
ACM 常用算法模板



kuangbin

<http://www.cnblogs.com/kuangbin/>

Email:kuangbin2009@126.com

Last build at July 8, 2018

Contents

1	字符串处理	5
1.1	KMP	5
1.2	e-KMP	8
1.3	Manacher	8
1.4	AC 自动机	9
1.5	后缀数组	11
1.5.1	DA	11
1.5.2	DC3	14
1.6	后缀自动机	15
1.6.1	基本函数	15
1.6.2	例题	16
1.7	字符串 hash	23
2	数学	25
2.1	素数	25
2.1.1	素数筛选 (判断 $<MAXN$ 的数是否素数)	25
2.1.2	素数筛选 (筛选出小于等于 $MAXN$ 的素数)	25
2.1.3	大区间素数筛选 (POJ 2689)	25
2.2	素数筛选和合数分解	27
2.3	扩展欧几里得算法 (求 $ax+by=gcd$ 的解以及逆元)	27
2.4	求逆元	28
2.4.1	扩展欧几里德法	28
2.4.2	简洁写法	28
2.4.3	利用欧拉函数	28
2.5	模线性方程组	28
2.6	随机素数测试和大数分解 (POJ 1811)	29
2.7	欧拉函数	32
2.7.1	分解质因素求欧拉函数	32
2.7.2	筛法欧拉函数	32
2.7.3	求单个数的欧拉函数	32
2.7.4	线性筛 (同时得到欧拉函数和素数表)	32
2.8	高斯消元 (浮点数)	33
2.9	FFT	34
2.10	高斯消元法求方程组的解	37
2.10.1	一类开关问题, 对 2 取模的 01 方程组	37
2.10.2	解同余方程组	39
2.11	整数拆分	41
2.12	求 A^B 的约数之和对 MOD 取模	43
2.13	莫比乌斯反演	44
2.13.1	莫比乌斯函数	44
2.13.2	例题: BZOJ2301	44
2.14	Baby-Step Giant-Step	46
2.15	自适应 simpson 积分	46
2.16	斐波那契数列取模循环节	46
2.17	原根	50
2.18	快速数论变换	51
2.18.1	HDU4656 卷积取模	51
2.19	其它公式	54
2.19.1	Polya	54

3	数据结构	56
3.1	划分树	56
3.2	RMQ	57
3.2.1	一维	57
3.2.2	二维	57
3.3	树链剖分	59
3.3.1	点权	59
3.3.2	边权	61
3.4	伸展树 (splay tree)	64
3.4.1	例题: HDU1890	64
3.4.2	例题: HDU3726	67
3.5	动态树	72
3.5.1	SPOJQTREE	72
3.5.2	SPOJQTREE2	75
3.5.3	SPOJQTREE4	78
3.5.4	SPOJQTREE5	82
3.5.5	SPOJQTREE6	84
3.5.6	SPOJQTREE7	87
3.5.7	HDU4010	91
3.6	主席树	95
3.6.1	查询区间多少个不同的数	95
3.6.2	静态区间第 k 大	97
3.6.3	树上路径点权第 k 大	98
3.6.4	动态第 k 大	102
3.7	Treap	105
3.8	KD 树	108
3.8.1	HDU4347 K 近邻	108
3.8.2	CF44G	111
3.8.3	HDU4742	114
3.9	替罪羊树 (ScapeGoat Tree)	116
3.9.1	CF455D	116
3.10	动态 KD 树	120
3.11	树套树	123
3.11.1	替罪羊树套 splay	123
4	图论	130
4.1	最短路	130
4.1.1	Dijkstra 单源最短路	130
4.1.2	Dijkstra 算法 + 堆优化	130
4.1.3	单源最短路 bellman_ford 算法	131
4.1.4	单源最短路 SPFA	132
4.2	最小生成树	133
4.2.1	Prim 算法	133
4.2.2	Kruskal 算法	134
4.3	次小生成树	135
4.4	有向图的强连通分量	136
4.4.1	Tarjan	136
4.4.2	Kosaraju	137
4.5	图的割点、桥和双连通分支的基本概念	138
4.6	割点与桥	139
4.6.1	模板	139

4.6.2	调用	141
4.7	边双连通分支	143
4.8	点双连通分支	144
4.9	最小树形图	147
4.10	二分图匹配	149
4.10.1	邻接矩阵 (匈牙利算法)	149
4.10.2	邻接表 (匈牙利算法)	150
4.10.3	Hopcroft-Karp 算法	151
4.11	二分图多重匹配	152
4.12	二分图最大权匹配 (KM 算法)	153
4.13	一般图匹配带花树	154
4.14	一般图最大加权匹配	157
4.15	生成树计数	159
4.16	最大流	161
4.16.1	SAP 邻接矩阵形式	161
4.16.2	SAP 邻接矩阵形式 2	162
4.16.3	ISAP 邻接表形式	163
4.16.4	ISAP+bfs 初始化 + 栈优化	165
4.16.5	dinic	166
4.16.6	最大流判断多解	168
4.17	最小费用最大流	169
4.17.1	SPFA 版费用流	169
4.17.2	zkw 费用流	170
4.18	2-SAT	172
4.18.1	染色法 (可以得到字典序最小的解)	172
4.18.2	强连通缩点法 (拓扑排序只能得到任意解)	173
4.19	曼哈顿最小生成树	177
4.20	LCA	179
4.20.1	dfs+ST 在线算法	179
4.20.2	离线 Tarjan 算法	181
4.20.3	LCA 倍增法	184
4.21	欧拉路	186
4.21.1	有向图	186
4.21.2	无向图	188
4.21.3	混合图	189
4.22	树分治	192
4.22.1	点分治 -HDU5016	192
4.22.2	* 点分治 -HDU4918	195
4.22.3	链分治 -HDU5039	199
5	搜索	205
5.1	Dancing Links	205
5.1.1	精确覆盖	205
5.1.2	可重复覆盖	207
5.2	八数码	209
5.2.1	HDU1043 反向搜索	209
6	动态规划	212
6.1	最长上升子序列 $O(n\log n)$	212
6.2	背包	212
6.3	插头 DP	213

6.3.1	HDU 4285	213
7	计算几何	218
7.1	二维几何	218
7.2	三维几何	238
7.3	平面最近点对	242
7.4	三维凸包	243
7.4.1	HDU4273	243
8	其他	249
8.1	高精度	249
8.2	完全高精度	250
8.3	strtok 和 sscanf 结合输入	256
8.4	解决爆栈, 手动加栈	256
8.5	STL	256
8.5.1	优先队列 priority_queue	256
8.5.2	set 和 multiset	257
8.6	输入输出外挂	258
8.7	莫队算法	258
8.7.1	分块	259
8.7.2	Manhattan MST 的 dfs 顺序求解	260
8.8	VIM 配置	264

1 字符串处理

1.1 KMP

```

1  /*
2  * next[] 的含义: x[i-next[i]...i-1]=x[0...next[i]-1]
3  * next[i] 为满足 x[i-z...i-1]=x[0...z-1] 的最大 z 值 (就是 x 的自身匹配)
4  */
5  void kmp_pre(char x[],int m,int next[]){
6      int i,j;
7      j=next[0]=-1;
8      i=0;
9      while(i<m){
10         while(-1!=j && x[i]!=x[j])j=next[j];
11         next[++i]=++j;
12     }
13 }
14 /*
15 * kmpNext[i] 的意思:next'[i]=next[next[...[next[i]]]] (直到
    next'[i]<0 或者 x[next'[i]]!=x[i])
16 * 这样的预处理可以快一些
17 */
18 void preKMP(char x[],int m,int kmpNext[]){
19     int i,j;
20     j=kmpNext[0]=-1;
21     i=0;
22     while(i<m){
23         while(-1!=j && x[i]!=x[j])j=kmpNext[j];
24         if(x[++i]==x[++j])kmpNext[i]=kmpNext[j];
25         else kmpNext[i]=j;
26     }
27 }
28 /*
29 * 返回 x 在 y 中出现的次数, 可以重叠
30 */
31 int next[10010];
32 int KMP_Count(char x[],int m,char y[],int n){//x 是模式串, y 是主串
33     int i,j;
34     int ans=0;
35     //preKMP(x,m,next);
36     kmp_pre(x,m,next);
37     i=j=0;
38     while(i<n){
39         while(-1!=j && y[i]!=x[j])j=next[j];
40         i++;j++;
41         if(j>=m){
42             ans++;
43             j=next[j];
44         }
45     }
46     return ans;

```

```

47 }
48 //经典题目: POJ 3167
49 /*
50 * POJ 3167 Cow Patterns
51 * 模式串可以浮动的模式匹配问题
52 * 给出模式串的相对大小, 需要找出模式串匹配次数和位置
53 * 比如说模式串: 1, 4, 4, 2, 3, 1 而主串: 5,6,2,10,10,7,3,2,9
54 * 那么 2,10,10,7,3,2 就是匹配的
55 *
56 * 统计比当前数小, 和于当前数相等的, 然后进行 kmp
57 */
58 const int MAXN=100010;
59 const int MAXM=25010;
60 int a[MAXN];
61 int b[MAXN];
62 int n,m,s;
63 int as[MAXN][30];
64 int bs[MAXM][30];
65 void init(){
66     for(int i=0;i<n;i++){
67         if(i==0){
68             for(int j=1;j<=25;j++)as[i][j]=0;
69         }
70         else{
71             for(int j=1;j<=25;j++)as[i][j]=as[i-1][j];
72         }
73         as[i][a[i]]++;
74     }
75     for(int i=0;i<m;i++){
76         if(i==0){
77             for(int j=1;j<=25;j++)bs[i][j]=0;
78         }
79         else{
80             for(int j=1;j<=25;j++)bs[i][j]=bs[i-1][j];
81         }
82         bs[i][b[i]]++;
83     }
84 }
85 int next[MAXM];
86 void kmp_pre(){
87     int i,j;
88     j=next[0]=-1;
89     i=0;
90     while(i<m){
91         int t11=0,t12=0,t21=0,t22=0;
92         for(int k=1;k<b[i];k++){
93             if(i-j>0)t11+=bs[i][k]-bs[i-j-1][k];
94             else t11+=bs[i][k];
95         }
96         if(i-j>0)t12=bs[i][b[i]]-bs[i-j-1][b[i]];
97         else t12=bs[i][b[i]];

```



```

98
99     for(int k=1;k<b[j];k++){
100         t21+=bs[j][k];
101     }
102     t22=bs[j][b[j]];
103     if(j==--1 || (t11==t21&& t12==t22)){
104         next[++i]=++j;
105     }
106     else j=next[j];
107 }
108 }
109 vector<int>ans;
110 void kmp(){
111     ans.clear();
112     int i,j;
113     kmp_pre();
114     i=j=0;
115     while(i<n){
116         int t11=0,t12=0,t21=0,t22=0;
117         for(int k=1;k<a[i];k++){
118             if(i-j>0)t11+=as[i][k]-as[i-j-1][k];
119             else t11+=as[i][k];
120         }
121         if(i-j>0)t12=as[i][a[i]]-as[i-j-1][a[i]];
122         else t12=as[i][a[i]];
123
124         for(int k=1;k<b[j];k++){
125             t21+=bs[j][k];
126         }
127         t22=bs[j][b[j]];
128         if(j==--1 || (t11==t21&& t12==t22)){
129             i++;j++;
130             if(j>=m){
131                 ans.push_back(i-m+1);
132                 j=next[j];
133             }
134         }
135         else j=next[j];
136     }
137 }
138 int main(){
139     while(scanf("%d%d%d",&n,&m,&s)==3){
140         for(int i=0;i<n;i++)scanf("%d",&a[i]);
141         for(int i=0;i<m;i++)scanf("%d",&b[i]);
142         init();
143         kmp();
144         printf("%d\n",ans.size());
145         for(int i=0;i<ans.size();i++)
146             printf("%d\n",ans[i]);
147     }
148     return 0;

```

149 | }

1.2 e-KMP

```

1  /*
2  * 扩展 KMP 算法
3  */
4  //next[i]:x[i...m-1] 与 x[0...m-1] 的最长公共前缀
5  //extend[i]:y[i...n-1] 与 x[0...m-1] 的最长公共前缀
6  void pre_EKMP(char x[],int m,int next[]){
7      next[0] = m;
8      int j = 0;
9      while( j+1 < m && x[j] == x[j+1] )j++;
10     next[1] = j;
11     int k = 1;
12     for(int i = 2; i < m; i++){
13         int p = next[k]+k-1;
14         int L = next[i-k];
15         if( i+L < p+1 )next[i] = L;
16         else{
17             j = max(0,p-i+1);
18             while( i+j < m && x[i+j] == x[j])j++;
19             next[i] = j;
20             k = i;
21         }
22     }
23 }
24 void EKMP(char x[],int m,char y[],int n,int next[],int extend[]){
25     pre_EKMP(x,m,next);
26     int j = 0;
27     while(j < n && j < m && x[j] == y[j])j++;
28     extend[0] = j;
29     int k = 0;
30     for(int i = 1;i < n;i++){
31         int p = extend[k]+k-1;
32         int L = next[i-k];
33         if(i+L < p+1)extend[i] = L;
34         else{
35             j = max(0,p-i+1);
36             while( i+j < n && j < m && y[i+j] == x[j] )j++;
37             extend[i] = j;
38             k = i;
39         }
40     }
41 }

```

1.3 Manacher

```

1  /*
2  * 求最长回文子串
3  */
4  const int MAXN=110010;

```

```

5 char Ma[MAXN*2];
6 int Mp[MAXN*2];
7 void Manacher(char s[],int len){
8     int l=0;
9     Ma[l++]='$';
10    Ma[l++]='#';
11    for(int i=0;i<len;i++){
12        Ma[l++]=s[i];
13        Ma[l++]='#';
14    }
15    Ma[l]=0;
16    int mx=0,id=0;
17    for(int i=0;i<l;i++){
18        Mp[i]=mx>i?min(Mp[2*id-i],mx-i):1;
19        while(Ma[i+Mp[i]]==Ma[i-Mp[i]])Mp[i]++;
20        if(i+Mp[i]>mx){
21            mx=i+Mp[i];
22            id=i;
23        }
24    }
25 }
26 /*
27  * abaaba
28  * i:      0 1 2 3 4 5 6 7 8 9 10 11 12 13
29  * Ma[i]: $ # a # b # a # a # b # a #
30  * Mp[i]: 1 1 2 1 4 1 2 7 2 1 4 1 2 1
31  */
32 char s[MAXN];
33 int main(){
34     while(scanf("%s",s)==1){
35         int len=strlen(s);
36         Manacher(s,len);
37         int ans=0;
38         for(int i=0;i<2*len+2;i++){
39             ans=max(ans,Mp[i]-1);
40         }
41         printf("%d\n",ans);
42     }
43     return 0;
44 }

```

1.4 AC 自动机

```

1 //=====
2 // HDU 2222
3 // 求目标串中出现了几个模式串
4 //=====
5 struct Trie{
6     int next[500010][26],fail[500010],end[500010];
7     int root,L;
8     int newnode(){
9         for(int i = 0;i < 26;i++)
10            next[L][i] = -1;

```

```

11         end[L++] = 0;
12         return L-1;
13     }
14     void init(){
15         L = 0;
16         root = newnode();
17     }
18     void insert(char buf[]){
19         int len = strlen(buf);
20         int now = root;
21         for(int i = 0; i < len; i++){
22             if(next[now][buf[i]-'a'] == -1)
23                 next[now][buf[i]-'a'] = newnode();
24             now = next[now][buf[i]-'a'];
25         }
26         end[now]++;
27     }
28     void build(){
29         queue<int>Q;
30         fail[root] = root;
31         for(int i = 0; i < 26; i++){
32             if(next[root][i] == -1)
33                 next[root][i] = root;
34             else{
35                 fail[next[root][i]] = root;
36                 Q.push(next[root][i]);
37             }
38         }
39         while( !Q.empty() ){
40             int now = Q.front();
41             Q.pop();
42             for(int i = 0; i < 26; i++){
43                 if(next[now][i] == -1)
44                     next[now][i] = next[fail[now]][i];
45                 else{
46                     fail[next[now][i]] = next[fail[now]][i];
47                     Q.push(next[now][i]);
48                 }
49             }
50         }
51     }
52     int query(char buf[]){
53         int len = strlen(buf);
54         int now = root;
55         int res = 0;
56         for(int i = 0; i < len; i++){
57             now = next[now][buf[i]-'a'];
58             int temp = now;
59             while( temp != root ){
60                 res += end[temp];
61                 end[temp] = 0;
62                 temp = fail[temp];

```

```

62         }
63     }
64     return res;
65 }
66 void debug(){
67     for(int i = 0;i < L;i++){
68         printf("id_=%3d,fail_=%3d,end_=%3d,chi_=",i,fail[i],end[i]);
69         for(int j = 0;j < 26;j++){
70             printf("%2d",next[i][j]);
71             printf("]\n");
72         }
73     }
74 };
75 char buf[1000010];
76 Trie ac;
77 int main(){
78     int T;
79     int n;
80     scanf("%d",&T);
81     while( T-- ){
82         scanf("%d",&n);
83         ac.init();
84         for(int i = 0;i < n;i++){
85             scanf("%s",buf);
86             ac.insert(buf);
87         }
88         ac.build();
89         scanf("%s",buf);
90         printf("%d\n",ac.query(buf));
91     }
92     return 0;
93 }

```

1.5 后缀数组

1.5.1 DA

```

1  /*
2  *suffix array
3  *倍增算法  $O(n \log n)$ 
4  *待排序数组长度为  $n$ ，放在  $0 \sim n-1$  中，在最后面补一个 0
5  *da(str ,sa,rank,height, n , );//注意是  $n$ ;
6  *例如:
7  *n = 8;
8  * num[] = { 1, 1, 2, 1, 1, 1, 1, 2, $ }; 注意 num 最后一位为 0，其他
   大于 0
9  *rank[] = 4, 6, 8, 1, 2, 3, 5, 7, 0 ;rank[0 ~ n-1] 为有效值，rank[n]
   必定为 0 无效值
10 *sa[] = 8, 3, 4, 5, 0, 6, 1, 7, 2 ;sa[1 ~ n] 为有效值，sa[0] 必定为  $n$  是
   无效值
11 *height[] = 0, 0, 3, 2, 3, 1, 2, 0, 1 ;height[2 ~ n] 为有效值

```

```

12 *
13 */
14 const int MAXN=20010;
15 int t1[MAXN],t2[MAXN],c[MAXN]; //求 SA 数组需要的中间变量, 不需要赋值
16 //待排序的字符串放在 s 数组中, 从 s[0] 到 s[n-1], 长度为 n, 且最大值小于 m,
17 //除 s[n-1] 外的所有 s[i] 都大于 0, r[n-1]=0
18 //函数结束以后结果放在 sa 数组中
19 bool cmp(int *r,int a,int b,int l){
20     return r[a] == r[b] && r[a+l] == r[b+l];
21 }
22 void da(int str[],int sa[],int rank[],int height[],int n,int m){
23     n++;
24     int i, j, p, *x = t1, *y = t2;
25     //第一轮基数排序, 如果 s 的最大值很大, 可改为快速排序
26     for(i = 0; i < m; i++) c[i] = 0;
27     for(i = 0; i < n; i++) c[x[i] = str[i]]++;
28     for(i = 1; i < m; i++) c[i] += c[i-1];
29     for(i = n-1; i >= 0; i--) sa[--c[x[i]]] = i;
30     for(j = 1; j <= n; j <= 1){
31         p = 0;
32         //直接利用 sa 数组排序第二关键字
33         for(i = n-j; i < n; i++) y[p++] = i; //后面的 j 个数第二关键字为
            空的最小
34         for(i = 0; i < n; i++) if(sa[i] >= j) y[p++] = sa[i] - j;
35         //这样数组 y 保存的就是按照第二关键字排序的结果
36         //基数排序第一关键字
37         for(i = 0; i < m; i++) c[i] = 0;
38         for(i = 0; i < n; i++) c[x[y[i]]]++;
39         for(i = 1; i < m; i++) c[i] += c[i-1];
40         for(i = n-1; i >= 0; i--) sa[--c[x[y[i]]]] = y[i];
41         //根据 sa 和 x 数组计算新的 x 数组
42         swap(x,y);
43         p = 1; x[sa[0]] = 0;
44         for(i = 1; i < n; i++)
45             x[sa[i]] = cmp(y,sa[i-1],sa[i],j)?p-1:p++;
46         if(p >= n) break;
47         m = p; //下次基数排序的最大值
48     }
49     int k = 0;
50     n--;
51     for(i = 0; i <= n; i++) rank[sa[i]] = i;
52     for(i = 0; i < n; i++){
53         if(k) k--;
54         j = sa[rank[i]-1];
55         while(str[i+k] == str[j+k]) k++;
56         height[rank[i]] = k;
57     }
58 }
59 int rank[MAXN],height[MAXN];
60 int RMQ[MAXN];
61 int mm[MAXN];

```

```

62 int best[20][MAXN];
63 void initRMQ(int n){
64     mm[0]=-1;
65     for(int i=1;i<=n;i++)
66         mm[i]=((i&(i-1))==0)?mm[i-1]+1:mm[i-1];
67     for(int i=1;i<=n;i++)best[0][i]=i;
68     for(int i=1;i<=mm[n];i++)
69         for(int j=1;j+(1<<i)-1<=n;j++){
70             int a=best[i-1][j];
71             int b=best[i-1][j+(1<<(i-1))];
72             if(RMQ[a]<RMQ[b])best[i][j]=a;
73             else best[i][j]=b;
74         }
75 }
76 int askRMQ(int a,int b){
77     int t;
78     t=mm[b-a+1];
79     b-=(1<<t)-1;
80     a=best[t][a];b=best[t][b];
81     return RMQ[a]<RMQ[b]?a:b;
82 }
83 int lcp(int a,int b){
84     a=rank[a];b=rank[b];
85     if(a>b)swap(a,b);
86     return height[askRMQ(a+1,b)];
87 }
88 char str[MAXN];
89 int r[MAXN];
90 int sa[MAXN];
91 int main()
92 {
93     while(scanf("%s",str) == 1){
94         int len = strlen(str);
95         int n = 2*len + 1;
96         for(int i = 0;i < len;i++)r[i] = str[i];
97         for(int i = 0;i < len;i++)r[len + 1 + i] = str[len - 1 - i]
98         ];
99         r[len] = 1;
100        r[n] = 0;
101        da(r,sa,rank,height,n,128);
102        for(int i=1;i<=n;i++)RMQ[i]=height[i];
103        initRMQ(n);
104        int ans=0,st;
105        int tmp;
106        for(int i=0;i<len;i++){
107            tmp=lcp(i,n-i);//偶对称
108            if(2*tmp>ans){
109                ans=2*tmp;
110                st=i-tmp;
111            }
112            tmp=lcp(i,n-i-1);//奇数对称

```

```

112         if(2*tmp-1>ans){
113             ans=2*tmp-1;
114             st=i-tmp+1;
115         }
116     }
117     str[st+ans]=0;
118     printf("%s\n",str+st);
119 }
120 return 0;
121 }

```

1.5.2 DC3

da[] 和 str[] 数组要开大三倍，相关数组也是三倍

```

1  /*
2  * 后缀数组
3  *  DC3  算法，复杂度 O(n)
4  *  所有的相关数组都要开三倍
5  */
6  const int MAXN = 2010;
7  #define F(x) ((x)/3+((x)%3==1?0:tb))
8  #define G(x) ((x)<tb?(x)*3+1:((x)-tb)*3+2)
9  int wa[MAXN*3],wb[MAXN*3],wv[MAXN*3],wss[MAXN*3];
10 int c0(int *r,int a,int b){
11     return r[a] == r[b] && r[a+1] == r[b+1] && r[a+2] == r[b+2];
12 }
13 int c12(int k,int *r,int a,int b){
14     if(k == 2)
15         return r[a] < r[b] || ( r[a] == r[b] && c12(1,r,a+1,b+1) );
16     else return r[a] < r[b] || ( r[a] == r[b] && wv[a+1] < wv[b+1] );
17 }
18 void sort(int *r,int *a,int *b,int n,int m){
19     int i;
20     for(i = 0;i < n;i++)wv[i] = r[a[i]];
21     for(i = 0;i < m;i++)wss[i] = 0;
22     for(i = 0;i < n;i++)wss[wv[i]]++;
23     for(i = 1;i < m;i++)wss[i] += wss[i-1];
24     for(i = n-1;i >= 0;i--)
25         b[--wss[wv[i]]] = a[i];
26 }
27 void dc3(int *r,int *sa,int n,int m){
28     int i, j, *rn = r + n;
29     int *san = sa + n, ta = 0, tb = (n+1)/3, tbc = 0, p;
30     r[n] = r[n+1] = 0;
31     for(i = 0;i < n;i++)if(i % 3 != 0)wa[tbc++] = i;
32     sort(r + 2, wa, wb, tbc, m);
33     sort(r + 1, wb, wa, tbc, m);
34     sort(r, wa, wb, tbc, m);
35     for(p = 1, rn[F(wb[0])] = 0, i = 1;i < tbc;i++)
36         rn[F(wb[i])] = c0(r, wb[i-1], wb[i]) ? p - 1 : p++;
37     if(p < tbc)dc3(rn,san,tbc,p);

```



```

38     else for(i = 0; i < tbc; i++) san[rn[i]] = i;
39     for(i = 0; i < tbc; i++) if(san[i] < tb) wb[ta++] = san[i] * 3;
40     if(n % 3 == 1) wb[ta++] = n - 1;
41     sort(r, wb, wa, ta, m);
42     for(i = 0; i < tbc; i++) wv[wb[i] = G(san[i])] = i;
43     for(i = 0, j = 0, p = 0; i < ta && j < tbc; p++)
44         sa[p] = c12(wb[j] % 3, r, wa[i], wb[j]) ? wa[i++] : wb[j
            ++];
45     for(; i < ta; p++) sa[p] = wa[i++];
46     for(; j < tbc; p++) sa[p] = wb[j++];
47 }
48 //str 和 sa 也要三倍
49 void da(int str[], int sa[], int rank[], int height[], int n, int m){
50     for(int i = n; i < n*3; i++)
51         str[i] = 0;
52     dc3(str, sa, n+1, m);
53     int i, j, k = 0;
54     for(i = 0; i <= n; i++) rank[sa[i]] = i;
55     for(i = 0; i < n; i++){
56         if(k) k--;
57         j = sa[rank[i]-1];
58         while(str[i+k] == str[j+k]) k++;
59         height[rank[i]] = k;
60     }
61 }

```

1.6 后缀自动机

1.6.1 基本函数

```

1  const int CHAR = 26;
2  const int MAXN = 250010;
3  struct SAM_Node{
4      SAM_Node *fa, *next[CHAR];
5      int len;
6      long long cnt;
7      void clear(){
8          fa = 0;
9          memset(next, 0, sizeof(next));
10         cnt = 0;
11     }
12 }pool[MAXN*2];
13 SAM_Node *root, *tail;
14 SAM_Node* newnode(int len){
15     SAM_Node* cur = tail++;
16     cur->clear();
17     cur->len = len;
18     return cur;
19 }
20 void SAM_init(){
21     tail = pool;

```

```

22     root = newnode(0);
23 }
24 SAM_Node* extend(SAM_Node* last,int x){
25     SAM_Node *p = last, *np = newnode(p->len+1);
26     while(p && !p->next[x])
27         p->next[x] = np, p = p->fa;
28     if(!p)np->fa = root;
29     else {
30         SAM_Node* q = p->next[x];
31         if(q->len == p->len+1)np->fa = q;
32         else {
33             SAM_Node* nq = newnode(p->len+1);
34             memcpy(nq->next,q->next,sizeof(q->next));
35             nq->fa = q->fa;
36             q->fa = np->fa = nq;
37             while(p && p->next[x] == q)
38                 p->next[x] = nq, p = p->fa;
39         }
40     }
41     return np;
42 }

```

1.6.2 例题

CC TSUBSTR

给了一个 Trie 树，求 Trie 子树上的第 k 大的子串。

```

1  /*
2   *  http://www.codechef.com/problems/TSUBSTR/
3  Input:
4  8 4
5  abcbBaca
6  1 2
7  2 3
8  1 4
9  4 5
10 4 6
11 4 7
12 1 8
13 abcdefghijklmnopqrstuvwxyz 5
14 abcdefghijklmnopqrstuvwxyz 1
15 bcdefghijklmnopqrstuvwxyz 5
16 abcdefghijklmnopqrstuvwxyz 100
17
18 Output:
19 12
20 aba
21
22 ba
23 -1
24 */
25 const int CHAR = 26;

```

```

26 const int MAXN = 250010;
27 struct SAM_Node{
28     SAM_Node *fa,*next[CHAR];
29     int len;
30     long long cnt;
31     void clear(){
32         fa = 0;
33         memset(next,0,sizeof(next));
34         cnt = 0;
35     }
36 }pool[MAXN*2];
37 SAM_Node *root,*tail;
38 SAM_Node* newnode(int len){
39     SAM_Node* cur = tail++;
40     cur->clear();
41     cur->len = len;
42     return cur;
43 }
44 void SAM_init(){
45     tail = pool;
46     root = newnode(0);
47 }
48 SAM_Node* extend(SAM_Node* last,int x){
49     SAM_Node *p = last, *np = newnode(p->len+1);
50     while(p && !p->next[x])
51         p->next[x] = np, p = p->fa;
52     if(!p)np->fa = root;
53     else {
54         SAM_Node* q = p->next[x];
55         if(q->len == p->len+1)np->fa = q;
56         else {
57             SAM_Node* nq = newnode(p->len+1);
58             memcpy(nq->next,q->next,sizeof(q->next));
59             nq->fa = q->fa;
60             q->fa = np->fa = nq;
61             while(p && p->next[x] == q)
62                 p->next[x] = nq, p = p->fa;
63         }
64     }
65     return np;
66 }
67 char str[MAXN];
68 struct Edge
69 {
70     int to,next;
71 }edge[MAXN*2];
72 int head[MAXN],tot;
73 void addedge(int u,int v){
74     edge[tot].to = v;
75     edge[tot].next = head[u];
76     head[u] = tot++;

```

```

77 }
78
79 SAM_Node *end[MAXN];
80 int topcnt[MAXN]; // 拓扑排序使用
81 SAM_Node *topsam[MAXN*2];
82 char s2[40];
83 int order[40];
84
85 int main()
86 {
87     int n,Q;
88     while(scanf("%d%d",&n,&Q) == 2){
89         scanf("%s",str+1);
90         memset(head,-1,sizeof(head));tot = 0;
91         int u,v;
92         for(int i = 1;i < n;i++){
93             scanf("%d%d",&u,&v);
94             addedge(u,v); addedge(v,u);
95         }
96         addedge(0,1);
97         SAM_init();
98         memset(end,0,sizeof(end));
99         end[0] = root;
100         queue<int>q;
101         q.push(0);
102         while(!q.empty()){
103             u = q.front();
104             q.pop();
105             for(int i = head[u];i != -1;i = edge[i].next){
106                 v = edge[i].to;
107                 if(end[v] != 0)continue;
108                 end[v] = extend(end[u],str[v]-'a');
109                 q.push(v);
110             }
111         }
112         memset(topcnt,0,sizeof(topcnt));
113         int num = tail - pool;
114         for(int i = 0;i < num;i++)topcnt[pool[i].len]++;
115         for(int i = 1;i <= n;i++)topcnt[i] += topcnt[i-1];
116         for(int i = 0;i < num;i++)topsam[--topcnt[pool[i].len]] = &
            pool[i];
117
118         for(int i = num-1;i >= 0;i--){
119             SAM_Node *p = topsam[i];
120             p->cnt = 1;
121             for(int i = 0;i < 26;i++)
122                 if(p->next[i])
123                     p->cnt += p->next[i]->cnt;
124         }
125         printf("%lld\n",root->cnt);
126         long long k;

```

```

127     while(Q--){
128         scanf("%s",s2);
129         for(int i = 0;i < 26;i++)order[i] = s2[i]-'a';
130         scanf("%lld",&k);
131         if(k > root->cnt){
132             printf("-1\n");
133             continue;
134         }
135         SAM_Node *p = root;
136         //这里的第 k 个子串是从空串算起的
137         while( (--k) > 0 ){
138             for(int i = 0;i < 26;i++)
139                 if(p->next[order[i]]){
140                     if(k <= p->next[order[i]]->cnt){
141                         printf("%c",'a'+order[i]);
142                         p = p->next[order[i]];
143                         break; //这个不要忘记
144                     }
145                     else k -= p->next[order[i]]->cnt;
146                 }
147             }
148         printf("\n");
149     }
150 }
151 return 0;
152 }

```

CF129 E

给了 n 个字符串，求每个字符串有多少个至少出现在 k 个字符串中的子串
fail 树，两遍 dfs, 经典题。

```

1  /* http://codeforces.com/contest/204/problem/E
2  input
3  3 1
4  abc
5  a
6  ab
7  output
8  6 1 3
9  input
10 7 4
11 rubik
12 furik
13 abab
14 baba
15 aaabbbababa
16 abababababa
17 zero
18 output
19 1 0 9 9 21 30 0
20 */
21 const int CHAR = 26;

```

```

22 const int MAXN = 100010;
23 //*****SAM*****
24 struct SAM_Node{
25     SAM_Node *fa,*next[CHAR];
26     int len;
27     void clear(){
28         fa = 0;
29         memset(next,0,sizeof(next));
30     }
31 }pool[MAXN*2];
32 SAM_Node *root,*tail;
33 SAM_Node* newnode(int len){
34     SAM_Node* cur = tail++;
35     cur->clear();
36     cur->len = len;
37     return cur;
38 }
39 void SAM_init(){
40     tail = pool;
41     root = newnode(0);
42 }
43 SAM_Node* extend(SAM_Node* last,int x){
44     SAM_Node *p = last, *np = newnode(p->len+1);
45     while(p && !p->next[x])
46         p->next[x] = np, p = p->fa;
47     if(!p)np->fa = root;
48     else {
49         SAM_Node* q = p->next[x];
50         if(q->len == p->len+1)np->fa = q;
51         else {
52             SAM_Node* nq = newnode(p->len+1);
53             memcpy(nq->next,q->next,sizeof(q->next));
54             nq->fa = q->fa;
55             q->fa = np->fa = nq;
56             while(p && p->next[x] == q)
57                 p->next[x] = nq, p = p->fa;
58         }
59     }
60     return np;
61 }
62 //*****Trie*****
63 struct Trie_Node{
64     int next[CHAR];
65     vector<int>belongs;
66 }trie[MAXN];
67 int trie_root,trie_tot;
68 int trie_newnode(){
69     memset(trie[trie_tot].next,-1,sizeof(trie[trie_tot].next));
70     trie[trie_tot].belongs.clear();
71     return trie_tot++;
72 }

```

```

73 void Trie_init(){
74     trie_tot = 0;
75     trie_root = trie_newnode();
76 }
77 void insert(char buf[],int id){
78     int now = trie_root;
79     int len = strlen(buf);
80     for(int i = 0;i < len;i++){
81         if(trie[now].next[buf[i]-'a'] == -1)
82             trie[now].next[buf[i]-'a'] = trie_newnode();
83         now = trie[now].next[buf[i]-'a'];
84         trie[now].belongs.push_back(id);
85     }
86 }
87 //***** fail 树*****
88 struct Edge{
89     int to,next;
90 }edge[MAXN*2];
91 int head[MAXN*2],tot;
92 void addedge(int u,int v){
93     edge[tot].to = v; edge[tot].next = head[u]; head[u] = tot++;
94 }
95 int MtoT[MAXN*2]; //SAM 结点映射到 Trie 结点
96 int cnt[MAXN*2];
97 int F[MAXN*2];
98 int find(int x){
99     if(F[x] == -1)return x;
100     return F[x] = find(F[x]);
101 }
102 void bing(int u,int v) //注意方向性
103 {
104     int t1 = find(u);
105     int t2 = find(v);
106     if(t1 != t2)F[t1] = t2;
107 }
108 int L[MAXN];
109 void Tarjan(int u){
110     for(int i = head[u];i != -1;i = edge[i].next){
111         Tarjan(edge[i].to);
112         bing(edge[i].to,u);
113     }
114     if(MtoT[u]){
115         int tt = MtoT[u];
116         int sz = trie[tt].belongs.size();
117         for(int i = 0;i < sz;i++){
118             int v = trie[tt].belongs[i];
119             cnt[find(L[v])]--;
120             cnt[u]++;
121             L[v] = u;
122         }
123     }

```

```

124 }
125 void dfs1(int u){
126     for(int i = head[u]; i != -1; i = edge[i].next){
127         dfs1(edge[i].to);
128         cnt[u] += cnt[edge[i].to];
129     }
130 }
131 long long ans[MAXN];
132 void dfs2(int u){
133     for(int i = head[u]; i != -1; i = edge[i].next){
134         int v = edge[i].to;
135         cnt[v] += cnt[u];
136         dfs2(v);
137     }
138     if(MtoT[u]){
139         int tt = MtoT[u];
140         int sz = trie[tt].belongs.size();
141         for(int i = 0; i < sz; i++){
142             int v = trie[tt].belongs[i];
143             ans[v] += cnt[u];
144         }
145     }
146 }
147
148 char str[MAXN];
149 SAM_Node *end[MAXN];
150 int main()
151 {
152     int n,k;
153     while(scanf("%d%d",&n,&k) == 2){
154         Trie_init();
155         for(int i = 0; i < n; i++){
156             scanf("%s",str);
157             insert(str,i);
158         }
159         SAM_init();
160         //根据 Trie 建立 SAM
161         memset(end,0,sizeof(end));
162         end[0] = root;
163         memset(MtoT,0,sizeof(MtoT));
164         MtoT[root-pool] = 0;
165         queue<int>q;
166         q.push(trie_root);
167         while(!q.empty()){
168             int u = q.front();
169             q.pop();
170             for(int i = 0; i < 26; i++){
171                 if(trie[u].next[i] == -1)continue;
172                 int v = trie[u].next[i];
173                 end[v] = extend(end[u],i);
174                 MtoT[end[v]-pool] = v;

```



```

175         q.push(v);
176     }
177 }
178 //建立 fail 树
179 int num = tail - pool;
180 memset(head,-1,sizeof(head));
181 tot = 0;
182 for(SAM_Node *p = pool+1;p < tail;p++)
183     addedge(p->fa - pool,p - pool);
184 memset(cnt,0,sizeof(cnt));
185 memset(F,-1,sizeof(F));
186 memset(L,0,sizeof(L));
187 Tarjan(0);
188 dfs1(0);
189 for(int i = 0;i < num;i++){
190     if(cnt[i] >= k)cnt[i] = pool[i].len - pool[i].fa->len;
191     else cnt[i] = 0;
192 }
193 memset(ans,0,sizeof(ans));
194 dfs2(0);
195 for(int i = 0;i < n;i++){
196     printf("%I64d",ans[i]);
197     if(i < n-1)printf("_");
198     else printf("\n");
199 }
200 }
201 return 0;
202 }

