.s).det(e-v.s);
167         return Point((s.x*a2-e.x*a1)/(a2-a1),(s.y*a2-e.y*a1)/(a2-a1));

```

```

165     }
166     // 点和直线关系, 1在上方, 2在下方, 3在直线上
167     int relation(Point p) {
168         int c = sign((p-s).det(e-s));
169         if(c < 0) return 1;
170         else if(c > 0) return 2;
171         else return 3;
172     }
173     // 点在线段上的判断
174     bool pointonseg(Point p) { return sign((p-s).det(e-s)) == 0 && sign((p-s).dot(p-e))
175         <= 0; }
176     // 两向量平行 (对应直线平行或重合)
177     bool parallel(Line v) { return sign((e-s).det(v.e-v.s)) == 0; }
178     // 线段比较, 用于map存直线
179     bool operator<(Line l) const {
180         if(s != l.s) return s < l.s;
181         else return e < l.e;
182     }
183     //两线段相交判断
184     //2 规范相交
185     //1 非规范相交
186     //0 不相交
187     int segcrossseg(Line v){
188         int d1 = sign((e-s).det(v.s-s));
189         int d2 = sign((e-s).det(v.e-s));
190         int d3 = sign((v.e-v.s).det(s-v.s));
191         int d4 = sign((v.e-v.s).det(e-v.s));
192         if((d1^d2) == -2 && (d3^d4) == -2) return 2;
193         return (d1 == 0 && sign((v.s-s).dot(v.s-e)) <= 0) ||
194             (d2 == 0 && sign((v.e-s).dot(v.e-e)) <= 0) ||
195             (d3 == 0 && sign((s-v.s).dot(s-v.e)) <= 0) ||
196             (d4 == 0 && sign((e-v.s).dot(e-v.e)) <= 0);
197     }
198     //点在线段上的判断
199     bool pointonseg(Point p){
200         return sign((p-s).det(e-s)) == 0 && sign((p-s).dot(p-e)) <= 0;
201     }
202 }
203 struct circle{
204     Point p; //圆心
205     db r; //半径
206     circle() {}
207     circle(Point _p, db _r){
208         p = _p;
209         r = _r;
210     }
211     circle(Point a,Point b,Point c){
212         Line u = Line((a+b)/2,((a+b)/2)+((b-a).rotleft()));
213         Line v = Line((b+c)/2,((b+c)/2)+((c-b).rotleft()));
214         p = u.crosspoint(v);
215         r = p.distTo(a);
216     }
217 };
218
219 int main()
220 {
221     Point p1(1,3), p2(2,3), p3(1.5,1), p4(1.5,5);
222     Line l1(p1,p2); l1.adjust();

```

```

223     printf("%f\n", l1.angle());
224
225     return 0;
226 }
227
228 /*
229 例题1: [hiocoder 1879]
230 有n个点, 求锐角三角形的个数/总面积 (2018 北京icpc )
231 n <= 2000, 可能三点共线
232 解法1:
233 (n,3)-直角-钝角, O((n^2)logn), 因为锐角三角形会出现重复, 因此计算直角和钝角, 直角和钝角不会重复
234
235
236
237 例题2:
238 点旋转, 用复数乘法来操作
239
240
241 例题3:
242 n个点, 查看是否有三点共线
243 枚举一个点, 将其它点进行极角排序, 查看是否有三点共线 O((n^2)logn)
244 如果是整数点, 还可以进行哈希
245 (x1,y1) (x2,y2)
246 d1 = gcd(x1,y1)
247 d2 = gcd(x2,y2)
248 hash (x1/d1, y1/d1), (x2/d2, y2/d2)
249 */

```

7.2 凸包三分

题意:

给出一个 N 个点的凸包, Q 次询问, 每次给出一个 idx 和 k , 表示现有一根长度无限的垂直 $(P[idx], P[idx + 1])$ 向量且距离 $P[idx]$ 为 k 的垂线。先问这根垂线倒落时凸包上第一个被碰到的点, 如果第一个被碰到的是直线, 则输出两个点。 $(1 \leq n \leq 10^5, 1 \leq Q \leq 10^5)$

思路:

稍微画图模拟一下就可以发现, 最终倒在哪个点上取决于凸包上的点与这个点连线, 连线之后与 $(P[idx], P[idx + 1])$ 之间的夹角大小, 夹角最大的点即为第一个碰到的点。若有两个相同的点, 即碰到的是直线。

这个夹角是先增后减的, 因此不能进行二分, 而是三分, 而且是整数三分。而整数三分的精髓就是先三分到大致区间, 然后在确定的小区间内暴力求解。

但是直接判断角度不好处理, 我们转化成向量的叉乘进行计算, 判断大小是变大还是变小即可。

总结:

1. 计算几何中判断大小的问题, 尽量转化成叉积或者点积进行计算。
2. 整数三分是先三分到一个小范围, 然后在小范围内直接暴力。

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
6 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
   << " , " << z1 << ":" << z2 << endl;