```

1.7 字符串 hash

HDU4622 求区间不相同子串个数

```

1  const int HASH = 10007;
2  const int MAXN = 2010;
3  struct HASHMAP{
4      int head[HASH],next[MAXN],size;
5      unsigned long long state[MAXN];
6      int f[MAXN];
7      void init(){
8          size = 0;
9          memset(head,-1,sizeof(head));
10     }
11     int insert(unsigned long long val,int _id){
12         int h = val%HASH;
13         for(int i = head[h]; i != -1;i = next[i])
14             if(val == state[i]){
15                 int tmp = f[i];
16                 f[i] = _id;
17                 return tmp;
18             }
19         f[size] = _id;

```

```

20     state[size] = val;
21     next[size] = head[h];
22     head[h] = size++;
23     return 0;
24 }
25 }H;
26 const int SEED = 13331;
27 unsigned long long P[MAXN];
28 unsigned long long S[MAXN];
29 char str[MAXN];
30 int ans[MAXN][MAXN];
31 int main(){
32     P[0] = 1;
33     for(int i = 1; i < MAXN; i++)
34         P[i] = P[i-1] * SEED;
35     int T;
36     scanf("%d",&T);
37     while(T--){
38         scanf("%s",str);
39         int n = strlen(str);
40         S[0] = 0;
41         for(int i = 1; i <= n; i++)
42             S[i] = S[i-1]*SEED + str[i-1];
43         memset(ans,0,sizeof(ans));
44         for(int L = 1; L <= n; L++){
45             H.init();
46             for(int i = 1; i + L - 1 <= n; i++){
47                 int l = H.insert(S[i+L-1] - S[i-1]*P[L],i);
48                 ans[i][i+L-1] ++;
49                 ans[l][i+L-1]--;
50             }
51         }
52         for(int i = n; i >= 0; i--)
53             for(int j = i; j <= n; j++)
54                 ans[i][j] += ans[i+1][j] + ans[i][j-1] - ans[i+1][j-1];
55         int m,u,v;
56         scanf("%d",&m);
57         while(m--){
58             scanf("%d%d",&u,&v);
59             printf("%d\n",ans[u][v]);
60         }
61     }
62     return 0;
63 }

```

2 数学

2.1 素数

2.1.1 素数筛选（判断 $\leq \text{MAXN}$ 的数是否素数）

```

1  /*
2   * 素数筛选，判断小于 MAXN 的数是不是素数。
3   * notprime 是一张表，为 false 表示是素数，true 表示不是素数
4   */
5  const int MAXN=1000010;
6  bool notprime[MAXN]; //值为 false 表示素数，值为 true 表示非素数
7  void init(){
8      memset(notprime,false,sizeof(notprime));
9      notprime[0]=notprime[1]=true;
10     for(int i=2;i<MAXN;i++){
11         if(!notprime[i]){
12             if(i>MAXN/i) continue; //防止后面 i*i 溢出（或者 i,j 用 long
                long）
13             //直接从 i*i 开始就可以，小于 i 倍的已经筛选过了，注意是 j+=i
14             for(int j=i*i;j<MAXN;j+=i)
15                 notprime[j]=true;
16         }
17     }

```

2.1.2 素数筛选（筛选出小于等于 MAXN 的素数）

```

1  /*
2   * 素数筛选，存在小于等于 MAXN 的素数
3   * prime[0] 存的是素数的个数
4   */
5  const int MAXN=10000;
6  int prime[MAXN+1];
7  void getPrime(){
8      memset(prime,0,sizeof(prime));
9      for(int i=2;i<=MAXN;i++){
10         if(!prime[i]) prime[++prime[0]]=i;
11         for(int j=1;j<=prime[0]&&prime[j]<=MAXN/i;j++){
12             prime[prime[j]*i]=1;
13             if(i%prime[j]==0) break;
14         }
15     }
16 }

```

2.1.3 大区间素数筛选（POJ 2689）

```

1  /*
2   * POJ 2689 Prime Distance
3   * 给出一个区间 [L,U]，找出区间内容、相邻的距离最近的两个素数和
4   * 距离最远的两个素数。
5   *  $1 \leq L < U \leq 2,147,483,647$  区间长度不超过 1,000,000
6   * 就是要筛选出 [L,U] 之间的素数
7   */

```

```

8  const int MAXN=100010;
9  int prime[MAXN+1];
10 void getPrime(){
11     memset(prime,0,sizeof(prime));
12     for(int i=2;i<=MAXN;i++){
13         if(!prime[i])prime[++prime[0]]=i;
14         for(int j=1;j<=prime[0]&&prime[j]<=MAXN/i;j++){
15             prime[prime[j]*i]=1;
16             if(i%prime[j]==0)break;
17         }
18     }
19 }
20 bool notprime[1000010];
21 int prime2[1000010];
22 void getPrime2(int L,int R){
23     memset(notprime,false,sizeof(notprime));
24     if(L<2)L=2;
25     for(int i=1;i<=prime[0]&&(long long)prime[i]*prime[i]<=R;i++){
26         int s=L/prime[i]+(L%prime[i]>0);
27         if(s==1)s=2;
28         for(int j=s;(long long)j*prime[i]<=R;j++){
29             if((long long)j*prime[i]>=L)
30                 notprime[j*prime[i]-L]=true;
31         }
32     }
33     prime2[0]=0;
34     for(int i=0;i<=R-L;i++){
35         if(!notprime[i])
36             prime2[++prime2[0]]=i+L;
37 }
38 int main(){
39     getPrime();
40     int L,U;
41     while(scanf("%d%d",&L,&U)==2){
42         getPrime2(L,U);
43         if(prime2[0]<2)printf("There are no adjacent primes.\n");
44         else{
45             int x1=0,x2=1000000000,y1=0,y2=0;
46             for(int i=1;i<prime2[0];i++){
47                 if(prime2[i+1]-prime2[i]<x2-x1){
48                     x1=prime2[i];
49                     x2=prime2[i+1];
50                 }
51                 if(prime2[i+1]-prime2[i]>y2-y1){
52                     y1=prime2[i];
53                     y2=prime2[i+1];
54                 }
55             }
56             printf("%d,%d are closest,%d,%d are most distant.\n",
57                 x1,x2,y1,y2);
58         }
59     }
60 }

```

58 | }

2.2 素数筛选和合数分解

```

1 //*****
2 //素数筛选和合数分解
3 const int MAXN=10000;
4 int prime[MAXN+1];
5 void getPrime(){
6     memset(prime,0,sizeof(prime));
7     for(int i=2;i<=MAXN;i++){
8         if(!prime[i])prime[++prime[0]]=i;
9         for(int j=1;j<=prime[0]&&prime[j]<=MAXN/i;j++){
10             prime[prime[j]*i]=1;
11             if(i%prime[j]==0) break;
12         }
13     }
14 }
15 long long factor[100][2];
16 int fatCnt;
17 int getFactors(long long x){
18     fatCnt=0;
19     long long tmp=x;
20     for(int i=1;prime[i]<=tmp/prime[i];i++){
21         factor[fatCnt][1]=0;
22         if(tmp%prime[i]==0){
23             factor[fatCnt][0]=prime[i];
24             while(tmp%prime[i]==0){
25                 factor[fatCnt][1]++;
26                 tmp/=prime[i];
27             }
28             fatCnt++;
29         }
30     }
31     if(tmp!=1){
32         factor[fatCnt][0]=tmp;
33         factor[fatCnt++][1]=1;
34     }
35     return fatCnt;
36 }
37 //*****

```

2.3 扩展欧几里得算法（求 $ax+by=gcd$ 的解以及逆元）

```

1 //*****
2 //返回 d=gcd(a,b); 和对应于等式  $ax+by=d$  中的  $x,y$ 
3 long long extend_gcd(long long a,long long b,long long &x,long long
    &y){
4     if(a==0&&b==0) return -1;//无最大公约数
5     if(b==0){x=1;y=0;return a;}
6     long long d=extend_gcd(b,a%b,y,x);
7     y-=a/b*x;

```

```

8     return d;
9 }
10 //***** 求逆元 *****
11 //ax = 1(mod n)
12 long long mod_reverse(long long a,long long n){
13     long long x,y;
14     long long d=extend_gcd(a,n,x,y);
15     if(d==1) return (x%n+n)%n;
16     else return -1;
17 }

```

2.4 求逆元

2.4.1 扩展欧几里德法

见上面的写法

2.4.2 简洁写法

注意：这个只能求 $a < m$ 的情况，而且必须保证 a 和 m 互质

```

1 //求 ax = 1( mod m) 的 x 值，就是逆元 (0<a<m)
2 long long inv(long long a,long long m){
3     if(a == 1)return 1;
4     return inv(m%a,m)*(m-m/a)%m;
5 }

```

2.4.3 利用欧拉函数

mod 为素数, 而且 a 和 m 互质

```

1 long long inv(long long a,long long mod)//为素数mod
2 {
3     return pow_m(a,mod-2,mod);
4 }

```

2.5 模线性方程组

```

1 long long extend_gcd(long long a,long long b,long long &x,long long
    &y){
2     if(a == 0 && b == 0)return -1;
3     if(b == 0 ){x = 1; y = 0;return a;}
4     long long d = extend_gcd(b,a%b,y,x);
5     y -= a/b*x;
6     return d;
7 }
8 int m[10],a[10];//模数为 m, 余数为 a,X % m = a
9 bool solve(int &m0,int &a0,int m,int a){
10     long long y,x;
11     int g = extend_gcd(m0,m,x,y);
12     if( abs(a - a0)%g )return false;
13     x *= (a - a0)/g;
14     x %= m/g;

```

```

15     a0 = (x*m0 + a0);
16     m0 *= m/g;
17     a0 %= m0;
18     if( a0 < 0 )a0 += m0;
19     return true;
20 }
21 /*
22  * 无解返回 false, 有解返回 true;
23  * 解的形式最后为 a0 + m0 * t (0<=a0<m0)
24  */
25 bool MLES(int &m0 ,int &a0,int n)//解为 X = a0 + m0 * k
26 {
27     bool flag = true;
28     m0 = 1;
29     a0 = 0;
30     for(int i = 0;i < n;i++)
31         if( !solve(m0,a0,m[i],a[i]) )
32         {
33             flag = false;
34             break;
35         }
36     return flag;
37 }

```

2.6 随机素数测试和大数分解 (POJ 1811)

```

1  /* *****
2  * Miller_Rabin 算法进行素数测试
3  * 速度快可以判断一个 < 2^63 的数是不是素数
4  *
5  * *****/
6  const int S = 8; //随机算法判定次数一般 8~10 就够了
7  // 计算 ret = (a*b)%c      a,b,c < 2^63
8  long long mult_mod(long long a,long long b,long long c){
9      a %= c;
10     b %= c;
11     long long ret = 0;
12     long long tmp = a;
13     while(b){
14         if(b & 1){
15             ret += tmp;
16             if(ret > c)ret -= c;//直接取模慢很多
17         }
18         tmp <<= 1;
19         if(tmp > c)tmp -= c;
20         b >>= 1;
21     }
22     return ret;
23 }
24 // 计算 ret = (a^n)%mod
25 long long pow_mod(long long a,long long n,long long mod){
26     long long ret = 1;

```

```

27     long long temp = a%mod;
28     while(n){
29         if(n & 1)ret = mult_mod(ret,temp,mod);
30         temp = mult_mod(temp,temp,mod);
31         n >>= 1;
32     }
33     return ret;
34 }
35 // 通过  $a^{(n-1)}=1 \pmod n$  来判断 n 是不是素数
36 //  $n-1 = x * 2^t$  中间使用二次判断
37 // 是合数返回 true, 不一定是合数返回 false
38 bool check(long long a,long long n,long long x,long long t){
39     long long ret = pow_mod(a,x,n);
40     long long last = ret;
41     for(int i = 1;i <= t;i++){
42         ret = mult_mod(ret,ret,n);
43         if(ret == 1 && last != 1 && last != n-1)return true;//合数
44         last = ret;
45     }
46     if(ret != 1)return true;
47     else return false;
48 }
49 //*****
50 // Miller_Rabin 算法
51 // 是素数返回 true,(可能是伪素数)
52 // 不是素数返回 false
53 //*****
54 bool Miller_Rabin(long long n){
55     if( n < 2)return false;
56     if( n == 2)return true;
57     if( (n&1) == 0)return false;//偶数
58     long long x = n - 1;
59     long long t = 0;
60     while( (x&1)==0 ){x >>= 1; t++;}
61
62     srand(time(NULL));/* ***** */
63
64     for(int i = 0;i < S;i++){
65         long long a = rand()%(n-1) + 1;
66         if( check(a,n,x,t) )
67             return false;
68     }
69     return true;
70 }
71
72 //*****
73 // pollard_rho 算法进行质因数分解
74 //*****
75 long long factor[100];//质因数分解结果（刚返回时时无序的）
76 int tol;//质因素的个数, 编号 0~tol-1
77

```



```

78 long long gcd(long long a,long long b){
79     long long t;
80     while(b){
81         t = a;
82         a = b;
83         b = t%b;
84     }
85     if(a >= 0)return a;
86     else return -a;
87 }
88
89 //找出一个因子
90 long long pollard_rho(long long x,long long c){
91     long long i = 1, k = 2;
92     srand(time(NULL));
93     long long x0 = rand()%(x-1) + 1;
94     long long y = x0;
95     while(1){
96         i ++;
97         x0 = (mult_mod(x0,x0,x) + c)%x;
98         long long d = gcd(y - x0,x);
99         if( d != 1 && d != x)return d;
100        if(y == x0)return x;
101        if(i == k){y = x0; k += k;}
102    }
103 }
104 //对 n 进行素因子分解, 存入 factor. k 设置为 107 左右即可
105 void findfac(long long n,int k){
106     if(n == 1)return;
107     if(Miller_Rabin(n))
108     {
109         factor[tol++] = n;
110         return;
111     }
112     long long p = n;
113     int c = k;
114     while( p >= n)p = pollard_rho(p,c--); //值变化, 防止死循环 k
115     findfac(p,k);
116     findfac(n/p,k);
117 }
118 //POJ 1811
119 //给出一个N( $2 \leq N < 2^{54}$ ),如果是素数, 输出"Prime", 否则输出最小的素因子
120 int main(){
121     int T;
122     long long n;
123     scanf("%d",&T);
124     while(T--){
125         scanf("%I64d",&n);
126         if(Miller_Rabin(n))printf("Prime\n");
127         else{
128             tol = 0;

```

```

129         findfac(n,107);
130         long long ans = factor[0];
131         for(int i = 1;i < tol;i++)
132             ans = min(ans,factor[i]);
133         printf("%I64d\n",ans);
134     }
135 }
136 return 0;
137 }

```

2.7 欧拉函数

2.7.1 分解质因素求欧拉函数

```

1 getFactors(n);
2 int ret = n;
3 for(int i = 0;i < fatCnt;i++){
4     ret = ret/factor[i][0]*(factor[i][0]-1);
5 }

```

2.7.2 筛法欧拉函数

```

1 int euler[3000001];
2 void getEuler(){
3     memset(euler,0,sizeof(euler));
4     euler[1] = 1;
5     for(int i = 2;i <= 3000000;i++)
6         if(!euler[i])
7             for(int j = i;j <= 3000000; j += i){
8                 if(!euler[j])
9                     euler[j] = j;
10                euler[j] = euler[j]/i*(i-1);
11            }
12 }

```

2.7.3 求单个数的欧拉函数

```

1 long long eular(long long n){
2     long long ans = n;
3     for(int i = 2;i*i <= n;i++){
4         if(n % i == 0){
5             ans -= ans/i;
6             while(n % i == 0)
7                 n /= i;
8         }
9     }
10     if(n > 1)ans -= ans/n;
11     return ans;
12 }

```

2.7.4 线性筛（同时得到欧拉函数和素数表）

```

1 const int MAXN = 10000000;
2 bool check[MAXN+10];

```

```

3  int phi[MAXN+10];
4  int prime[MAXN+10];
5  int tot;//素数的个数
6  void phi_and_prime_table(int N){
7      memset(check,false,sizeof(check));
8      phi[1] = 1;
9      tot = 0;
10     for(int i = 2; i <= N; i++){
11         if( !check[i] ){
12             prime[tot++] = i;
13             phi[i] = i-1;
14         }
15         for(int j = 0; j < tot; j++){
16             if(i * prime[j] > N)break;
17             check[i * prime[j]] = true;
18             if( i % prime[j] == 0){
19                 phi[i * prime[j]] = phi[i] * prime[j];
20                 break;
21             }
22             else{
23                 phi[i * prime[j]] = phi[i] * (prime[j] - 1);
24             }
25         }
26     }
27 }

```

2.8 高斯消元（浮点数）

```

1  #define eps 1e-9
2  const int MAXN=220;
3  double a[MAXN][MAXN],x[MAXN];//方程的左边的矩阵和等式右边的值，求解之后 x
   存的就是结果
4  int equ,var;//方程数和未知数个数
5  /*
6  * 返回 0 表示无解,1 表示有解
7  */
8  int Gauss(){
9      int i,j,k,col,max_r;
10     for(k=0,col=0;k<equ&&col<var;k++,col++){
11         max_r=k;
12         for(i=k+1;i<equ;i++){
13             if(fabs(a[i][col])>fabs(a[max_r][col]))
14                 max_r=i;
15             if(fabs(a[max_r][col])<eps)return 0;
16             if(k!=max_r){
17                 for(j=col;j<var;j++){
18                     swap(a[k][j],a[max_r][j]);
19                     swap(x[k],x[max_r]);
20                 }
21                 x[k]/=a[k][col];
22                 for(j=col+1;j<var;j++)a[k][j]/=a[k][col];
23                 a[k][col]=1;

```

```

24     for(i=0;i<=u;i++)
25         if(i!=k){
26             x[i]-=x[k]*a[i][col];
27             for(j=col+1;j<var;j++)a[i][j]-=a[k][j]*a[i][col];
28             a[i][col]=0;
29         }
30     }
31     return 1;
32 }

```

2.9 FFT

```

1 //HDU 1402 求高精度乘法
2 const double PI = acos(-1.0);
3 //复数结构体
4 struct Complex{
5     double x,y;//实部和虚部 x+yi
6     Complex(double _x = 0.0,double _y = 0.0){
7         x = _x;
8         y = _y;
9     }
10    Complex operator -(const Complex &b)const{
11        return Complex(x-b.x,y-b.y);
12    }
13    Complex operator +(const Complex &b)const{
14        return Complex(x+b.x,y+b.y);
15    }
16    Complex operator *(const Complex &b)const{
17        return Complex(x*b.x-y*b.y,x*b.y+y*b.x);
18    }
19 };
20 /*
21  * 进行 FFT 和 IFFT 前的反转变换。
22  * 位置 i 和 (i 二进制反转后位置) 互换
23  * len 必须为 2 的幂
24  */
25 void change(Complex y[],int len){
26     int i,j,k;
27     for(i = 1, j = len/2;i < len-1;i++){
28         if(i < j)swap(y[i],y[j]);
29         //交换互为小标反转的元素, i<j 保证交换一次
30         //i 做正常的 +1, j 左反转类型的 +1, 始终保持 i 和 j 是反转的
31         k = len/2;
32         while(j >= k){
33             j -= k;
34             k /= 2;
35         }
36         if(j < k)j += k;
37     }
38 }
39 /*
40  * 做 FFT

```

```

41  * len 必须为2^k形式
42  * on==1 时是 DFT, on== -1 时是 IDFT
43  */
44  void fft(Complex y[],int len,int on){
45      change(y,len);
46      for(int h = 2; h <= len; h <= 1){
47          Complex wn(cos(-on*2*PI/h),sin(-on*2*PI/h));
48          for(int j = 0;j < len;j+=h){
49              Complex w(1,0);
50              for(int k = j;k < j+h/2;k++){
51                  Complex u = y[k];
52                  Complex t = w*y[k+h/2];
53                  y[k] = u+t;
54                  y[k+h/2] = u-t;
55                  w = w*wn;
56              }
57          }
58      }
59      if(on == -1)
60          for(int i = 0;i < len;i++)
61              y[i].x /= len;
62  }
63  const int MAXN = 200010;
64  Complex x1[MAXN],x2[MAXN];
65  char str1[MAXN/2],str2[MAXN/2];
66  int sum[MAXN];
67  int main(){
68      while(scanf("%s%s",str1,str2)==2){
69          int len1 = strlen(str1);
70          int len2 = strlen(str2);
71          int len = 1;
72          while(len < len1*2 || len < len2*2)len<=1;
73          for(int i = 0;i < len1;i++)
74              x1[i] = Complex(str1[len1-1-i]-'0',0);
75          for(int i = len1;i < len;i++)
76              x1[i] = Complex(0,0);
77          for(int i = 0;i < len2;i++)
78              x2[i] = Complex(str2[len2-1-i]-'0',0);
79          for(int i = len2;i < len;i++)
80              x2[i] = Complex(0,0);
81          //求 DFT
82          fft(x1,len,1);
83          fft(x2,len,1);
84          for(int i = 0;i < len;i++)
85              x1[i] = x1[i]*x2[i];
86          fft(x1,len,-1);
87          for(int i = 0;i < len;i++)
88              sum[i] = (int)(x1[i].x+0.5);
89          for(int i = 0;i < len;i++){
90              sum[i+1]+=sum[i]/10;
91              sum[i]%=10;

```

```

92         }
93         len = len1+len2-1;
94         while(sum[len] <= 0 && len > 0)len--;
95         for(int i = len;i >= 0;i--)
96             printf("%c",sum[i]+'0');
97         printf("\n");
98     }
99     return 0;
100 }
101
102 //HDU 4609
103 //给出 n 条线段长度, 问任取 3 根, 组成三角形的概率。
104 //n<=10^5 用 FFT 求可以组成三角形的取法有几种
105 const int MAXN = 400040;
106 Complex x1[MAXN];
107 int a[MAXN/4];
108 long long num[MAXN]; //100000*100000 会超 int
109 long long sum[MAXN];
110 int main(){
111     int T;
112     int n;
113     scanf("%d",&T);
114     while(T--){
115         scanf("%d",&n);
116         memset(num,0,sizeof(num));
117         for(int i = 0;i < n;i++){
118             scanf("%d",&a[i]);
119             num[a[i]]++;
120         }
121         sort(a,a+n);
122         int len1 = a[n-1]+1;
123         int len = 1;
124         while( len < 2*len1 )len <= 1;
125         for(int i = 0;i < len1;i++)
126             x1[i] = Complex(num[i],0);
127         for(int i = len1;i < len;i++)
128             x1[i] = Complex(0,0);
129         fft(x1,len,1);
130         for(int i = 0;i < len;i++)
131             x1[i] = x1[i]*x1[i];
132         fft(x1,len,-1);
133         for(int i = 0;i < len;i++)
134             num[i] = (long long)(x1[i].x+0.5);
135         len = 2*a[n-1];
136         //减掉取两个相同的组合
137         for(int i = 0;i < n;i++)
138             num[a[i]+a[i]]--;
139         for(int i = 1;i <= len;i++)num[i]/=2;
140         sum[0] = 0;
141         for(int i = 1;i <= len;i++)
142             sum[i] = sum[i-1]+num[i];

```

```

143     long long cnt = 0;
144     for(int i = 0; i < n; i++){
145         cnt += sum[len] - sum[a[i]];
146         //减掉一个取大, 一个取小的
147         cnt -= (long long)(n-1-i)*i;
148         //减掉一个取本身, 另外一个取其它
149         cnt -= (n-1);
150         cnt -= (long long)(n-1-i)*(n-i-2)/2;
151     }
152     long long tot = (long long)n*(n-1)*(n-2)/6;
153     printf("%.7lf\n", (double)cnt/tot);
154 }
155 return 0;
156 }

```

2.10 高斯消元法求方程组的解

2.10.1 一类开关问题, 对 2 取模的 01 方程组

POJ 1681 需要枚举自由变元, 找解中 1 个数最少的

```

1 //对 2 取模的 01 方程组
2 const int MAXN = 300;
3 //有 equ 个方程, var 个变元。增广矩阵行数为 equ, 列数为 var+1, 分别为 0 到
  var
4 int equ, var;
5 int a[MAXN][MAXN]; //增广矩阵
6 int x[MAXN]; //解集
7 int free_x[MAXN]; //用来存储自由变元 (多解枚举自由变元可以使用)
8 int free_num; //自由变元的个数
9
10 //返回值为 -1 表示无解, 为 0 是唯一解, 否则返回自由变元个数
11 int Gauss(){
12     int max_r, col, k;
13     free_num = 0;
14     for(k = 0, col = 0; k < equ && col < var; k++, col++){
15         max_r = k;
16         for(int i = k+1; i < equ; i++){
17             if(abs(a[i][col]) > abs(a[max_r][col]))
18                 max_r = i;
19         }
20         if(a[max_r][col] == 0){
21             k--;
22             free_x[free_num++] = col; //这个是自由变元
23             continue;
24         }
25         if(max_r != k){
26             for(int j = col; j < var+1; j++)
27                 swap(a[k][j], a[max_r][j]);
28         }
29         for(int i = k+1; i < equ; i++){
30             if(a[i][col] != 0){
31                 for(int j = col; j < var+1; j++)

```

```

32         a[i][j] ^= a[k][j];
33     }
34 }
35 }
36 for(int i = k; i < equ; i++)
37     if(a[i][col] != 0)
38         return -1; //无解
39 if(k < var) return var - k; //自由变元个数
40 //唯一解, 回代
41 for(int i = var - 1; i >= 0; i--){
42     x[i] = a[i][var];
43     for(int j = i + 1; j < var; j++)
44         x[i] ^= (a[i][j] && x[j]);
45 }
46 return 0;
47 }
48 int n;
49 void init(){
50     memset(a, 0, sizeof(a));
51     memset(x, 0, sizeof(x));
52     equ = n * n;
53     var = n * n;
54     for(int i = 0; i < n; i++)
55         for(int j = 0; j < n; j++){
56             int t = i * n + j;
57             a[t][t] = 1;
58             if(i > 0) a[(i - 1) * n + j][t] = 1;
59             if(i < n - 1) a[(i + 1) * n + j][t] = 1;
60             if(j > 0) a[i * n + j - 1][t] = 1;
61             if(j < n - 1) a[i * n + j + 1][t] = 1;
62         }
63 }
64 void solve(){
65     int t = Gauss();
66     if(t == -1){
67         printf("inf\n");
68         return;
69     }
70     else if(t == 0){
71         int ans = 0;
72         for(int i = 0; i < n * n; i++)
73             ans += x[i];
74         printf("%d\n", ans);
75         return;
76     }
77     else
78     {
79         //枚举自由变元
80         int ans = 0x3f3f3f3f;
81         int tot = (1 << t);
82         for(int i = 0; i < tot; i++){

```



```

83         int cnt = 0;
84         for(int j = 0;j < t;j++){
85             if(i&(1<<j)){
86                 x[free_x[j]] = 1;
87                 cnt++;
88             }
89             else x[free_x[j]] = 0;
90         }
91         for(int j = var-t-1;j >= 0;j--){
92             int idx;
93             for(idx = j;idx < var;idx++){
94                 if(a[j][idx])
95                     break;
96                 x[idx] = a[j][var];
97                 for(int l = idx+1;l < var;l++){
98                     if(a[j][l])
99                         x[idx] ^= x[l];
100                 cnt += x[idx];
101             }
102             ans = min(ans,cnt);
103         }
104         printf("%d\n",ans);
105     }
106 }
107 char str[30][30];
108 int main(){
109     int T;
110     scanf("%d",&T);
111     while(T--){
112         scanf("%d",&n);
113         init();
114         for(int i = 0;i < n;i++){
115             scanf("%s",str[i]);
116             for(int j = 0;j < n;j++){
117                 if(str[i][j] == 'y')
118                     a[i*n+j][n*n] = 0;
119                 else a[i*n+j][n*n] = 1;
120             }
121         }
122         solve();
123     }
124     return 0;
125 }

```

2.10.2 解同余方程组

POJ 2947 Widget Factory

```

1 //求解对 MOD 取模的方程组
2 const int MOD = 7;
3 const int MAXN = 400;
4 int a[MAXN][MAXN]; //增广矩阵

```

```

5  int x[MAXN]; //最后得到的解集
6  inline int gcd(int a,int b){
7      while(b != 0){
8          int t = b;
9          b = a%b;
10         a = t;
11     }
12     return a;
13 }
14 inline int lcm(int a,int b){
15     return a/gcd(a,b)*b;
16 }
17 long long inv(long long a,long long m){
18     if(a == 1)return 1;
19     return inv(m%a,m)*(m-m/a)%m;
20 }
21 int Gauss(int equ,int var){
22     int max_r,col,k;
23     for(k = 0, col = 0; k < equ && col < var; k++,col++){
24         max_r = k;
25         for(int i = k+1; i < equ;i++){
26             if(abs(a[i][col]) > abs(a[max_r][col]))
27                 max_r = i;
28             if(a[max_r][col] == 0){
29                 k--;
30                 continue;
31             }
32             if(max_r != k)
33                 for(int j = col; j < var+1;j++)
34                     swap(a[k][j],a[max_r][j]);
35             for(int i = k+1;i < equ;i++){
36                 if(a[i][col] != 0){
37                     int LCM = lcm(abs(a[i][col]),abs(a[k][col]));
38                     int ta = LCM/abs(a[i][col]);
39                     int tb = LCM/abs(a[k][col]);
40                     if(a[i][col]*a[k][col] < 0)tb = -tb;
41                     for(int j = col;j < var+1;j++){
42                         a[i][j] = ((a[i][j]*ta - a[k][j]*tb)%MOD + MOD)
43                             %MOD;
44                     }
45                 }
46             }
47             for(int i = k;i < equ;i++){
48                 if(a[i][col] != 0)
49                     return -1;//无解
50             }
51             if(k < var) return var-k;//多解
52             for(int i = var-1;i >= 0;i--){
53                 int temp = a[i][var];
54                 for(int j = i+1; j < var;j++){
55                     temp -= a[i][j]*x[j];
56                 }
57             }
58         }
59     }
60     return 0;
61 }

```

```

55         temp = (temp%MOD + MOD)%MOD;
56     }
57 }
58     x[i] = (temp*inv(a[i][i],MOD))%MOD;
59 }
60     return 0;
61 }
62 int change(char s[]){
63     if(strcmp(s,"MON") == 0) return 1;
64     else if(strcmp(s,"TUE")==0) return 2;
65     else if(strcmp(s,"WED")==0) return 3;
66     else if(strcmp(s,"THU")==0) return 4;
67     else if(strcmp(s,"FRI")==0) return 5;
68     else if(strcmp(s,"SAT")==0) return 6;
69     else return 7;
70 }
71 int main(){
72     int n,m;
73     while(scanf("%d%d",&n,&m) == 2){
74         if(n == 0 && m == 0)break;
75         memset(a,0,sizeof(a));
76         char str1[10],str2[10];
77         int k;
78         for(int i = 0;i < m;i++){
79             scanf("%d%s",&k,str1,str2);
80             a[i][n] = ((change(str2) - change(str1) + 1)%MOD + MOD)
                        %MOD;
81             int t;
82             while(k--){
83                 scanf("%d",&t);
84                 t--;
85                 a[i][t] ++;
86                 a[i][t]%MOD;
87             }
88         }
89         int ans = Gauss(m,n);
90         if(ans == 0){
91             for(int i = 0;i < n;i++)
92                 if(x[i] <= 2)
93                     x[i] += 7;
94             for(int i = 0;i < n-1;i++)printf("%d_",x[i]);
95             printf("%d\n",x[n-1]);
96         }
97         else if(ans == -1)printf("Inconsistent_data.\n");
98         else printf("Multiple_solutions.\n");
99     }
100     return 0;
101 }

```

2.11 整数拆分

```

1 //HDU 4651
2 //把数 n 拆成几个数（小于等于 n）相加的形式，问有多少种拆法。
3 const int MOD = 1e9+7;
4 int dp[100010];
5 void init(){
6     memset(dp,0,sizeof(dp));
7     dp[0] = 1;
8     for(int i = 1;i <= 100000;i++){
9         for(int j = 1, r = 1; i - (3 * j * j - j) / 2 >= 0; j++, r
            *= -1){
10             dp[i] += dp[i - (3 * j * j - j) / 2] * r;
11             dp[i] %= MOD;
12             dp[i] = (dp[i]+MOD)%MOD;
13             if( i - (3 * j * j + j) / 2 >= 0 ){
14                 dp[i] += dp[i - (3 * j * j + j) / 2] * r;
15                 dp[i] %= MOD;
16                 dp[i] = (dp[i]+MOD)%MOD;
17             }
18         }
19     }
20 }
21 int main(){
22     int T;
23     int n;
24     init();
25     scanf("%d",&T);
26     while(T--){
27         scanf("%d",&n);
28         printf("%d\n",dp[n]);
29     }
30     return 0;
31 }
32
33 //HDU 4658
34 //数 n(<=10^5) 的划分,相同的数重复不能超过 k 个。
35 const int MOD = 1e9+7;
36 int dp[100010];
37 void init(){
38     memset(dp,0,sizeof(dp));
39     dp[0] = 1;
40     for(int i = 1;i <= 100000;i++){
41         for(int j = 1, r = 1; i - (3 * j * j - j) / 2 >= 0; j++, r
            *= -1){
42             dp[i] += dp[i - (3 * j * j - j) / 2] * r;
43             dp[i] %= MOD;
44             dp[i] = (dp[i]+MOD)%MOD;
45             if( i - (3 * j * j + j) / 2 >= 0 ){
46                 dp[i] += dp[i - (3 * j * j + j) / 2] * r;
47                 dp[i] %= MOD;
48                 dp[i] = (dp[i]+MOD)%MOD;
49             }

```

```

50     }
51 }
52 }
53 int solve(int n,int k){
54     int ans = dp[n];
55     for(int j = 1, r = -1; n - k*(3 * j * j - j) / 2 >= 0; j++, r
        *= -1){
56         ans += dp[n - k*(3 * j * j - j) / 2] * r;
57         ans %= MOD;
58         ans = (ans+MOD)%MOD;
59         if( n - k*(3 * j * j + j) / 2 >= 0 ){
60             ans += dp[n - k*(3 * j * j + j) / 2] * r;
61             ans %= MOD;
62             ans = (ans+MOD)%MOD;
63         }
64     }
65     return ans;
66 }
67 int main(){
68     init();
69     int T;
70     int n,k;
71     scanf("%d",&T);
72     while(T--){
73         scanf("%d%d",&n,&k);
74         printf("%d\n",solve(n,k));
75     }
76     return 0;
77 }

```

2.12 求 A^B 的约数之和对 MOD 取模

```

1 //参考 POJ 1845
2 //里面有一种求 $1+p+p^2+p^3+\dots+p^n$ 的方法。
3 //需要素数筛选和合数分解的程序，需要先调用 getPrime();
4 long long pow_m(long long a,long long n){
5     long long ret = 1;
6     long long tmp = a%MOD;
7     while(n){
8         if(n&1)ret = (ret*tmp)%MOD;
9         tmp = tmp*tmp%MOD;
10        n >>= 1;
11    }
12    return ret;
13 }
14 //计算 $1+p+p^2+\dots+p^n$ 
15 long long sum(long long p,long long n){
16     if(p == 0)return 0;
17     if(n == 0)return 1;
18     if(n & 1){
19         return ((1+pow_m(p,n/2+1))%MOD*sum(p,n/2)%MOD)%MOD;
20     }

```

```

21     else return ((1+pow_m(p,n/2+1))%MOD*sum(p,n/2-1)+pow_m(p,n/2)%
22         MOD)%MOD;
23 }
24 //返回A^B的约数之和 % MOD
25 long long solve(long long A,long long B){
26     getFactors(A);
27     long long ans = 1;
28     for(int i = 0;i < fatCnt;i++){
29         ans *= sum(factor[i][0],B*factor[i][1])%MOD;
30         ans %= MOD;
31     }
32     return ans;
33 }

```

2.13 莫比乌斯反演

2.13.1 莫比乌斯函数

```

1  const int MAXN = 1000000;
2  bool check[MAXN+10];
3  int prime[MAXN+10];
4  int mu[MAXN+10];
5  void Moblus(){
6      memset(check,false,sizeof(check));
7      mu[1] = 1;
8      int tot = 0;
9      for(int i = 2; i <= MAXN; i++){
10         if( !check[i] ){
11             prime[tot++] = i;
12             mu[i] = -1;
13         }
14         for(int j = 0; j < tot; j++){
15             if(i * prime[j] > MAXN) break;
16             check[i * prime[j]] = true;
17             if( i % prime[j] == 0){
18                 mu[i * prime[j]] = 0;
19                 break;
20             }
21             else{
22                 mu[i * prime[j]] = -mu[i];
23             }
24         }
25     }
26 }

```

2.13.2 例题：BZOJ2301

对于给出的 n 个询问，每次求有多少个数对 (x,y) ，满足 $a \leq x \leq b, c \leq y \leq d$ ，且 $\gcd(x,y) = k$ ， $\gcd(x,y)$ 函数为 x 和 y 的最大公约数。 $1 \leq n \leq 50000, 1 \leq a \leq b \leq 50000, 1 \leq c \leq d \leq 50000, 1 \leq k \leq 50000$

```

1  const int MAXN = 1000000;
2  bool check[MAXN+10];
3  int prime[MAXN+10];

```

```

4  int mu[MAXN+10];
5  void Moblus(){
6      memset(check,false,sizeof(check));
7      mu[1] = 1;
8      int tot = 0;
9      for(int i = 2; i <= MAXN; i++){
10         if( !check[i] ){
11             prime[tot++] = i;
12             mu[i] = -1;
13         }
14         for(int j = 0; j < tot; j ++){
15             if( i * prime[j] > MAXN) break;
16             check[i * prime[j]] = true;
17             if( i % prime[j] == 0){
18                 mu[i * prime[j]] = 0;
19                 break;
20             }
21             else{
22                 mu[i * prime[j]] = -mu[i];
23             }
24         }
25     }
26 }
27 int sum[MAXN+10];
28 //找 [1,n],[1,m] 内互质的数的对数
29 long long solve(int n,int m){
30     long long ans = 0;
31     if(n > m)swap(n,m);
32     for(int i = 1, la = 0; i <= n; i = la+1){
33         la = min(n/(n/i),m/(m/i));
34         ans += (long long)(sum[la] - sum[i-1])*(n/i)*(m/i);
35     }
36     return ans;
37 }
38 int main(){
39     Moblus();
40     sum[0] = 0;
41     for(int i = 1;i <= MAXN;i++)
42         sum[i] = sum[i-1] + mu[i];
43     int a,b,c,d,k;
44     int T;
45     scanf("%d",&T);
46     while(T--){
47         scanf("%d%d%d%d",&a,&b,&c,&d,&k);
48         long long ans = solve(b/k,d/k) - solve((a-1)/k,d/k) - solve
            (b/k,(c-1)/k) + solve((a-1)/k,(c-1)/k);
49         printf("%lld\n",ans);
50     }
51     return 0;
52 }