```

```

7  typedef long long ll;
8  typedef double db;
9  const int N = 1e5+100;
10 const int M = 1e5+100;
11 const db EPS = 1e-9;
12 #define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y))
13 #define cross0p(p1,p2,p3) sign((cross(p1,p2,p3)))
14 using namespace std;
15
16 inline int sign(db a) {return a < -EPS ? -1 : a > EPS; }
17 inline int cmp(db a,db b) {return sign(a-b);}
18
19 struct Point{
20     db x,y;
21     Point() {}
22     Point(db _x, db _y) : x(_x), y(_y) {}
23     Point operator-(Point p) {return {x-p.x,y-p.y};}
24     db dot(Point p) {return x*p.x+y*p.y;}
25     db det(Point p) { return x*p.y-y*p.x; } //叉积
26     db abs() { return sqrt(abs2()); }
27     db abs2() { return x*x+y*y; }
28     db distTo(Point p) { return (*this-p).abs(); }
29 }P[N];
30
31 int T,Q;
32
33 int calc(Point p2,Point p3,Point p4){
34     Point t1 = p3-p2, t2 = p4-p2;
35     return sign(t2.det(t1));
36 }
37
38 int main()
39 {
40     scanf("%d%d",&T,&Q);
41     rep(i,0,T-1) scanf("%lf%lf",&P[i].x,&P[i].y);
42     int n = T;
43     rep(i,1,Q){
44         int pos; db x; scanf("%d%lf",&pos,&x);
45         db len = P[pos].distTo(P[(pos+1)%n]);
46         db tx = ((db)(P[(pos+1)%n].x-P[pos].x)*(db)x/len)+P[pos].x;
47         db ty = ((db)(P[(pos+1)%n].y-P[pos].y)*(db)x/len)+P[pos].y;
48         int l = 1, r = T-1, lmid, rmid;
49         //三分先确定一个小范围
50         while(r-l >= 15){
51             lmid = l+(r-l)/3;
52             rmid = r-(r-l)/3;
53             if(calc({tx,ty},P[(pos+lmid)%n],P[(pos+rmid)%n]) == 1){
54                 l = lmid;
55             }
56             else r = rmid;
57         }
58         int idx1 = 1, idx2 = 0;
59         rep(k,l,r){
60             int jud = calc({tx,ty},P[(pos+k)%n],P[(pos+idx1)%n]);
61             if(jud == -1) idx1 = k, idx2 = 0;
62             else if(jud == 0) idx2 = k;
63         }
64         if(idx2 == 0) printf("%d\n", (idx1+pos)%n);
65         else printf("%d %d\n", (idx1+pos)%n, (idx2+pos)%n);

```

```

66     }
67     return 0;
68 }
```

7.3 锐角三角形计数

题意：

给出 n 个点，先从中任意选取三个点，使得这三个点为一个锐角三角形，问一共有多少个锐角三角形。 $(3 \leq n \leq 2000)$

思路：

正则反，很明显正向思考这题难度非常高，因此考虑逆向思考，即能够构造出的直角和钝角三角形个数。

我们枚举一个点，然后按这个点对其他所有点进行极角排序，然后进行双指针枚举，枚举一个刚好变成直角的点，再枚举一个刚好不是钝角的点，然后直接计算答案即可，具体内容可以看代码。

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
6 ;
6 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
    << " , " << z1 << ":" << z2 << endl;
7 typedef long long ll;
8 typedef double db;
9 const int N = 2000+100;
10 const int M = 1e5+100;
11 const db EPS = 1e-9;
12 using namespace std;
13
14 inline int sign(db a) {return a < -EPS ? -1 : a > EPS;}
15
16 struct Point{
17     ll x,y;
18     Point() {}
19     Point(ll _x, ll _y) : x(_x), y(_y) {}
20     Point operator+(Point p) {return {x+p.x, y+p.y};}
21     Point operator-(Point p) {return {x-p.x, y-p.y};}
22     ll dot(Point p) {return x*p.x+y*p.y;} //点积
23     ll det(Point p) {return x*p.y-y*p.x;} //叉积
24     int quad() const { return sign(y) == 1 || (sign(y) == 0 && sign(x) >= 0); }
25 }P[N],Q[2*N];
26
27 db rad(Point p1, Point p2){
28     return atan2l(p1.det(p2), p1.dot(p2));
29 }
30
31 bool cmp1(Point a, Point b){ //极角排序
32     if(a.quad() != b.quad()) return a.quad() < b.quad();
33     else return sign(a.det(b)) > 0;
34 }
35
36 int n;
37
38 int main()
39 {
40     int _; scanf("%d",&_);

```

```

41     while(_--){
42         ll ans = 0;
43         scanf("%d",&n);
44         rep(i,1,n){
45             scanf("%lld%lld",&P[i].x,&P[i].y);
46         }
47         int l = 0, r = 0;
48         rep(i,1,n){
49             int tot = 0;
50             rep(j,1,n){
51                 if(j == i) continue;
52                 Q[++tot].x = P[j].x-P[i].x, Q[tot].y = P[j].y-P[i].y;
53             }
54             sort(Q+1,Q+1+tot,cmp1);
55             // rep(j,tot+1,2*tot) Q[j].x = Q[j-tot].x, Q[j].y = Q[j-tot].y;
56             int l = 1, r = 1;
57             rep(j,1,tot){
58                 while(l <= tot && Q[j].det(Q[l]) >= 0 && Q[j].dot(Q[l]) > 0) l++;
59                 while(r <= tot && (Q[j].det(Q[r]) >= 0 || Q[j].dot(Q[r]) <= 0)) r++;
60                 ans += r-l;
61             }
62         }
63         ans = ((n*(n-1ll)*(n-2ll))/6ll)-ans;
64         printf("%lld\n",ans);
65     }
66     return 0;
67 }
```

7.4 点与凸包的切线

题意：

给定一个 n 个点的凸包，再给出 m 个光照点，每个光照点的照射范围为 360 度，问最少选取几个光照点可以照亮整个凸包，要求输出方案，保证不会出现一个光照点位于凸包的延长线上，共 200 组数据。 $(1 \leq n, m \leq 1000)$

思路：

其实这题思路比较明显，就是先求出每个光照点所能照射到的一段连续范围，然后问题就变成了给定一个长度为 n 的环形区域，以及 m 段区间，要求选取最少的区间覆盖整个区域。

首先求每个光照点对应的一段连续区域，比赛时的思路是极角排序，然后选择最两边的边。这样的复杂度是 $O(n^2 \log n)$ ，再加上 200 组数据，成功 TLE... 其实仔细思考一下可以发现并不需要进行极角排序，利用叉积即可解决该问题。因为照射范围的两个端点一定是光照点与凸包点连线形成的直线中最两边的两个点。

因此其它所有端点与光照点连线形成的直线都是顺时针或逆时针才能到达两个端点，即照射范围的两个端点是顺时针旋转的两个边界点，因此直接用叉积判断即可。

求出每个光照点范围之后，就只需要处理区间覆盖问题了。环形区域，我们只需要枚举起始点， $O(n^2)$ 求解即可。我们记录 $add[i]$ 数组表示所有覆盖第 i 条的光照点中所能覆盖的最远距离， $id[i]$ 记录光照点编号，然后直接贪心即可解决该问题。

```

1 #include <bits/stdc++.h>
2 #define mem(a,b) memset(a,b,sizeof a);
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define per(i,a,b) for(int i = a; i >= b; i--)
5 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6 typedef long long ll;
7 typedef double db;
8 #define pi acos(-1.0)
```

```

9 const int N = 1e3+100;
10 const db EPS = 1e-9;
11 using namespace std;
12
13 void dbg() {cout << "\n";}
14 template<typename T, typename... A> void dbg(T a, A... x) {cout << a << ' ' ; dbg(x...)}
15 #define logs(x...) {cout << #x << " -> "; dbg(x);}
16 inline int sign(db a) {return a < -EPS ? -1 : a > EPS;}
17 inline int cmp(db a,db b) {return sign(a-b);}
18
19 struct Point{
20     ll x,y;
21     Point operator-(Point p) {return {x-p.x,y-p.y};}
22     ll dot(Point p) {return x*p.x+y*p.y;}
23     ll det(Point p) {return x*p.y-y*p.x;} //叉积
24 }P[N],H[N];
25
26 int n,m,L[N],R[N],add[N],id[N],ans;
27 vector<int> base;
28
29 void init(){
30     scanf("%d%d",&n,&m);
31     rep(i,0,n-1) scanf("%lld%lld",&P[i].x,&P[i].y);
32     rep(i,0,m-1) scanf("%lld%lld",&H[i].x,&H[i].y);
33     rep(i,0,n-1) add[i] = id[i] = 0;
34     rep(i,0,m-1){
35         L[i] = 0, R[i] = 0;
36         rep(j,1,n-1){
37             //利用叉积求切线
38             if((P[L[i]]-H[i]).det(P[j]-H[i]) > 0) L[i] = j;
39             if((P[R[i]]-H[i]).det(P[j]-H[i]) < 0) R[i] = j;
40         }
41         int len = (R[i]-L[i]+n)%n;
42         rep(j,0,len-1){
43             int now = (L[i]+j)%n;
44             int tlen = (R[i]-now+n)%n;
45             if(tlen > add[now]) add[now] = tlen, id[now] = i;
46         }
47     }
48 }
49
50 void solve(){
51     ans = 1e5; base.clear();
52     rep(i,0,n-1){
53         int pos = i, num = 0, left = n;
54         vector<int> hp; hp.clear();
55         while(left > 0){
56             if(add[pos] == 0) {num = 1e5; break;}
57             left -= add[pos];
58             hp.push_back(id[pos]);
59             pos = (pos+add[pos])%n;
60             num++;
61         }
62         if(num < ans){
63             ans = num;
64             base = hp;
65         }
66     }
}

```