```

2.14 Baby-Step Giant-Step

```

1 // (POJ 2417, 3243)
2 // baby_step giant_step
3 //  $a^x = b \pmod n$   $n$  是素数和不是素数都可以
4 // 求解上式  $0 \leq x < n$  的解
5 #define MOD 76543
6 int hs[MOD], head[MOD], next[MOD], id[MOD], top;
7 void insert(int x, int y){
8     int k = x%MOD;
9     hs[top] = x, id[top] = y, next[top] = head[k], head[k] = top++;
10 }
11 int find(int x){
12     int k = x%MOD;
13     for(int i = head[k]; i != -1; i = next[i])
14         if(hs[i] == x)
15             return id[i];
16     return -1;
17 }
18 int BSGS(int a, int b, int n){
19     memset(head, -1, sizeof(head));
20     top = 1;
21     if(b == 1) return 0;
22     int m = sqrt(n*1.0), j;
23     long long x = 1, p = 1;
24     for(int i = 0; i < m; ++i, p = p*a%n) insert(p*b%n, i);
25     for(long long i = m; ; i += m){
26         if((j = find(x = x*p%n)) != -1) return i-j;
27         if(i > n) break;
28     }
29     return -1;
30 }

```

2.15 自适应 simpson 积分

```

1 double simpson(double a, double b){
2     double c = a + (b-a)/2;
3     return (F(a) + 4*F(c) + F(b))*(b-a)/6;
4 }
5 double asr(double a, double b, double eps, double A){
6     double c = a + (b-a)/2;
7     double L = simpson(a, c), R = simpson(c, b);
8     if(fabs(L + R - A) <= 15*eps) return L + R + (L + R - A)/15.0;
9     return asr(a, c, eps/2, L) + asr(c, b, eps/2, R);
10 }
11 double asr(double a, double b, double eps){
12     return asr(a, b, eps, simpson(a, b));
13 }

```

2.16 斐波那契数列取模循环节

必要时要上 unsigned long long
 HDU3977


```

1 long long gcd(long long a,long long b){
2     if(b == 0)return a;
3     return gcd(b,a%b);
4 }
5 long long lcm(long long a,long long b){
6     return a/gcd(a,b)*b;
7 }
8 struct Matrix{
9     long long mat[2][2];
10 };
11 Matrix mul_M(Matrix a,Matrix b,long long mod){
12     Matrix ret;
13     for(int i = 0;i < 2;i++){
14         for(int j = 0;j < 2;j++){
15             ret.mat[i][j] = 0;
16             for(int k = 0;k < 2;k++){
17                 ret.mat[i][j] += a.mat[i][k]*b.mat[k][j]%mod;
18                 if(ret.mat[i][j] >= mod)ret.mat[i][j] -= mod;
19             }
20         }
21     }
22     return ret;
23 }
24 Matrix pow_M(Matrix a,long long n,long long mod){
25     Matrix ret;
26     memset(ret.mat,0,sizeof(ret.mat));
27     for(int i = 0;i < 2;i++)ret.mat[i][i] = 1;
28     Matrix tmp = a;
29     while(n){
30         if(n&1)ret = mul_M(ret,tmp,mod);
31         tmp = mul_M(tmp,tmp,mod);
32         n >>= 1;
33     }
34     return ret;
35 }
36 long long pow_m(long long a,long long n,long long mod)//a^b % mod{
37     long long ret = 1;
38     long long tmp = a%mod;
39     while(n){
40         if(n&1)ret = ret*tmp%mod;
41         tmp = tmp*tmp%mod;
42         n >>= 1;
43     }
44     return ret;
45 }
46 //素数筛选和合数分解
47 const int MAXN = 1000000;
48 int prime[MAXN+1];
49 void getPrime(){
50     memset(prime,0,sizeof(prime));
51     for(int i = 2;i <= MAXN;i++){
52         if(!prime[i])prime[++prime[0]] = i;

```

```

52     for(int j = 1;j <= prime[0] && prime[j] <= MAXN/i;j++){
53         prime[prime[j]*i] = 1;
54         if(i%prime[j] == 0)break;
55     }
56 }
57 }
58 long long factor[100][2];
59 int fatCnt;
60 int getFactors(long long x){
61     fatCnt = 0;
62     long long tmp = x;
63     for(int i = 1;prime[i] <= tmp/prime[i];i++){
64         factor[fatCnt][1] = 0;
65         if(tmp%prime[i] == 0){
66             factor[fatCnt][0] = prime[i];
67             while(tmp%prime[i] == 0){
68                 factor[fatCnt][1]++;
69                 tmp /= prime[i];
70             }
71             fatCnt++;
72         }
73     }
74     if(tmp != 1){
75         factor[fatCnt][0] = tmp;
76         factor[fatCnt++][1] = 1;
77     }
78     return fatCnt;
79 }
80 //勒让德符号
81 int legendre(long long a,long long p){
82     if(pow_m(a,(p-1)>>1,p) == 1)return 1;
83     else return -1;
84 }
85 int f0 = 1;
86 int f1 = 1;
87 long long getFib(long long n,long long mod){
88     if(mod == 1)return 0;
89     Matrix A;
90     A.mat[0][0] = 0;
91     A.mat[1][0] = 1;
92     A.mat[0][1] = 1;
93     A.mat[1][1] = 1;
94     Matrix B = pow_M(A,n,mod);
95     long long ret = f0*B.mat[0][0] + f1*B.mat[1][0];
96     return ret%mod;
97 }
98 long long fac[1000000];
99 long long G(long long p){
100     long long num;
101     if(legendre(5,p) == 1)num = p-1;
102     else num = 2*(p+1);

```

```

103 //找出 num 的所有约数
104 int cnt = 0;
105 for(long long i = 1;i*i <= num;i++)
106     if(num%i == 0){
107         fac[cnt++] = i;
108         if(i*i != num)
109             fac[cnt++] = num/i;
110     }
111 sort(fac,fac+cnt);
112 long long ans;
113 for(int i = 0;i < cnt;i++){
114     if(getFib(fac[i],p) == f0 && getFib(fac[i]+1,p) == f1){
115         ans = fac[i];
116         break;
117     }
118 }
119 return ans;
120 }
121 long long find_loop(long long n){
122     getFactors(n);
123     long long ans = 1;
124     for(int i = 0;i < fatCnt;i++){
125         long long record = 1;
126         if(factor[i][0] == 2)record = 3;
127         else if(factor[i][0] == 3)record = 8;
128         else if(factor[i][0] == 5)record = 20;
129         else record = G(factor[i][0]);
130         for(int j = 1;j < factor[i][1];j++)
131             record *= factor[i][0];
132         ans = lcm(ans,record);
133     }
134     return ans;
135 }
136 void init(){
137     getPrime();
138 }
139 int main(){
140     init();
141     int T;
142     int iCase = 0;
143     int n;
144     scanf("%d",&T);
145     while(T--){
146         iCase++;
147         scanf("%d",&n);
148         printf("Case_#%d:_%I64d\n",iCase,find_loop(n));
149     }
150     return 0;
151 }

```

2.17 原根

定义：设 $m > 1, \gcd(a, m) = 1$, 使得 $a^d \equiv 1 \pmod{m}$ 成立的最小的正整数 d 为 a 对模 m 的阶，记为 $\delta_m(a)$.

如果 $\delta_m(a) = \varphi(m)$, 则称 a 是模 m 的原根.

定理：若 $m > 1, \gcd(a, m) = 1$, 正整数 d 满足 $a^d \equiv 1 \pmod{m}$, 则 $\delta_m(a)$ 整除 d .

定理：模 m 有原根的充要条件是 $m = 2, 4, p^n, 2p^n$, 其中 p 是奇质数, n 是任意正整数.

定理：如果模 m 有原根，那么它一定有 $\varphi(\varphi(m))$ 个原根.

定理：如果 p 是素数，那么素数 p 一定有原根，并且模 p 的原根的个数为 $\varphi(p-1)$.

求模素数 p 原根的方法：对 $p-1$ 素因子分解，即 $p-1 = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k}$ 的标准分解式，若恒有

$$g^{\frac{p-1}{p_i}} \not\equiv 1 \pmod{p}$$

成立，则 g 就是 p 的原根。（对于合数求原根，只需要把 $p-1$ 换成 $\varphi(p)$ ）即可。

求素数的最小原根程序

```

1 //*****
2 //素数筛选和合数分解
3 const int MAXN=100000;
4 int prime[MAXN+1];
5 void getPrime(){
6     memset(prime,0,sizeof(prime));
7     for(int i=2;i<=MAXN;i++){
8         if(!prime[i])prime[++prime[0]]=i;
9         for(int j=1;j<=prime[0]&&prime[j]<=MAXN/i;j++){
10             prime[prime[j]*i]=1;
11             if(i%prime[j]==0) break;
12         }
13     }
14 }
15 long long factor[100][2];
16 int fatCnt;
17 int getFactors(long long x){
18     fatCnt=0;
19     long long tmp=x;
20     for(int i=1;prime[i]<=tmp/prime[i];i++){
21         factor[fatCnt][1]=0;
22         if(tmp%prime[i]==0){
23             factor[fatCnt][0]=prime[i];
24             while(tmp%prime[i]==0){
25                 factor[fatCnt][1]++;
26                 tmp/=prime[i];
27             }
28             fatCnt++;
29         }
30     }
31     if(tmp!=1){
32         factor[fatCnt][0]=tmp;
33         factor[fatCnt++][1]=1;
34     }
35     return fatCnt;
36 }
```

```

37 //*****
38 long long pow_m(long long a,long long n,long long mod){
39     long long ret = 1;
40     long long tmp = a%mod;
41     while(n){
42         if(n&1)ret = ret*tmp%mod;
43         tmp = tmp*tmp%mod;
44         n >>= 1;
45     }
46     return ret;
47 }
48 //求素数 P 的最小的原根
49 void solve(int P){
50     if(P == 2){
51         printf("1\n");
52         return;
53     }
54     getFactors(P-1);
55     for(int g = 2; g < P;g++){
56         bool flag = true;
57         for(int i = 0;i < fatCnt;i++){
58             int t = (P-1)/factor[i][0];
59             if(pow_m(g,t,P) == 1){
60                 flag = false;
61                 break;
62             }
63         }
64         if(flag){
65             printf("%d\n",g);
66             return;
67         }
68     }
69 }
70 int main(){
71     getPrime();
72     int T;
73     int P;
74     scanf("%d",&T);
75     while(T--){
76         scanf("%d",&P);
77         solve(P);
78     }
79     return 0;
80 }

```

2.18 快速数论变换

2.18.1 HDU4656 卷积取模

HDU4656

$x_k = b * c^{(2^k)} + d$, $F(x) = a_0x_0 + a_1x_1 + a_2x_2 + \dots + a_{n-1}x_{n-1}$ Given $n, b, c, d, a_0, \dots, a_{n-1}$,

calculate $F(x_0), \dots, F(x_{n-1})$.

$$\begin{aligned}
 F_{x_k} &= \sum_{i=0}^{n-1} a_i (bc^{2k} + d)^i \\
 &= \sum_{i=0}^{n-1} a_i \sum_{j=0}^i C_i^j (bc^{2k})^j d^{i-j} \\
 &= \sum_{j=0}^{n-1} (bc^{2k})^j j!^{-1} \sum_{i=j}^{n-1} a_i d^{i-j} i! (i-j)!^{-1} \\
 &= \sum_{j=0}^{n-1} (bc^{2k})^j j!^{-1} \sum_{i=0}^{n-1-j} a_{n-1-i} (n-1-i)! d^{n-1-i-j} (n-1-i-j)!^{-1} \\
 &= \sum_{j=0}^{n-1} (bc^{2k})^j j!^{-1} p_j \\
 &= \sum_{j=0}^{n-1} b^j j!^{-1} p_j c^{2jk} \\
 &= c^{k^2} \sum_{j=0}^{n-1} b^j j!^{-1} p_j c^{j^2} c^{-(k-j)^2} \\
 &= c^{k^2} q_k
 \end{aligned}$$

其中 p_j 和 q_k 都是卷积，可以使用 NTT 进行快速计算。

```

1 //*****
2 //快速数论变换 (NTT)
3 //求 A 和 B 的卷积，结果对 P 取模
4 //做长度为 N1 的变换，选取两个质数 P1 和 P2
5 //P1-1 和 P2-1 必须是 N1 的倍数
6 //E1 和 E2 分别是 P1,P2 的原根
7 //F1 是 E1 模 P1 的逆元,F2 是 E2 模 P2 的逆元
8 //I1 是 N1 对模 P1 的逆元,I2 是 N1 对模 P2 的逆元
9 //
10 //然后使用中国剩余定理，保证了结果是小于 MM=P1*P2 的
11 //M1 = (P2 对 P1 的逆元)*P2
12 //M2 = (P1 对 P2 的逆元)*P1
13
14 const int P = 1000003;//结果对 P 取模
15 const int N1 = 262144;// 2^18
16 const int N2 = N1+1;//数组大小
17 const int P1 = 998244353;//P1 = 2^23 * 7 * 17 + 1
18 const int P2 = 995622913;//P2 = 2^19 * 3 * 3 * 211 + 1
19 const int E1 = 996173970;
20 const int E2 = 88560779;
21 const int F1 = 121392023;//E1*F1 = 1(mod P1)
22 const int F2 = 840835547;//E2*F2 = 1(mod P2)
23 const int I1 = 998240545;//I1*N1 = 1(mod P1)
24 const int I2 = 995619115;//I2*N1 = 1(mod P2)
25 const long long M1 = 397550359381069386LL;

```

```

26 const long long M2 = 596324591238590904LL;
27 const long long MM = 993874950619660289LL;//MM = P1*P2
28 //计算 x*y 对 z 取模
29 long long mul(long long x,long long y,long long z){
30     return (x*y - (long long)(x/(long double)z*y+1e-3)*z+z)%z;
31 }
32 int trf(int x1,int x2){
33     return (mul(M1,x1,MM)+mul(M2,x2,MM))%MM%P;
34 }
35 int A[N2],B[N2],C[N2];
36 int A1[N2],B1[N2],C1[N2];
37 void fft(int *A,int PM,int PW){
38     for(int m = N1,h;h = m/2, m >= 2;PW = (long long)PW*PW%PM,m=h)
39         for(int i = 0,w=1;i < h;i++, w = (long long)w*PW%PM)
40             for(int j = i;j < N1;j += m){
41                 int k = j+h, x = (A[j]-A[k]+PM)%PM;
42                 (A[j]+=A[k])%=PM;
43                 A[k] = (long long)w*x%PM;
44             }
45     for(int i = 0,j = 1;j < N1-1;j++){
46         for(int k = N1/2; k > (i^=k);k /= 2);
47         if(j < i)swap(A[i],A[j]);
48     }
49 }
50 //计算 A 和 B 的卷积, 结果保存在 C 中, 结果对 P 取模
51 void mul(){
52     memset(C,0,sizeof(C));
53     memcpy(A1,A,sizeof(A));
54     memcpy(B1,B,sizeof(B));
55     fft(A1,P1,E1); fft(B1,P1,E1);
56     for(int i = 0;i < N1;i++)C1[i] = (long long)A1[i]*B1[i]%P1;
57     fft(C1,P1,F1);
58     for(int i = 0;i < N1;i++)C1[i] = (long long)C1[i]*I1%P1;
59     fft(A,P2,E2); fft(B,P2,E2);
60     for(int i = 0;i < N1;i++)C[i] = (long long)A[i]*B[i]%P2;
61     fft(C,P2,F2);
62     for(int i = 0;i < N1;i++)C[i] = (long long)C[i]*I2%P2;
63     for(int i = 0;i < N1;i++)C[i] = trf(C1[i],C[i]);
64 }
65 int INV[P];//逆元
66 const int MAXN = 100010;
67 int F[MAXN];//阶乘
68 int a[MAXN];
69 int pd[MAXN];
70 int pb[MAXN];
71 int pc2[MAXN];
72 int p[MAXN];
73 int main()
74 {
75     //预处理逆元
76     INV[1] = 1;

```

```

77     for(int i = 2; i < P; i++)
78         INV[i] = (long long)P/i*(P-INV[P%i])%P;
79     F[0] = 1;
80     for(int i = 1; i < MAXN; i++)
81         F[i] = (long long)F[i-1]*i%P;
82     int n,b,c,d;
83     while(scanf("%d%d%d%d",&n,&b,&c,&d) == 4){
84         for(int i = 0; i < n; i++)scanf("%d",&a[i]);
85         pd[0] = 1;
86         for(int i = 1; i < n; i++)
87             pd[i] = (long long)pd[i-1]*d%P;
88         memset(A,0,sizeof(A));
89         memset(B,0,sizeof(B));
90         for(int i = 0; i < n; i++)
91             A[i] = (long long)a[n-1-i]*F[n-1-i]%P;
92         for(int i = 0; i < n; i++)
93             B[i] = (long long)pd[i]*INV[F[i]]%P;
94         mul();
95         for(int i = 0; i < n; i++)p[i] = C[i];
96         reverse(p,p+n);
97         memset(A,0,sizeof(A));
98         pb[0] = 1;
99         for(int i = 1; i < n; i++)
100             pb[i] = (long long)pb[i-1]*b%P;
101         pc2[0] = 1;
102         int c2 = (long long)c*c%P;
103         for(int i = 1, s = c; i < n; i++){
104             pc2[i] = (long long)pc2[i-1]*s%P;
105             s = (long long)s*c2%P;
106         }
107         for(int i = 0; i < n; i++)
108             A[i] = (long long)pb[i]*INV[F[i]]%P*p[i]%P*pc2[i]%P;
109         memset(B,0,sizeof(B));
110         B[0] = 1;
111         for(int i = 1; i < n; i++)
112             B[i] = B[n-1-i] = INV[pc2[i]];
113         mul();
114         for(int i = 0; i < n; i++)C[i] = (long long)C[i]*pc2[i]%P;
115         for(int i = 0; i < n; i++)
116             printf("%d\n",C[i]);
117     }
118     return 0;
119 }

```

2.19 其它公式

2.19.1 Polya

设 G 是 p 个对象的一个置换群，用 k 种颜色去染这 p 个对象，若一种染色方案在群 G 的作用下变为另一种方案，则这两个方案当作是同一种方案，这样的不同染色方案数为：

$$L = \frac{1}{|G|} \times \sum (k^{C(f)}), f \in G$$

$C(f)$ 为循环节, $|G|$ 表示群的置换方法数

对于有 n 个位置的手镯, 有 n 种旋转置换和 n 种翻转置换

对于旋转置换:

$C(f_i) = \gcd(n, i)$, i 表示一次转过 i 颗宝石, $i = 0$ 时 $c = n$;

对于翻转置换:

如果 n 为偶数: 则有 $\frac{n}{2}$ 个置换 $C(f) = \frac{n}{2}$, 有 $\frac{n}{2}$ 个置换 $C(f) = \frac{n}{2} + 1$

如果 n 为奇数: $C(f) = \frac{n}{2} + 1$

3 数据结构

3.1 划分树

```

1  /*
2   * 划分树（查询区间第 k 大）
3   */
4  const int MAXN = 100010;
5  int tree[20][MAXN]; //表示每层每个位置的值
6  int sorted[MAXN]; //已经排序好的数
7  int toleft[20][MAXN]; //toleft[p][i] 表示第 i 层从 1 到 i 有数分入左边
8
9  void build(int l, int r, int dep){
10     if(l == r) return;
11     int mid = (l+r)>>1;
12     int same = mid - l + 1; //表示等于中间值而且被分入左边的个数
13     for(int i = l; i <= r; i++) //注意是 l, 不是 one
14         if(tree[dep][i] < sorted[mid])
15             same--;
16     int lpos = l;
17     int rpos = mid+1;
18     for(int i = l; i <= r; i++){
19         if(tree[dep][i] < sorted[mid])
20             tree[dep+1][lpos++] = tree[dep][i];
21         else if(tree[dep][i] == sorted[mid] && same > 0){
22             tree[dep+1][lpos++] = tree[dep][i];
23             same--;
24         }
25         else
26             tree[dep+1][rpos++] = tree[dep][i];
27         toleft[dep][i] = toleft[dep][l-1] + lpos - l;
28     }
29     build(l, mid, dep+1);
30     build(mid+1, r, dep+1);
31 }
32
33 //查询区间第 k 大的数, [L,R] 是大区间, [l,r] 是要查询的小区间
34 int query(int L, int R, int l, int r, int dep, int k){
35     if(l == r) return tree[dep][l];
36     int mid = (L+R)>>1;
37     int cnt = toleft[dep][r] - toleft[dep][l-1];
38     if(cnt >= k){
39         int newl = L + toleft[dep][l-1] - toleft[dep][L-1];
40         int newr = newl + cnt - 1;
41         return query(L, mid, newl, newr, dep+1, k);
42     }
43     else{
44         int newr = r + toleft[dep][R] - toleft[dep][r];
45         int newl = newr - (r-l-cnt);
46         return query(mid+1, R, newl, newr, dep+1, k-cnt);
47     }
}

```

```

48 }
49 int main(){
50     int n,m;
51     while (scanf("%d%d",&n,&m)==2){
52         memset(tree,0,sizeof(tree));
53         for(int i = 1;i <= n;i++){
54             scanf("%d",&tree[0][i]);
55             sorted[i] = tree[0][i];
56         }
57         sort(sorted+1,sorted+n+1);
58         build(1,n,0);
59         int s,t,k;
60         while(m--){
61             scanf("%d%d%d",&s,&t,&k);
62             printf("%d\n",query(1,n,s,t,0,k));
63         }
64     }
65     return 0;
66 }

```

3.2 RMQ

3.2.1 一维

求最大值，数组下标从 1 开始。

求最小值，或者最大最小值下标，或者数组从 0 开始对应修改即可。

```

1  const int MAXN = 50010;
2  int dp[MAXN][20];
3  int mm[MAXN];
4  //初始化 RMQ, b 数组下标从 1 开始, 从 0 开始简单修改
5  void initRMQ(int n,int b[]){
6      mm[0] = -1;
7      for(int i = 1; i <= n;i++){
8          mm[i] = ((i&(i-1)) == 0)?mm[i-1]+1:mm[i-1];
9          dp[i][0] = b[i];
10     }
11     for(int j = 1; j <= mm[n];j++){
12         for(int i = 1;i + (1<<j) -1 <= n;i++){
13             dp[i][j] = max(dp[i][j-1],dp[i+(1<<(j-1))][j-1]);
14         }
15     }
16     //查询最大值
17     int rmq(int x,int y){
18         int k = mm[y-x+1];
19         return max(dp[x][k],dp[y-(1<<k)+1][k]);
20     }

```

3.2.2 二维

```

1  /*
2  * 二维 RMQ, 预处理复杂度 n*m*log*(n)*log(m)
3  * 数组下标从 1 开始

```

```

4  */
5  int val[310][310];
6  int dp[310][310][9][9]; //最大值
7  int mm[310]; //二进制位数减一，使用前初始化
8  void initRMQ(int n, int m){
9      for(int i = 1; i <= n; i++)
10         for(int j = 1; j <= m; j++)
11             dp[i][j][0][0] = val[i][j];
12     for(int ii = 0; ii <= mm[n]; ii++)
13         for(int jj = 0; jj <= mm[m]; jj++)
14             if(ii+jj)
15                 for(int i = 1; i + (1<<ii) - 1 <= n; i++)
16                     for(int j = 1; j + (1<<jj) - 1 <= m; j++){
17                         if(ii) dp[i][j][ii][jj] = max(dp[i][j][ii-1][jj], dp[i+(1<<(ii-1))][j][ii-1][jj]);
18                         else dp[i][j][ii][jj] = max(dp[i][j][ii][jj-1], dp[i][j+(1<<(jj-1))][ii][jj-1]);
19                     }
20 }
21 //查询矩形内的最大值 (x1<=x2,y1<=y2)
22 int rmq(int x1, int y1, int x2, int y2){
23     int k1 = mm[x2-x1+1];
24     int k2 = mm[y2-y1+1];
25     x2 = x2 - (1<<k1) + 1;
26     y2 = y2 - (1<<k2) + 1;
27     return max(max(dp[x1][y1][k1][k2], dp[x1][y2][k1][k2]), max(dp[x2][y1][k1][k2], dp[x2][y2][k1][k2]));
28 }
29 int main(){
30     //在外面对 mm 数组进行初始化
31     mm[0] = -1;
32     for(int i = 1; i <= 305; i++)
33         mm[i] = ((i&(i-1))==0)?mm[i-1]+1:mm[i-1];
34     int n, m;
35     int Q;
36     int r1, c1, r2, c2;
37     while(scanf("%d%d", &n, &m) == 2){
38         for(int i = 1; i <= n; i++)
39             for(int j = 1; j <= m; j++)
40                 scanf("%d", &val[i][j]);
41         initRMQ(n, m);
42         scanf("%d", &Q);
43         while(Q--){
44             scanf("%d%d%d%d", &r1, &c1, &r2, &c2);
45             if(r1 > r2) swap(r1, r2);
46             if(c1 > c2) swap(c1, c2);
47             int tmp = rmq(r1, c1, r2, c2);
48             printf("%d_", tmp);
49             if(tmp == val[r1][c1] || tmp == val[r1][c2] || tmp == val[r2][c1] || tmp == val[r2][c2])
50                 printf("yes\n");

```

```

51         else printf("no\n");
52     }
53 }
54 return 0;
55 }

```

3.3 树链剖分

3.3.1 点权

基于点权，查询单点值，修改路径的上的点权（HDU 3966 树链剖分 + 树状数组）

```

1  const int MAXN = 50010;
2  struct Edge{
3      int to,next;
4  }edge[MAXN*2];
5  int head[MAXN],tot;
6  int top[MAXN]; //top[v] 表示 v 所在的重链的顶端节点
7  int fa[MAXN]; //父亲节点
8  int deep[MAXN]; //深度
9  int num[MAXN]; //num[v] 表示以 v 为根的子树的节点数
10 int p[MAXN]; //p[v] 表示 v 对应的位置
11 int fp[MAXN]; //fp 和 p 数组相反
12 int son[MAXN]; //重儿子
13 int pos;
14 void init(){
15     tot = 0;
16     memset(head,-1,sizeof(head));
17     pos = 1; //使用树状数组，编号从头 1 开始
18     memset(son,-1,sizeof(son));
19 }
20 void addedge(int u,int v){
21     edge[tot].to = v; edge[tot].next = head[u]; head[u] = tot++;
22 }
23 void dfs1(int u,int pre,int d){
24     deep[u] = d;
25     fa[u] = pre;
26     num[u] = 1;
27     for(int i = head[u]; i != -1; i = edge[i].next){
28         int v = edge[i].to;
29         if(v != pre){
30             dfs1(v,u,d+1);
31             num[u] += num[v];
32             if(son[u] == -1 || num[v] > num[son[u]])
33                 son[u] = v;
34         }
35     }
36 }
37 void getpos(int u,int sp){
38     top[u] = sp;
39     p[u] = pos++;
40     fp[p[u]] = u;

```

```

41     if(son[u] == -1) return;
42     getpos(son[u],sp);
43     for(int i = head[u];i != -1;i = edge[i].next){
44         int v = edge[i].to;
45         if( v != son[u] && v != fa[u])
46             getpos(v,v);
47     }
48 }
49
50 //树状数组
51 int lowbit(int x){
52     return x&(-x);
53 }
54 int c[MAXN];
55 int n;
56 int sum(int i){
57     int s = 0;
58     while(i > 0)
59     {
60         s += c[i];
61         i -= lowbit(i);
62     }
63     return s;
64 }
65 void add(int i,int val){
66     while(i <= n){
67         c[i] += val;
68         i += lowbit(i);
69     }
70 }
71 //u->v 的路径上点的值改变 val
72 void Change(int u,int v,int val){
73     int f1 = top[u], f2 = top[v];
74     int tmp = 0;
75     while(f1 != f2){
76         if(deep[f1] < deep[f2]){
77             swap(f1,f2);
78             swap(u,v);
79         }
80         add(p[f1],val);
81         add(p[u]+1,-val);
82         u = fa[f1];
83         f1 = top[u];
84     }
85     if(deep[u] > deep[v]) swap(u,v);
86     add(p[u],val);
87     add(p[v]+1,-val);
88 }
89 int a[MAXN];
90 int main(){
91     int M,P;

```

```

92     while (scanf("%d%d%d",&n,&M,&P) == 3){
93         int u,v;
94         int C1,C2,K;
95         char op[10];
96         init();
97         for(int i = 1;i <= n;i++){
98             scanf("%d",&a[i]);
99         }
100        while(M--){
101            scanf("%d%d",&u,&v);
102            addedge(u,v);
103            addedge(v,u);
104        }
105        dfs1(1,0,0);
106        getpos(1,1);
107        memset(c,0,sizeof(c));
108        for(int i = 1;i <= n;i++){
109            add(p[i],a[i]);
110            add(p[i]+1,-a[i]);
111        }
112        while(P--){
113            scanf("%s",op);
114            if(op[0] == 'Q'){
115                scanf("%d",&u);
116                printf("%d\n",sum(p[u]));
117            }
118            else{
119                scanf("%d%d%d",&C1,&C2,&K);
120                if(op[0] == 'D')
121                    K = -K;
122                Change(C1,C2,K);
123            }
124        }
125    }
126    return 0;
127 }

```

3.3.2 边权

基于边权，修改单条边权，查询路径边权最大值（SPOJ QTREE 树链剖分 + 线段树）

```

1  const int MAXN = 10010;
2  struct Edge{
3      int to,next;
4  }edge[MAXN*2];
5  int head[MAXN],tot;
6  int top[MAXN]; //top[v] 表示 v 所在的重链的顶端节点
7  int fa[MAXN]; //父亲节点
8  int deep[MAXN]; //深度
9  int num[MAXN]; //num[v] 表示以 v 为根的子树的节点数
10 int p[MAXN]; //p[v] 表示 v 与其父亲节点的连边在线段树中的位置

```

```

11 int fp[MAXN]; //和 p 数组相反
12 int son[MAXN]; //重儿子
13 int pos;
14 void init(){
15     tot = 0;
16     memset(head,-1,sizeof(head));
17     pos = 0;
18     memset(son,-1,sizeof(son));
19 }
20 void addedge(int u,int v){
21     edge[tot].to = v; edge[tot].next = head[u]; head[u] = tot++;
22 }
23 //第一遍 dfs 求出 fa,deep,num,son
24 void dfs1(int u,int pre,int d){
25     deep[u] = d;
26     fa[u] = pre;
27     num[u] = 1;
28     for(int i = head[u]; i != -1; i = edge[i].next){
29         int v = edge[i].to;
30         if(v != pre){
31             dfs1(v,u,d+1);
32             num[u] += num[v];
33             if(son[u] == -1 || num[v] > num[son[u]])
34                 son[u] = v;
35         }
36     }
37 }
38 //第二遍 dfs 求出 top 和 p
39 void getpos(int u,int sp){
40     top[u] = sp;
41     p[u] = pos++;
42     fp[p[u]] = u;
43     if(son[u] == -1) return;
44     getpos(son[u],sp);
45     for(int i = head[u] ; i != -1; i = edge[i].next){
46         int v = edge[i].to;
47         if(v != son[u] && v != fa[u])
48             getpos(v,v);
49     }
50 }
51
52 //线段树
53 struct Node{
54     int l,r;
55     int Max;
56 }segTree[MAXN*3];
57 void build(int i,int l,int r){
58     segTree[i].l = l;
59     segTree[i].r = r;
60     segTree[i].Max = 0;
61     if(l == r)return;

```



```

62     int mid = (l+r)/2;
63     build(i<<1,l,mid);
64     build((i<<1)|1,mid+1,r);
65 }
66 void push_up(int i){
67     segTree[i].Max = max(segTree[i<<1].Max,segTree[(i<<1)|1].Max);
68 }
69 // 更新线段树的第 k 个值为 val
70 void update(int i,int k,int val){
71     if(segTree[i].l == k && segTree[i].r == k){
72         segTree[i].Max = val;
73         return;
74     }
75     int mid = (segTree[i].l + segTree[i].r)/2;
76     if(k <= mid)update(i<<1,k,val);
77     else update((i<<1)|1,k,val);
78     push_up(i);
79 }
80 //查询线段树中 [l,r] 的最大值
81 int query(int i,int l,int r){
82     if(segTree[i].l == l && segTree[i].r == r)
83         return segTree[i].Max;
84     int mid = (segTree[i].l + segTree[i].r)/2;
85     if(r <= mid)return query(i<<1,l,r);
86     else if(l > mid)return query((i<<1)|1,l,r);
87     else return max(query(i<<1,l,mid),query((i<<1)|1,mid+1,r));
88 }
89 //查询 u->v 边的最大值
90 int find(int u,int v){
91     int f1 = top[u], f2 = top[v];
92     int tmp = 0;
93     while(f1 != f2){
94         if(deep[f1] < deep[f2]){
95             swap(f1,f2);
96             swap(u,v);
97         }
98         tmp = max(tmp,query(1,p[f1],p[u]));
99         u = fa[f1]; f1 = top[u];
100     }
101     if(u == v)return tmp;
102     if(deep[u] > deep[v]) swap(u,v);
103     return max(tmp,query(1,p[son[u]],p[v]));
104 }
105 int e[MAXN][3];
106 int main(){
107     int T;
108     int n;
109     scanf("%d",&T);
110     while(T--){
111         init();
112         scanf("%d",&n);

```

```

113     for(int i = 0;i < n-1;i++){
114         scanf("%d%d%d",&e[i][0],&e[i][1],&e[i][2]);
115         addedge(e[i][0],e[i][1]);
116         addedge(e[i][1],e[i][0]);
117     }
118     dfs1(1,0,0);
119     getpos(1,1);
120     build(1,0,pos-1);
121     for(int i = 0;i < n-1; i++){
122         if(deep[e[i][0]] > deep[e[i][1]])
123             swap(e[i][0],e[i][1]);
124         update(1,p[e[i][1]],e[i][2]);
125     }
126     char op[10];
127     int u,v;
128     while(scanf("%s",op) == 1){
129         if(op[0] == 'D')break;
130         scanf("%d%d",&u,&v);
131         if(op[0] == 'Q')
132             printf("%d\n",find(u,v)); //查询 u->v 路径上边权的最大值
133         else update(1,p[e[u-1][1]],v); //修改第 u 条边的长度为 v
134     }
135 }
136 return 0;
137 }

```

3.4 伸展树 (splay tree)

3.4.1 例题: HDU1890

```

1  const int MAXN = 100010;
2  struct Node;
3  Node* null;
4  struct Node{
5      Node *ch[2],*fa;
6      int size;
7      int rev;
8      Node(){
9          ch[0] = ch[1] = fa = null; rev = 0;
10     }
11     inline void push_up(){
12         if(this == null)return;
13         size = ch[0]->size + ch[1]->size + 1;
14     }
15     inline void setc(Node* p,int d){
16         ch[d] = p;
17         p->fa = this;
18     }
19     inline bool d(){
20         return fa->ch[1] == this;
21     }

```

```

22     void clear(){
23         size = 1;
24         ch[0] = ch[1] = fa = null;
25         rev = 0;
26     }
27     void Update_Rev(){
28         if(this == null) return;
29         swap(ch[0], ch[1]);
30         rev ^= 1;
31     }
32     inline void push_down(){
33         if(this == null) return;
34         if(rev){
35             ch[0] -> Update_Rev();
36             ch[1] -> Update_Rev();
37             rev = 0;
38         }
39     }
40     inline bool isroot(){
41         return fa == null || this != fa -> ch[0] && this != fa -> ch
           [1];
42     }
43 };
44 inline void rotate(Node* x)
45 {
46     Node *f = x -> fa, *ff = x -> fa -> fa;
47     f -> push_down();
48     x -> push_down();
49     int c = x -> d(), cc = f -> d();
50     f -> setc(x -> ch[!c], c);
51     x -> setc(f, !c);
52     if(ff -> ch[cc] == f) ff -> setc(x, cc);
53     else x -> fa = ff;
54     f -> push_up();
55 }
56 inline void splay(Node* &root, Node* x, Node* goal)
57 {
58     while(x -> fa != goal){
59         if(x -> fa -> fa == goal) rotate(x);
60         else {
61             x -> fa -> fa -> push_down();
62             x -> fa -> push_down();
63             x -> push_down();
64             bool f = x -> fa -> d();
65             x -> d() == f ? rotate(x -> fa) : rotate(x);
66             rotate(x);
67         }
68     }
69     x -> push_up();
70     if(goal == null) root = x;
71 }

```

```

72 Node* get_kth(Node* r,int k)
73 {
74     Node* x = r;
75     x->push_down();
76     while(x->ch[0]->size+1 != k){
77         if(k < x->ch[0]->size+1)x = x->ch[0];
78         else{
79             k -= x->ch[0]->size+1;
80             x = x->ch[1];
81         }
82         x->push_down();
83     }
84     return x;
85 }
86 Node* get_next(Node* p){
87     p->push_down();
88     p = p->ch[1];
89     p->push_down();
90     while(p->ch[0] != null){
91         p = p->ch[0];
92         p->push_down();
93     }
94     return p;
95 }
96 Node pool[MAXN],*tail;
97 Node *node[MAXN];
98 Node *root;
99 void build(Node* &x,int l,int r,Node* fa)
100 {
101     if(l > r)return;
102     int mid = (l+r)/2;
103     x = tail++;
104     x->clear();
105     x->fa = fa;
106     node[mid] = x;
107     build(x->ch[0],l,mid-1,x);
108     build(x->ch[1],mid+1,r,x);
109     x->push_up();
110 }
111 void init(int n)
112 {
113     tail = pool;
114     null = tail++;
115     null->fa = null->ch[0] = null->ch[1] = null;
116     null->size = 0; null->rev = 0;
117     Node *p = tail++;
118     p->clear();
119     root = p;
120     p = tail++;
121     p->clear();
122     root->setc(p,1);

```

```

123     build(root->ch[1]->ch[0],1,n,root->ch[1]);
124     root->ch[1]->push_up();
125     root->push_up();
126 }
127 int a[MAXN];
128 int b[MAXN];
129 bool cmp(int i,int j)
130 {
131     if(a[i] != a[j])return a[i] < a[j];
132     else return i < j;
133 }
134 int main()
135 {
136     int n;
137     while(scanf("%d",&n) == 1 && n){
138         for(int i = 1;i <= n;i++){
139             scanf("%d",&a[i]);
140             b[i] = i;
141         }
142         init(n);
143         sort(b+1,b+n+1,cmp);
144         for(int i = 1;i <= n;i++){
145             splay(root,node[b[i]],null);
146             int sz = root->ch[0]->size;
147             printf("%d",root->ch[0]->size);
148             if(i == n)printf("\n");
149             else printf(" ");
150             splay(root,get_kth(root,i),null);
151             splay(root,get_kth(root,sz+2),root);
152             root->ch[1]->ch[0]->Update_Rev();
153         }
154     }
155     return 0;
156 }

```

3.4.2 例题：HDU3726

```

1  const int MAXN = 20010;
2  struct Node;
3  Node* null;
4  struct Node
5  {
6      Node *ch[2],*fa;//指向儿子和父亲结点
7      int size,key;
8      Node(){
9          ch[0] = ch[1] = fa = null;
10     }
11     inline void setc(Node* p,int d){
12         ch[d] = p;
13         p->fa = this;
14     }
15     inline bool d(){

```

```

16         return fa->ch[1] == this;
17     }
18     void push_up(){
19         size = ch[0]->size + ch[1]->size + 1;
20     }
21     void clear(){
22         size = 1;
23         ch[0] = ch[1] = fa = null;
24     }
25     inline bool isroot()
26     {
27         return fa == null || this != fa->ch[0] && this != fa->ch
           [1];
28     }
29 };
30 inline void rotate(Node* x)
31 {
32     Node *f = x->fa, *ff = x->fa->fa;
33     int c = x->d(), cc = f->d();
34     f->setc(x->ch[!c],c);
35     x->setc(f,!c);
36     if(ff->ch[cc] == f) ff->setc(x,cc);
37     else x->fa = ff;
38     f->push_up();
39 }
40 inline void splay(Node* &root,Node* x,Node* goal)
41 {
42     while(x->fa != goal){
43         if(x->fa->fa == goal) rotate(x);
44         else {
45             bool f = x->fa->d();
46             x->d() == f ? rotate(x->fa) : rotate(x);
47             rotate(x);
48         }
49     }
50     x->push_up();
51     if(goal == null) root = x;
52 }
53 //找到 r 子树里面的第 k 个
54 Node* get_kth(Node* r,int k)
55 {
56     Node* x = r;
57     while(x->ch[0]->size+1 != k){
58         if(k < x->ch[0]->size+1) x = x->ch[0];
59         else {
60             k -= x->ch[0]->size+1;
61             x = x->ch[1];
62         }
63     }
64     return x;
65 }

```

```

66 //在 root 的树中删掉 x
67 void erase(Node* &root,Node* x)
68 {
69     splay(root,x,null);
70     Node* t = root;
71     if(t->ch[1] != null){
72         root = t->ch[1];
73         splay(root,get_kth(root,1),null);
74         root->setc(t->ch[0],0);
75     }
76     else{
77         root = root->ch[0];
78     }
79     root->fa = null;
80     if(root != null)root->push_up();
81 }
82 void insert(Node* &root,Node* x)
83 {
84     if(root == null){
85         root = x;
86         return;
87     }
88     Node* now = root;
89     Node* pre = root->fa;
90     while(now != null){
91         pre = now;
92         now = now->ch[x->key >= now->key];
93     }
94     x->clear();
95     pre->setc(x,x->key >= pre->key);
96     splay(root,x,null);
97 }
98 void merge(Node* &A,Node* B)
99 {
100     if(A->size <= B->size)swap(A,B);
101     queue<Node*>Q;
102     Q.push(B);
103     while(!Q.empty()){
104         Node* fr = Q.front();
105         Q.pop();
106         if(fr->ch[0] != null)Q.push(fr->ch[0]);
107         if(fr->ch[1] != null)Q.push(fr->ch[1]);
108         fr->clear();
109         insert(A,fr);
110     }
111 }
112 Node pool[MAXN],*tail;
113
114 struct Edge
115 {
116     int u,v;

```

```

117 }edge[60010];
118 int a[MAXN];
119 bool del[60010];
120 struct QUERY
121 {
122     char op[10];
123     int u,v;
124 }query[500010];
125 int y[500010];
126
127 Node* node[MAXN];
128 Node* root[MAXN];
129 int F[MAXN];
130 int find(int x)
131 {
132     if(F[x] == -1)return x;
133     return F[x] = find(F[x]);
134 }
135 void debug(Node *root)
136 {
137     if(root == null)return;
138     debug(root->ch[0]);
139     printf("size: %d, key= %d\n",root->size,root->key);
140     debug(root->ch[1]);
141 }
142
143 int main()
144 {
145     int n,m;
146     int iCase = 0;
147     while(scanf("%d%d",&n,&m) == 2)
148     {
149         if(n == 0 && m == 0)break;
150         iCase++;
151         memset(F,-1,sizeof(F));
152         tail = pool;
153         null = tail++;
154         null->size = 0; null->ch[0] = null->ch[1] = null->fa = null
            ;
155         null->key = 0;
156         for(int i = 1;i <= n;i++)scanf("%d",&a[i]);
157         for(int i = 0;i < m;i++)
158         {
159             scanf("%d%d",&edge[i].u,&edge[i].v);
160             del[i] = false;
161         }
162         int Q = 0;
163         while(1)
164         {
165             scanf("%s",&query[Q].op);
166             if(query[Q].op[0] == 'E')break;

```



```

167         if(query[Q].op[0] == 'D'){
168             scanf("%d",&query[Q].u);
169             query[Q].u--;
170             del[query[Q].u] = true;
171         }
172         else if(query[Q].op[0] == 'Q'){
173             scanf("%d%d",&query[Q].u,&query[Q].v);
174         }
175         else{
176             scanf("%d%d",&query[Q].u,&query[Q].v);
177             y[Q] = a[query[Q].u];
178             a[query[Q].u] = query[Q].v;
179         }
180         Q++;
181     }
182     for(int i = 1;i <= n;i++){
183         node[i] = tail++;
184         node[i]—>clear();
185         node[i]—>key = a[i];
186         root[i] = node[i];
187     }
188     for(int i = 0;i < m;i++){
189         if(!del[i]){
190             int u = edge[i].u;
191             int v = edge[i].v;
192             int t1 = find(u);
193             int t2 = find(v);
194             if(t1 == t2)continue;
195             F[t2] = t1;
196             merge(root[t1],root[t2]);
197         }
198     }
199     vector<int>ans;
200     for(int i = Q-1;i >= 0;i—){
201         if(query[i].op[0] == 'D'){
202             int u = edge[query[i].u].u;
203             int v = edge[query[i].u].v;
204             int t1 = find(u);
205             int t2 = find(v);
206             if(t1 == t2)continue;
207             F[t2] = t1;
208             merge(root[t1],root[t2]);
209         }
210         else if(query[i].op[0] == 'Q'){
211             int u = query[i].u;
212             int k = query[i].v;
213             u = find(u);
214             if(k <= 0 || k > root[u]—>size){
215                 ans.push_back(0);
216             }
217             else{
218                 k = root[u]—>size - k + 1;

```

```

218         Node* p = get_kth(root[u],k);
219         ans.push_back(p->key);
220     }
221 }
222 else{
223     int u = query[i].u;
224     int t1 = find(u);
225     Node* p = node[u];
226     erase(root[t1],p);
227     p->clear();
228     p->key = y[i];
229     a[u] = y[i];
230     insert(root[t1],p);
231 }
232 }
233 double ret = 0;
234 int sz = ans.size();
235 for(int i = 0;i < sz;i++)ret += ans[i];
236 if(sz)ret /= sz;
237 printf("Case_%d: %.6lf\n",iCase,ret);
238 }
239 return 0;
240 }

```

3.5 动态树

3.5.1 SPOJQTREE

给定一棵 n 个结点的树，树的边上有权。有两种操作：

1. 修改一条边上的权值。
2. 查询两个结点 x 和 y 之间的最短路径中经过的最大的边的权值。

其中 $n \leq 10^4$

```

1 // http://www.spoj.com/problems/QTREE/
2 const int MAXN = 10010;
3 struct Node *null;
4 struct Node{
5     Node *fa,*ch[2];
6     int Max,key;
7     inline void push_up(){
8         if(this == null)return;
9         Max = max(key,max(ch[0]->Max,ch[1]->Max));
10    }
11    inline void setc(Node *p,int d){
12        ch[d] = p;
13        p->fa = this;
14    }
15    inline bool d(){
16        return fa->ch[1] == this;
17    }
18    inline bool isroot() {

```

```

19         return fa == null || fa->ch[0] != this && fa->ch[1] != this
20         ;
21     }
22     inline void rot(){
23         Node *f = fa,*ff = fa->fa;
24         int c = d(), cc = fa->d();
25         f->setc(ch[!c],c);
26         this->setc(f,!c);
27         if(ff->ch[cc] == f)ff->setc(this,cc);
28         else this->fa = ff;
29         f->push_up();
30     }
31     inline Node* splay(){
32         while(!isroot()){
33             if(!fa->isroot())
34                 d()==fa->d() ? fa->rot() : rot();
35             rot();
36         }
37         push_up();
38         return this;
39     }
40     inline Node* access(){
41         for(Node *p = this,*q = null; p != null; q = p, p = p->fa){
42             p->splay()->setc(q,1);
43             p->push_up();
44         }
45         return splay();
46     }
47     };
48     Node pool[MAXN],*tail;
49     Node *node[MAXN];
50     void init(int n){
51         tail = pool;
52         null = tail++;
53         null->fa = null->ch[0] = null->ch[1] = null;
54         null->Max = null->key = 0;
55         for(int i = 1;i <= n;i++){
56             node[i] = tail++;
57             node[i]->fa = node[i]->ch[0] = node[i]->ch[1] = null;
58             node[i]->Max = node[i]->key = 0;
59         }
60     }
61     struct Edge{
62         int to,next;
63         int w,id;
64     }edge[MAXN*2];
65     int head[MAXN],tot;
66     inline int addedge(int u,int v,int w,int id){
67         edge[tot].to = v;
68         edge[tot].w = w;
69         edge[tot].id = id;

```

```

69     edge[tot].next = head[u];
70     head[u] = tot++;
71 }
72 Node *ee[MAXN];
73 bool vis[MAXN];
74 void bfs(int n){
75     for(int i = 1;i <= n;i++)vis[i] = false;
76     queue<int>q;
77     q.push(1);
78     vis[1] = true;
79     while(!q.empty()){
80         int u = q.front();
81         q.pop();
82         for(int i = head[u];i != -1;i = edge[i].next){
83             int v = edge[i].to;
84             if(vis[v])continue;
85             vis[v] = true;
86             q.push(v);
87             ee[edge[i].id] = node[v];
88             node[v]—>key = edge[i].w;
89             node[v]—>push_up();
90             node[v]—>fa = node[u];
91         }
92     }
93 }
94 inline int ask(Node *x,Node *y){
95     x—>access();
96     for(x = null; y != null; x = y, y = y—>fa){
97         y—>splay();
98         if(y—>fa == null)return max(y—>ch[1]—>Max,x—>Max);
99         y—>setc(x,1);
100        y—>push_up();
101    }
102 }
103 int main()
104 {
105     int T;
106     scanf("%d",&T);
107     int n;
108     while(T—){
109         scanf("%d",&n);
110         for(int i = 1;i <= n;i++)head[i] = -1;
111         tot = 0;
112         init(n);
113         int u,v,w;
114         for(int i = 1;i < n;i++){
115             scanf("%d%d%d",&u,&v,&w);
116             addedge(u,v,w,i);
117             addedge(v,u,w,i);
118         }
119         bfs(n);

```

```

120     char op[20];
121     int x,y;
122     while(scanf("%s",op) == 1){
123         if(strcmp(op,"DONE") == 0)break;
124         scanf("%d%d",&x,&y);
125         if(op[0] == 'Q'){
126             printf("%d\n",ask(node[x],node[y]));
127         }
128         else {
129             ee[x]→splay()→key = y;
130             ee[x]→push_up();
131         }
132     }
133 }
134 return 0;
135 }

```

3.5.2 SPOJQTREE2

给定一棵 n 个结点的树，树的边上有权。有两种操作：

1. 查询两个结点 x 和 y 之间的最短路径长度。
2. 查询从 x 到 y 的最短路径的第 K 条边的长度。

其中 $n \leq 10^4$

```

1 // http://www.spoj.com/problems/QTREE2/
2 const int MAXN = 10010;
3 struct Node *null;
4 struct Node{
5     Node *fa,*ch[2];
6     int sum,val;
7     int size;
8     int id;
9     void clear(){
10         fa = ch[0] = ch[1] = null;
11         sum = val = 0;
12         size = 1;
13     }
14     inline void push_up(){
15         if(this == null)return;
16         sum = val + ch[0]→sum + ch[1]→sum;
17         size = ch[0]→size + ch[1]→size + 1;
18     }
19     inline void setc(Node *p,int d){
20         ch[d] = p;
21         p→fa = this;
22     }
23     inline bool d(){
24         return fa→ch[1] == this;
25     }
26     inline bool isroot(){

```

```

27         return fa == null || fa->ch[0] != this && fa->ch[1] != this
28         ;
29     }
30     inline void rot(){
31         Node *f = fa, *ff = fa->fa;
32         int c = d(), cc = fa->d();
33         f->setc(ch[!c],c);
34         this->setc(f,!c);
35         if(ff->ch[cc] == f) ff->setc(this,cc);
36         else this->fa = ff;
37         f->push_up();
38     }
39     inline Node* splay(){
40         while(!isroot()){
41             if(!fa->isroot())
42                 d()==fa->d() ? fa->rot() : rot();
43             rot();
44         }
45         push_up();
46         return this;
47     }
48     inline Node* access(){
49         for(Node *p = this,*q = null; p != null; q = p, p = p->fa){
50             p->splay()->setc(q,1);
51             p->push_up();
52         }
53         return splay();
54     };
55     Node pool[MAXN],*tail;
56     Node *node[MAXN];
57     void init(int n){
58         tail = pool;
59         null = tail++;
60         null->fa = null->ch[0] = null->ch[1] = null;
61         null->size = null->sum = null->val = 0;
62         for(int i = 1;i <= n;i++){
63             node[i] = tail++;
64             node[i]->id = i;
65             node[i]->clear();
66         }
67     }
68     struct Edge{
69         int to,next;
70         int w;
71     }edge[MAXN*2];
72     int head[MAXN],tot;
73     inline void addedge(int u,int v,int w){
74         edge[tot].to = v;
75         edge[tot].w = w;
76         edge[tot].next = head[u];

```

```

77     head[u] = tot++;
78 }
79 void dfs(int u,int pre){
80     for(int i = head[u];i != -1;i = edge[i].next){
81         int v = edge[i].to;
82         if(v == pre)continue;
83         dfs(v,u);
84         node[v]→val = edge[i].w;
85         node[v]→push_up();
86         node[v]→fa = node[u];
87     }
88 }
89 //查询 x→y 的距离
90 inline int query_sum(Node *x,Node *y){
91     x→access();
92     for(x = null; y != null; x = y, y = y→fa){
93         y→splay();
94         if(y→fa == null)
95             return y→ch[1]→sum + x→sum;
96         y→setc(x,1);
97         y→push_up();
98     }
99 }
100 //在 splay 中得到第 k 个点
101 Node* get_kth(Node* r,int k){
102     Node *x = r;
103     while(x→ch[0]→size+1 != k){
104         if(k < x→ch[0]→size+1)x = x→ch[0];
105         else {
106             k -= x→ch[0]→size+1;
107             x = x→ch[1];
108         }
109     }
110     return x;
111 }
112 //查询 x→y 路径上的第 k 个点
113 inline int query_kth(Node *x,Node *y,int k){
114     x→access();
115     for(x = null; y != null; x = y, y = y→fa){
116         y→splay();
117         if(y→fa == null){
118             if(y→ch[1]→size+1 == k)return y→id;
119             else if(y→ch[1]→size+1 > k)
120                 return get_kth(y→ch[1],y→ch[1]→size+1-k)→id;
121             else return get_kth(x,k-(y→ch[1]→size+1))→id;
122         }
123         y→setc(x,1);
124         y→push_up();
125     }
126 }
127 int main()

```

```

128 {
129     int T,n;
130     scanf("%d",&T);
131     while(T--){
132         scanf("%d",&n);
133         for(int i = 1;i <= n;i++)head[i] = -1;
134         tot = 0;
135         init(n);
136         int u,v,w;
137         for(int i = 1;i < n;i++){
138             scanf("%d%d%d",&u,&v,&w);
139             addedge(u,v,w);
140             addedge(v,u,w);
141         }
142         dfs(1,1);
143         char op[20];
144         while(scanf("%s",op) == 1){
145             if(strcmp(op,"DONE") == 0)break;
146             if(op[0] == 'D'){
147                 scanf("%d%d",&u,&v);
148                 printf("%d\n",query_sum(node[u],node[v]));
149             }
150             else {
151                 int k; scanf("%d%d%d",&u,&v,&k);
152                 printf("%d\n",query_kth(node[u],node[v],k));
153             }
154         }
155     }
156     return 0;
157 }

```

3.5.3 SPOJQTREE4

给定一棵 n 个结点的树，树的边上有权，每个结点有黑白两色，初始时所有的结点都是白的。有两种操作：

1. 对一个结点执行反色操作（白变黑，黑变白）
2. 查询树中距离最远的两个白点的距离。

其中 $n \leq 10^5$ ，查询数目不超过 10^5 。

```

1 //http://www.spoj.com/problems/QTREE4/
2 const int MAXN = 100010;
3 const int INF = 0x3f3f3f3f;
4 struct Node *null;
5 struct Node{
6     Node *fa,*ch[2];
7     multiset<int>st0,st1;//st0 是链，st1 是路径
8     int dd,d0;//d0 是该点对应边的长度，dd 是重链长度
9     int w0;//白点值为 0，黑点值为 -INF
10    int ls,rs,ms;
11    inline void clear(){
12        fa = ch[0] = ch[1] = null;

```



```

13         st0.clear(); st1.clear();
14         st0.insert(-INF);
15         st0.insert(-INF);
16         st1.insert(-INF);
17         w0 = 0; dd = d0 = 0;
18         ls = rs = ms = -INF;
19     }
20     inline void push_up(){
21         if(this == null) return;
22         dd = d0 + ch[0]→dd + ch[1]→dd;
23         int m0 = max(w0,*st0.rbegin()), ml = max(m0,ch[0]→rs+d0),
            mr = max(m0,ch[1]→ls);
24         ls = max(ch[0]→ls,ch[0]→dd + d0 + mr);
25         rs = max(ch[1]→rs,ch[1]→dd + ml);
26         multiset<int>::reverse_iterator it = st0.rbegin();
27         ++it;
28         int t0 = max((*st0.rbegin()) + (*it) , *st1.rbegin());
29         if(w0 == 0)
30             t0 = max(t0,max(0,*st0.rbegin()));
31         ms = max(max(max(ml+ch[1]→ls,mr+d0+ch[0]→rs),max(ch[0]→
            ms,ch[1]→ms)),t0);
32     }
33     inline void setc(Node *p,int d){
34         ch[d] = p;
35         p→fa = this;
36     }
37     inline bool d(){
38         return fa→ch[1] == this;
39     }
40     inline bool isroot(){
41         return fa == null || fa→ch[0] != this && fa→ch[1] != this
            ;
42     }
43     inline void rot(){
44         Node *f = fa, *ff = fa→fa;
45         int c = d(), cc = fa→d();
46         f→setc(ch[!c],c);
47         this→setc(f,!c);
48         if(ff→ch[cc] == f) ff→setc(this,cc);
49         else this→fa = ff;
50         f→push_up();
51     }
52     inline Node* splay(){
53         while(!isroot()){
54             if(!fa→isroot())
55                 d()==fa→d() ? fa→rot() : rot();
56             rot();
57         }
58         push_up();
59         return this;
60     }

```

```

61     inline Node* access(){
62         for(Node *p = this,*q = null; p != null; q = p, p = p->fa){
63             p->splay();
64             if(p->ch[1] != null){
65                 p->st0.insert(p->ch[1]->ls);
66                 p->st1.insert(p->ch[1]->ms);
67             }
68             if(q != null){
69                 p->st0.erase(p->st0.find(q->ls));
70                 p->st1.erase(p->st1.find(q->ms));
71             }
72             p->setc(q,1);
73             p->push_up();
74         }
75         return splay();
76     }
77 };
78 Node pool[MAXN],*tail;
79 Node *node[MAXN];
80 inline void init(int n){
81     tail = pool;
82     null = tail++;
83     null->fa = null->ch[0] = null->ch[1] = null;
84     null->st0.clear(); null->st1.clear();
85     null->ls = null->rs = null->ms = -INF;
86     null->w0 = -INF;
87     null->d0 = null->dd = 0;
88     for(int i = 1;i <= n;i++){
89         node[i] = tail++;
90         node[i]->clear();
91     }
92 }
93 struct Edge{
94     int to,next,w;
95 }edge[MAXN*2];
96 int head[MAXN],tot;
97 inline void addedge(int u,int v,int w){
98     edge[tot].to = v;
99     edge[tot].w = w;
100    edge[tot].next = head[u];
101    head[u] = tot++;
102 }
103 inline void dfs(int u,int pre){
104     for(int i = head[u];i != -1;i = edge[i].next){
105         int v = edge[i].to;
106         if(v == pre)continue;
107         node[v]->fa = node[u];
108         node[v]->d0 = edge[i].w;
109         dfs(v,u);
110         node[u]->st0.insert(node[v]->ls);
111         node[u]->st1.insert(node[v]->ms);

```

```

112     }
113     node[u]→push_up();
114 }
115 template <class T>
116 inline bool scan_d(T &ret) {
117     char c; int sgn;
118     if(c=getchar(),c==EOF) return 0;
119     while(c!='-'&&(c<'0' || c>'9')) c=getchar();
120     sgn=(c=='-')?-1:1;
121     ret=(c=='-')?0:(c-'0');
122     while(c=getchar(),c>='0'&&c<='9') ret=ret*10+(c-'0');
123     ret*=sgn;
124     return 1;
125 }
126 int main()
127 {
128     int n;
129     while(scanf("%d",&n) == 1){
130         for(int i = 1;i <= n;i++)head[i] = -1;
131         tot = 0;
132         init(n);
133         int u,v,w;
134         for(int i = 1;i < n;i++){
135             scan_d(u); scan_d(v);scan_d(w);
136             addedge(u,v,w);
137             addedge(v,u,w);
138         }
139         dfs(1,1);
140         int ans = node[1]→ms;
141         int Q;
142         char op[10];
143         scanf("%d",&Q);
144         while(Q—){
145             scanf("%s",op);
146             if(op[0] == 'C'){
147                 scan_d(u);
148                 node[u]→access();
149                 node[u]→splay();
150                 if(node[u]→w0 == 0)node[u]→w0 = -INF;
151                 else node[u]→w0 = 0;
152                 node[u]→push_up();
153                 ans = node[u]→ms;
154             }
155             else{
156                 if(ans < 0)puts("They have disappeared.");
157                 else printf("%d\n",ans);
158             }
159         }
160     }
161     return 0;
162 }

```

3.5.4 SPOJQTREE5

给定一棵 n 个结点的树，边权均为 1。每个结点有黑白两色，初始时所有结点都是黑的。两种查询操作：

1. 对一个结点执行反色操作（白变黑，黑变白）
2. 查询距离某个特定结点 i 最远的白点的距离。

其中 $\leq 10^5$ ，查询数目不超过 10^5 。

```

1 //http://www.spoj.com/problems/QTREE5/
2 const int MAXN = 100010;
3 const int INF = 0x3f3f3f3f;
4 struct Node *null;
5 struct Node{
6     Node *fa,*ch[2];
7     multiset<int>st;
8     int dd,d0;
9     int w0;
10    int ls,rs;
11    inline void clear(){
12        fa = ch[0] = ch[1] = null;
13        st.clear();
14        st.insert(INF);
15        w0 = INF; dd = d0 = 0;
16        ls = rs = INF;
17    }
18    inline void push_up(){
19        if(this == null)return;
20        dd = d0 + ch[0]->dd + ch[1]->dd;
21        int m0 = min(w0,*st.begin()), ml = min(m0,ch[0]->rs+d0), mr
            = min(m0,ch[1]->ls);
22        ls = min(ch[0]->ls,ch[0]->dd + d0 + mr);
23        rs = min(ch[1]->rs,ch[1]->dd + ml);
24    }
25    inline void setc(Node *p,int d){
26        ch[d] = p;
27        p->fa = this;
28    }
29    inline bool d(){
30        return fa->ch[1] == this;
31    }
32    inline bool isroot(){
33        return fa == null || fa->ch[0] != this && fa->ch[1] != this
            ;
34    }
35    inline void rot(){
36        Node *f = fa, *ff = fa->fa;
37        int c = d(), cc = fa->d();
38        f->setc(ch[!c],c);
39        this->setc(f,!c);
40        if(ff->ch[cc] == f)ff->setc(this,cc);
41        else this->fa = ff;

```

```

42         f->push_up();
43     }
44     inline Node* splay(){
45         while(!isroot()){
46             if(!fa->isroot())
47                 d()==fa->d() ? fa->rot() : rot();
48             rot();
49         }
50         push_up();
51         return this;
52     }
53     inline Node* access(){
54         for(Node *p = this,*q = null; p != null; q = p, p = p->fa){
55             p->splay();
56             if(p->ch[1] != null){
57                 p->st.insert(p->ch[1]->ls);
58             }
59             if(q != null){
60                 p->st.erase(p->st.find(q->ls));
61             }
62             p->setc(q,1);
63             p->push_up();
64         }
65         return splay();
66     }
67 };
68 Node pool[MAXN],*tail;
69 Node *node[MAXN];
70 inline void init(int n){
71     tail = pool;
72     null = tail++;
73     null->fa = null->ch[0] = null->ch[1] = null;
74     null->st.clear();
75     null->ls = null->rs = INF;
76     null->w0 = INF;
77     null->dd = null->d0 = 0;
78     for(int i = 1;i <= n;i++){
79         node[i] = tail++;
80         node[i]->clear();
81     }
82 }
83 struct Edge{
84     int to,next;
85 }edge[MAXN*2];
86 int head[MAXN],tot;
87 inline void addedge(int u,int v){
88     edge[tot].to = v;
89     edge[tot].next = head[u];
90     head[u] = tot++;
91 }
92 inline void dfs(int u,int pre){

```

```

93     for(int i = head[u]; i != -1; i = edge[i].next){
94         int v = edge[i].to;
95         if(v == pre)continue;
96         node[v]→fa = node[u];
97         node[v]→d0 = 1;
98         dfs(v,u);
99         node[u]→st.insert(node[v]→ls);
100     }
101     node[u]→push_up();
102 }
103 int main()
104 {
105     int n;
106     while(scanf("%d",&n) == 1){
107         init(n);
108         for(int i = 1; i <= n; i++)head[i] = -1;
109         tot = 0;
110         int u,v;
111         for(int i = 1; i < n; i++){
112             scanf("%d%d",&u,&v);
113             addedge(u,v);
114             addedge(v,u);
115         }
116         dfs(1,1);
117         int Q;
118         scanf("%d",&Q);
119         int op;
120         while(Q—){
121             scanf("%d%d",&op,&v);
122             if(op == 0){
123                 node[v]→access();
124                 node[v]→splay();
125                 if(node[v]→w0 == 0)node[v]→w0 = INF;
126                 else node[v]→w0 = 0;
127                 node[v]→push_up();
128             }
129             else {
130                 node[v]→access();
131                 node[v]→splay();
132                 if(node[v]→rs < INF)printf("%d\n",node[v]→rs);
133                 else printf("-1\n");
134             }
135         }
136     }
137     return 0;
138 }

```

3.5.5 SPOJQTREE6

给定一棵 n 个结点的树，每个结点有黑白两色，初始时所有结点都是黑的。你被要求支持：

1. 对一个结点执行反色操作（白变黑，黑变白）

2. 询问有多少个点与 u 相连。两个结点 u, v 相连当且仅当 u, v 路径上所有点的颜色相同。
其中 $n \leq 10^5$, 查询数目不超过 10^5 .

```

1 //http://www.spoj.com/problems/QTREE6/
2 const int MAXN = 100010;
3 struct Node *null;
4 struct Node{
5     Node *fa,*ch[2];
6     int co;//0 is black, 1 is white
7     int lco,rco;
8     int ls,rs;
9     int s[2];
10    int sum[2];//the sum of black and white
11    inline void clear(){
12        fa = ch[0] = ch[1] = null;
13        co = lco = rco = 0;
14        ls = rs = 1;
15        s[0] = s[1] = 0;
16        sum[0] = 1; sum[1] = 0;
17    }
18    inline void push_up(){
19        if(this == null)return;
20        if(ch[0] != null)lco = ch[0]->lco;
21        else lco = co;
22        if(ch[1] != null)rco = ch[1]->rco;
23        else rco = co;
24        sum[0] = ch[0]->sum[0] + ch[1]->sum[0] + (co == 0);
25        sum[1] = ch[0]->sum[1] + ch[1]->sum[1] + (co == 1);
26        int ml = 1 + s[co] + (co==ch[0]->rco?ch[0]->rs:0);
27        int mr = 1 + s[co] + (co==ch[1]->lco?ch[1]->ls:0);
28        ls = ch[0]->ls;
29        if(lco == co && ch[0]->sum[!co] == 0)ls += mr;
30        rs = ch[1]->rs;
31        if(rco == co && ch[1]->sum[!co] == 0)rs += ml;
32    }
33    inline void setc(Node *p,int d){
34        ch[d] = p;
35        p->fa = this;
36    }
37    inline bool d(){
38        return fa->ch[1] == this;
39    }
40    inline bool isroot(){
41        return fa == null || fa->ch[0] != this && fa->ch[1] != this
42            ;
43    }
44    inline void rot(){
45        Node *f = fa, *ff = fa->fa;
46        int c = d(), cc = fa->d();
47        f->setc(ch[!c],c);
48        this->setc(f,!c);

```

```

48     if(ff->ch[cc] == f)ff->setc(this,cc);
49     else this->fa = ff;
50     f->push_up();
51 }
52 inline Node* splay(){
53     while(!isroot()){
54         if(!fa->isroot())
55             d()==fa->d() ? fa->rot() : rot();
56         rot();
57     }
58     push_up();
59     return this;
60 }
61 inline Node* access(){
62     for(Node *p = this,*q = null; p != null; q = p, p = p->fa){
63         p->splay();
64         if(p->ch[1] != null)
65             p->s[p->ch[1]->lco] += p->ch[1]->ls;
66         if(q != null)
67             p->s[q->lco] -= q->ls;
68         p->setc(q,1);
69         p->push_up();
70     }
71     return splay();
72 }
73 };
74 Node pool[MAXN],*tail;
75 Node *node[MAXN];
76 void init(int n){
77     tail = pool;
78     null = tail++;
79     null->fa = null->ch[0] = null->ch[1] = null;
80     null->s[0] = null->s[1] = 0;
81     null->ls = null->rs = 0;
82     null->sum[0] = null->sum[1] = 0;
83     null->co = null->lco = null->rco = 0;
84     for(int i = 1;i <= n;i++){
85         node[i] = tail++;
86         node[i]->clear();
87     }
88 }
89 struct Edge{
90     int to,next;
91 }edge[MAXN*2];
92 int head[MAXN],tot;
93 inline void addedge(int u,int v){
94     edge[tot].to = v; edge[tot].next = head[u]; head[u] = tot++;
95 }
96 void dfs(int u,int pre){
97     for(int i = head[u];i != -1;i = edge[i].next){
98         int v = edge[i].to;

```



```

99         if(v == pre)continue;
100        node[v]→fa = node[u];
101        dfs(v,u);
102        node[u]→s[node[v]→lco] += node[v]→ls;
103    }
104    node[u]→push_up();
105 }
106 int main()
107 {
108     int n;
109     while(scanf("%d",&n) == 1){
110         init(n);
111         for(int i = 1;i <= n;i++)head[i] = -1;
112         tot = 0;
113         int u,v;
114         for(int i = 1;i < n;i++){
115             scanf("%d%d",&u,&v);
116             addedge(u,v);
117             addedge(v,u);
118         }
119         dfs(1,1);
120         int Q;
121         int op;
122         scanf("%d",&Q);
123         while(Q—){
124             scanf("%d%d",&op,&u);
125             if(op == 0){
126                 node[u]→access();
127                 node[u]→splay();
128                 printf("%d\n",node[u]→rs);
129             }
130             else{
131                 node[u]→access();
132                 node[u]→splay();
133                 node[u]→co ^= 1;
134                 node[u]→push_up();
135             }
136         }
137         return 0;
138     }
139     return 0;
140 }

```

3.5.6 SPOJQTREE7

给定一棵 n 个结点的树，每个结点有黑白两色和权值。三种操作：

1. 对一个结点执行反色操作（白变黑，黑变白）
2. 询问与 u 相连的点中点权的最大值。两个结点 u,v 相连当且仅当 u,v 路径上所有点的颜色相同。
3. 改变一个点的点权。

其中 $n \leq 10^5$ ，查询数目不超过 10^5 。

```

1 //http://www.spoj.com/problems/QTREE7/
2 const int MAXN = 100010;
3 const int INF = 0x3f3f3f3f;
4 struct Node *null;
5 struct Node{
6     Node *fa,*ch[2];
7     int co;
8     int lco,rco;
9     int ls,rs;
10    int w0;
11    multiset<int>st[2];
12    int sum[2];
13    inline void clear(int _co = 0, int _w0 = 0){
14        fa = ch[0] = ch[1] = null;
15        co = lco = rco = _co;
16        w0 = _w0;
17        ls = rs = _w0;
18        st[0].clear(); st[1].clear();
19        st[0].insert(-INF); st[1].insert(-INF);
20        sum[0] = sum[1] = 0; sum[_co]++;
21    }
22    inline void push_up(){
23        if(this == null)return;
24        if(ch[0] != null)lco = ch[0]->lco;
25        else lco = co;
26        if(ch[1] != null)rco = ch[1]->rco;
27        else rco = co;
28        sum[0] = ch[0]->sum[0] + ch[1]->sum[0] + (co == 0);
29        sum[1] = ch[0]->sum[1] + ch[1]->sum[1] + (co == 1);
30        int ml = max(w0,max(*st[co].rbegin(),co==ch[0]->rco?ch[0]->
31            rs:-INF));
32        int mr = max(w0,max(*st[co].rbegin(),co==ch[1]->lco?ch[1]->
33            ls:-INF));
34        ls = ch[0]->ls;
35        if(lco == co && ch[0]->sum[!co] == 0)ls = max(ls,mr);
36        rs = ch[1]->rs;
37        if(rco == co && ch[1]->sum[!co] == 0)rs = max(rs,ml);
38    }
39    inline void setc(Node *p,int d){
40        ch[d] = p;
41        p->fa = this;
42    }
43    inline bool d(){
44        return fa->ch[1] == this;
45    }
46    inline bool isroot(){
47        return fa == null || fa->ch[0] != this && fa->ch[1] != this
48            ;
49    }
50    inline void rot(){

```

```

48     Node *f = fa, *ff = fa->fa;
49     int c = d(), cc = fa->d();
50     f->setc(ch[!c],c);
51     this->setc(f,!c);
52     if(ff->ch[cc] == f)ff->setc(this,cc);
53     else this->fa = ff;
54     f->push_up();
55 }
56 inline Node* splay(){
57     while(!isroot()){
58         if(!fa->isroot())
59             d()==fa->d() ? fa->rot() : rot();
60         rot();
61     }
62     push_up();
63     return this;
64 }
65 inline Node* access(){
66     for(Node *p = this,*q = null; p != null; q = p, p = p->fa){
67         p->splay();
68         if(p->ch[1] != null)
69             p->st[p->ch[1]->lco].insert(p->ch[1]->ls);
70         if(q != null)
71             p->st[q->lco].erase(p->st[q->lco].find(q->ls));
72         p->setc(q,1);
73         p->push_up();
74     }
75     return splay();
76 }
77 };
78 Node pool[MAXN],*tail;
79 Node *node[MAXN];
80 int color[MAXN],val[MAXN];
81 void init(int n){
82     tail = pool;
83     null = tail++;
84     null->fa = null->ch[0] = null->ch[1] = null;
85     null->st[0].clear(); null->st[1].clear();
86     null->ls = null->rs = -INF;
87     null->sum[0] = null->sum[1] = 0;
88     null->co = null->lco = null->rco = 0;
89     for(int i = 1;i <= n;i++){
90         node[i] = tail++;
91         node[i]->clear(color[i],val[i]);
92     }
93 }
94 struct Edge{
95     int to,next;
96 }edge[MAXN*2];
97 int head[MAXN],tot;
98 inline void addedge(int u,int v){

```

```

99     edge[tot].to = v; edge[tot].next = head[u]; head[u] = tot++;
100 }
101 void dfs(int u,int pre){
102     for(int i = head[u];i != -1;i = edge[i].next){
103         int v = edge[i].to;
104         if(v == pre)continue;
105         node[v]→fa = node[u];
106         dfs(v,u);
107         node[u]→st[node[v]→lco].insert(node[v]→ls);
108     }
109     node[u]→push_up();
110 }
111 int main()
112 {
113     int n;
114     while(scanf("%d",&n) == 1){
115         for(int i = 1;i <= n;i++)head[i] = -1;
116         tot = 0;
117         int u,v;
118         for(int i = 1;i < n;i++){
119             scanf("%d%d",&u,&v);
120             addedge(u,v);
121             addedge(v,u);
122         }
123         for(int i = 1;i <= n;i++)scanf("%d",&color[i]);
124         for(int i = 1;i <= n;i++)scanf("%d",&val[i]);
125         init(n);
126         dfs(1,1);
127         int Q;
128         int w,op;
129         scanf("%d",&Q);
130         while(Q—){
131             scanf("%d",&op);
132             if(op == 0){
133                 scanf("%d",&u);
134                 node[u]→access(); node[u]→splay();
135                 printf("%d\n",node[u]→rs);
136             }
137             else if(op == 1){
138                 scanf("%d",&u);
139                 node[u]→access(); node[u]→splay();
140                 node[u]→co ^= 1;
141                 node[u]→push_up();
142             }
143             else {
144                 scanf("%d%d",&u,&w);
145                 node[u]→access(); node[u]→splay();
146                 node[u]→w0 = w;
147                 node[u]→push_up();
148             }
149         }

```

```

150     }
151     return 0;
152 }

```

3.5.7 HDU4010

支持:

- 1 x y: 如果 x,y 不在同一颗子树中, 则通过在 x,y 之间连边的方式, 连接这两颗子树
 - 2 x y: 如果 x,y 在同一颗子树中, 且 $x \neq y$, 则将 x 视为这颗子树的根以后, 切断 y 与其父亲结点的连接
 - 3 w x y: 如果 x,y 在同一颗子树中, 则将 x,y 之间路径上所有点的点权增加 w
 - 4 x y: 如果 x,y 在同一颗子树中, 输出 x,y 之间路径上点权的最大值
- 非法操作输出 -1

```

1  const int MAXN = 300010;
2  const int INF = 0x3f3f3f3f;
3  struct Node *null;
4  struct Node{
5      Node *fa,*ch[2];
6      int Max,val;
7      int rev;//旋转标记
8      int add;
9      inline void clear(int _val){
10         fa = ch[0] = ch[1] = null;
11         val = Max = _val;
12         rev = 0;
13         add = 0;
14     }
15     inline void push_up(){
16         Max = max(val,max(ch[0]->Max,ch[1]->Max));
17     }
18     inline void setc(Node *p,int d){
19         ch[d] = p;
20         p->fa = this;
21     }
22     inline bool d(){
23         return fa->ch[1] == this;
24     }
25     inline bool isroot(){
26         return fa == null || fa->ch[0] != this && fa->ch[1] != this
27             ;
28     }
29     //翻转
30     inline void flip(){
31         if(this == null)return;
32         swap(ch[0],ch[1]);
33         rev ^= 1;
34     }
35     inline void update_add(int w){
36         if(this == null)return;
37         val += w;

```

```

37         add += w;
38         Max += w;
39     }
40     inline void push_down(){
41         if(rev){
42             ch[0]→flip(); ch[1]→flip(); rev = 0;
43         }
44         if(add){
45             ch[0]→update_add(add); ch[1]→update_add(add);
46             add = 0;
47         }
48     }
49     //直接标记下放
50     inline void go(){
51         if(!isroot())fa→go();
52         push_down();
53     }
54     inline void rot(){
55         Node *f = fa, *ff = fa→fa;
56         int c = d(), cc = fa→d();
57         f→setc(ch[!c],c);
58         this→setc(f,!c);
59         if(ff→ch[cc] == f)ff→setc(this,cc);
60         else this→fa = ff;
61         f→push_up();
62     }
63     inline Node* splay(){
64         go();
65         while(!isroot()){
66             if(!fa→isroot())
67                 d()==fa→d() ? fa→rot() : rot();
68             rot();
69         }
70         push_up();
71         return this;
72     }
73     inline Node* access(){
74         for(Node *p = this,*q = null; p != null; q = p, p = p→fa){
75             p→splay()→setc(q,1);
76             p→push_up();
77         }
78         return splay();
79     }
80     //找该点的根
81     inline Node* find_root(){
82         Node *x;
83         for(x = access(); x→push_down(), x→ch[0] != null; x = x→ch[0]);
84         return x;
85     }
86     //变为树根 (换根操作)

```

```

87     void make_root(){
88         access()->flip();
89     }
90     //切断该点和父亲结点的边
91     void cut(){
92         access();
93         ch[0]->fa = null;
94         ch[0] = null;
95         push_up();
96     }
97     //切断该点以 x 为根时, 该点和父亲结点的根
98     //要求这个点和 x 在同一颗树而且不能相同
99     //x 变为所在树的树根
100    void cut(Node* x){
101        if(this == x || find_root() != x->find_root())
102            puts("-1");
103        else {
104            x->make_root();
105            cut();
106        }
107    }
108    //该点连接到 x
109    //假如是有虚边信息的, 需要先 x->access() 再连接
110    void link(Node *x){
111        if(find_root() == x->find_root())
112            puts("-1");
113        else {
114            make_root(); fa = x;
115        }
116    }
117 };
118 Node pool[MAXN],*tail;
119 Node *node[MAXN];
120 struct Edge{
121     int to,next;
122 }edge[MAXN*2];
123 int head[MAXN],tot;
124 inline void addedge(int u,int v){
125     edge[tot].to = v;
126     edge[tot].next = head[u];
127     head[u] = tot++;
128 }
129 void dfs(int u,int pre){
130     for(int i = head[u];i != -1;i = edge[i].next){
131         int v = edge[i].to;
132         if(v == pre)continue;
133         node[v]->fa = node[u];
134         dfs(v,u);
135     }
136 }
137 void ADD(Node *x,Node *y,int w){

```

```

138     x->access();
139     for(x = null; y != null; x = y, y = y->fa){
140         y->splay();
141         if(y->fa == null){
142             y->ch[1]->update_add(w);
143             x->update_add(w);
144             y->val += w;
145             y->push_up();
146             return;
147         }
148         y->setc(x,1);
149         y->push_up();
150     }
151 }
152 int ask(Node *x, Node *y){
153     x->access();
154     for(x = null; y != null; x = y, y = y->fa){
155         y->splay();
156         if(y->fa == null)
157             return max(y->val, max(y->ch[1]->Max, x->Max));
158         y->setc(x,1);
159         y->push_up();
160     }
161 }
162 int main()
163 {
164     int n;
165     while(scanf("%d",&n) == 1){
166         for(int i = 1; i <= n; i++) head[i] = -1;
167         tot = 0;
168         int u, v;
169         for(int i = 1; i < n; i++){
170             scanf("%d%d",&u,&v);
171             addedge(u,v);
172             addedge(v,u);
173         }
174         tail = pool;
175         null = tail++;
176         null->clear(-INF);
177         for(int i = 1; i <= n; i++){
178             node[i] = tail++;
179             scanf("%d",&v);
180             node[i]->clear(v);
181         }
182         dfs(1,1);
183         int m, op;
184         int x, y, w;
185         scanf("%d",&m);
186         while(m--){
187             scanf("%d",&op);
188             if(op == 1){

```



```

189         scanf("%d%d",&x,&y);
190         node[x]→link(node[y]);
191     }
192     else if(op == 2){
193         scanf("%d%d",&x,&y);
194         node[y]→cut(node[x]);
195     }
196     else if(op == 3){
197         scanf("%d%d%d",&w,&x,&y);
198         if(node[x]→find_root() != node[y]→find_root())
199             printf("-1\n");
200         else ADD(node[x],node[y],w);
201     }
202     else{
203         scanf("%d%d",&x,&y);
204         if(node[x]→find_root() != node[y]→find_root())
205             printf("-1\n");
206         else printf("%d\n",ask(node[x],node[y]));
207     }
208 }
209 printf("\n");
210 }
211 return 0;
212 }

```

3.6 主席树

3.6.1 查询区间多少个不同的数

查询区间有多少个不同的数 (SPOJ DQUERY)

```

1  /*
2   *  给出一个序列，查询区间内有多少个不相同的数
3   */
4  const int MAXN = 30010;
5  const int M = MAXN * 100;
6  int n,q,tot;
7  int a[MAXN];
8  int T[MAXN],lson[M],rson[M],c[M];
9  int build(int l,int r){
10     int root = tot++;
11     c[root] = 0;
12     if(l != r){
13         int mid = (l+r)>>1;
14         lson[root] = build(l,mid);
15         rson[root] = build(mid+1,r);
16     }
17     return root;
18 }
19 int update(int root,int pos,int val){
20     int newroot = tot++, tmp = newroot;
21     c[newroot] = c[root] + val;

```

```

22     int l = 1, r = n;
23     while(l < r){
24         int mid = (l+r)>>1;
25         if(pos <= mid){
26             lson[newroot] = tot++; rson[newroot] = rson[root];
27             newroot = lson[newroot]; root = lson[root];
28             r = mid;
29         }
30         else{
31             rson[newroot] = tot++; lson[newroot] = lson[root];
32             newroot = rson[newroot]; root = rson[root];
33             l = mid+1;
34         }
35         c[newroot] = c[root] + val;
36     }
37     return tmp;
38 }
39 int query(int root,int pos){
40     int ret = 0;
41     int l = 1, r = n;
42     while(pos < r){
43         int mid = (l+r)>>1;
44         if(pos <= mid){
45             r = mid;
46             root = lson[root];
47         }
48         else{
49             ret += c[lson[root]];
50             root = rson[root];
51             l = mid+1;
52         }
53     }
54     return ret + c[root];
55 }
56 int main(){
57     while(scanf("%d",&n) == 1){
58         tot = 0;
59         for(int i = 1;i <= n;i++){
60             scanf("%d",&a[i]);
61             T[n+1] = build(1,n);
62             map<int,int>mp;
63             for(int i = n;i >= 1;i--){
64                 if(mp.find(a[i]) == mp.end()){
65                     T[i] = update(T[i+1],i,1);
66                 }
67                 else{
68                     int tmp = update(T[i+1],mp[a[i]],-1);
69                     T[i] = update(tmp,i,1);
70                 }
71                 mp[a[i]] = i;
72             }

```

```

73         scanf("%d",&q);
74         while(q--){
75             int l,r;
76             scanf("%d%d",&l,&r);
77             printf("%d\n",query(T[l],r));
78         }
79     }
80     return 0;
81 }

```

3.6.2 静态区间第 k 大

POJ2104

```

1  const int MAXN = 100010;
2  const int M = MAXN * 30;
3  int n,q,m,tot;
4  int a[MAXN], t[MAXN];
5  int T[MAXN], lson[M], rson[M], c[M];
6  void Init_hash(){
7      for(int i = 1; i <= n;i++)
8          t[i] = a[i];
9      sort(t+1,t+1+n);
10     m = unique(t+1,t+1+n)-t-1;
11 }
12 int build(int l,int r){
13     int root = tot++;
14     c[root] = 0;
15     if(l != r){
16         int mid = (l+r)>>1;
17         lson[root] = build(l,mid);
18         rson[root] = build(mid+1,r);
19     }
20     return root;
21 }
22 int hash(int x){
23     return lower_bound(t+1,t+1+m,x) - t;
24 }
25 int update(int root,int pos,int val){
26     int newroot = tot++, tmp = newroot;
27     c[newroot] = c[root] + val;
28     int l = 1, r = m;
29     while(l < r){
30         int mid = (l+r)>>1;
31         if(pos <= mid){
32             lson[newroot] = tot++; rson[newroot] = rson[root];
33             newroot = lson[newroot]; root = lson[root];
34             r = mid;
35         }
36         else{
37             rson[newroot] = tot++; lson[newroot] = lson[root];
38             newroot = rson[newroot]; root = rson[root];

```

```

39         l = mid+1;
40     }
41     c[newroot] = c[root] + val;
42 }
43 return tmp;
44 }
45 int query(int left_root,int right_root,int k){
46     int l = 1, r = m;
47     while( l < r){
48         int mid = (l+r)>>1;
49         if(c[lson[left_root]]-c[lson[right_root]] >= k ){
50             r = mid;
51             left_root = lson[left_root];
52             right_root = lson[right_root];
53         }
54         else{
55             l = mid + 1;
56             k -= c[lson[left_root]] - c[lson[right_root]];
57             left_root = rson[left_root];
58             right_root = rson[right_root];
59         }
60     }
61     return l;
62 }
63 int main(){
64     while(scanf("%d%d",&n,&q) == 2){
65         tot = 0;
66         for(int i = 1;i <= n;i++){
67             scanf("%d",&a[i]);
68             Init_hash();
69             T[n+1] = build(1,m);
70             for(int i = n;i ;i--){
71                 int pos = hash(a[i]);
72                 T[i] = update(T[i+1],pos,1);
73             }
74             while(q--){
75                 int l,r,k;
76                 scanf("%d%d%d",&l,&r,&k);
77                 printf("%d\n",t[query(T[l],T[r+1],k)]);
78             }
79         }
80         return 0;
81     }

```

3.6.3 树上路径点权第 k 大

树上路径点权第 k 大 (SPOJ COT)

LCA + 主席树

```

1 //主席树部分 *****
2 const int MAXN = 200010;
3 const int M = MAXN * 40;

```

```

4  int n,q,m,TOT;
5  int a[MAXN], t[MAXN];
6  int T[MAXN], lson[M], rson[M], c[M];
7  void Init_hash(){
8      for(int i = 1; i <= n;i++)
9          t[i] = a[i];
10     sort(t+1,t+1+n);
11     m = unique(t+1,t+n+1)-t-1;
12 }
13 int build(int l,int r){
14     int root = TOT++;
15     c[root] = 0;
16     if(l != r){
17         int mid = (l+r)>>1;
18         lson[root] = build(l,mid);
19         rson[root] = build(mid+1,r);
20     }
21     return root;
22 }
23 int hash(int x){
24     return lower_bound(t+1,t+1+m,x) - t;
25 }
26 int update(int root,int pos,int val){
27     int newroot = TOT++, tmp = newroot;
28     c[newroot] = c[root] + val;
29     int l = 1, r = m;
30     while( l < r){
31         int mid = (l+r)>>1;
32         if(pos <= mid){
33             lson[newroot] = TOT++; rson[newroot] = rson[root];
34             newroot = lson[newroot]; root = lson[root];
35             r = mid;
36         }
37         else{
38             rson[newroot] = TOT++; lson[newroot] = lson[root];
39             newroot = rson[newroot]; root = rson[root];
40             l = mid+1;
41         }
42         c[newroot] = c[root] + val;
43     }
44     return tmp;
45 }
46 int query(int left_root,int right_root,int LCA,int k){
47     int lca_root = T[LCA];
48     int pos = hash(a[LCA]);
49     int l = 1, r = m;
50     while(l < r){
51         int mid = (l+r)>>1;
52         int tmp = c[lson[left_root]] + c[lson[right_root]] - 2*c[
                    lson[lca_root]] + (pos >= l && pos <= mid);
53         if(tmp >= k){

```

```

54         left_root = lson[left_root];
55         right_root = lson[right_root];
56         lca_root = lson[lca_root];
57         r = mid;
58     }
59     else{
60         k -= tmp;
61         left_root = rson[left_root];
62         right_root = rson[right_root];
63         lca_root = rson[lca_root];
64         l = mid + 1;
65     }
66 }
67 return l;
68 }
69
70 //LCA 部分
71 int rmq[2*MAXN]; //rmq 数组, 就是欧拉序列对应的深度序列
72 struct ST{
73     int mm[2*MAXN];
74     int dp[2*MAXN][20]; //最小值对应的下标
75     void init(int n){
76         mm[0] = -1;
77         for(int i = 1; i <= n; i++){
78             mm[i] = ((i & (i-1)) == 0) ? mm[i-1] + 1 : mm[i-1];
79             dp[i][0] = i;
80         }
81         for(int j = 1; j <= mm[n]; j++){
82             for(int i = 1; i + (1 << j) - 1 <= n; i++){
83                 dp[i][j] = rmq[dp[i][j-1]] < rmq[dp[i+(1 << (j-1))][j-1]] ? dp[i][j-1] : dp[i+(1 << (j-1))][j-1];
84             }
85             //查询 [a,b] 之间最小值的下标
86             int query(int a, int b){
87                 if(a > b) swap(a, b);
88                 int k = mm[b-a+1];
89                 return rmq[dp[a][k]] < rmq[dp[b-(1 << k)+1][k]] ? dp[a][k] : dp[b-(1 << k)+1][k];
90             }
91 };
92 //边的结构体定义
93 struct Edge{
94     int to, next;
95 };
96 Edge edge[MAXN*2];
97 int tot, head[MAXN];
98
99 int F[MAXN*2]; //欧拉序列, 就是 dfs 遍历的顺序, 长度为 2*n-1, 下标从 1 开始
100 int P[MAXN]; //P[i] 表示点 i 在 F 中第一次出现的位置
101 int cnt;
102

```

```

103 ST st;
104 void init(){
105     tot = 0;
106     memset(head,-1,sizeof(head));
107 }
108 //加边, 无向边需要加两次
109 void addedge(int u,int v){
110     edge[tot].to = v;
111     edge[tot].next = head[u];
112     head[u] = tot++;
113 }
114 void dfs(int u,int pre,int dep){
115     F[++cnt] = u;
116     rmq[cnt] = dep;
117     P[u] = cnt;
118     for(int i = head[u]; i != -1; i = edge[i].next){
119         int v = edge[i].to;
120         if(v == pre)continue;
121         dfs(v,u,dep+1);
122         F[++cnt] = u;
123         rmq[cnt] = dep;
124     }
125 }
126 //查询 LCA 前的初始化
127 void LCA_init(int root,int node_num){
128     cnt = 0;
129     dfs(root,root,0);
130     st.init(2*node_num-1);
131 }
132 //查询 u,v 的 lca 编号
133 int query_lca(int u,int v){
134     return F[st.query(P[u],P[v])];
135 }
136 void dfs_build(int u,int pre){
137     int pos = hash(a[u]);
138     T[u] = update(T[pre],pos,1);
139     for(int i = head[u]; i != -1; i = edge[i].next){
140         int v = edge[i].to;
141         if(v == pre)continue;
142         dfs_build(v,u);
143     }
144 }
145 int main(){
146     while(scanf("%d%d",&n,&q) == 2){
147         for(int i = 1; i <= n; i++){
148             scanf("%d",&a[i]);
149             Init_hash();
150             init();
151             TOT = 0;
152             int u,v;
153             for(int i = 1; i < n; i++){

```

```

154         scanf("%d%d",&u,&v);
155         addedge(u,v);
156         addedge(v,u);
157     }
158     LCA_init(1,n);
159     T[n+1] = build(1,m);
160     dfs_build(1,n+1);
161     int k;
162     while(q--){
163         scanf("%d%d%d",&u,&v,&k);
164         printf("%d\n",t[query(T[u],T[v],query_lca(u,v),k)]);
165     }
166     return 0;
167 }
168 return 0;
169 }

```

3.6.4 动态第 k 大

树状数组套主席树 ZOJ 2112

```

1  const int MAXN = 60010;
2  const int M = 2500010;
3  int n,q,m,tot;
4  int a[MAXN], t[MAXN];
5  int T[MAXN], lson[M], rson[M],c[M];
6  int S[MAXN];
7  struct Query{
8      int kind;
9      int l,r,k;
10 }query[10010];
11
12 void Init_hash(int k){
13     sort(t,t+k);
14     m = unique(t,t+k) - t;
15 }
16 int hash(int x){
17     return lower_bound(t,t+m,x)-t;
18 }
19 int build(int l,int r){
20     int root = tot++;
21     c[root] = 0;
22     if(l != r){
23         int mid = (l+r)/2;
24         lson[root] = build(l,mid);
25         rson[root] = build(mid+1,r);
26     }
27     return root;
28 }
29
30 int Insert(int root,int pos,int val){
31     int newroot = tot++, tmp = newroot;

```



```

32     int l = 0, r = m-1;
33     c[newroot] = c[root] + val;
34     while(l < r){
35         int mid = (l+r)>>1;
36         if(pos <= mid){
37             lson[newroot] = tot++; rson[newroot] = rson[root];
38             newroot = lson[newroot]; root = lson[root];
39             r = mid;
40         }
41         else{
42             rson[newroot] = tot++; lson[newroot] = lson[root];
43             newroot = rson[newroot]; root = rson[root];
44             l = mid+1;
45         }
46         c[newroot] = c[root] + val;
47     }
48     return tmp;
49 }
50
51 int lowbit(int x){
52     return x&(-x);
53 }
54 int use[MAXN];
55 void add(int x,int pos,int val){
56     while(x <= n){
57         S[x] = Insert(S[x],pos,val);
58         x += lowbit(x);
59     }
60 }
61 int sum(int x){
62     int ret = 0;
63     while(x > 0){
64         ret += c[lson[use[x]]];
65         x -= lowbit(x);
66     }
67     return ret;
68 }
69 int Query(int left,int right,int k){
70     int left_root = T[left-1];
71     int right_root = T[right];
72     int l = 0, r = m-1;
73     for(int i = left-1;i;i -= lowbit(i)) use[i] = S[i];
74     for(int i = right;i;i -= lowbit(i)) use[i] = S[i];
75     while(l < r){
76         int mid = (l+r)/2;
77         int tmp = sum(right) - sum(left-1) + c[lson[right_root]] -
              c[lson[left_root]];
78         if(tmp >= k){
79             r = mid;
80             for(int i = left-1; i ;i -= lowbit(i))
81                 use[i] = lson[use[i]];

```

```

82         for(int i = right; i; i -= lowbit(i))
83             use[i] = lson[use[i]];
84         left_root = lson[left_root];
85         right_root = lson[right_root];
86     }
87     else{
88         l = mid+1;
89         k -= tmp;
90         for(int i = left-1; i; i -= lowbit(i))
91             use[i] = rson[use[i]];
92         for(int i = right; i; i -= lowbit(i))
93             use[i] = rson[use[i]];
94         left_root = rson[left_root];
95         right_root = rson[right_root];
96     }
97 }
98 return l;
99 }
100 void Modify(int x,int p,int d){
101     while(x <= n){
102         S[x] = Insert(S[x],p,d);
103         x += lowbit(x);
104     }
105 }
106
107 int main(){
108     int Tcase;
109     scanf("%d",&Tcase);
110     while(Tcase--){
111         scanf("%d%d",&n,&q);
112         tot = 0;
113         m = 0;
114         for(int i = 1; i <= n; i++){
115             scanf("%d",&a[i]);
116             t[m++] = a[i];
117         }
118         char op[10];
119         for(int i = 0; i < q; i++){
120             scanf("%s",op);
121             if(op[0] == 'Q'){
122                 query[i].kind = 0;
123                 scanf("%d%d%d",&query[i].l,&query[i].r,&query[i].k)
124                     ;
125             }
126             else{
127                 query[i].kind = 1;
128                 scanf("%d%d",&query[i].l,&query[i].r);
129                 t[m++] = query[i].r;
130             }
131         }
132         Init_hash(m);

```

```

132     T[0] = build(0,m-1);
133     for(int i = 1;i <= n;i++)
134         T[i] = Insert(T[i-1],hash(a[i]),1);
135     for(int i = 1;i <= n;i++)
136         S[i] = T[0];
137     for(int i = 0;i < q;i++){
138         if(query[i].kind == 0)
139             printf("%d\n",t[Query(query[i].l,query[i].r,query[i]
140                                     ].k)]);
141         else{
142             Modify(query[i].l,hash(a[query[i].l]),-1);
143             Modify(query[i].l,hash(query[i].r),1);
144             a[query[i].l] = query[i].r;
145         }
146     }
147     return 0;
148 }

```

3.7 Treap

ZOJ3765

```

1 long long gcd(long long a,long long b){
2     if(b == 0)return a;
3     else return gcd(b,a%b);
4 }
5 const int MAXN = 300010;
6 int num[MAXN],st[MAXN];
7 struct Treap{
8     int tot1;
9     int s[MAXN],tot2;//内存池和容量
10    int ch[MAXN][2];
11    int key[MAXN],size[MAXN];
12    int sum0[MAXN],sum1[MAXN];
13    int status[MAXN];
14    void Init(){
15        tot1 = tot2 = 0;
16        size[0] = 0;
17        ch[0][0] = ch[0][1] = 0;
18        sum0[0] = sum1[0] = 0;
19    }
20    bool random(double p){
21        return (double)rand() / RAND_MAX < p;
22    }
23    int newnode(int val,int _status){
24        int r;
25        if(tot2)r = s[tot2--];
26        else r = ++tot1;
27        size[r] = 1;
28        key[r] = val;
29        status[r] = _status;

```

```

30     ch[r][0] = ch[r][1] = 0;
31     sum0[r] = sum1[r] = 0; //需要push_up
32     return r;
33 }
34 void del(int r){
35     if(!r) return;
36     s[++tot2] = r;
37     del(ch[r][0]);
38     del(ch[r][1]);
39 }
40 void push_up(int r){
41     int lson = ch[r][0], rson = ch[r][1];
42     size[r] = size[lson] + size[rson] + 1;
43     sum0[r] = gcd(sum0[lson], sum0[rson]);
44     sum1[r] = gcd(sum1[lson], sum1[rson]);
45     if(status[r] == 0)
46         sum0[r] = gcd(sum0[r], key[r]);
47     else sum1[r] = gcd(sum1[r], key[r]);
48 }
49 void merge(int &p, int x, int y){
50     if(!x || !y)
51         p = x | y;
52     else if(random((double) size[x] / (size[x] + size[y]))) {
53         merge(ch[x][1], ch[x][1], y);
54         push_up(p=x);
55     }
56     else{
57         merge(ch[y][0], x, ch[y][0]);
58         push_up(p=y);
59     }
60 }
61 void split(int p, int &x, int &y, int k){
62     if(!k){
63         x = 0; y = p;
64         return;
65     }
66     if(size[ch[p][0]] >= k){
67         y = p;
68         split(ch[p][0], x, ch[y][0], k);
69         push_up(y);
70     }
71     else{
72         x = p;
73         split(ch[p][1], ch[x][1], y, k - size[ch[p][0]] - 1);
74         push_up(x);
75     }
76 }
77 void build(int &p, int l, int r){
78     if(l > r) return;
79     int mid = (l + r) / 2;
80     p = newnode(num[mid], st[mid]);

```

```

81     build(ch[p][0],l,mid-1);
82     build(ch[p][1],mid+1,r);
83     push_up(p);
84 }
85 void debug(int root){
86     if(root == 0) return;
87     printf("%d左儿子: %d右儿子: %dsize=%dkey=%d\n",
88         root,ch[root][0],ch[root][1],size[root],key[root]);
89     debug(ch[root][0]);
90     debug(ch[root][1]);
91 }
92 Treap T;
93 char op[10];
94 int main(){
95     int n,q;
96     while(scanf("%d%d",&n,&q) == 2){
97         int root = 0;
98         T.Init();
99         for(int i = 1;i <= n;i++){
100             scanf("%d%d",&num[i],&st[i]);
101             T.build(root,1,n);
102             while(q--){
103                 scanf("%s",op);
104                 if(op[0] == 'Q'){
105                     int l,r,s;
106                     scanf("%d%d%d",&l,&r,&s);
107                     int x,y,z;
108                     T.split(root,x,z,r);
109                     T.split(x,x,y,l-1);
110                     if(s == 0)
111                         printf("%d\n",T.sum0[y] == 0? -1:T.sum0[y]);
112                     else
113                         printf("%d\n",T.sum1[y] == 0?-1:T.sum1[y]);
114                     T.merge(x,x,y);
115                     T.merge(root,x,z);
116                 }
117                 else if(op[0] == 'I'){
118                     int v,s,loc;
119                     scanf("%d%d%d",&loc,&v,&s);
120                     int x,y;
121                     T.split(root,x,y,loc);
122                     T.merge(x,x,T.newnode(v,s));
123                     T.merge(root,x,y);
124                 }
125                 else if(op[0] == 'D'){
126                     int loc;
127                     scanf("%d",&loc);
128                     int x,y,z;
129                     T.split(root,x,z,loc);
130                     T.split(x,x,y,loc-1);

```

```

131         T.del(y);
132         T.merge(root,x,z);
133     }
134     else if(op[0] == 'R'){
135         int loc;
136         scanf("%d",&loc);
137         int x,y,z;
138         T.split(root,x,z,loc);
139         T.split(x,x,y,loc-1);
140         T.status[y] = 1-T.status[y];
141         T.push_up(y);
142         T.merge(x,x,y);
143         T.merge(root,x,z);
144     }
145     else{
146         int loc,v;
147         scanf("%d%d",&loc,&v);
148         int x,y,z;
149         T.split(root,x,z,loc);
150         T.split(x,x,y,loc-1);
151         T.key[y] = v;
152         T.push_up(y);
153         T.merge(x,x,y);
154         T.merge(root,x,z);
155     }
156 }
157 }
158 return 0;
159 }

```

3.8 KD 树

3.8.1 HDU4347 K 近邻

模板题，求出最近的 K 个点。

```

1  const int MAXN = 50010;
2  const int DIM = 10;
3  inline double sqr(double x){return x*x;}
4  namespace KDTree{
5      int K;//维数
6      struct Point{
7          int x[DIM];
8          double distance(const Point &b)const{
9              double ret = 0;
10             for(int i = 0;i < K;i++)
11                 ret += sqr(x[i]-b.x[i]);
12             return ret;
13         }
14         void input(){
15             for(int i = 0;i < K;i++)scanf("%d",&x[i]);
16         }

```

```

17     void output(){
18         for(int i = 0;i < K;i++)
19             printf("%d%c",x[i],i < K-1?' ':'\n');
20     }
21 };
22 struct qnode{
23     Point p;
24     double dis;
25     qnode(){}
26     qnode(Point _p,double _dis){
27         p = _p; dis = _dis;
28     }
29     bool operator <(const qnode &b)const{
30         return dis < b.dis;
31     }
32 };
33 priority_queue<qnode>q;
34 struct cmpx{
35     int div;
36     cmpx(const int &_div){div = _div;}
37     bool operator()(const Point &a,const Point &b){
38         for(int i = 0;i < K;i++)
39             if(a.x[(div+i)%K] != b.x[(div+i)%K])
40                 return a.x[(div+i)%K] < b.x[(div+i)%K];
41         return true;
42     }
43 };
44 bool cmp(const Point &a,const Point &b,int div){
45     cmpx cp = cmpx(div);
46     return cp(a,b);
47 }
48 struct Node{
49     Point e;
50     Node *lc,*rc;
51     int div;
52 }pool[MAXN],*tail,*root;
53 void init(){
54     tail = pool;
55 }
56 Node* build(Point *a,int l,int r,int div){
57     if(l >= r)return NULL;
58     Node *p = tail++;
59     p->div = div;
60     int mid = (l+r)/2;
61     nth_element(a+l,a+mid,a+r,cmpx(div));
62     p->e = a[mid];
63     p->lc = build(a,l,mid,(div+1)%K);
64     p->rc = build(a,mid+1,r,(div+1)%K);
65     return p;
66 }
67 void search(Point p,Node *x,int div,int m){

```

```

68         if(!x) return;
69         if(cmp(p,x->e,div)){
70             search(p,x->lc,(div+1)%K,m);
71             if(q.size() < m){
72                 q.push(qnode(x->e,p.distance(x->e)));
73                 search(p,x->rc,(div+1)%K,m);
74             }
75             else {
76                 if(p.distance(x->e) < q.top().dis){
77                     q.pop();
78                     q.push(qnode(x->e,p.distance(x->e)));
79                 }
80                 if(sqr(x->e.x[div]-p.x[div]) < q.top().dis)
81                     search(p,x->rc,(div+1)%K,m);
82             }
83         }
84         else {
85             search(p,x->rc,(div+1)%K,m);
86             if(q.size() < m){
87                 q.push(qnode(x->e,p.distance(x->e)));
88                 search(p,x->lc,(div+1)%K,m);
89             }
90             else {
91                 if(p.distance(x->e) < q.top().dis){
92                     q.pop();
93                     q.push(qnode(x->e,p.distance(x->e)));
94                 }
95                 if(sqr(x->e.x[div]-p.x[div]) < q.top().dis)
96                     search(p,x->lc,(div+1)%K,m);
97             }
98         }
99     }
100     void search(Point p,int m){
101         while(!q.empty())q.pop();
102         search(p,root,0,m);
103     }
104 };
105 KDTree::Point p[MAXN];
106 int main()
107 {
108     int n,k;
109     while(scanf("%d%d",&n,&k) == 2){
110         KDTree::K = k;
111         for(int i = 0;i < n;i++)p[i].input();
112         KDTree::init();
113         KDTree::root = KDTree::build(p,0,n,0);
114         int Q;
115         scanf("%d",&Q);
116         KDTree::Point o;
117         while(Q--){
118             o.input();

```



```

119         int m;
120         scanf("%d",&m);
121         KDTree::search(o,m);
122         printf("the closest %d points are:\n",m);
123         int cnt = 0;
124         while(!KDTree::q.empty()){
125             p[cnt++] = KDTree::q.top().p;
126             KDTree::q.pop();
127         }
128         for(int i = 0;i < m;i++)p[m-1-i].output();
129     }
130 }
131 return 0;
132 }

```

3.8.2 CF44G

给定若干个靶子 (xl, xr, yl, yr, z) , z 为该靶子离射击位置的距离, 所有靶子都可以看成是二维平面上平行于坐标轴的矩形。然后按顺序给定若干子弹的射击位置 (x, y) , 子弹射到一个靶子就会将靶子打碎, 并掉落到地上。问每个子弹射到的靶子是谁。保证靶子的 z 值不相同。

找矩形内权值最小的点, 支持删除操作

```

1  const int MAXN = 100010;
2  const int INF = 0x3f3f3f3f;
3  struct Point{
4      int x,y,id;
5      bool operator ==(const Point &b)const{
6          return x == b.x && y == b.y && id == b.id;
7      }
8  };
9  struct Node{
10     Point e;
11     Node *lc,*rc;
12     bool div;
13     int sub,cur;
14     int size;
15     bool exist;
16     void push_up(){
17         size = lc->size + rc->size + exist;
18         sub = min(cur,min(lc->sub,rc->sub));
19     }
20 }pool[MAXN],*root,*tail,*null;
21 inline bool cmpX(const Point &a,const Point &b){return a.x < b.x ||
    (a.x == b.x && a.y < b.y) || (a.x == b.x && a.y == b.y && a.id
    < b.id);}
22 inline bool cmpY(const Point &a,const Point &b){return a.y < b.y ||
    (a.y == b.y && a.x < b.x) || (a.y == b.y && a.x == b.x && a.id
    < b.id);}
23 inline bool cmp(const Point &a,const Point &b,bool div){return div?
    cmpY(a,b):cmpX(a,b);}
24 Node* build(Point *a,int l,int r,bool div){

```

```

25     if(l >= r)return null;
26     Node *p = tail++;
27     p->div = div;
28     int mid = (l+r)/2;
29     nth_element(a+l,a+mid,a+r,div?cmpY:cmpX);
30     p->e = a[mid];
31     p->lc = build(a,l,mid,!div);
32     p->rc = build(a,mid+1,r,!div);
33     p->exist = 1;
34     p->cur = p->e.id;
35     p->push_up();
36     return p;
37 }
38 void remove(Node *p,Point o){
39     if(p->e == o){
40         p->exist = 0;
41         p->cur = INF;
42         p->size--;
43     }
44     else {
45         if(cmp(p->e,o,p->div))remove(p->rc,o);
46         else remove(p->lc,o);
47     }
48     p->push_up();
49 }
50 int getMin(Node *p,int xl,int xr,int yl,int yr,int minx,int maxx,
51 int miny,int maxy){
52     if(p == null || p->size == 0)return INF;
53     if(xl <= minx && xr >= maxx && yl <= miny && yr >= maxy)return
54         p->sub;
55     if(xl > maxx || xr < minx || yl > maxy || yr < miny)return INF;
56     int ret = INF;
57     if(p->e.x >= xl && p->e.x <= xr && p->e.y >= yl && p->e.y <= yr
58         )
59         ret = min(ret,p->cur);
60     if(p->div){
61         if(yl <= p->e.y)
62             ret = min(ret,getMin(p->lc,xl,xr,yl,min(yr,p->e.y),minx
63                 ,maxx,miny,min(maxy,p->e.y)));
64         if(yr >= p->e.y)
65             ret = min(ret,getMin(p->rc,xl,xr,max(yl,p->e.y),yr,minx
66                 ,maxx,max(miny,p->e.y),maxy));
67     }
68     else {
69         if(xl <= p->e.x)
70             ret = min(ret,getMin(p->lc,xl,min(xr,p->e.x),yl,yr,minx
71                 ,min(maxx,p->e.x),miny,maxy));
72         if(xr >= p->e.x)
73             ret = min(ret,getMin(p->rc,max(xl,p->e.x),xr,yl,yr,max(
74                 minx,p->e.x),maxx,miny,maxy));
75     }
76 }

```

```

69     return ret;
70 }
71 Point pp[MAXN],pp2[MAXN];
72 struct REC{
73     int xl,xr,yl,yr,z;
74     int id;
75     void input(){
76         scanf("%d%d%d%d%d",&xl,&xr,&yl,&yr,&z);
77     }
78     bool operator <(const REC &b)const{
79         return z < b.z;
80     }
81 }rec[MAXN];
82 int ans[MAXN];
83 int main()
84 {
85     int n,m;
86     while(scanf("%d",&n) == 1){
87         for(int i = 0;i < n;i++){
88             rec[i].input();
89             rec[i].id = i+1;
90         }
91         sort(rec,rec+n);
92         scanf("%d",&m);
93         for(int i = 0;i < m;i++){
94             scanf("%d%d",&pp[i].x,&pp[i].y);
95             pp[i].id = i;
96             pp2[i] = pp[i]; //备份
97         }
98         tail = pool;
99         null = tail++;
100        null->size = 0;
101        null->sub = null->cur = INF;
102        null->lc = null->rc = null;
103        root = build(pp,0,m,0);
104        memset(ans,0,sizeof(ans));
105        for(int i = 0;i < n;i++){
106            int tmp = getMin(root,rec[i].xl,rec[i].xr,rec[i].yl,rec
                [i].yr,-INF,INF,-INF,INF);
107            if(tmp == INF)continue;
108            ans[tmp] = rec[i].id;
109            remove(root,pp2[tmp]);
110        }
111        for(int i = 0;i < m;i++)printf("%d\n",ans[i]);
112    }
113    return 0;
114 }

```

3.8.3 HDU4742

三维 LIS。即每个点有个三维坐标，两个点能放在一前一后当且仅当 $x_i < x_j, y_i < y_j, z_i < z_j$ ，求最长的序列，并该条件下的方案数。

```

1  const int MAXN = 100010;
2  const int MOD = 1<<30;
3  const int INF = 0x7fffffff;//这个一定要够大
4  struct Node{
5      pair<int,int>e,sub,cur;
6      bool div;
7      Node *lc,*rc;
8  };
9  Node pool[MAXN],*tail;
10 Node *root;
11 bool cmpX(const pair<int,int> &a,const pair<int,int> &b){return a.
    first < b.first || (a.first == b.first && a.second < b.second)
    };
12 bool cmpY(const pair<int,int> &a,const pair<int,int> &b){return a.
    second < b.second || (a.second == b.second && a.first < b.first)
    };
13 bool cmp(const pair<int,int> &a,const pair<int,int> &b,bool div){
    return div?cmpY(a,b):cmpX(a,b);}
14 Node* build(pair<int,int> *a,int l,int r,bool div){
15     if(l >= r)return NULL;
16     Node *p = tail++;
17     p->div = div;
18     int mid = (l+r)/2;
19     nth_element(a+l,a+mid,a+r,div?cmpY:cmpX);
20     p->e = a[mid];
21     p->cur = p->sub = make_pair(0,0);
22     p->lc = build(a,l,mid,!div);
23     p->rc = build(a,mid+1,r,!div);
24     return p;
25 }
26 inline void update(pair<int,int> &a,pair<int,int> b){
27     if(a.first < b.first)a = b;
28     else if(a.first == b.first){
29         a.second += b.second;
30         if(a.second >= MOD)a.second -= MOD;
31     }
32 }
33 void add(Node *p,pair<int,int> e,pair<int,int> v){
34     update(p->sub,v);
35     if(e == p->e){
36         update(p->cur,v);
37         return;
38     }
39     else {
40         if(cmp(p->e,e,p->div))add(p->rc,e,v);
41         else add(p->lc,e,v);
42     }