```
67     if(ans == 1e5) printf("-1\n");
68     else{
69         printf("%d\n",ans);
70         rep(i,0,ans-1) printf("%d%c",base[i]+1," \n"[i==ans-1]);
71     }
72 }
73
74 int main()
75 {
76     int _; scanf("%d",&_);
77     while(_--){
78         init();
79         solve();
80     }
81     return 0;
82 }
```

8 其他

8.1 三分

```

1 #include <cstdio>
2 #include <iostream>
3 #include <cstring>
4 #include <algorithm>
5 #define rep(i,a,b) for(int i = a; i <= b; i++)
6 using namespace std;
7 const int N = 20;
8 const double eps = 1e-7;
9 double coe[N];
10 int n;
11
12 double func(double x)
13 {
14     double ans = 0;
15     double base = 1;
16     rep(i,0,n)
17     {
18         ans += base*coe[i];
19         base *= x;
20     }
21     return ans;
22 }
23
24 void solve(double l, double r)
25 {
26     while(l + eps < r)
27     {
28         double m1 = (2*l+r)/3, m2 = (2*r+l)/3;
29         double r1 = func(m1), r2 = func(m2);
30         if(r1 >= r2)
31             r = m2;
32         else
33             l = m1;
34     }
35     printf("%.5f\n",r);
36 }
37
38 int main()
39 {
40     double l,r;
41     scanf("%d",&n);
42     scanf("%lf%lf",&l,&r);
43     for(int i = n; i >= 0; i--)
44         scanf("%lf",&coe[i]);
45     solve(l,r);
46     return 0;
47 }
```

8.2 bitset 优化暴力

题意：

给出一个 $n * m$ 的矩阵，矩阵每个点为 1 或 0，每行每列均可翻转且只能翻转一次。问能否将矩阵通过翻转变成一个从 $a_{1,1} \dots a_{1,m}$ $a_{2,1} \dots a_{n,m}$ 数值不下降的状态，如果可以给出每行每列的翻转状态，否则输出 NO。 $(1 \leq n, m \leq 200)$

思路：

稍微考虑一下这个题目，就可以发现此题难点主要在于一个点是否翻转由该点所代表的行列同时决定，因此我们难以进行判断。

所以我们可以思考能否事先确定行或列的状态，这样判断起来就很方便了。然后就可以发现只要确定了矩阵的最终状态，再确定了第一行的状态，我们就可以确定每一列的状态。确定了每一列状态和矩阵最终状态，就可以不断向下循环确定每一行的状态，如果一直都不发生冲突，则为 YES。

我们来考虑一下复杂度。首先枚举矩形的最终状态，即从 (n, m) 向 $(1, 1)$ 不断填 1，因此一共有 $n * m$ 个最终状态。对于每个最终状态，我们枚举第一行是否翻转，然后确定每一列的状态，再对剩下的每一行进行判断。因此复杂度为 $O(n * m * n * m)$ 。

这样的复杂度肯定是要 T 的，因此考虑 bitset 优化，对于每一行直接按位异或得到答案，可以将复杂度优化到 $O(n * m * n * m) / 32$ ，因此可以通过此题。具体异或细节见代码。

```

1 #include <cstdio>
2 #include <cstdlib>
3 #include <iostream>
4 #include <cstring>
5 #include <bitset>
6 #include <algorithm>
7 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
8 #define rep(i,a,b) for(int i = a; i <= b; i++)
9 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
10 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2 << endl
11 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << " , " << y1 << ":" << y2
12     << " , " << z1 << ":" << z2 << endl;
13 typedef long long ll;
14 typedef double db;
15 const int N = 200+10;
16 const int M = 1e5+100;
17 const db EPS = 1e-9;
18 using namespace std;
19 bitset<N> row,col,normal[2];
20 bitset<N> mp[N],jud;
21 bitset<N> base[N];
22 int n,m;
23
24 void test()
25 {
26     col = mp[0]^base[0]^normal[row[0]]; //枚举第一行状态，确定每一列是否翻转
27     rep(i,1,n-1){
28         jud = base[i]^mp[i]^col; //判断这一行的row是否翻转
29         if(jud.count() == m) row.set(i,1);
30         else if(jud.count() == 0) row.set(i,0);
31         else return;
32     }
33     printf("YES\n");
34     rep(i,0,n-1) printf("%d",row[i]==1); printf("\n");
35     rep(i,0,m-1) printf("%d",col[i]==1); printf("\n");
36     exit(0);
37 }
38
39 int main()
40 {

```