```

```

43 }
44 pair<int,int>ans;
45 //查询最大值
46 void get(Node *p,pair<int,int>e,int maxx,int maxy){
47     if(!p)return;
48     if(p->sub.first < ans.first)return;
49     if(maxx <= e.first && maxy <= e.second)
50         update(ans,p->sub);
51     else {
52         if(p->e.first <= e.first && p->e.second <= e.second)update(
53             ans,p->cur);
54         if(p->div){
55             if(p->e.second <= e.second)get(p->rc,e,maxx,maxy);
56             get(p->lc,e,maxx,min(maxy,p->e.second));
57         }
58         else {
59             if(p->e.first <= e.first)get(p->rc,e,maxx,maxy);
60             get(p->lc,e,min(maxx,p->e.first),maxy);
61         }
62     }
63 struct TNode{
64     int x,y,z;
65     void input(){
66         scanf("%d%d%d",&x,&y,&z);
67     }
68     bool operator < (const TNode &b)const{
69         if(x != b.x)return x < b.x;
70         else if(y != b.y)return y < b.y;
71         else return z < b.z;
72     }
73 }node[MAXN];
74 pair<int,int>p[MAXN];
75 pair<int,int>dp[MAXN];
76 int main()
77 {
78     int T;
79     int n;
80     scanf("%d",&T);
81     while(T--){
82         scanf("%d",&n);
83         int cnt = 0;
84         for(int i = 0;i < n;i++){
85             node[i].input();
86             p[cnt++] = make_pair(node[i].y,node[i].z);
87         }
88         sort(node,node+n);
89         sort(p,p+cnt);
90         cnt = unique(p,p+cnt)-p;
91         tail = pool;
92         root = build(p,0,cnt,0);

```

```

93     for(int i = 0;i < n;i++)dp[i] = make_pair(1,1);
94     for(int i = 0;i < n;i++){
95         ans = make_pair(0,0);
96         get(root,make_pair(node[i].y,node[i].z),INF,INF);
97         ans.first++;
98         update(dp[i],ans);
99         add(root,make_pair(node[i].y,node[i].z),dp[i]);
100    }
101    printf("%d_%d\n",root->sub.first,root->sub.second);
102 }
103 return 0;
104 }

```

3.9 替罪羊树 (ScapeGoat Tree)

3.9.1 CF455D

<http://codeforces.com/contest/455/problem/D>

题意：给了一个序列，1 操作把一个区间的末尾的数插入到头部，2 操作是询问一个区间里面等于某个数的个数。

使用替罪羊树，里面套一个 map 来统计区间的个数。

```

1  const int MAXN = 200010;
2  const double alpha = 0.75;
3  struct Node{
4      Node *ch[2];
5      int size,key,nodeCount;
6      bool exist;
7      map<int,int>mp;
8      bool isBad(){
9          return ch[0]->nodeCount > alpha*nodeCount+5 || ch[1]->
              nodeCount > alpha*nodeCount + 5;
10     }
11     void push_up(){
12         size = exist + ch[0]->size + ch[1]->size;
13         nodeCount = 1 + ch[0]->nodeCount + ch[1]->nodeCount;
14         mp.clear();
15         if(exist)mp[key]++;
16         for(map<int,int>::iterator it = ch[0]->mp.begin();it != ch
            [0]->mp.end();it++)
17             mp[(*it).first] += (*it).second;
18         for(map<int,int>::iterator it = ch[1]->mp.begin();it != ch
            [1]->mp.end();it++)
19             mp[(*it).first] += (*it).second;
20     }
21 };
22 struct ScapeGoatTree{
23     Node pool[MAXN];
24     Node *tail,*root,*null;
25     Node *bc[MAXN]; //内存回收
26     int bc_top;
27     void init(){

```

```

28     tail = pool;
29     null = tail++;
30     null->ch[0] = null->ch[1] = null;
31     null->size = null->key = null->nodeCount = 0;
32     null->mp.clear();
33     root = null;
34     bc_top = 0;
35 }
36 inline Node *newNode(int key){
37     Node *p;
38     if(bc_top)p = bc[--bc_top];
39     else p = tail++;
40     p->ch[0] = p->ch[1] = null;
41     p->size = p->nodeCount = 1;
42     p->key = key;
43     p->exist = true;
44     p->mp.clear();
45     p->mp[key] = 1;
46     return p;
47 }
48 Node *buildTree(int *a,int l,int r){
49     if(l >= r)return null;
50     int mid = (l+r)>>1;
51     Node *p = newNode(a[mid]);
52     p->ch[0] = buildTree(a,l,mid);
53     p->ch[1] = buildTree(a,mid+1,r);
54     p->push_up();
55     return p;
56 }
57 inline void Travel(Node *p,vector<Node *>&v){
58     if(p == null)return;
59     Travel(p->ch[0],v);
60     if(p->exist)v.push_back(p);
61     else bc[bc_top++] = p;
62     Travel(p->ch[1],v);
63 }
64 inline Node *divide(vector<Node *>&v,int l,int r){
65     if(l >= r)return null;
66     int mid = (l+r)/2;
67     Node *p = v[mid];
68     p->ch[0] = divide(v,l,mid);
69     p->ch[1] = divide(v,mid+1,r);
70     p->push_up();
71     return p;
72 }
73 //重构, 注意 p 要引用
74 inline void rebuild(Node *&p){
75     vector<Node *>v;
76     Travel(p,v);
77     p = divide(v,0,v.size());
78 }

```

```

79 //删除第 id 个元素, 返回第 id 个元素的值
80 inline int erase(Node *p,int id){
81     if(p->exist && id == p->ch[0]->size + 1){
82         p->exist = 0;
83         p->mp[p->key]--;
84         p->size--;
85         return p->key;
86     }
87     p->size--;
88     int res;
89     if(p->ch[0]->size >= id)
90         res = erase(p->ch[0],id);
91     else res = erase(p->ch[1],id - p->ch[0]->size - p->exist);
92     p->mp[res]--;
93     return res;
94 }
95 //删除一定的点以后重构
96 void check_erase(){
97     if(root->size < 0.5*root->nodeCount)
98         rebuild(root);
99 }
100 Node **insert(Node *&p,int id,int val){
101     if(p == null){
102         p = newNode(val);
103         return &null;
104     }
105     else {
106         p->size++;
107         p->nodeCount++;
108         p->mp[val]++;
109         Node ** res;
110         if(id <= p->ch[0]->size+p->exist)
111             res = insert(p->ch[0],id,val);
112         else res = insert(p->ch[1],id-p->ch[0]->size-p->exist,
113             val);
114         if(p->isBad())res = &p;
115         return res;
116     }
117 }
118 //在第 id 个位置插入数 val
119 void insert(int id,int val){
120     Node **p = insert(root,id,val);
121     if(*p != null)rebuild(*p);
122 }
123 //查询 [l,r] 之间值为 val 的数的个数
124 int query(Node *p,int l,int r,int val){
125     if(p == null)return 0;
126     if(l <= 1 && p->size <= r)
127         return p->mp.count(val)?p->mp[val]:0;
128     else {
129         int ans = 0;

```



```

129         if(l <= p->ch[0]->size)
130             ans += query(p->ch[0],l,r,val);
131         if(r > p->ch[0]->size+p->exist)
132             ans += query(p->ch[1],l - p->ch[0]->size - p->exist
133                 , r - p->ch[0]->size - p->exist,val);
134         if(p->exist && p->key == val && l <= p->ch[0]->size+1
135             && r >= p->ch[0]->size+1)
136             ans++;
137         return ans;
138     }
139 }tree;
140 int a[MAXN];
141 int main()
142 {
143     int n;
144     while(scanf("%d",&n) == 1){
145         tree.init();
146         for(int i = 0;i < n;i++)scanf("%d",&a[i]);
147         tree.root = tree.buildTree(a,0,n);
148         int m;
149         int op,l,r,k;
150         scanf("%d",&m);
151         int ans = 0;
152         while(m--){
153             scanf("%d",&op);
154             if(op == 1){
155                 scanf("%d%d",&l,&r);
156                 l = ((l+ans-1)%n)+1;
157                 r = ((r+ans-1)%n)+1;
158                 if(l > r)swap(l,r);
159                 int v = tree.erase(tree.root,r);
160                 //tree.check_erase(); //有时候可以加上删除重构
161                 tree.insert(l,v);
162             }
163             else {
164                 scanf("%d%d%d",&l,&r,&k);
165                 l = ((l+ans-1)%n)+1;
166                 r = ((r+ans-1)%n)+1;
167                 k = ((k+ans-1)%n)+1;
168                 if(l > r)swap(l,r);
169                 ans = tree.query(tree.root,l,r,k);
170                 printf("%d\n",ans);
171             }
172         }
173     }
174     return 0;
175 }

```

3.10 动态 KD 树

动态 KD 树就是结合了 KD 树和替罪羊树。支持 KD 树的插入删除操作，用替罪羊树的思想来保存平衡。

UVALive6045

题意：给了二维平面上的 N 个整点 ($N \leq 50000$)。每次操作给了点 (x_i, y_i) ，需要曼哈顿距离小于 E 的点进行一个变换。输出最后的点的坐标，保证变换次数不超过 50000。

```

1  const int MAXN = 100010;
2  const double alpha = 0.75;
3  struct Point{
4      int x,y,id;
5  };
6  struct Node{
7      Point e;
8      int size,nodeCount;
9      Node *lc,*rc;
10     bool div;
11     bool exist;
12     bool isBad(){
13         return lc->nodeCount > alpha*nodeCount+5 || rc->nodeCount >
            alpha*nodeCount+5;
14     }
15     inline void push_up(){
16         size = exist + lc->size + rc->size;
17         nodeCount = 1+lc->nodeCount+rc->nodeCount;
18     }
19 };
20 Node pool[MAXN],*tail,*root,*null;
21 Node *bc[MAXN];
22 int bc_top;
23 void init(){
24     tail = pool;
25     null = tail++;
26     null->lc = null->rc = null;
27     null->size = null->nodeCount = 0;
28     root = null;
29     bc_top = 0;
30 }
31 Node *newNode(Point e){
32     Node *p;
33     if(bc_top)p = bc[--bc_top];
34     else p = tail++;
35     p->e = e;
36     p->lc = p->rc = null;
37     p->size = p->nodeCount = 1;
38     p->exist = true;
39     return p;
40 }
41 inline bool cmpX(const Point &a,const Point &b){
42     return a.x < b.x || (a.x == b.x && a.y < b.y) || (a.x == b.x &&
        a.y == b.y && a.id < b.id);

```

```

43 }
44 inline bool cmpY(const Point &a,const Point &b){
45     return a.y < b.y || (a.y == b.y && a.x < b.x) || (a.y == b.y &&
        a.x == b.x && a.id < b.id);
46 }
47 inline bool cmp(const Point &a,const Point &b,bool div){
48     return div?cmpY(a,b):cmpX(a,b);
49 }
50 //注意 a 需要备份, 否则就乱序
51 Node *build(Point *a,int l,int r,bool div){
52     if(l >= r)return null;
53     int mid = (l+r)/2;
54     nth_element(a+l,a+mid,a+r,div?cmpY:cmpX);
55     Node *p = newNode(a[mid]);
56     p->div = div;
57     p->lc = build(a,l,mid,!div);
58     p->rc = build(a,mid+1,r,!div);
59     p->push_up();
60     return p;
61 }
62 void Travel(Node *p,vector<Point>&v){
63     if(p == null)return;
64     Travel(p->lc,v);
65     if(p->exist)v.push_back(p->e);
66     bc[bc_top++] = p;
67     Travel(p->rc,v);
68 }
69 Node *divide(vector<Point>&v,int l,int r,bool div){
70     if(l >= r)return null;
71     int mid = (l+r)/2;
72     nth_element(v.begin()+l,v.begin()+mid,v.begin()+r,div?cmpY:cmpX);
73     Node *p = newNode(v[mid]);
74     p->div = div;
75     p->lc = divide(v,l,mid,!div);
76     p->rc = divide(v,mid+1,r,!div);
77     p->push_up();
78     return p;
79 }
80 inline void rebuild(Node *&p){
81     vector<Point>v;
82     Travel(p,v);
83     p = divide(v,0,v.size(),p->div);
84 }
85 Node **insert(Node *&p,Point a,bool div){
86     if(p == null){
87         p = newNode(a);
88         p->div = div;
89         return &null;
90     }
91     else {

```

```

92         p->nodeCount++;
93         p->size++;
94         Node **res;
95         if(cmp(a,p->e,div))
96             res = insert(p->lc,a,!div);
97         else res = insert(p->rc,a,!div);
98         if(p->isBad())res = &p;
99         return res;
100     }
101 }
102 void insert(Point e){
103     Node **p = insert(root,e,0);
104     if(*p != null)rebuild(*p);
105 }
106 vector<int>vec;
107 void getvec(Node *p,int minx,int maxx,int miny,int maxy){
108     if(p->size == 0)return;
109     if(p->exist && minx <= p->e.x && p->e.x <= maxx && miny <= p->e
        .y && p->e.y <= maxy){
110         vec.push_back(p->e.id);
111         p->exist = 0;
112         p->size--;
113     }
114     if(p->div? miny <= p->e.y : minx <= p->e.x)getvec(p->lc,minx,
        maxx,miny,maxy);
115     if(p->div? maxy >= p->e.y : maxx >= p->e.x)getvec(p->rc,minx,
        maxx,miny,maxy);
116     p->push_up();
117 }
118 Point p[MAXN],p2[MAXN];
119 Point p3[MAXN];
120 int main()
121 {
122     int T;
123     scanf("%d",&T);
124     int iCase = 0;
125     int N,Q,W,H;
126     while(T--){
127         iCase++;
128         scanf("%d%d%d%d",&N,&Q,&W,&H);
129         init();
130         for(int i = 0;i < N;i++){
131             scanf("%d%d",&p[i].x,&p[i].y);
132             p[i].id = p2[i].id = i;
133             p2[i].x = p[i].x+p[i].y;
134             p2[i].y = p[i].x-p[i].y;
135             p3[i] = p2[i];
136         }
137         root = build(p3,0,N,0);
138         int X,Y,E,a,b,c,d,e,f;
139         while(Q--){

```

```

140         scanf("%d%d%d%d%d%d%d%d", &X, &Y, &E, &a, &b, &c, &d, &e, &f);
141         vec.clear();
142         int minx = X+Y-E;
143         int maxx = X+Y+E;
144         int miny = X-Y-E;
145         int maxy = X-Y+E;
146         getvec(root, minx, maxx, miny, maxy);
147         int sz = vec.size();
148         for(int i = 0; i < sz; i++){
149             int id = vec[i];
150             long long tx = p[id].x;
151             long long ty = p[id].y;
152             p[id].x = (tx*a+ty*b+(long long)(id+1)*c)%W;
153             p[id].y = (tx*d+ty*e+(long long)(id+1)*f)%H;
154             p2[id].x = p[id].x+p[id].y;
155             p2[id].y = p[id].x-p[id].y;
156             insert(p2[id]);
157         }
158     }
159     printf("Case_#%d:\n", iCase);
160     for(int i = 0; i < N; i++)
161         printf("%d_ %d\n", p[i].x, p[i].y);
162 }
163 return 0;
164 }

```

3.11 树套树

3.11.1 替罪羊树套 splay

BZOJ 3065: 带插入区间 K 小值

带插入、修改的区间 k 小值在线查询。

1. Q x y k: 询问从左至右第 x 只跳蚤到从左至右第 y 只跳蚤中，弹跳力第 k 小的跳蚤的弹跳力是多少。 ($1 \leq x \leq y \leq m, 1 \leq k \leq y - x + 1$)
2. M x val: 将从左至右第 x 只跳蚤的弹跳力改为 val。 ($1 \leq x \leq m$)
3. I x val: 在从左至右第 x 只跳蚤的前面插入一只弹跳力为 val 的跳蚤。即插入后从左至右第 x 只跳蚤是我刚插入的跳蚤。 ($1 \leq x \leq m + 1$)

```

1  const int MAXN = 70010;
2  namespace Splay{
3      struct Node *null;
4      struct Node{
5          Node *ch[2], *fa;
6          int size, key, cnt;
7          inline void setc(Node *p, int d){
8              ch[d] = p;
9              p->fa = this;
10         }
11         inline bool d(){
12             return fa->ch[1] == this;
13         }
14         inline void push_up(){

```

```

15         size = ch[0]→size + ch[1]→size + cnt;
16     }
17     void clear(int _key){
18         size = cnt = 1;
19         key = _key;
20         ch[0] = ch[1] = fa = null;
21     }
22     inline bool isroot(){
23         return fa == null || this != fa→ch[0] && this != fa→
            ch[1];
24     }
25 };
26 Node pool[MAXN*20],*tail;
27 Node *bc[MAXN*20];
28 int bc_top;//内存回收
29 void init(){
30     tail = pool;
31     bc_top = 0;
32     null = tail++;
33     null→size = null→cnt = 0;
34     null→ch[0] = null→ch[1] = null→fa = null;
35 }
36 inline void rotate(Node *x){
37     Node *f = x→fa, *ff = x→fa→fa;
38     int c = x→d(), cc = f→d();
39     f→setc(x→ch[!c],c);
40     x→setc(f,!c);
41     if(ff→ch[cc] == f)ff→setc(x,cc);
42     else x→fa = ff;
43     f→push_up();
44 }
45 inline void splay(Node* &root,Node* x,Node* goal){
46     while(x→fa != goal){
47         if(x→fa→fa == goal)rotate(x);
48         else {
49             bool f = x→fa→d();
50             x→d() == f ? rotate(x→fa) : rotate(x);
51             rotate(x);
52         }
53     }
54     x→push_up();
55     if(goal == null)root = x;
56 }
57 //找到 r 子树里面的最左边那个
58 Node* get_left(Node* r){
59     Node* x = r;
60     while(x→ch[0] != null)x = x→ch[0];
61     return x;
62 }
63 //在 root 的树中删掉 x
64 void erase(Node* &root,Node* x){

```

```

65     splay(root,x,null);
66     Node* t = root;
67     if(t->ch[1] != null){
68         root = t->ch[1];
69         splay(root,get_left(t->ch[1]),null);
70         root->setc(t->ch[0],0);
71     }
72     else root = root->ch[0];
73     bc[bc_top++] = x;
74     root->fa = null;
75     if(root != null)root->push_up();
76 }
77 Node* newNode(int key){
78     Node* p;
79     if(bc_top)p = bc[--bc_top];
80     else p = tail++;
81     p->clear(key);
82     return p;
83 }
84 //插入一个值 key
85 void insert(Node* &root,int key){
86     if(root == null){
87         root = newNode(key);
88         return;
89     }
90     Node* now = root;
91     Node* pre = root->fa;
92     while(now != null){
93         if(now->key == key){
94             now->cnt++;
95             splay(root,now,null);
96             return;
97         }
98         pre = now;
99         now = now->ch[key >= now->key];
100     }
101     Node *x = newNode(key);
102     pre->setc(x,key >= pre->key);
103     splay(root,x,null);
104 }
105 //删除一个值 key
106 void erase(Node* &root,int key){
107     Node* now = root;
108     while(now->key != key){
109         now = now->ch[key >= now->key];
110     }
111     now->cnt--;
112     if(now->cnt == 0)erase(root,now);
113     else splay(root,now,null);
114 }
115 void Travel(Node* r){

```

```

116         if(r == null)return;
117         Travel(r->ch[0]);
118         bc[bc_top++] = r;
119         Travel(r->ch[1]);
120     }
121     void CLEAR(Node* &root){
122         Travel(root);
123         root = null;
124     }
125     //查询小于等于 val 的个数
126     int query(Node *root,int val){
127         int ans = 0;
128         Node *x = root;
129         while(x != null){
130             if(val < x->key)x = x->ch[0];
131             else{
132                 ans += x->ch[0]->size + x->cnt;
133                 x = x->ch[1];
134             }
135         }
136         return ans;
137     }
138 };
139 namespace ScapeGoatTree{
140     const double alpha = 0.75;
141     struct Node{
142         Node *ch[2];
143         int size,nodeCount,key;
144         Splay::Node *root;
145         bool isBad(){
146             return ch[0]->nodeCount > alpha*nodeCount+5 || ch[1]->
                nodeCount > alpha*nodeCount+5;
147         }
148         void push_up(){
149             size = 1+ch[0]->size+ch[1]->size;
150             nodeCount = 1+ch[0]->nodeCount+ch[1]->nodeCount;
151         }
152     };
153     Node pool[MAXN];
154     Node *tail,*root,*null;
155     Node *bc[MAXN];
156     int bc_top;
157     void init(){
158         tail = pool;
159         null = tail++;
160         null->ch[0] = null->ch[1] = null;
161         null->size = null->nodeCount = 0;
162         null->root = Splay::null;
163         bc_top = 0;
164     }
165     inline Node* newNode(int key){

```



```

166     Node *p;
167     if(bc_top)p = bc[--bc_top];
168     else p = tail++;
169     p->key = key;
170     p->ch[0] = p->ch[1] = null;
171     p->size = p->nodeCount = 1;
172     p->root = Splay::null;
173     return p;
174 }
175 Node *buildTree(int *a,int l,int r){
176     if(l >= r)return null;
177     int mid = (l+r)/2;
178     Node *p = newNode(a[mid]);
179     for(int i = l;i < r;i++)
180         Splay::insert(p->root,a[i]);
181     p->ch[0] = buildTree(a,l,mid);
182     p->ch[1] = buildTree(a,mid+1,r);
183     p->push_up();
184     return p;
185 }
186 void Travel(Node *p,vector<int>&v){
187     if(p == null)return;
188     Travel(p->ch[0],v);
189     v.push_back(p->key);
190     Splay::CLEAR(p->root);
191     bc[bc_top++] = p;
192     Travel(p->ch[1],v);
193 }
194 Node *divide(vector<int>&v,int l,int r){
195     if(l == r)return null;
196     int mid = (l+r)/2;
197     Node *p = newNode(v[mid]);
198     for(int i = l;i < r;i++)
199         Splay::insert(p->root,v[i]);
200     p->ch[0] = divide(v,l,mid);
201     p->ch[1] = divide(v,mid+1,r);
202     p->push_up();
203     return p;
204 }
205 inline void rebuild(Node *&p){
206     vector<int>v;
207     Travel(p,v);
208     p = divide(v,0,v.size());
209 }
210 //将第 id 个值修改为 val
211 int Modify(Node *p,int id,int val){
212     if(id == p->ch[0]->size+1){
213         int v = p->key;
214         Splay::erase(p->root,v);
215         Splay::insert(p->root,val);
216         p->key = val;

```

```

217         return v;
218     }
219     int res;
220     if(p->ch[0]->size >= id)
221         res = Modify(p->ch[0],id,val);
222     else res = Modify(p->ch[1],id-p->ch[0]->size-1,val);
223     Splay::erase(p->root,res);
224     Splay::insert(p->root,val);
225     return res;
226 }
227 Node **insert(Node *&p,int id,int val){
228     if(p == null){
229         p = newNode(val);
230         Splay::insert(p->root,val);
231         return &null;
232     }
233     else {
234         p->size++;
235         p->nodeCount++;
236         Splay::insert(p->root,val);
237         Node ** res;
238         if(id <= p->ch[0]->size+1)
239             res = insert(p->ch[0],id,val);
240         else res = insert(p->ch[1],id-p->ch[0]->size-1,val);
241         if(p->isBad())res = &p;
242         return res;
243     }
244 }
245 void insert(int id,int val){
246     Node **p = insert(root,id,val);
247     if(*p != null)rebuild(*p);
248 }
249 //查询在 [l,r] 区间, 值小于等于 val 的个数
250 int query(Node *p,int l,int r,int val){
251     if(p == null)return 0;
252     if(l <= 1 && p->size <= r)
253         return Splay::query(p->root,val);
254     else {
255         int ans = 0;
256         if(l <= p->ch[0]->size)
257             ans += query(p->ch[0],l,r,val);
258         if(r > p->ch[0]->size+1)
259             ans += query(p->ch[1],l-p->ch[0]->size-1,r-p->ch[0]->size-1,val);
260         if(p->key <= val && l <= p->ch[0]->size+1 && p->ch[0]->size+1 <= r)
261             ans++;
262         return ans;
263     }
264 }
265 int query(int L,int R,int k){

```

```

266     int ans;
267     int l = 0, r = 100000;
268     while(l <= r){
269         int mid = (l+r)/2;
270         if(query(root,L,R,mid) >= k){
271             ans = mid;
272             r = mid-1;
273         }
274         else l = mid+1;
275     }
276     return ans;
277 }
278 };
279 int a[MAXN];
280 int main()
281 {
282     int n;
283     while(scanf("%d",&n) == 1){
284         Splay::init();
285         ScapeGoatTree::init();
286         for(int i = 0;i < n;i++)scanf("%d",&a[i]);
287         ScapeGoatTree::root = ScapeGoatTree::buildTree(a,0,n);
288         int m;
289         char op[10];
290         scanf("%d",&m);
291         int ans = 0;
292         while(m--){
293             scanf("%s",op);
294             if(op[0] == 'Q'){
295                 int x,y,k;
296                 scanf("%d%d%d",&x,&y,&k);
297                 x ^= ans; y ^= ans; k ^= ans;
298                 ans = ScapeGoatTree::query(x,y,k);
299                 printf("%d\n",ans);
300             }
301             else if(op[0] == 'M'){
302                 int x,val;
303                 scanf("%d%d",&x,&val);
304                 x ^= ans; val ^= ans;
305                 ScapeGoatTree::Modify(ScapeGoatTree::root,x,val);
306             }
307             else if(op[0] == 'I'){
308                 int x,val;
309                 scanf("%d%d",&x,&val);
310                 x ^= ans; val ^= ans;
311                 ScapeGoatTree::insert(x,val);
312             }
313         }
314     }
315     return 0;
316 }

```

4 图论

4.1 最短路

4.1.1 Dijkstra 单源最短路

权值必须是非负

```

1  /*
2   * 单源最短路径, Dijkstra 算法, 邻接矩阵形式, 复杂度为 $O(n^2)$ 
3   * 求出源 beg 到所有点的最短路径, 传入图的顶点数, 和邻接矩阵 cost[][]
4   * 返回各点的最短路径 lowcost[], 路径 pre[].pre[i] 记录 beg 到 i 路径上的
      父结点, pre[beg]=-1
5   * 可更改路径权类型, 但是权值必须为非负
6   */
7  const int MAXN=1010;
8  #define typec int
9  const typec INF=0x3f3f3f3f;//防止后面溢出, 这个不能太大
10 bool vis[MAXN];
11 int pre[MAXN];
12 void Dijkstra(typec cost[][MAXN],typec lowcost[],int n,int beg){
13     for(int i=0;i<n;i++){
14         lowcost[i]=INF;vis[i]=false;pre[i]=-1;
15     }
16     lowcost[beg]=0;
17     for(int j=0;j<n;j++){
18         int k=-1;
19         int Min=INF;
20         for(int i=0;i<n;i++){
21             if(!vis[i]&&lowcost[i]<Min){
22                 Min=lowcost[i];
23                 k=i;
24             }
25         if(k===-1)break;
26         vis[k]=true;
27         for(int i=0;i<n;i++){
28             if(!vis[i]&&lowcost[k]+cost[k][i]<lowcost[i]){
29                 lowcost[i]=lowcost[k]+cost[k][i];
30                 pre[i]=k;
31             }
32         }
33     }

```

4.1.2 Dijkstra 算法 + 堆优化

使用优先队列优化, 复杂度 $O(E \log E)$

```

1  /*
2   * 使用优先队列优化 Dijkstra 算法
3   * 复杂度  $O(E \log E)$ 
4   * 注意对 vector<Edge>E[MAXN] 进行初始化后加边
5   */
6  const int INF=0x3f3f3f3f;

```

```

7  const int MAXN=1000010;
8  struct qnode{
9      int v;
10     int c;
11     qnode(int _v=0,int _c=0):v(_v),c(_c){}
12     bool operator <(const qnode &r)const{
13         return c>r.c;
14     }
15 };
16 struct Edge{
17     int v,cost;
18     Edge(int _v=0,int _cost=0):v(_v),cost(_cost){}
19 };
20 vector<Edge>E[MAXN];
21 bool vis[MAXN];
22 int dist[MAXN];
23 //点的编号从 1 开始
24 void Dijkstra(int n,int start){
25     memset(vis,false,sizeof(vis));
26     for(int i=1;i<=n;i++)dist[i]=INF;
27     priority_queue<qnode>que;
28     while(!que.empty())que.pop();
29     dist[start]=0;
30     que.push(qnode(start,0));
31     qnode tmp;
32     while(!que.empty()){
33         tmp=que.top();
34         que.pop();
35         int u=tmp.v;
36         if(vis[u])continue;
37         vis[u]=true;
38         for(int i=0;i<E[u].size();i++){
39             int v=E[u][i].v;
40             int cost=E[u][i].cost;
41             if(!vis[v]&&dist[v]>dist[u]+cost){
42                 dist[v]=dist[u]+cost;
43                 que.push(qnode(v,dist[v]));
44             }
45         }
46     }
47 }
48 void addedge(int u,int v,int w){
49     E[u].push_back(Edge(v,w));
50 }

```

4.1.3 单源最短路 bellman_ford 算法

```

1  /*
2  * 单源最短路 bellman_ford 算法, 复杂度 O(VE)
3  * 可以处理负边权图。
4  * 可以判断是否存在负环回路。返回 true, 当且仅当图中不包含从源点可达的负权回路

```

```

5  * vector<Edge>E; 先 E.clear() 初始化, 然后加入所有边
6  * 点的编号从 1 开始 (从 0 开始简单修改就可以了)
7  */
8  const int INF=0x3f3f3f3f;
9  const int MAXN=550;
10 int dist[MAXN];
11 struct Edge{
12     int u,v;
13     int cost;
14     Edge(int _u=0,int _v=0,int _cost=0):u(_u),v(_v),cost(_cost){}
15 };
16 vector<Edge>E;
17 //点的编号从 1 开始
18 bool bellman_ford(int start,int n){
19     for(int i=1;i<=n;i++)dist[i]=INF;
20     dist[start]=0;
21     //最多做 n-1 次
22     for(int i=1;i<n;i++){
23         bool flag=false;
24         for(int j=0;j<E.size();j++){
25             int u=E[j].u;
26             int v=E[j].v;
27             int cost=E[j].cost;
28             if(dist[v]>dist[u]+cost){
29                 dist[v]=dist[u]+cost;
30                 flag=true;
31             }
32         }
33         if(!flag)return true;//没有负环回路
34     }
35     for(int j=0;j<E.size();j++)
36         if(dist[E[j].v]>dist[E[j].u]+E[j].cost)
37             return false;//有负环回路
38     return true;//没有负环回路
39 }

```

4.1.4 单源最短路 SPFA

```

1  /*
2  * 单源最短路 SPFA
3  * 时间复杂度  $O(kE)$ 
4  * 这个是队列实现, 有时候改成栈实现会更加快, 很容易修改
5  * 这个复杂度是不定的
6  */
7  const int MAXN=1010;
8  const int INF=0x3f3f3f3f;
9  struct Edge{
10     int v;
11     int cost;
12     Edge(int _v=0,int _cost=0):v(_v),cost(_cost){}
13 };
14 vector<Edge>E[MAXN];

```

```

15 void addedge(int u,int v,int w){
16     E[u].push_back(Edge(v,w));
17 }
18 bool vis[MAXN]; //在队列标志
19 int cnt[MAXN]; //每个点的入队列次数
20 int dist[MAXN];
21 bool SPFA(int start,int n){
22     memset(vis,false,sizeof(vis));
23     for(int i=1;i<=n;i++) dist[i]=INF;
24     vis[start]=true;
25     dist[start]=0;
26     queue<int> que;
27     while(!que.empty()) que.pop();
28     que.push(start);
29     memset(cnt,0,sizeof(cnt));
30     cnt[start]=1;
31     while(!que.empty()){
32         int u=que.front();
33         que.pop();
34         vis[u]=false;
35         for(int i=0;i<E[u].size();i++){
36             int v=E[u][i].v;
37             if(dist[v]>dist[u]+E[u][i].cost){
38                 dist[v]=dist[u]+E[u][i].cost;
39                 if(!vis[v]){
40                     vis[v]=true;
41                     que.push(v);
42                     if(++cnt[v]>n) return false;
43                     //cnt[i] 为入队列次数，用来判定是否存在负环回路
44                 }
45             }
46         }
47     }
48     return true;
49 }

```

4.2 最小生成树

4.2.1 Prim 算法

```

1  /*
2   * Prim 求 MST
3   * 耗费矩阵 cost[][], 标号从 0 开始, 0~n-1
4   * 返回最小生成树的权值, 返回 -1 表示原图不连通
5   */
6  const int INF=0x3f3f3f3f;
7  const int MAXN=110;
8  bool vis[MAXN];
9  int lowc[MAXN];
10 //点是 0 n-1
11 int Prim(int cost[][MAXN],int n){
12     int ans=0;

```

```

13     memset(vis, false, sizeof(vis));
14     vis[0]=true;
15     for(int i=1; i<n; i++) lowc[i]=cost[0][i];
16     for(int i=1; i<n; i++){
17         int minc=INF;
18         int p=-1;
19         for(int j=0; j<n; j++){
20             if(!vis[j]&&minc>lowc[j]){
21                 minc=lowc[j];
22                 p=j;
23             }
24             if(minc==INF) return -1; //原图不连通
25             ans+=minc;
26             vis[p]=true;
27             for(int j=0; j<n; j++){
28                 if(!vis[j]&&lowc[j]>cost[p][j])
29                     lowc[j]=cost[p][j];
30             }
31         }
32     }

```

4.2.2 Kruskal 算法

```

1  /*
2   * Kruskal 算法求 MST
3   */
4  const int MAXN=110; //最大点数
5  const int MAXM=10000; //最大边数
6  int F[MAXN]; //并查集使用
7  struct Edge{
8      int u,v,w;
9  }edge[MAXM]; //存储边的信息, 包括起点/终点/权值
10 int tol; //边数, 加边前赋值为 0
11 void addedge(int u, int v, int w){
12     edge[tol].u=u;
13     edge[tol].v=v;
14     edge[tol++].w=w;
15 }
16 //排序函数, 讲边按照权值从小到大排序
17 bool cmp(Edge a, Edge b){
18     return a.w<b.w;
19 }
20 int find(int x){
21     if(F[x]==-1) return x;
22     else return F[x]=find(F[x]);
23 }
24 //传入点数, 返回最小生成树的权值, 如果不连通返回 -1
25 int Kruskal(int n){
26     memset(F, -1, sizeof(F));
27     sort(edge, edge+tol, cmp);
28     int cnt=0; //计算加入的边数
29     int ans=0;

```



```

30     for(int i=0;i<tol;i++){
31         int u=edge[i].u;
32         int v=edge[i].v;
33         int w=edge[i].w;
34         int t1=find(u);
35         int t2=find(v);
36         if(t1!=t2){
37             ans+=w;
38             F[t1]=t2;
39             cnt++;
40         }
41         if(cnt==n-1)break;
42     }
43     if(cnt<n-1)return -1;//不连通
44     else return ans;
45 }

```

4.3 次小生成树

```

1  /*
2  * 次小生成树
3  * 求最小生成树时,用数组 Max[i][j] 来表示 MST 中 i 到 j 最大边权
4  * 求完后,直接枚举所有不在 MST 中的边,替换掉最大边权的边,更新答案
5  * 点的编号从 0 开始
6  */
7  const int MAXN=110;
8  const int INF=0x3f3f3f3f;
9  bool vis[MAXN];
10 int lowc[MAXN];
11 int pre[MAXN];
12 int Max[MAXN][MAXN]; //Max[i][j] 表示在最小生成树中从 i 到 j 的路径中的最
    大边权
13 bool used[MAXN][MAXN];
14 int Prim(int cost[][MAXN],int n){
15     int ans=0;
16     memset(vis,false,sizeof(vis));
17     memset(Max,0,sizeof(Max));
18     memset(used,false,sizeof(used));
19     vis[0]=true;
20     pre[0]=-1;
21     for(int i=1;i<n;i++){
22         lowc[i]=cost[0][i];
23         pre[i]=0;
24     }
25     lowc[0]=0;
26     for(int i=1;i<n;i++){
27         int minc=INF;
28         int p=-1;
29         for(int j=0;j<n;j++){
30             if(!vis[j]&&minc>lowc[j]){
31                 minc=lowc[j];
32                 p=j;

```

```

33         }
34         if(minc==INF)return -1;
35         ans+=minc;
36         vis[p]=true;
37         used[p][pre[p]]=used[pre[p]][p]=true;
38         for(int j=0;j<n;j++){
39             if(vis[j] && j != p)Max[j][p]=Max[p][j]=max(Max[j][pre[
                p]],lowc[p]);
40             if(!vis[j]&&lowc[j]>cost[p][j]){
41                 lowc[j]=cost[p][j];
42                 pre[j]=p;
43             }
44         }
45     }
46     return ans;
47 }

```

4.4 有向图的强连通分量

4.4.1 Tarjan

```

1  /*
2   * Tarjan 算法
3   * 复杂度 O(N+M)
4   */
5  const int MAXN = 20010;//点数
6  const int MAXM = 50010;//边数
7  struct Edge{
8      int to,next;
9  }edge[MAXN];
10 int head[MAXN],tot;
11 int Low[MAXN],DFN[MAXN],Stack[MAXN],Belong[MAXN];//Belong 数组的值是
    1 ~ scc
12 int Index,top;
13 int scc;//强连通分量的个数
14 bool Instack[MAXN];
15 int num[MAXN];//各个强连通分量包含点的个数，数组编号 1 ~ scc
16 //num 数组不一定需要，结合实际情况
17
18 void addedge(int u,int v){
19     edge[tot].to = v;edge[tot].next = head[u];head[u] = tot++;
20 }
21 void Tarjan(int u){
22     int v;
23     Low[u] = DFN[u] = ++Index;
24     Stack[top++] = u;
25     Instack[u] = true;
26     for(int i = head[u];i != -1;i = edge[i].next){
27         v = edge[i].to;
28         if( !DFN[v] ){
29             Tarjan(v);
30             if( Low[u] > Low[v] )Low[u] = Low[v];

```

```

31     }
32     else if(Instack[v] && Low[u] > DFN[v])
33         Low[u] = DFN[v];
34 }
35 if(Low[u] == DFN[u]){
36     scc++;
37     do{
38         v = Stack[--top];
39         Instack[v] = false;
40         Belong[v] = scc;
41         num[scc]++;
42     }
43     while( v != u);
44 }
45 }
46 void solve(int N){
47     memset(DFN,0,sizeof(DFN));
48     memset(Instack,false,sizeof(Instack));
49     memset(num,0,sizeof(num));
50     Index = scc = top = 0;
51     for(int i = 1;i <= N;i++)
52         if(!DFN[i])
53             Tarjan(i);
54 }
55 void init(){
56     tot = 0;
57     memset(head,-1,sizeof(head));
58 }