```

41     scanf("%d%d",&n,&m);
42     rep(i,0,n-1)
43         rep(j,0,m-1){
44             int xx; scanf("%d",&xx);
45             if(xx == 1) mp[i].set(j);
46         }
47     rep(i,0,m-1) norml[1].set(i);
48     for(int i = n-1; i >= 0; i--)
49         for(int j = m-1; j >= 0; j--){
50             base[i].set(j);
51             row.set(0,1);
52             test();
53             row.set(0,0);
54             test();
55         }
56     printf("NO\n");
57     return 0;
58 }
```

8.3 IDA*

```

1 #include <cstdio>
2 #include <iostream>
3 #include <algorithm>
4 #include <cmath>
5
6 int n, shell[21];
7
8 // swap [x1, x2] and [x2 + 1, x3]
9 void move (int x1, int x2, int x3)
10 {
11     int tmp[21], i, j;
12     for (i = x2 + 1, j = 0; i <= x3; i++, j++)
13         tmp[j] = shell[i];
14     for (i = x1; i <= x2; i++, j++)
15         tmp[j] = shell[i];
16     for (i = x1, j = 0; i <= x3; i++, j++)
17         shell[i] = tmp[j];
18     return;
19 }
20
21 int hfunc ()
22 {
23     int i, ans = 0;
24     for (i = 0; i < n - 1; i++)
25         if (shell[i + 1] != shell[i] + 1) ans++;
26     if (shell[n - 1] != n) ans++;
27     return ans;
28 }
29
30 int maxdepth;
31 int dfs (int depth)
32 {
33     int x1, x2, x3, h;
34     for (x1 = 0; x1 <= n - 2; x1++) //枚举移动区间的左端点
35     {
36         for (x2 = x1; x2 <= n - 2; x2++) //枚举移动区间的右端点
37         {
```

```

38         for (x3 = x2 + 1; x3 <= n - 1; x3++) //枚举插入点
39     {
40         move(x1, x2, x3); //进行移动
41         h = hfunc();
42         if (h == 0) return 1;
43         else if (3 * depth + h <= 3 * maxdepth) //IDA*, 限制深度
44     {
45             if (dfs(depth + 1)) return 1;
46         }
47         move(x1, x1 - x2 + x3 - 1, x3); //如果不可行, 则返回原状
48     }
49 }
50 } return 0;
51 }
52
53 int main ()
54 {
55     int kase, i;
56     scanf("%d", &kase);
57     for (; kase > 0; kase--)
58     {
59         scanf("%d", &n); //几本书
60         for (i = 0; i < n; i++)
61             scanf("%d", &shell[i]); //各个位置的序号
62         maxdepth = (int)ceil((double)hfunc() / 3);
63         if (maxdepth) while (maxdepth < 5 && dfs(1) == 0) maxdepth++; //不断加大搜索深度
64         if (maxdepth == 5) printf("5 or more\n");
65         else printf("%d\n", maxdepth);
66     }
67     return 0;
68 }
69
70 /*
71 题意:
72 1-n本书, 可以选择任意一段连续的书, 再将这一段书插入到其他某个位置。
73 目标是将书按照1-n的顺序依次排列, 求最少需要多少次操作。
74 如果更换次数 >= 5, 则直接输出 "5 or more"
75
76 解法:
77 限制dfs的搜索深度, 并且设计一个估价函数。
78 可以发现, 将一段书插入到某一个位置, 最多改变3个数的后继值, 因此对于每一个状态, 记录该状态下i与i+1不匹配
    的个数
79 然后ceil(这个个数/3)的值就是该状态达到目标状态所至少需要的操作数
80
81 对于每个状态, 当前深度+目标操作数 < 5才继续进行深度搜索, 由此大大降低了搜索的复杂度
82 */

```

8.4 左右手路径

```

1 #include <bits/stdc++.h>
2 #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
3 #define rep(i,a,b) for(int i = a; i <= b; i++)
4 #define LOG1(x1,x2) cout << x1 << ":" << x2 << endl;
5 #define LOG2(x1,x2,y1,y2) cout << x1 << ":" << x2 << ", " << y1 << ":" << y2 << endl
6 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ":" << x2 << ", " << y1 << ":" << y2
    << ", " << z1 << ":" << z2 << endl;
7 typedef long long ll;

```