```

4.4.2 Kosaraju

```

1  /*
2   * Kosaraju 算法, 复杂度 O(N+M)
3   */
4  const int MAXN = 20010;
5  const int MAXM = 50010;
6  struct Edge{
7      int to,next;
8  }edge1[MAXN],edge2[MAXN];
9  //edge1 是原图 G, edge2 是逆图 GT
10 int head1[MAXN],head2[MAXN];
11 bool mark1[MAXN],mark2[MAXN];
12 int tot1,tot2;
13 int cnt1,cnt2;
14 int st[MAXN]; //对原图进行 dfs, 点的结束时间从小到大排序
15 int Belong[MAXN]; //每个点属于哪个连通分量 (0~cnt2-1)
16 int num; //中间变量, 用来数某个连通分量中点的个数
17 int setNum[MAXN]; //强连通分量中点的个数, 编号 0~cnt2-1
18 void addedge(int u,int v){
19     edge1[tot1].to = v;edge1[tot1].next = head1[u];head1[u] = tot1++;

```

```

20     edge2[tot2].to = u; edge2[tot2].next = head2[v]; head2[v] = tot2
      ++;
21 }
22 void DFS1(int u){
23     mark1[u] = true;
24     for(int i = head1[u]; i != -1; i = edge1[i].next)
25         if(!mark1[edge1[i].to])
26             DFS1(edge1[i].to);
27     st[cnt1++] = u;
28 }
29 void DFS2(int u){
30     mark2[u] = true;
31     num++;
32     Belong[u] = cnt2;
33     for(int i = head2[u]; i != -1; i = edge2[i].next)
34         if(!mark2[edge2[i].to])
35             DFS2(edge2[i].to);
36 }
37 //点的编号从 1 开始
38 void solve(int n){
39     memset(mark1, false, sizeof(mark1));
40     memset(mark2, false, sizeof(mark2));
41     cnt1 = cnt2 = 0;
42     for(int i = 1; i <= n; i++)
43         if(!mark1[i])
44             DFS1(i);
45     for(int i = cnt1-1; i >= 0; i--)
46         if(!mark2[st[i]]){
47             num = 0;
48             DFS2(st[i]);
49             setNum[cnt2++] = num;
50         }
51 }

```

4.5 图的割点、桥和双连通分支的基本概念

[点连通度与边连通度] 在一个无向连通图中，如果有一个顶点集合，删除这个顶点集合，以及这个集合中所有顶点相关联的边以后，原图变成多个连通块，就称这个点集为割点集合。一个图的点连通度的定义为，最小割点集合中的顶点数。

类似的，如果有一个边集合，删除这个边集合以后，原图变成多个连通块，就称这个点集为割边集合。一个图的边连通度的定义为，最小割边集合中的边数。

[双连通图、割点与桥]

如果一个无向连通图的点连通度大于 1，则称该图是点双连通的 (point biconnected)，简称双连通或重连通。一个图有割点，当且仅当这个图的点连通度为 1，则割点集合的唯一元素被称为割点 (cut point)，又叫关节点 (articulation point)。

如果一个无向连通图的边连通度大于 1，则称该图是边双连通的 (edge biconnected)，简称双连通或重连通。一个图有桥，当且仅当这个图的边连通度为 1，则割边集合的唯一元素被称为桥 (bridge)，又叫关节边 (articulation edge)。

可以看出，点双连通与边双连通都可以简称为双连通，它们之间是有着某种联系的，下文中提到的双连通，均既可指点双连通，又可指边双连通。

[双连通分支]

在图 G 的所有子图 G' 中，如果 G' 是双连通的，则称 G' 为双连通子图。如果一个双连

通子图 G' 它不是任何一个双连通子图的真子集, 则 G' 为极大双连通子图。双连通分支 (biconnected component), 或重连通分支, 就是图的极大双连通子图。特殊的, 点双连通分支又叫做块。[求割点与桥]

该算法是 R.Tarjan 发明的。对图深度优先搜索, 定义 $DFS(u)$ 为 u 在搜索树 (以下简称为树) 中被遍历到的次序号。定义 $Low(u)$ 为 u 或 u 的子树中能通过非父子边追溯到的最早的节点, 即 DFS 序号最小的节点。根据定义, 则有:

$Low(u) = \min DFS(u) DFS(v)$ (u, v 为后向边 (返祖边) 等价于 $DFS(v) < DFS(u)$ 且 v 不为 u 的父亲节点 $Low(v)$ (u, v 为树枝边 (父子边) 一个顶点 u 是割点, 当且仅当满足 (1) 或 (2) (1) u 为树根, 且 u 有多于一个子树。(2) u 不为树根, 且满足存在 (u, v) 为树枝边 (或称父子边, 即 u 为 v 在搜索树中的父亲), 使得 $DFS(u) \leq Low(v)$ 。

一条无向边 (u, v) 是桥, 当且仅当 (u, v) 为树枝边, 且满足 $DFS(u) < Low(v)$ 。

[求双连通分支]

下面要分开讨论点双连通分支与边双连通分支的求法。

对于点双连通分支, 实际上在求割点的过程中就能顺便把每个点双连通分支求出。建立一个栈, 存储当前双连通分支, 在搜索图时, 每找到一条树枝边或后向边 (非横叉边), 就把这条边加入栈中。如果遇到某时满足 $DFS(u) \leq Low(v)$, 说明 u 是一个割点, 同时把边从栈顶一个个取出, 直到遇到了边 (u, v), 取出的这些边与其关联的点, 组成一个点双连通分支。割点可以属于多个点双连通分支, 其余点和每条边只属于且属于一个点双连通分支。

对于边双连通分支, 求法更为简单。只需在求出所有的桥以后, 把桥边删除, 原图变成了多个连通块, 则每个连通块就是一个边双连通分支。桥不属于任何一个边双连通分支, 其余的边和每个顶点都属于且只属于一个边双连通分支。

[构造双连通图]

一个有桥的连通图, 如何把它通过加边变成边双连通图? 方法为首先求出所有的桥, 然后删除这些桥边, 剩下的每个连通块都是一个双连通子图。把每个双连通子图收缩为一个顶点, 再把桥边加回来, 最后的这个图一定是一棵树, 边连通度为 1。

统计出树中度为 1 的节点的个数, 即为叶节点的个数, 记为 $leaf$ 。则至少在树上添加 $(leaf+1)/2$ 条边, 就能使树达到边二连通, 所以至少添加的边数就是 $(leaf+1)/2$ 。具体方法为, 首先把两个最近公共祖先最远的两个叶节点之间连接一条边, 这样可以把这两个点到祖先的路径上所有点收缩到一起, 因为一个形成的环一定是双连通的。然后再找两个最近公共祖先最远的两个叶节点, 这样一对一对找完, 恰好是 $(leaf+1)/2$ 次, 把所有点收缩到了一起。

4.6 割点与桥

4.6.1 模板

```

1  /*
2  *   求无向图的割点和桥
3  *   可以找出割点和桥, 求删掉每个点后增加的连通块。
4  *   需要注意重边的处理, 可以先用矩阵存, 再转邻接表, 或者进行判重
5  */
6  const int MAXN = 10010;
7  const int MAXM = 100010;
8  struct Edge{
9      int to,next;
10     bool cut;//是否为桥的标记
11 }edge[MAXN];
12 int head[MAXN],tot;
13 int Low[MAXN],DFN[MAXN],Stack[MAXN];
14 int Index,top;
15 bool Instack[MAXN];

```

```

16 bool cut[MAXN];
17 int add_block[MAXN]; //删除一个点后增加的连通块
18 int bridge;
19 void addedge(int u, int v){
20     edge[tot].to = v; edge[tot].next = head[u]; edge[tot].cut = false
21     ;
22     head[u] = tot++;
23 }
24 void Tarjan(int u, int pre){
25     int v;
26     Low[u] = DFN[u] = ++Index;
27     Stack[top++] = u;
28     Instack[u] = true;
29     int son = 0;
30     int pre_cnt = 0; //处理重边, 如果不需要可以去掉
31     for(int i = head[u]; i != -1; i = edge[i].next){
32         v = edge[i].to;
33         if(v == pre && pre_cnt == 0){pre_cnt++; continue;}
34         if( !DFN[v] ){
35             son++;
36             Tarjan(v, u);
37             if(Low[u] > Low[v]) Low[u] = Low[v];
38             //桥
39             //一条无向边 (u,v) 是桥, 当且仅当 (u,v) 为树枝边, 且满足
40             DFS(u) < Low(v)。
41             if(Low[v] > DFN[u]){
42                 bridge++;
43                 edge[i].cut = true;
44                 edge[i^1].cut = true;
45             }
46             //割点
47             //一个顶点 u 是割点, 当且仅当满足 (1) 或 (2) (1) u 为树根, 且
48             u 有多于一个子树。
49             //(2) u 不为树根, 且满足存在 (u,v) 为树枝边 (或称父子边,
50             //即 u 为 v 在搜索树中的父亲), 使得 DFS(u) <= Low(v)
51             if(u != pre && Low[v] >= DFN[u]){ //不是树根
52                 cut[u] = true;
53                 add_block[u]++;
54             }
55         }
56         else if( Low[u] > DFN[v] )
57             Low[u] = DFN[v];
58     }
59     //树根, 分支数大于 1
60     if(u == pre && son > 1) cut[u] = true;
61     if(u == pre) add_block[u] = son - 1;
62     Instack[u] = false;
63     top--;
64 }

```

4.6.2 调用

1) UVA 796 Critical Links 给出一个无向图，按顺序输出桥

```

1 void solve(int N){
2     memset(DFN,0,sizeof(DFN));
3     memset(Instack,false,sizeof(Instack));
4     memset(add_block,0,sizeof(add_block));
5     memset(cut,false,sizeof(cut));
6     Index = top = 0;
7     bridge = 0;
8     for(int i = 1;i <= N;i++){
9         if( !DFN[i] )
10             Tarjan(i,i);
11     }
12     printf("%d critical links\n",bridge);
13     vector<pair<int,int> >ans;
14     for(int u = 1;u <= N;u++){
15         for(int i = head[u];i != -1;i = edge[i].next)
16             if(edge[i].cut && edge[i].to > u)
17                 ans.push_back(make_pair(u,edge[i].to));
18     }
19     sort(ans.begin(),ans.end());
20     //按顺序输出桥
21     for(int i = 0;i < ans.size();i++){
22         printf("%d - %d\n",ans[i].first-1,ans[i].second-1);
23     }
24     printf("\n");
25 }
26 void init(){
27     tot = 0;
28     memset(head,-1,sizeof(head));
29 }
30 //处理重边
31 map<int,int>mapit;
32 inline bool isHash(int u,int v){
33     if(mapit[u*MAXN+v])return true;
34     if(mapit[v*MAXN+u])return true;
35     mapit[u*MAXN+v] = mapit[v*MAXN+u] = 1;
36     return false;
37 }
38 int main(){
39     int n;
40     while(scanf("%d",&n) == 1){
41         init();
42         int u;
43         int k;
44         int v;
45         //mapit.clear();
46         for(int i = 1;i <= n;i++){
47             scanf("%d %d",&u,&k);
48             u++;
49             //这样加边，要保证正边和反边是相邻的，建无向图

```

```

49         while(k--){
50             scanf("%d",&v);
51             v++;
52             if(v <= u)continue;
53             //if(isHash(u,v))continue;
54             addedge(u,v);
55             addedge(v,u);
56         }
57     }
58     solve(n);
59 }
60 return 0;
61 }

```

2) POJ 2117 求删除一个点后，图中最多有多少个连通块

```

1 void solve(int N){
2     memset(DFN,0,sizeof(DFN));
3     memset(Instack,0,sizeof(Instack));
4     memset(add_block,0,sizeof(add_block));
5     memset(cut,false,sizeof(cut));
6     Index = top = 0;
7     int cnt = 0;//原来的连通块数
8     for(int i = 1;i <= N;i++){
9         if( !DFN[i] ){
10             Tarjan(i,i);//找割点调用必须是 Tarjan(i,i)
11             cnt++;
12         }
13     }
14     int ans = 0;
15     for(int i = 1;i <= N;i++){
16         ans = max(ans,cnt+add_block[i]);
17     }
18     printf("%d\n",ans);
19 }
20 void init(){
21     tot = 0;
22     memset(head,-1,sizeof(head));
23 }
24 int main(){
25     int n,m;
26     int u,v;
27     while(scanf("%d%d",&n,&m)==2){
28         if(n==0 && m == 0)break;
29         init();
30         while(m--){
31             scanf("%d%d",&u,&v);
32             u++;v++;
33             addedge(u,v);
34             addedge(v,u);
35         }
36         solve(n);
37     }
38     return 0;
39 }

```


4.7 边双连通分支

去掉桥，其余的连通分支就是边双连通分支了。一个有桥的连通图要变成边双连通图的话，把双连通子图收缩为一个点，形成一颗树。需要加的边为 $(leaf+1)/2$ (leaf 为叶子结点个数)
POJ 3177 给定一个连通的无向图 G，至少要添加几条边，才能使其变为双连通图。

```

1  const int MAXN = 5010; //点数
2  const int MAXM = 20010; //边数，因为是无向图，所以这个值要 *2
3  struct Edge{
4      int to,next;
5      bool cut; //是否是桥标记
6  }edge[MAXN];
7  int head[MAXN],tot;
8  int Low[MAXN],DFN[MAXN],Stack[MAXN],Belong[MAXN]; //Belong 数组的值是
    1 ~ block
9  int Index,top;
10 int block; //边双连通块数
11 bool Instack[MAXN];
12 int bridge; //桥的数目
13 void addedge(int u,int v){
14     edge[tot].to = v; edge[tot].next = head[u]; edge[tot].cut=false;
15     head[u] = tot++;
16 }
17 void Tarjan(int u,int pre){
18     int v;
19     Low[u] = DFN[u] = ++Index;
20     Stack[top++] = u;
21     Instack[u] = true;
22     int pre_cnt = 0;
23     for(int i = head[u]; i != -1; i = edge[i].next){
24         v = edge[i].to;
25         if(v == pre && pre_cnt == 0){pre_cnt++; continue;}
26         if( !DFN[v] ){
27             Tarjan(v,u);
28             if( Low[u] > Low[v] )Low[u] = Low[v];
29             if(Low[v] > DFN[u]){
30                 bridge++;
31                 edge[i].cut = true;
32                 edge[i^1].cut = true;
33             }
34         }
35         else if( Instack[v] && Low[u] > DFN[v] )
36             Low[u] = DFN[v];
37     }
38     if(Low[u] == DFN[u]){
39         block++;
40         do{
41             v = Stack[--top];
42             Instack[v] = false;
43             Belong[v] = block;
44         }

```

```

45     while( v!=u );
46 }
47 }
48 void init(){
49     tot = 0;
50     memset(head,-1,sizeof(head));
51 }
52 int du[MAXN]; //缩点后形成树, 每个点的度数
53 void solve(int n){
54     memset(DFN,0,sizeof(DFN));
55     memset(Instack,false,sizeof(Instack));
56     Index = top = block = 0;
57     Tarjan(1,0);
58     int ans = 0;
59     memset(du,0,sizeof(du));
60     for(int i = 1; i <= n; i++){
61         for(int j = head[i]; j != -1; j = edge[j].next){
62             if(edge[j].cut)
63                 du[Belong[i]]++;
64         }
65         for(int i = 1; i <= block; i++){
66             if(du[i]==1)
67                 ans++;
68         }
69         //找叶子结点的个数 ans, 构造边双连通图需要加边 (ans+1)/2
70         printf("%d\n", (ans+1)/2);
71     }
72 }
73 int main(){
74     int n,m;
75     int u,v;
76     while(scanf("%d%d",&n,&m)==2){
77         init();
78         while(m--){
79             scanf("%d%d",&u,&v);
80             addedge(u,v);
81             addedge(v,u);
82         }
83         solve(n);
84     }
85     return 0;
86 }

```

4.8 点双连通分支

对于点双连通分支, 实际上在求割点的过程中就能顺便把每个点双连通分支求出。建立一个栈, 存储当前双连通分支, 在搜索图时, 每找到一条树枝边或后向边 (非横叉边), 就把这条边加入栈中。如果遇到某时满足 $DFS(u) \leq Low(v)$, 说明 u 是一个割点, 同时把边从栈顶一个个取出, 直到遇到了边 (u,v) , 取出的这些边与其关联的点, 组成一个点双连通分支。割点可以属于多个点双连通分支, 其余点和每条边只属于且属于一个点双连通分支。

POJ 2942

奇圈, 二分图判断的染色法, 求点双连通分支

```

1  /*
2  POJ 2942 Knights of the Round Table

```

```

3 | 亚瑟王要在圆桌上召开骑士会议，为了不引发骑士之间的冲突，
4 | 并且能够让会议的议题有令人满意的结果，每次开会前都必须对出席会议的骑士有如下要
   | 求：
5 | 1、相互憎恨的两个骑士不能坐在直接相邻的 2 个位置；
6 | 2、出席会议的骑士数必须是奇数，这是为了让投票表决议题时都能有结果。
7 |
8 | 注意：1、所给出的憎恨关系一定是双向的，不存在单向憎恨关系。
9 | 2、由于是圆桌会议，则每个出席的骑士身边必定刚好有 2 个骑士。
10 | 即每个骑士的座位两边都必定各有一个骑士。
11 | 3、一个骑士无法开会，就是说至少有 3 个骑士才可能开会。
12 |
13 | 首先根据给出的互相憎恨的图中得到补图。
14 | 然后就相当于找出不能形成奇圈的点。
15 | 利用下面两个定理：
16 | （1）如果一个双连通分量内的某些顶点在一个奇圈中（即双连通分量含有奇圈），
17 | 那么这个双连通分量的其他顶点也在某个奇圈中；
18 | （2）如果一个双连通分量含有奇圈，则他必定不是一个二分图。反过来也成立，这是一个
   | 充要条件。
19 |
20 | 所以本题的做法，就是对补图求点双连通分量。
21 | 然后对于求得的点双连通分量，使用染色法判断是不是二分图，不是二分图，这个双连通分
   | 量的点是可以存在的
22 | */
23 |
24 | const int MAXN = 1010;
25 | const int MAXM = 2000010;
26 | struct Edge{
27 |     int to,next;
28 | }edge[MAXN];
29 | int head[MAXN],tot;
30 | int Low[MAXN],DFN[MAXN],Stack[MAXN],Belong[MAXN];
31 | int Index,top;
32 | int block;//点双连通分量的个数
33 | bool Instack[MAXN];
34 | bool can[MAXN];
35 | bool ok[MAXN]);//标记
36 | int tmp[MAXN]);//暂时存储双连通分量中的点
37 | int cc;//tmp 的计数
38 | int color[MAXN]);//染色
39 | void addedge(int u,int v){
40 |     edge[tot].to = v;edge[tot].next = head[u];head[u] = tot++;
41 | }
42 | //染色判断二分图
43 | bool dfs(int u,int col){
44 |     color[u] = col;
45 |     for(int i = head[u];i != -1;i = edge[i].next){
46 |         int v = edge[i].to;
47 |         if( !ok[v] )continue;
48 |         if(color[v] != -1){
49 |             if(color[v]==col)return false;
50 |             continue;

```

```

51         }
52         if(!dfs(v,!col))return false;
53     }
54     return true;
55 }
56 void Tarjan(int u,int pre){
57     int v;
58     Low[u] = DFN[u] = ++Index;
59     Stack[top++] = u;
60     Instack[u] = true;
61     int pre_cnt = 0;
62     for(int i = head[u];i != -1;i = edge[i].next){
63         v = edge[i].to;
64         if(v == pre && pre_cnt == 0){pre_cnt++; continue;}
65         if( !DFN[v] ){
66             Tarjan(v,u);
67             if(Low[u] > Low[v])Low[u] = Low[v];
68             if( Low[v] >= DFN[u]){
69                 block++;
70                 int vn;
71                 cc = 0;
72                 memset(ok,false,sizeof(ok));
73                 do{
74                     vn = Stack[--top];
75                     Belong[vn] = block;
76                     Instack[vn] = false;
77                     ok[vn] = true;
78                     tmp[cc++] = vn;
79                 }
80                 while( vn!=v );
81                 ok[u] = 1;
82                 memset(color,-1,sizeof(color));
83                 if( !dfs(u,0) ){
84                     can[u] = true;
85                     while(cc--)can[tmp[cc]]=true;
86                 }
87             }
88         }
89         else if(Instack[v] && Low[u] > DFN[v])
90             Low[u] = DFN[v];
91     }
92 }
93 void solve(int n){
94     memset(DFN,0,sizeof(DFN));
95     memset(Instack,false,sizeof(Instack));
96     Index = block = top = 0;
97     memset(can,false,sizeof(can));
98     for(int i = 1;i <= n;i++){
99         if(!DFN[i])
100             Tarjan(i,-1);
101     }
102     int ans = n;

```

```

102     for(int i = 1;i <= n;i++)
103         if(can[i])
104             ans—;
105     printf("%d\n",ans);
106 }
107 void init(){
108     tot = 0;
109     memset(head,-1,sizeof(head));
110 }
111 int g[MAXN][MAXN];
112 int main(){
113     int n,m;
114     int u,v;
115     while(scanf("%d%d",&n,&m)==2){
116         if(n==0 && m==0)break;
117         init();
118         memset(g,0,sizeof(g));
119         while(m—){
120             scanf("%d%d",&u,&v);
121             g[u][v]=g[v][u]=1;
122         }
123         for(int i = 1;i <= n;i++)
124             for(int j = 1;j <= n;j++)
125                 if(i != j && g[i][j]==0)
126                     addedge(i,j);
127         solve(n);
128     }
129     return 0;
130 }

```

4.9 最小树形图

```

1  /*
2  * 最小树形图
3  * int 型
4  * 复杂度 O(NM)
5  * 点从 0 开始
6  */
7  const int INF = 0x3f3f3f3f;
8  const int MAXN = 1010;
9  const int MAXM = 40010;
10 struct Edge{
11     int u,v,cost;
12 };
13 Edge edge[MAXM];
14 int pre[MAXN],id[MAXN],visit[MAXN],in[MAXN];
15 int zhuliu(int root,int n,int m,Edge edge[]){
16     int res = 0,u,v;
17     while(1){
18         for(int i = 0;i < n;i++)
19             in[i] = INF;

```

```

20     for(int i = 0;i < m;i++)
21         if(edge[i].u != edge[i].v && edge[i].cost < in[edge[i].
22             v]){
23             pre[edge[i].v] = edge[i].u;
24             in[edge[i].v] = edge[i].cost;
25         }
26     for(int i = 0;i < n;i++)
27         if(i != root && in[i] == INF)
28             return -1;//不存在最小树形图
29     int tn = 0;
30     memset(id,-1,sizeof(id));
31     memset(visit,-1,sizeof(visit));
32     in[root] = 0;
33     for(int i = 0;i < n;i++){
34         res += in[i];
35         v = i;
36         while( visit[v] != i && id[v] == -1 && v != root){
37             visit[v] = i;
38             v = pre[v];
39         }
40         if( v != root && id[v] == -1 ){
41             for(int u = pre[v]; u != v ;u = pre[u])
42                 id[u] = tn;
43             id[v] = tn++;
44         }
45     }
46     if(tn == 0)break;//没有有向环
47     for(int i = 0;i < n;i++)
48         if(id[i] == -1)
49             id[i] = tn++;
50     for(int i = 0;i < m;){
51         v = edge[i].v;
52         edge[i].u = id[edge[i].u];
53         edge[i].v = id[edge[i].v];
54         if(edge[i].u != edge[i].v)
55             edge[i].cost -= in[v];
56         else
57             swap(edge[i],edge[--m]);
58     }
59     n = tn;
60     root = id[root];
61 }
62 return res;
63 }
64 int g[MAXN][MAXN];
65 int main(){
66     int n,m;
67     int iCase = 0;
68     int T;
69     scanf("%d",&T);
70     while( T-- ){

```

```

70         iCase ++;
71         scanf("%d%d",&n,&m);
72         for(int i = 0;i < n;i++)
73             for(int j = 0;j < n;j++)
74                 g[i][j] = INF;
75         int u,v,cost;
76         while(m--){
77             scanf("%d%d%d",&u,&v,&cost);
78             if(u == v)continue;
79             g[u][v] = min(g[u][v],cost);
80         }
81         int L = 0;
82         for(int i = 0;i < n;i++)
83             for(int j = 0;j < n;j++)
84                 if(g[i][j] < INF){
85                     edge[L].u = i;
86                     edge[L].v = j;
87                     edge[L++].cost = g[i][j];
88                 }
89         int ans = zhuliu(0,n,L,edge);
90         printf("Case_#%d:_",iCase);
91         if(ans == -1)printf("Possums!\n");
92         else printf("%d\n",ans);
93     }
94     return 0;
95 }

```

4.10 二分图匹配

4.10.1 邻接矩阵（匈牙利算法）

```

1  /* *****
2  //二分图匹配（匈牙利算法的 DFS 实现）（邻接矩阵形式）
3  //初始化: g[][] 两边顶点的划分情况
4  //建立 g[i][j] 表示 i->j 的有向边就可以了，是左边向右边的匹配
5  //g 没有边相连则初始化为 0
6  //uN 是匹配左边的顶点数，vN 是匹配右边的顶点数
7  //调用: res=hungary(); 输出最大匹配数
8  //优点: 适用于稠密图，DFS 找增广路，实现简洁易于理解
9  //时间复杂度:O(VE)
10 //*****/
11 //顶点编号从 0 开始的
12 const int MAXN = 510;
13 int uN,vN;//u,v 的数目，使用前面必须赋值
14 int g[MAXN][MAXN];//邻接矩阵
15 int linker[MAXN];
16 bool used[MAXN];
17 bool dfs(int u){
18     for(int v = 0; v < vN;v++){
19         if(g[u][v] && !used[v]){
20             used[v] = true;
21             if(linker[v] == -1 || dfs(linker[v])){

```

```

22         linker[v] = u;
23         return true;
24     }
25 }
26 return false;
27 }
28 int hungary(){
29     int res = 0;
30     memset(linker,-1,sizeof(linker));
31     for(int u = 0;u < uN;u++){
32         memset(used,false,sizeof(used));
33         if(dfs(u))res++;
34     }
35     return res;
36 }

```

4.10.2 邻接表（匈牙利算法）

```

1  /*
2   * 匈牙利算法邻接表形式
3   * 使用前用 init() 进行初始化，给 uN 赋值
4   * 加边使用函数 addedge(u,v)
5   *
6   */
7  const int MAXN = 5010;//点数的最大值
8  const int MAXM = 50010;//边数的最大值
9  struct Edge{
10     int to,next;
11 }edge[MAXN];
12 int head[MAXN],tot;
13 void init(){
14     tot = 0;
15     memset(head,-1,sizeof(head));
16 }
17 void addedge(int u,int v){
18     edge[tot].to = v; edge[tot].next = head[u];
19     head[u] = tot++;
20 }
21 int linker[MAXN];
22 bool used[MAXN];
23 int uN;
24 bool dfs(int u){
25     for(int i = head[u]; i != -1 ;i = edge[i].next){
26         int v = edge[i].to;
27         if(!used[v]){
28             used[v] = true;
29             if(linker[v] == -1 || dfs(linker[v])){
30                 linker[v] = u;
31                 return true;
32             }
33         }
34     }
35 }

```



```

35     return false;
36 }
37 int hungary(){
38     int res = 0;
39     memset(linker,-1,sizeof(linker));
40     //点的编号 0~uN-1
41     for(int u = 0; u < uN;u++){
42         memset(used,false,sizeof(used));
43         if(dfs(u))res++;
44     }
45     return res;
46 }

```

4.10.3 Hopcroft-Karp 算法

```

1  /* *****
2  * 二分图匹配 (Hopcroft-Karp 算法)
3  * 复杂度  $O(\sqrt{n} \times E)$ 
4  * 邻接表存图, vector 实现
5  * vector 先初始化, 然后假如边
6  * uN 为左端的顶点数, 使用前赋值 (点编号 0 开始)
7  */
8  const int MAXN = 3000;
9  const int INF = 0x3f3f3f3f;
10 vector<int>G[MAXN];
11 int uN;
12 int Mx[MAXN],My[MAXN];
13 int dx[MAXN],dy[MAXN];
14 int dis;
15 bool used[MAXN];
16 bool SearchP(){
17     queue<int>Q;
18     dis = INF;
19     memset(dx,-1,sizeof(dx));
20     memset(dy,-1,sizeof(dy));
21     for(int i = 0 ; i < uN; i++){
22         if(Mx[i] == -1){
23             Q.push(i);
24             dx[i] = 0;
25         }
26     }
27     while(!Q.empty()){
28         int u = Q.front();
29         Q.pop();
30         if(dx[u] > dis)break;
31         int sz = G[u].size();
32         for(int i = 0;i < sz;i++){
33             int v = G[u][i];
34             if(dy[v] == -1){
35                 dy[v] = dx[u] + 1;
36                 if(My[v] == -1)dis = dy[v];
37                 else{
38                     dx[My[v]] = dy[v] + 1;

```

```

38         Q.push(My[v]);
39     }
40 }
41 }
42 }
43     return dis != INF;
44 }
45 bool DFS(int u){
46     int sz = G[u].size();
47     for(int i = 0;i < sz;i++){
48         int v = G[u][i];
49         if(!used[v] && dy[v] == dx[u] + 1){
50             used[v] = true;
51             if(My[v] != -1 && dy[v] == dis)continue;
52             if(My[v] == -1 || DFS(My[v])){
53                 My[v] = u;
54                 Mx[u] = v;
55                 return true;
56             }
57         }
58     }
59     return false;
60 }
61 int MaxMatch(){
62     int res = 0;
63     memset(Mx,-1,sizeof(Mx));
64     memset(My,-1,sizeof(My));
65     while(SearchP()){
66         memset(used,false,sizeof(used));
67         for(int i = 0;i < uN;i++){
68             if(Mx[i] == -1 && DFS(i))
69                 res++;
70         }
71     }
72     return res;
73 }

```

4.11 二分图多重匹配

```

1  const int MAXN = 1010;
2  const int MAXM = 510;
3  int uN,vN;
4  int g[MAXN][MAXM];
5  int linker[MAXM][MAXN];
6  bool used[MAXM];
7  int num[MAXM]; //右边最大的匹配数
8  bool dfs(int u){
9      for(int v = 0; v < vN;v++){
10         if(g[u][v] && !used[v]){
11             used[v] = true;
12             if(linker[v][0] < num[v]){
13                 linker[v][++linker[v][0]] = u;
14                 return true;

```

```

15         }
16         for(int i = 1; i <= num[v]; i++)
17             if(dfs(linker[v][i])){
18                 linker[v][i] = u;
19                 return true;
20             }
21     }
22     return false;
23 }
24 int hungary(){
25     int res = 0;
26     for(int i = 0; i < vN; i++){
27         linker[i][0] = 0;
28         for(int u = 0; u < uN; u++){
29             memset(used, false, sizeof(used));
30             if(dfs(u)) res++;
31         }
32     }
33     return res;
34 }

```

4.12 二分图最大权匹配 (KM 算法)

```

1  /* KM 算法
2   *   复杂度  $O(n \times n \times n)$ 
3   *   求最大权匹配
4   *   若求最小权匹配, 可将权值取相反数, 结果取相反数
5   *   点的编号从 0 开始
6   */
7  const int N = 310;
8  const int INF = 0x3f3f3f3f;
9  int nx, ny; // 两边的点数
10 int g[N][N]; // 二分图描述
11 int linker[N], lx[N], ly[N]; // y 中各点匹配状态, x, y 中的点标号
12 int slack[N];
13 bool visx[N], visy[N];
14 bool DFS(int x){
15     visx[x] = true;
16     for(int y = 0; y < ny; y++){
17         if(visy[y]) continue;
18         int tmp = lx[x] + ly[y] - g[x][y];
19         if(tmp == 0){
20             visy[y] = true;
21             if(linker[y] == -1 || DFS(linker[y])){
22                 linker[y] = x;
23                 return true;
24             }
25         }
26         else if(slack[y] > tmp)
27             slack[y] = tmp;
28     }
29     return false;
30 }

```

```

31 int KM(){
32     memset(linker,-1,sizeof(linker));
33     memset(ly,0,sizeof(ly));
34     for(int i = 0;i < nx;i++){
35         lx[i] = -INF;
36         for(int j = 0;j < ny;j++){
37             if(g[i][j] > lx[i])
38                 lx[i] = g[i][j];
39         }
40         for(int x = 0;x < nx;x++){
41             for(int i = 0;i < ny;i++){
42                 slack[i] = INF;
43                 while(true){
44                     memset(visx,false,sizeof(visx));
45                     memset(visy,false,sizeof(visy));
46                     if(DFS(x))break;
47                     int d = INF;
48                     for(int i = 0;i < ny;i++){
49                         if(!visy[i] && d > slack[i])
50                             d = slack[i];
51                     }
52                     for(int i = 0;i < nx;i++){
53                         if(visx[i])
54                             lx[i] -= d;
55                     }
56                     for(int i = 0;i < ny;i++){
57                         if(visy[i])ly[i] += d;
58                         else slack[i] -= d;
59                     }
60                 }
61             }
62         }
63         int res = 0;
64         for(int i = 0;i < ny;i++){
65             if(linker[i] != -1)
66                 res += g[linker[i]][i];
67         }
68         return res;
69     }
70 }
71 //HDU 2255
72 int main(){
73     int n;
74     while(scanf("%d",&n) == 1){
75         for(int i = 0;i < n;i++){
76             for(int j = 0;j < n;j++){
77                 scanf("%d",&g[i][j]);
78             }
79             nx = ny = n;
80             printf("%d\n",KM());
81         }
82     }
83     return 0;
84 }

```

4.13 一般图匹配带花树

URAL 1099

```
1 const int MAXN = 250;
```

```

2  int N; //点的个数, 点的编号从 1 到 N
3  bool Graph[MAXN][MAXN];
4  int Match[MAXN];
5  bool InQueue[MAXN], InPath[MAXN], InBlossom[MAXN];
6  int Head, Tail;
7  int Queue[MAXN];
8  int Start, Finish;
9  int NewBase;
10 int Father[MAXN], Base[MAXN];
11 int Count; //匹配数, 匹配对数是 Count/2
12 void CreateGraph(){
13     int u, v;
14     memset(Graph, false, sizeof(Graph));
15     scanf("%d", &N);
16     while(scanf("%d%d", &u, &v) == 2){
17         Graph[u][v] = Graph[v][u] = true;
18     }
19 }
20 void Push(int u){
21     Queue[Tail] = u;
22     Tail++;
23     InQueue[u] = true;
24 }
25 int Pop(){
26     int res = Queue[Head];
27     Head++;
28     return res;
29 }
30 int FindCommonAncestor(int u, int v){
31     memset(InPath, false, sizeof(InPath));
32     while(true){
33         u = Base[u];
34         InPath[u] = true;
35         if(u == Start) break;
36         u = Father[Match[u]];
37     }
38     while(true){
39         v = Base[v];
40         if(InPath[v]) break;
41         v = Father[Match[v]];
42     }
43     return v;
44 }
45 void ResetTrace(int u){
46     int v;
47     while(Base[u] != NewBase){
48         v = Match[u];
49         InBlossom[Base[u]] = InBlossom[Base[v]] = true;
50         u = Father[v];
51         if(Base[u] != NewBase) Father[u] = v;
52     }

```

```

53 }
54 void BlossomContract(int u,int v){
55     NewBase = FindCommonAncestor(u,v);
56     memset(InBlossom,false,sizeof(InBlossom));
57     ResetTrace(u);
58     ResetTrace(v);
59     if(Base[u] != NewBase) Father[u] = v;
60     if(Base[v] != NewBase) Father[v] = u;
61     for(int tu = 1; tu <= N; tu++){
62         if(InBlossom[Base[tu]]){
63             Base[tu] = NewBase;
64             if(!InQueue[tu]) Push(tu);
65         }
66     }
67 void FindAugmentingPath(){
68     memset(InQueue,false,sizeof(InQueue));
69     memset(Father,0,sizeof(Father));
70     for(int i = 1;i <= N;i++){
71         Base[i] = i;
72     }
73     Head = Tail = 1;
74     Push(Start);
75     Finish = 0;
76     while(Head < Tail){
77         int u = Pop();
78         for(int v = 1; v <= N; v++){
79             if(Graph[u][v] && (Base[u] != Base[v]) && (Match[u] !=
              v)){
80                 if((v == Start) || ((Match[v] > 0) && Father[Match[
              v]] > 0))
81                     BlossomContract(u,v);
82                 else if(Father[v] == 0){
83                     Father[v] = u;
84                     if(Match[v] > 0)
85                         Push(Match[v]);
86                     else{
87                         Finish = v;
88                         return;
89                     }
90                 }
91             }
92         }
93     }
94 void AugmentPath(){
95     int u,v,w;
96     u = Finish;
97     while(u > 0){
98         v = Father[u];
99         w = Match[v];
100         Match[v] = u;
101         Match[u] = v;
102         u = w;

```

```

102     }
103 }
104 void Edmonds(){
105     memset(Match,0,sizeof(Match));
106     for(int u = 1; u <= N; u++){
107         if(Match[u] == 0){
108             Start = u;
109             FindAugmentingPath();
110             if(Finish > 0)AugmentPath();
111         }
112     }
113 void PrintMatch(){
114     Count = 0;
115     for(int u = 1; u <= N;u++){
116         if(Match[u] > 0)
117             Count++;
118     }
119     printf("%d\n",Count);
120     for(int u = 1; u <= N; u++){
121         if(u < Match[u])
122             printf("%d_ %d\n",u,Match[u]);
123 }
124 int main(){
125     CreateGraph();//建图
126     Edmonds();//进行匹配
127     PrintMatch();//输出匹配数和匹配
128     return 0;
129 }

```

4.14 一般图最大加权匹配

```

1 //一般图的最大加权匹配模板
2 //注意 G 的初始化，需要偶数个点，刚好可以形成 n/2 个匹配
3 //如果要求最小权匹配，可以取相反数，或者稍加修改就可以了
4 //点的编号从 0 开始的
5 const int MAXN = 110;
6 const int INF = 0x3f3f3f3f;
7 int G[MAXN][MAXN];
8 int cnt_node;//点的个数
9 int dis[MAXN];
10 int match[MAXN];
11 bool vis[MAXN];
12 int sta[MAXN],top;//堆栈
13 bool dfs(int u){
14     sta[top++] = u;
15     if(vis[u])return true;
16     vis[u] = true;
17     for(int i = 0;i < cnt_node;i++){
18         if(i != u && i != match[u] && !vis[i]){
19             int t = match[i];
20             if(dis[t] < dis[u] + G[u][i] - G[i][t]){
21                 dis[t] = dis[u] + G[u][i] - G[i][t];