```

8  typedef double db;
9  const int N = 1e5+100;
10 const int M = 1e5+100;
11 const db EPS = 1e-9;
12 using namespace std;
13
14 //左手路径，优先序循环为顺时针，第一个优先选择为前一个方位
15 //右手路径，优先序循环为逆时针，第一个优先选择为后一个方位
16 string choice[4] = {"UP", "RIGHT", "DOWN", "LEFT"}, op;
17 int now = 2;
18
19 int main()
20 {
21     //右手路径，初始 now = "DOWN"
22     while(1){
23         rep(i,0,3){
24             string next = choice[(now-i+5)%4];
25             cout << "LOOK " << next << endl;
26             cin >> op;
27             if(op == "SAFE"){
28                 cout << "GO " << next << endl;
29                 cin >> op;
30                 now = (now-i+5)%4;
31                 if(op == "YES") exit(0);
32                 break;
33             }
34         }
35     }
36
37     //左手路径，初始 now = "RIGHT"
38     // while(1){
39     //     rep(i,0,3){
40     //         string next = choice[(now+i+3)%4];
41     //         cout << "LOOK " << next << endl;
42     //         cin >> op;
43     //         if(op == "SAFE"){
44     //             cout << "GO " << next << endl;
45     //             cin >> op;
46     //             now = (now+i+3)%4;
47     //             if(op == "YES") exit(0);
48     //             break;
49     //         }
50     //     }
51     // }
52     return 0;
53 }

```

8.5 圆周率小数点后第 n 位

```

1 // 求小数点后n位
2 #include <iostream>
3 #define MAX_C 56000
4 int a = 10000, b, c = MAX_C, d, e, f[MAX_C + 1], g, n, ans, cnt;
5 using namespace std;
6 int main() {
7     while(~scanf("%d", &n)){
8         for (; b - c; )
9             f[b++] = a / 5;

```

```

10    for (; d = 0, g = c * 2;
11        c -= 14, ans = e + d / a, e = d % a, cnt++)
12    { if (cnt * 4 > n) break;
13        for (b = c; d += f[b]*a, f[b] = d % --g, d /= g--, --b; d *= b); }
14    if (n % 4 == 0) cout << (ans / 1000);
15    else if (n % 4 == 1) cout << ((ans / 100) % 10);
16    else if (n % 4 == 2) cout << ((ans / 10) % 10);
17    else if (n % 4 == 3) cout << (ans % 10);
18    printf("\n");
19    return 0;
20 }

```

8.6 BM 板子

8.6.1 BM 板子

```

1 #include <cstdio>
2 #include <cstring>
3 #include <cmath>
4 #include <algorithm>
5 #include <vector>
6 #include <string>
7 #include <map>
8 #include <set>
9 #include <cassert>
10 #include<bits/stdc++.h>
11 using namespace std;
12 #define rep(i,a,n) for (ll i=a;i<n;i++)
13 #define per(i,a,n) for (ll i=n-1;i>=a;i--)
14 #define pb push_back
15 #define mp make_pair
16 #define all(x) (x).begin(),(x).end()
17 #define fi first
18 #define se second
19 #define SZ(x) ((int)(x).size())
20 typedef long long ll;
21 typedef vector<ll> VI;
22
23 typedef pair<ll,ll> PII;
24 const ll mod=1000000007;
25 ll powmod(ll a,ll b) {ll res=1;a%=mod; assert(b>=0); for(;b;b>>=1){if(b&1)res=res*a%mod
   ;a=a*a%mod;}return res;}
26 // head
27
28 ll _,n;
29 namespace linear_seq {
30     const ll N=10010;
31     ll res[N],base[N],_c[N],_md[N];
32     vector<ll> Md;
33     void mul(ll *a,ll *b,ll k) {
34         rep(i,0,k+k) _c[i]=0;
35         rep(i,0,k) if (a[i]) rep(j,0,k) _c[i+j]=(_c[i+j]+a[i]*b[j])%mod;
36         for (ll i=k+k-1;i>=k;i--) if (_c[i])
37             rep(j,0,SZ(Md)) _c[i-k+Md[j]]=(_c[i-k+Md[j]]-_c[i]*_md[Md[j]])%mod;
38         rep(i,0,k) a[i]=_c[i];
39     }
40     ll solve(ll n,VI a,VI b) { // a 系数 b 初值 b[n+1]=a[0]*b[n]+...
41 //         printf("%lld\n",SZ(b));
42     ll ans=0,pnt=0;

```

```

43     ll k=SZ(a);
44     assert(SZ(a)==SZ(b));
45     rep(i,0,k) _md[k-1-i]=-a[i];_md[k]=1;
46     Md.clear();
47     rep(i,0,k) if (_md[i]!=0) Md.push_back(i);
48     rep(i,0,k) res[i]=base[i]=0;
49     res[0]=1;
50     while ((1ll<<pnt)<=n) pnt++;
51     for (ll p=pnt;p>=0;p--) {
52         mul(res,res,k);
53         if ((n>p)&1) {
54             for (ll i=k-1;i>=0;i--) res[i+1]=res[i];res[0]=0;
55             rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]]-res[k]*_md[Md[j]])%mod;
56         }
57     }
58     rep(i,0,k) ans=(ans+res[i]*b[i])%mod;
59     if (ans<0) ans+=mod;
60     return ans;
61 }
62 VI BM(VI s) {
63     VI C(1,1),B(1,1);
64     ll L=0,m=1,b=1;
65     rep(n,0,SZ(s)) {
66         ll d=0;
67         rep(i,0,L+1) d=(d+(ll)C[i]*s[n-i])%mod;
68         if (d==0) ++m;
69         else if (2*L<=n) {
70             VI T=C;
71             ll c=mod-d*powmod(b,mod-2)%mod;
72             while (SZ(C)<SZ(B)+m) C.pb(0);
73             rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
74             L=n+1-L; B=T; b=d; m=1;
75         } else {
76             ll c=mod-d*powmod(b,mod-2)%mod;
77             while (SZ(C)<SZ(B)+m) C.pb(0);
78             rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
79             ++m;
80         }
81     }
82     return C;
83 }
84 ll gao(VI a,ll n) {
85     VI c=BM(a);
86     c.erase(c.begin());
87     rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
88     return solve(n,c,VI(a.begin(),a.begin()+SZ(c)));
89 }
90 };
91 ll dp[10005];
92 int main() {
93     ll t;
94     scanf("%lld",&t);
95     ll k;
96     ll n;
97     while (t--) {
98         scanf("%lld%lld",&k,&n);
99         ll p=powmod(k,mod-2);
100        if(n==-1){
101            printf("%lld\n",2ll*powmod(k+1,mod-2)%mod);

```

```

102         continue;
103     }
104     vector<ll>v;
105     dp[0]=1;
106     rep(i,1,11){
107         dp[i]=0;
108         rep(j,max(0ll,i-k),i){
109             dp[i]=(dp[i]+dp[j]*p)%mod;
110         }
111         v.push_back(dp[i]);
112         //cout<<i<<"!"<<dp[i]<<endl;
113     }
114     int len=v.size();
115     //for(int i=0;i<len;i++)cout<<v[i]<<endl;
116     //VI{1,2,4,7,13,24}
117     printf("%lld\n",linear_seq::gao(v,n-1));
118 }
119 }
```

8.6.2 BM 递推式

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define inf 0x3f3f3f3f
4 #define rep(i,a,b) for(int i=a;i<=b;i++)
5 #define per(i,a,b) for(int i=a;i>=b;i--)
6 #define ll long long
7 #define VI vector<ll>
8 const int MAXN = 105;
9 const int MOD = 1e9+7;
10 int a[5]={2,3,7};
11 int b[MAXN];
12 const int N = 50010;///多项式系数最大值
13 int64_t res[N],c[N],md[N],COEF[N]/**COEF是多项式系数*/,Mod=10000007;
14 vector<ll> Md;
15
16 inline static int64_t gcdEx(int64_t a, int64_t b, int64_t&x, int64_t& y)
17 {
18     if(!b) {x=1;y=0;return a;}
19     int64_t d = gcdEx(b,a%b,y,x);
20     y -= (a/b)*x;
21     return d;
22 }
23
24 static int64_t Inv(int64_t a, int64_t Mod) {
25     int64_t x, y;
26     return gcdEx(a, Mod, x, y)==1?(x%Mod+Mod)%Mod:-1;
27 };
28
29 VI BM(VI s) {///BM算法求模数是质数的递推式子的通项公式,可以单独用
30     VI C(1,1),B(1,1);
31     int L(0),m(1),b(1);
32     for(size_t n(0);n<s.size();++n) {
33         int64_t d(0);
34         for(int i(0);i<=L;++i) d=(d+C[i]*s[n-i])%Mod;
35         if (!d) ++m;
36         else {
37             VI T(C);

```

```

38         int64_t c(Mod-d*Inv(b,Mod)%Mod);
39         while (C.size()<B.size()+m) C.push_back(0);
40         for (size_t i(0);i<B.size();++i)
41             C[i+m]=(C[i+m]+c*B[i])%Mod;
42         if (2*L<=(int)n) {L=n+1-L; B=T; b=d; m=1;}
43         else ++m ;
44     }
45 }
46 //下边这样写能够输出递推式的系数.
47 printf("F[n] = ");
48 for(size_t i(0);i<C.size();++i) {
49     COEF[i+1] = min((int64_t)C[i],Mod-C[i]) ;
50     if(i>0) {
51         if(i != 1) printf(" + ");
52         printf("%lld*F[n-%d]",COEF[i+1],i) ;
53         putchar(i+1==C.size()?'\\n':' ') ;
54     }
55 }
56 return C;
57 }
58 int main()
59 {
60     VI v = {0,1,2,4,11,25} ;
61     BM(v);
62     return 0;
63 }
```

8.7 高精度

8.7.1 高精度

```

1 #include <cstdio>
2
3 struct HighPrec{
4     int L , A[10001];
5 };
6 inline HighPrec Init(){
7     // 函数作用: 返回一个初值为 1 的高精度
8     HighPrec H;
9     H.L = 1 , H.A[1] = 1;
10    return H;
11 }
12 HighPrec Mul(HighPrec A , int k){
13     // 函数作用: 高精度乘上单精度
14     HighPrec H;
15     for(int i = 1 ; i <= A.L ; i++)
16         H.A[i] = A.A[i] * k; // 乘
17     for(int i = 2 ; i <= A.L ; i++)
18         H.A[i] += H.A[i - 1] / 10,
19         H.A[i - 1] %= 10; // 进位
20     H.L = A.L;
21     while(H.A[H.L] > 10) // 看原先最高位能不能进位
22         H.A[H.L + 1] = H.A[H.L] / 10,
23         H.A[H.L] %= 10, // 进位
24         H.L++; // 增加长度
25     return H;
26 }
27 HighPrec Div(HighPrec A , int k){
28     // 函数作用: 高精度除以单精度
29 }
```

```

29     HighPrec H;
30     int t = 0;
31     for(int i = A.L ; i >= 1 ; i--)
32         t = t * 10 + A.A[i], // 小学竖式
33         H.A[i] = t / k , t %= k;
34     H.L = A.L;
35     while(H.A[H.L] == 0)
36         H.L--; // 看原先最高位有没有被除为 0 并不断退位
37     return H;
38 }
39 void OutPut(HighPrec A){
40     // 函数作用：输出高精度
41     for(int i = A.L ; i >= 1 ; i--)
42         printf("%d" , A.A[i]); // 正常套路，逆序输出
43 }
44
45 HighPrec T;
46 int n;
47
48 int main(){
49     scanf("%d" , &n);
50     T = Init(); // 赋初值 1
51     // 下面的乘除是原公式简化的结果，大家请手动去简化
52     for(int i = n + 2 ; i <= n * 2 ; i++)
53         T = Mul(T , i);
54     for(int i = 1 ; i <= n ; i++)
55         T = Div(T , i);
56     OutPut(T); // 输出
57     return 0;
58 }
```

8.7.2 大数模板

```

1 #include <bits/stdc++.h>
2 typedef long long ll;
3 using namespace std;
4
5 struct BigInteger{
6     ll A[50];
7     enum{MOD = 1000000};
8     BigInteger(){memset(A, 0, sizeof(A)); A[0]=1;}
9     void set(ll x){memset(A, 0, sizeof(A)); A[0]=1; A[1]=x;}
10    void print(){
11        printf("%lld", A[A[0]]);
12        for (ll i=A[0]-1; i>0; i--){
13            if (A[i]==0){printf("000000"); continue;}
14            for (ll k=10; k*A[i]<MOD; k*=10ll) printf("0");
15            printf("%lld", A[i]);
16        }
17        printf("\n");
18    }
19    ll& operator [] (int p) {return A[p];}
20    const ll& operator [] (int p) const {return A[p];}
21    BigInteger operator + (const BigInteger& B){
22        BigInteger C;
23        C[0]=max(A[0], B[0]);
24        for (ll i=1; i<=C[0]; i++)
25            C[i]+=A[i]+B[i], C[i+1]+=C[i]/MOD, C[i]%=MOD;
```

```
26         if (C[C[0]+1] > 0) C[0]++;
27         return C;
28     }
29     BigInteger operator * (const BigInteger& B){
30         BigInteger C;
31         C[0]=A[0]+B[0];
32         for (ll i=1; i<=A[0]; i++){
33             for (ll j=1; j<=B[0]; j++){
34                 C[i+j-1]+=A[i]*B[j], C[i+j]+=C[i+j-1]/MOD, C[i+j-1]%=MOD;
35             }
36             if (C[C[0]] == 0) C[0]--;
37         }
38     }
39 };
40
41 int main(){
42     BigInteger A,B,C,D;
43     A.set(10000);
44     B.set(10000);
45     C = A+B;
46     D = A*B;
47     C.print();
48     D.print();
49 }
```