```

```

22         if(dfs(t))return true;
23     }
24 }
25 top--;
26 vis[u] = false;
27 return false;
28 }
29 int P[MAXN];
30 //返回最大匹配权值
31 int get_Match(int N){
32     cnt_node = N;
33     for(int i = 0;i < cnt_node;i++)P[i] = i;
34     for(int i = 0;i < cnt_node;i += 2){
35         match[i] = i+1;
36         match[i+1] = i;
37     }
38     int cnt = 0;
39     while(1){
40         memset(dis,0,sizeof(dis));
41         memset(vis,false,sizeof(vis));
42         top = 0;
43         bool update = false;
44         for(int i = 0;i < cnt_node;i++){
45             if(dfs(P[i])){
46                 update = true;
47                 int tmp = match[sta[top-1]];
48                 int j = top-2;
49                 while(sta[j] != sta[top-1]){
50                     match[tmp] = sta[j];
51                     swap(tmp,match[sta[j]]);
52                     j--;
53                 }
54                 match[tmp] = sta[j];
55                 match[sta[j]] = tmp;
56                 break;
57             }
58             if(!update){
59                 cnt++;
60                 if(cnt >= 3)break;
61                 random_shuffle(P,P+cnt_node);
62             }
63         }
64         int ans = 0;
65         for(int i = 0;i < cnt_node;i++){
66             int v = match[i];
67             if(i < v)ans += G[i][v];
68         }
69         return ans;
70     }

```


4.15 生成树计数

Matrix-Tree 定理 (Kirchhoff 矩阵-树定理)

1、G 的度数矩阵 $D[G]$ 是一个 $n \times n$ 的矩阵, 并且满足: 当 $i \neq j$ 时, $d_{ij}=0$; 当 $i=j$ 时, d_{ij} 等于 v_i 的度数。

2、G 的邻接矩阵 $A[G]$ 也是一个 $n \times n$ 的矩阵, 并且满足: 如果 v_i 、 v_j 之间有边直接相连, 则 $a_{ij}=1$, 否则为 0。

我们定义 G 的 Kirchhoff 矩阵 (也称为拉普拉斯算子) $C[G]$ 为 $C[G]=D[G]-A[G]$, 则 Matrix-Tree 定理可以描述为:

G 的所有不同的生成树的个数等于其 Kirchhoff 矩阵 $C[G]$ 任何一个 $n-1$ 阶主子式的行列式的绝对值。

所谓 $n-1$ 阶主子式, 就是对于 $r(1 \leq r \leq n)$, 将 $C[G]$ 的第 r 行、第 r 列同时去掉后得到的新矩阵, 用 $Cr[G]$ 表示。

//HDU 4305

//求生成树计数部分代码, 计数对 10007 取模

```

1  const int MOD = 10007;
2  int INV[MOD];
3  //求 ax = 1( mod m) 的 x 值, 就是逆元 (0<a<m)
4  long long inv(long long a,long long m){
5      if(a == 1)return 1;
6      return inv(m%a,m)*(m-m/a)%m;
7  }
8  struct Matrix{
9      int mat[330][330];
10     void init(){
11         memset(mat,0,sizeof(mat));
12     }
13     //求行列式的值模上, 需要使用逆元MOD
14     int det(int n){
15         for(int i = 0;i < n;i++){
16             for(int j = 0;j < n;j++){
17                 mat[i][j] = (mat[i][j]%MOD+MOD)%MOD;
18             }
19             int res = 1;
20             for(int i = 0;i < n;i++){
21                 for(int j = i;j < n;j++){
22                     if(mat[j][i]!=0){
23                         for(int k = i;k < n;k++){
24                             swap(mat[i][k],mat[j][k]);
25                         }
26                         if(i != j)
27                             res = (-res+MOD)%MOD;
28                         break;
29                     }
30                 }
31                 if(mat[i][i] == 0){
32                     res = -1;//不存在也就是行列式值为(0)
33                     break;
34                 }
35                 for(int j = i+1;j < n;j++){
36                     //int mut = (mat[j][i]*INV[mat[i][i]])%MOD打表逆元;
37                     int mut = (mat[j][i]*inv(mat[i][i],MOD))%MOD;

```

```

35         for(int k = i;k < n;k++)
36             mat[j][k] = (mat[j][k]-(mat[i][k]*mut)%MOD+MOD)
37                 %MOD;
38     }
39     res = (res * mat[i][i])%MOD;
40 }
41 return res;
42 }
43 };
44
45 Matrix ret;
46 ret.init();
47 for(int i = 0;i < n;i++)
48     for(int j = 0;j < n;j++)
49         if(i != j && g[i][j]){
50             ret.mat[i][j] = -1;
51             ret.mat[i][i]++;
52         }
53 printf("%d\n",ret.det(n-1));

```

计算生成树个数，不取模，SPOJ 104

```

1  const double eps = 1e-8;
2  const int MAXN = 110;
3  int sgn(double x){
4      if(fabs(x) < eps)return 0;
5      if(x < 0)return -1;
6      else return 1;
7  }
8  double b[MAXN][MAXN];
9  double det(double a[][MAXN],int n){
10     int i, j, k, sign = 0;
11     double ret = 1;
12     for(i = 0;i < n;i++)
13         for(j = 0;j < n;j++)
14             b[i][j] = a[i][j];
15     for(i = 0;i < n;i++){
16         if(sgn(b[i][i]) == 0){
17             for(j = i + 1; j < n;j++)
18                 if(sgn(b[j][i]) != 0)
19                     break;
20             if(j == n)return 0;
21             for(k = i;k < n;k++)
22                 swap(b[i][k],b[j][k]);
23             sign++;
24         }
25         ret *= b[i][i];
26         for(k = i + 1;k < n;k++)
27             b[i][k]/=b[i][i];
28         for(j = i+1;j < n;j++)
29             for(k = i+1;k < n;k++)
30                 b[j][k] -= b[j][i]*b[i][k];
31     }

```

```

32     if(sign & 1)ret = -ret;
33     return ret;
34 }
35 double a[MAXN][MAXN];
36 int g[MAXN][MAXN];
37 int main(){
38     int T;
39     int n,m;
40     int u,v;
41     scanf("%d",&T);
42     while(T--){
43         scanf("%d%d",&n,&m);
44         memset(g,0,sizeof(g));
45         while(m--){
46             scanf("%d%d",&u,&v);
47             u--;v--;
48             g[u][v] = g[v][u] = 1;
49         }
50         memset(a,0,sizeof(a));
51         for(int i = 0;i < n;i++)
52             for(int j = 0;j < n;j++)
53                 if(i != j && g[i][j]){
54                     a[i][i]++;
55                     a[i][j] = -1;
56                 }
57         double ans = det(a,n-1);
58         printf("%.0lf\n",ans);
59     }
60     return 0;
61 }

```

4.16 最大流

4.16.1 SAP 邻接矩阵形式

```

1  /*
2   * SAP 算法（矩阵形式）
3   * 结点编号从 0 开始
4   */
5  const int MAXN=1100;
6  int maze[MAXN][MAXN];
7  int gap[MAXN],dis[MAXN],pre[MAXN],cur[MAXN];
8  int sap(int start,int end,int n){
9      memset(cur,0,sizeof(cur));
10     memset(dis,0,sizeof(dis));
11     memset(gap,0,sizeof(gap));
12     int u=pre[start]=start,maxflow=0,ug=-1;
13     gap[0]=n;
14     while(dis[start]<n){
15         loop:
16         for(int v=cur[u];v<n;v++)

```

```

17         if(maze[u][v] && dis[u]==dis[v]+1){
18             if(aug== -1 || aug>maze[u][v])aug=maze[u][v];
19             pre[v]=u;
20             u=cur[u]=v;
21             if(v==end){
22                 maxflow+=aug;
23                 for(u=pre[u];v!=start;v=u,u=pre[u]){
24                     maze[u][v]-=aug;
25                     maze[v][u]+=aug;
26                 }
27                 aug=-1;
28             }
29             goto loop;
30         }
31         int mindis=nodenum-1;
32         for(int v=0;v<nodenum;v++)
33             if(maze[u][v]&&mindis>dis[v]){
34                 cur[u]=v;
35                 mindis=dis[v];
36             }
37         if((--gap[dis[u]])==0)break;
38         gap[dis[u]=mindis+1]++;
39         u=pre[u];
40     }
41     return maxflow;
42 }

```

4.16.2 SAP 邻接矩阵形式 2

保留原矩阵，可用于多次使用最大流

```

1  /*
2   * SAP 邻接矩阵形式
3   * 点的编号从 0 开始
4   * 增加个 flow 数组，保留原矩阵 maze，可用于多次使用最大流
5   */
6  const int MAXN=1100;
7  int maze[MAXN][MAXN];
8  int gap[MAXN],dis[MAXN],pre[MAXN],cur[MAXN];
9  int flow[MAXN][MAXN]; //存最大流的容量
10 int sap(int start,int end,int nodenum){
11     memset(cur,0,sizeof(cur));
12     memset(dis,0,sizeof(dis));
13     memset(gap,0,sizeof(gap));
14     memset(flow,0,sizeof(flow));
15     int u=pre[start]=start,maxflow=0,aug=-1;
16     gap[0]=nodenum;
17     while(dis[start]<nodenum){
18         loop:
19         for(int v=cur[u];v<nodenum;v++){
20             if(maze[u][v]-flow[u][v] && dis[u]==dis[v]+1){
21                 if(aug== -1 || aug>maze[u][v]-flow[u][v])aug=maze[u][v]-flow[u][v];

```

```

22         pre[v]=u;
23         u=cur[u]=v;
24         if(v==end){
25             maxflow+=aug;
26             for(u=pre[u];v!=start;v=u,u=pre[u]){
27                 flow[u][v]+=aug;
28                 flow[v][u]-=aug;
29             }
30             aug=-1;
31         }
32         goto loop;
33     }
34     int mindis=nodenum-1;
35     for(int v=0;v<nodenum;v++)
36         if(maze[u][v]-flow[u][v]&&mindis>dis[v]){
37             cur[u]=v;
38             mindis=dis[v];
39         }
40     if((--gap[dis[u]])==0)break;
41     gap[dis[u]=mindis+1]++;
42     u=pre[u];
43 }
44 return maxflow;
45 }

```

4.16.3 ISAP 邻接表形式

```

1  const int MAXN = 100010; //点数的最大值
2  const int MAXM = 400010; //边数的最大值
3  const int INF = 0x3f3f3f3f;
4  struct Edge{
5      int to,next,cap,flow;
6  }edge[MAXM]; //注意是 MAXM
7  int tol;
8  int head[MAXN];
9  int gap[MAXN],dep[MAXN],pre[MAXN],cur[MAXN];
10 void init(){
11     tol = 0;
12     memset(head,-1,sizeof(head));
13 }
14 //加边, 单向图三个参数, 双向图四个参数
15 void addedge(int u,int v,int w,int rw=0){
16     edge[tol].to = v;edge[tol].cap = w;edge[tol].next = head[u];
17     edge[tol].flow = 0;head[u] = tol++;
18     edge[tol].to = u;edge[tol].cap = rw;edge[tol].next = head[v];
19     edge[tol].flow = 0;head[v]=tol++;
20 }
21 //输入参数: 起点、终点、点的总数
22 //点的编号没有影响, 只要输入点的总数
23 int sap(int start,int end,int N){
24     memset(gap,0,sizeof(gap));

```

```

25     memset(dep,0,sizeof(dep));
26     memcpy(cur,head,sizeof(head));
27     int u = start;
28     pre[u] = -1;
29     gap[0] = N;
30     int ans = 0;
31     while(dep[start] < N){
32         if(u == end){
33             int Min = INF;
34             for(int i = pre[u]; i != -1; i = pre[edge[i^1].to])
35                 if(Min > edge[i].cap - edge[i].flow)
36                     Min = edge[i].cap - edge[i].flow;
37             for(int i = pre[u]; i != -1; i = pre[edge[i^1].to]){
38                 edge[i].flow += Min;
39                 edge[i^1].flow -= Min;
40             }
41             u = start;
42             ans += Min;
43             continue;
44         }
45         bool flag = false;
46         int v;
47         for(int i = cur[u]; i != -1; i = edge[i].next){
48             v = edge[i].to;
49             if(edge[i].cap - edge[i].flow && dep[v]+1 == dep[u])
50             {
51                 flag = true;
52                 cur[u] = pre[v] = i;
53                 break;
54             }
55         }
56         if(flag){
57             u = v;
58             continue;
59         }
60         int Min = N;
61         for(int i = head[u]; i != -1; i = edge[i].next)
62             if(edge[i].cap - edge[i].flow && dep[edge[i].to] < Min)
63             {
64                 Min = dep[edge[i].to];
65                 cur[u] = i;
66             }
67         gap[dep[u]]--;
68         if(!gap[dep[u]])return ans;
69         dep[u] = Min+1;
70         gap[dep[u]]++;
71         if(u != start) u = edge[pre[u]^1].to;
72     }
73     return ans;
74 }

```

4.16.4 ISAP+bfs 初始化 + 栈优化

```

1  const int MAXN = 100010;//点数的最大值
2  const int MAXM = 400010;//边数的最大值
3  const int INF = 0x3f3f3f3f;
4  struct Edge{
5      int to,next,cap,flow;
6  }edge[MAXM];//注意是 MAXM
7  int tol;
8  int head[MAXN];
9  int gap[MAXN],dep[MAXN],cur[MAXN];
10 void init(){
11     tol = 0;
12     memset(head,-1,sizeof(head));
13 }
14 void addedge(int u,int v,int w,int rw = 0){
15     edge[tol].to = v; edge[tol].cap = w; edge[tol].flow = 0;
16     edge[tol].next = head[u]; head[u] = tol++;
17     edge[tol].to = u; edge[tol].cap = rw; edge[tol].flow = 0;
18     edge[tol].next = head[v]; head[v] = tol++;
19 }
20 int Q[MAXN];
21 void BFS(int start,int end){
22     memset(dep,-1,sizeof(dep));
23     memset(gap,0,sizeof(gap));
24     gap[0] = 1;
25     int front = 0, rear = 0;
26     dep[end] = 0;
27     Q[rear++] = end;
28     while(front != rear){
29         int u = Q[front++];
30         for(int i = head[u]; i != -1; i = edge[i].next){
31             int v = edge[i].to;
32             if(dep[v] != -1)continue;
33             Q[rear++] = v;
34             dep[v] = dep[u] + 1;
35             gap[dep[v]]++;
36         }
37     }
38 }
39 int S[MAXN];
40 int sap(int start,int end,int N){
41     BFS(start,end);
42     memcpy(cur,head,sizeof(head));
43     int top = 0;
44     int u = start;
45     int ans = 0;
46     while(dep[start] < N){
47         if(u == end){
48             int Min = INF;
49             int inser;

```

```

50         for(int i = 0; i < top; i++)
51             if(Min > edge[S[i]].cap - edge[S[i]].flow){
52                 Min = edge[S[i]].cap - edge[S[i]].flow;
53                 inser = i;
54             }
55         for(int i = 0; i < top; i++){
56             edge[S[i]].flow += Min;
57             edge[S[i]^1].flow -= Min;
58         }
59         ans += Min;
60         top = inser;
61         u = edge[S[top]^1].to;
62         continue;
63     }
64     bool flag = false;
65     int v;
66     for(int i = cur[u]; i != -1; i = edge[i].next){
67         v = edge[i].to;
68         if(edge[i].cap - edge[i].flow && dep[v]+1 == dep[u]){
69             flag = true;
70             cur[u] = i;
71             break;
72         }
73     }
74     if(flag){
75         S[top++] = cur[u];
76         u = v;
77         continue;
78     }
79     int Min = N;
80     for(int i = head[u]; i != -1; i = edge[i].next)
81         if(edge[i].cap - edge[i].flow && dep[edge[i].to] < Min)
82             {
83                 Min = dep[edge[i].to];
84                 cur[u] = i;
85             }
86     gap[dep[u]]--;
87     if(!gap[dep[u]]) return ans;
88     dep[u] = Min + 1;
89     gap[dep[u]]++;
90     if(u != start) u = edge[S[top]^1].to;
91 }
92 return ans;
93 }

```

4.16.5 dinic

```

1 const int MAXN = 2010; //点数的最大值
2 const int MAXM = 1200010; //边数的最大值
3 const int INF = 0x3f3f3f3f;
4 struct Edge{
5     int to, next, cap, flow;

```



```

6 }edge[MAXM]; //注意是 MAXM
7 int tol;
8 int head[MAXN];
9 void init(){
10     tol = 2;
11     memset(head, -1, sizeof(head));
12 }
13 void addedge(int u, int v, int w, int rw = 0){
14     edge[tol].to = v; edge[tol].cap = w; edge[tol].flow = 0;
15     edge[tol].next = head[u]; head[u] = tol++;
16     edge[tol].to = u; edge[tol].cap = rw; edge[tol].flow = 0;
17     edge[tol].next = head[v]; head[v] = tol++;
18 }
19 int Q[MAXN];
20 int dep[MAXN], cur[MAXN], sta[MAXN];
21 bool bfs(int s, int t, int n){
22     int front = 0, tail = 0;
23     memset(dep, -1, sizeof(dep[0]) * (n + 1));
24     dep[s] = 0;
25     Q[tail++] = s;
26     while(front < tail){
27         int u = Q[front++];
28         for(int i = head[u]; i != -1; i = edge[i].next){
29             int v = edge[i].to;
30             if(edge[i].cap > edge[i].flow && dep[v] == -1){
31                 dep[v] = dep[u] + 1;
32                 if(v == t) return true;
33                 Q[tail++] = v;
34             }
35         }
36     }
37     return false;
38 }
39 int dinic(int s, int t, int n){
40     int maxflow = 0;
41     while(bfs(s, t, n)){
42         for(int i = 0; i < n; i++) cur[i] = head[i];
43         int u = s, tail = 0;
44         while(cur[s] != -1){
45             if(u == t){
46                 int tp = INF;
47                 for(int i = tail - 1; i >= 0; i--)
48                     tp = min(tp, edge[sta[i]].cap - edge[sta[i]].flow);
49                 maxflow += tp;
50                 for(int i = tail - 1; i >= 0; i--){
51                     edge[sta[i]].flow += tp;
52                     edge[sta[i]^1].flow -= tp;
53                     if(edge[sta[i]].cap - edge[sta[i]].flow == 0)
54                         tail = i;
55                 }

```

```

56         u = edge[sta[tail]^1].to;
57     }
58     else if(cur[u] != -1 && edge[cur[u]].cap > edge[cur[u]
59         ].flow && dep[u] + 1 == dep[edge[cur[u]].to]){
60         sta[tail++] = cur[u];
61         u = edge[cur[u]].to;
62     }
63     else {
64         while(u != s && cur[u] == -1)
65             u = edge[sta[--tail]^1].to;
66         cur[u] = edge[cur[u]].next;
67     }
68 }
69 return maxflow;
70 }

```

4.16.6 最大流判断多解

```

1 //判断最大流多解就是在残留网络中找正环
2 bool vis[MAXN],no[MAXN];
3 int Stack[MAXN],top;
4 bool dfs(int u,int pre,bool flag){
5     vis[u] = 1;
6     Stack[top++] = u;
7     for(int i = head[u];i != -1;i = edge[i].next)
8     {
9         int v = edge[i].to;
10        if(edge[i].cap <= edge[i].flow)continue;
11        if(v == pre)continue;
12        if(!vis[v]){
13            if(dfs(v,u,edge[i^1].flow < edge[i^1].cap))return true;
14        }
15        else if(!no[v])return true;
16    }
17    if(!flag){
18        while(1){
19            int v = Stack[--top];
20            no[v] = true;
21            if(v == u)break;
22        }
23    }
24    return false;
25 }
26 //执行完最大流后可进行调用
27 memset(vis,false,sizeof(vis));
28 memset(no,false,sizeof(no));
29 top = 0;
30 bool flag = dfs(end,end,0);

```

4.17 最小费用最大流

4.17.1 SPFA 版费用流

最小费用最大流，求最大费用只需要取相反数，结果取相反数即可。

点的总数为 N ，点的编号 $0 \sim N-1$

```

1  const int MAXN = 10000;
2  const int MAXM = 100000;
3  const int INF = 0x3f3f3f3f;
4  struct Edge{
5      int to,next,cap,flow,cost;
6  }edge[MAXM];
7  int head[MAXN],tol;
8  int pre[MAXN],dis[MAXN];
9  bool vis[MAXN];
10 int N;//节点总个数，节点编号从 0~N-1
11 void init(int n){
12     N = n;
13     tol = 0;
14     memset(head,-1,sizeof(head));
15 }
16 void addedge(int u,int v,int cap,int cost){
17     edge[tol].to = v;
18     edge[tol].cap = cap;
19     edge[tol].cost = cost;
20     edge[tol].flow = 0;
21     edge[tol].next = head[u];
22     head[u] = tol++;
23     edge[tol].to = u;
24     edge[tol].cap = 0;
25     edge[tol].cost = -cost;
26     edge[tol].flow = 0;
27     edge[tol].next = head[v];
28     head[v] = tol++;
29 }
30 bool spfa(int s,int t){
31     queue<int>q;
32     for(int i = 0;i < N;i++){
33         dis[i] = INF;
34         vis[i] = false;
35         pre[i] = -1;
36     }
37     dis[s] = 0;
38     vis[s] = true;
39     q.push(s);
40     while(!q.empty()){
41         int u = q.front();
42         q.pop();
43         vis[u] = false;
44         for(int i = head[u]; i != -1;i = edge[i].next){
45             int v = edge[i].to;

```

```

46         if(edge[i].cap > edge[i].flow && dis[v] > dis[u] + edge
           [i].cost )
47         {
48             dis[v] = dis[u] + edge[i].cost;
49             pre[v] = i;
50             if(!vis[v]){
51                 vis[v] = true;
52                 q.push(v);
53             }
54         }
55     }
56 }
57 if(pre[t] == -1)return false;
58 else return true;
59 }
60 //返回的是最大流, cost 存的是最小费用
61 int minCostMaxflow(int s,int t,int &cost){
62     int flow = 0;
63     cost = 0;
64     while(spfa(s,t)){
65         int Min = INF;
66         for(int i = pre[t];i != -1;i = pre[edge[i^1].to]){
67             if(Min > edge[i].cap - edge[i].flow)
68                 Min = edge[i].cap - edge[i].flow;
69         }
70         for(int i = pre[t];i != -1;i = pre[edge[i^1].to]){
71             edge[i].flow += Min;
72             edge[i^1].flow -= Min;
73             cost += edge[i].cost * Min;
74         }
75         flow += Min;
76     }
77     return flow;
78 }

```

4.17.2 zkw 费用流

对于二分图类型的比较高效

```

1  const int MAXN = 100;
2  const int MAXM = 20000;
3  const int INF = 0x3f3f3f3f;
4  struct Edge{
5      int to,next,cap,flow,cost;
6      Edge(int _to = 0,int _next = 0,int _cap = 0,int _flow = 0,int
          _cost = 0):
7          to(_to),next(_next),cap(_cap),flow(_flow),cost(_cost){}
8  }edge[MAXN];
9  struct ZKW_MinCostMaxFlow{
10     int head[MAXN],tot;
11     int cur[MAXN];
12     int dis[MAXN];

```

```

13  bool vis[MAXN];
14  int ss,tt,N;//源点、汇点和点的总个数（编号是 0~N-1），不需要额外赋值，
    调用会直接赋值
15  int min_cost, max_flow;
16  void init(){
17      tot = 0;
18      memset(head,-1,sizeof(head));
19  }
20  void addedge(int u,int v,int cap,int cost){
21      edge[tot] = Edge(v,head[u],cap,0,cost);
22      head[u] = tot++;
23      edge[tot] = Edge(u,head[v],0,0,-cost);
24      head[v] = tot++;
25  }
26  int aug(int u,int flow){
27      if(u == tt)return flow;
28      vis[u] = true;
29      for(int i = cur[u];i != -1;i = edge[i].next){
30          int v = edge[i].to;
31          if(edge[i].cap > edge[i].flow && !vis[v] && dis[u] ==
              dis[v] + edge[i].cost){
32              int tmp = aug(v,min(flow,edge[i].cap-edge[i].flow))
              ;
33              edge[i].flow += tmp;
34              edge[i^1].flow -= tmp;
35              cur[u] = i;
36              if(tmp)return tmp;
37          }
38      }
39      return 0;
40  }
41  bool modify_label(){
42      int d = INF;
43      for(int u = 0;u < N;u++){
44          if(vis[u])
45              for(int i = head[u];i != -1;i = edge[i].next){
46                  int v = edge[i].to;
47                  if(edge[i].cap > edge[i].flow && !vis[v])
48                      d = min(d,dis[v]+edge[i].cost-dis[u]);
49              }
50          if(d == INF)return false;
51          for(int i = 0;i < N;i++){
52              if(vis[i]){
53                  vis[i] = false;
54                  dis[i] += d;
55              }
56          }
57          return true;
58      }
59  /*
60      * 直接调用获取最小费用和最大流

```

```

61     * 输入: start-源点, end-汇点, n-点的总个数 (编号从 0 开始)
62     * 返回值: pair<int,int> 第一个是最小费用, 第二个是最大流
63     */
64     pair<int,int> mincostmaxflow(int start,int end,int n){
65         ss = start, tt = end, N = n;
66         min_cost = max_flow = 0;
67         for(int i = 0;i < n;i++)dis[i] = 0;
68         while(1){
69             for(int i = 0;i < n;i++)cur[i] = head[i];
70             while(1){
71                 for(int i = 0;i < n;i++)vis[i] = false;
72                 int tmp = aug(ss,INF);
73                 if(tmp == 0)break;
74                 max_flow += tmp;
75                 min_cost += tmp*dis[ss];
76             }
77             if(!modify_label())break;
78         }
79         return make_pair(min_cost,max_flow);
80     }
81 }solve;

```

4.18 2-SAT

4.18.1 染色法 (可以得到字典序最小的解)

HDU 1814

```

1  const int MAXN = 20020;
2  const int MAXM = 100010;
3  struct Edge
4  {
5      int to,next;
6  }edge[MAXN];
7  int head[MAXN],tot;
8  void init()
9  {
10     tot = 0;
11     memset(head,-1,sizeof(head));
12 }
13 void addedge(int u,int v)
14 {
15     edge[tot].to = v;edge[tot].next = head[u];head[u] = tot++;
16 }
17 bool vis[MAXN]; //染色标记, 为 true 表示选择
18 int S[MAXN],top; //栈
19 bool dfs(int u)
20 {
21     if(vis[u^1])return false;
22     if(vis[u])return true;
23     vis[u] = true;
24     S[top++] = u;

```

```

25     for(int i = head[u]; i != -1; i = edge[i].next)
26         if(!dfs(edge[i].to))
27             return false;
28     return true;
29 }
30 bool Twosat(int n)
31 {
32     memset(vis, false, sizeof(vis));
33     for(int i = 0; i < n; i += 2)
34     {
35         if(vis[i] || vis[i^1]) continue;
36         top = 0;
37         if(!dfs(i))
38         {
39             while(top) vis[S[--top]] = false;
40             if(!dfs(i^1)) return false;
41         }
42     }
43     return true;
44 }
45 int main()
46 {
47     int n, m;
48     int u, v;
49     while(scanf("%d%d", &n, &m) == 2)
50     {
51         init();
52         while(m--)
53         {
54             scanf("%d%d", &u, &v);
55             u--; v--;
56             addedge(u, v^1);
57             addedge(v, u^1);
58         }
59         if(Twosat(2*n))
60         {
61             for(int i = 0; i < 2*n; i++)
62                 if(vis[i])
63                     printf("%d\n", i+1);
64         }
65         else printf("NIE\n");
66     }
67     return 0;
68 }

```

4.18.2 强连通缩点法（拓扑排序只能得到任意解）

POJ 3648 Wedding

```

1 //*****
2 //2-SAT 强连通缩点
3 const int MAXN = 1010;

```

```

4  const int MAXM = 100010;
5  struct Edge
6  {
7      int to,next;
8  }edge[MAXM];
9  int head[MAXN],tot;
10 void init()
11 {
12     tot = 0;
13     memset(head,-1,sizeof(head));
14 }
15 void addedge(int u,int v)
16 {
17     edge[tot].to = v; edge[tot].next = head[u]; head[u] = tot++;
18 }
19 int Low[MAXN],DFN[MAXN],Stack[MAXN],Belong[MAXN]; //Belong 数组的值 1
    ~ scc
20 int Index,top;
21 int scc;
22 bool Instack[MAXN];
23 int num[MAXN];
24 void Tarjan(int u)
25 {
26     int v;
27     Low[u] = DFN[u] = ++Index;
28     Stack[top++] = u;
29     Instack[u] = true;
30     for(int i = head[u]; i != -1; i = edge[i].next)
31     {
32         v = edge[i].to;
33         if( !DFN[v] )
34         {
35             Tarjan(v);
36             if(Low[u] > Low[v])Low[u] = Low[v];
37         }
38         else if(Instack[v] && Low[u] > DFN[v])
39             Low[u] = DFN[v];
40     }
41     if(Low[u] == DFN[u])
42     {
43         scc++;
44         do
45         {
46             v = Stack[--top];
47             Instack[v] = false;
48             Belong[v] = scc;
49             num[scc]++;
50         }
51         while(v != u);
52     }
53 }

```



```

54 bool solvable(int n)//n 是总个数, 需要选择一半
55 {
56     memset(DFN,0,sizeof(DFN));
57     memset(Instack,false,sizeof(Instack));
58     memset(num,0,sizeof(num));
59     Index = scc = top = 0;
60     for(int i = 0;i < n;i++)
61         if(!DFN[i])
62             Tarjan(i);
63     for(int i = 0;i < n;i += 2)
64     {
65         if(Belong[i] == Belong[i^1])
66             return false;
67     }
68     return true;
69 }
70 //*****
71
72 //拓扑排序求任意一组解部分
73 queue<int>q1,q2;
74 vector<vector<int> > dag;//缩点后的逆向 DAG 图
75 char color[MAXN];//染色, 为'R' 是选择的
76 int indeg[MAXN];//入度
77 int cf[MAXN];
78 void solve(int n)
79 {
80     dag.assign(scc+1,vector<int>());
81     memset(indeg,0,sizeof(indeg));
82     memset(color,0,sizeof(color));
83     for(int u = 0;u < n;u++)
84         for(int i = head[u];i != -1;i = edge[i].next)
85         {
86             int v = edge[i].to;
87             if(Belong[u] != Belong[v])
88             {
89                 dag[Belong[v]].push_back(Belong[u]);
90                 indeg[Belong[u]]++;
91             }
92         }
93     for(int i = 0;i < n;i += 2)
94     {
95         cf[Belong[i]] = Belong[i^1];
96         cf[Belong[i^1]] = Belong[i];
97     }
98     while(!q1.empty())q1.pop();
99     while(!q2.empty())q2.pop();
100    for(int i = 1;i <= scc;i++)
101        if(indeg[i] == 0)
102            q1.push(i);
103    while(!q1.empty())
104    {

```

```

105     int u = q1.front();
106     q1.pop();
107     if(color[u] == 0)
108     {
109         color[u] = 'R';
110         color[cf[u]] = 'B';
111     }
112     int sz = dag[u].size();
113     for(int i = 0;i < sz;i++)
114     {
115         indeg[dag[u][i]]--;
116         if(indeg[dag[u][i]] == 0)
117             q1.push(dag[u][i]);
118     }
119 }
120 }
121
122 int change(char s[])
123 {
124     int ret = 0;
125     int i = 0;
126     while(s[i]>='0' && s[i]<='9')
127     {
128         ret *= 10;
129         ret += s[i]-'0';
130         i++;
131     }
132     if(s[i] == 'w')return 2*ret;
133     else return 2*ret+1;
134 }
135 int main()
136 {
137     int n,m;
138     char s1[10],s2[10];
139     while(scanf("%d%d",&n,&m) == 2)
140     {
141         if(n == 0 && m == 0)break;
142         init();
143         while(m--)
144         {
145             scanf("%s%s",s1,s2);
146             int u = change(s1);
147             int v = change(s2);
148             addedge(u^1,v);
149             addedge(v^1,u);
150         }
151         addedge(1,0);
152         if(solvable(2*n))
153         {
154             solve(2*n);
155             for(int i = 1;i < n;i++)

```

```

156         {
157             //注意这一定是判断 color[Belong]
158             if(color[Belong[2*i]] == 'R')printf("%dw",i);
159             else printf("%dh",i);
160             if(i < n-1)printf("_");
161             else printf("\n");
162         }
163     }
164     else printf("bad luck\n");
165 }
166 return 0;
167 }

```

4.19 曼哈顿最小生成树

POJ 3241 求曼哈顿最小生成树上第 k 大的边

```

1  const int MAXN = 100010;
2  const int INF = 0x3f3f3f3f;
3  struct Point{
4      int x,y,id;
5  }p[MAXN];
6  bool cmp(Point a,Point b){
7      if(a.x != b.x) return a.x < b.x;
8      else return a.y < b.y;
9  }
10 //树状数组，找 y-x 大于当前的，但是 y+x 最小的
11 struct BIT{
12     int min_val,pos;
13     void init()
14     {
15         min_val = INF;
16         pos = -1;
17     }
18 }bit[MAXN];
19 //所有有效边
20 struct Edge{
21     int u,v,d;
22 }edge[MAXN<<2];
23 bool cmpedge(Edge a,Edge b){
24     return a.d < b.d;
25 }
26 int tot;
27 int n;
28 int F[MAXN];
29 int find(int x){
30     if(F[x] == -1) return x;
31     else return F[x] = find(F[x]);
32 }
33 void addedge(int u,int v,int d){
34     edge[tot].u = u;
35     edge[tot].v = v;

```

```

36     edge[tot++].d = d;
37 }
38 int lowbit(int x){
39     return x&(-x);
40 }
41 void update(int i,int val,int pos){
42     while(i > 0){
43         if(val < bit[i].min_val){
44             bit[i].min_val = val;
45             bit[i].pos = pos;
46         }
47         i -= lowbit(i);
48     }
49 }
50 //查询 [i,m] 的最小值位置
51 int ask(int i,int m){
52     int min_val = INF,pos = -1;
53     while(i <= m){
54         if(bit[i].min_val < min_val){
55             min_val = bit[i].min_val;
56             pos = bit[i].pos;
57         }
58         i += lowbit(i);
59     }
60     return pos;
61 }
62 int dist(Point a,Point b){
63     return abs(a.x - b.x) + abs(a.y - b.y);
64 }
65 void Manhattan_minimum_spanning_tree(int n,Point p[]){
66     int a[MAXN],b[MAXN];
67     tot = 0;
68     for(int dir = 0; dir < 4;dir++){
69         //4 种坐标变换
70         if(dir == 1 || dir == 3){
71             for(int i = 0;i < n;i++){
72                 swap(p[i].x,p[i].y);
73             }
74         }
75         else if(dir == 2){
76             for(int i = 0;i < n;i++){
77                 p[i].x = -p[i].x;
78             }
79         }
80         sort(p,p+n,cmp);
81         for(int i = 0;i < n;i++){
82             a[i] = b[i] = p[i].y - p[i].x;
83         }
84         sort(b,b+n);
85         int m = unique(b,b+n) - b;
86         for(int i = 1;i <= m;i++){
87             bit[i].init();
88         }
89         for(int i = n-1 ;i >= 0;i--){
90             int pos = lower_bound(b,b+m,a[i]) - b + 1;
91             update(i,a[i],pos);
92         }
93     }
94 }

```

```

87         int ans = ask(pos,m);
88         if(ans != -1)
89             addedge(p[i].id,p[ans].id,dist(p[i],p[ans]));
90         update(pos,p[i].x+p[i].y,i);
91     }
92 }
93 }
94 int solve(int k){
95     Manhattan_minimum_spanning_tree(n,p);
96     memset(F,-1,sizeof(F));
97     sort(edge,edge+tot,cmpedge);
98     for(int i = 0;i < tot;i++){
99         int u = edge[i].u;
100        int v = edge[i].v;
101        int t1 = find(u), t2 = find(v);
102        if(t1 != t2){
103            F[t1] = t2;
104            k--;
105            if(k == 0)return edge[i].d;
106        }
107    }
108 }
109 int main()
110 {
111     //freopen("in.txt","r",stdin);
112     //freopen("out.txt","w",stdout);
113     int k;
114     while(scanf("%d%d",&n,&k)==2 && n){
115         for(int i = 0;i < n;i++){
116             scanf("%d%d",&p[i].x,&p[i].y);
117             p[i].id = i;
118         }
119         printf("%d\n",solve(n-k));
120     }
121     return 0;
122 }

```

4.20 LCA

4.20.1 dfs+ST 在线算法

```

1  /*
2   * LCA (POJ 1330)
3   * 在线算法 DFS + ST
4   */
5  const int MAXN = 10010;
6  int rmq[2*MAXN]; //rmq 数组, 就是欧拉序列对应的深度序列
7  struct ST{
8      int mm[2*MAXN];
9      int dp[2*MAXN][20]; //最小值对应的下标
10     void init(int n){

```

```

11     mm[0] = -1;
12     for(int i = 1; i <= n; i++){
13         mm[i] = ((i & (i-1)) == 0) ? mm[i-1] + 1 : mm[i-1];
14         dp[i][0] = i;
15     }
16     for(int j = 1; j <= mm[n]; j++){
17         for(int i = 1; i + (1 << j) - 1 <= n; i++){
18             dp[i][j] = rmq[dp[i][j-1]] < rmq[dp[i+(1<<(j-1))][j-1]] ? dp[i][j-1] : dp[i+(1<<(j-1))][j-1];
19         }
20         //查询 [a,b] 之间最小值的下标
21         int query(int a, int b)
22         {
23             if(a > b) swap(a, b);
24             int k = mm[b-a+1];
25             return rmq[dp[a][k]] <= rmq[dp[b-(1<<k)+1][k]] ? dp[a][k] : dp[b-(1<<k)+1][k];
26         }
27     };
28     //边的结构体定义
29     struct Edge{
30         int to, next;
31     };
32     Edge edge[MAXN*2];
33     int tot, head[MAXN];
34
35     int F[MAXN*2]; //欧拉序列, 就是 dfs 遍历的顺序, 长度为 2*n-1, 下标从 1 开始
36     int P[MAXN]; //P[i] 表示点 i 在 F 中第一次出现的位置
37     int cnt;
38     ST st;
39     void init(){
40         tot = 0;
41         memset(head, -1, sizeof(head));
42     }
43     //加边, 无向边需要加两次
44     void addedge(int u, int v){
45         edge[tot].to = v;
46         edge[tot].next = head[u];
47         head[u] = tot++;
48     }
49     void dfs(int u, int pre, int dep){
50         F[++cnt] = u;
51         rmq[cnt] = dep;
52         P[u] = cnt;
53         for(int i = head[u]; i != -1; i = edge[i].next){
54             int v = edge[i].to;
55             if(v == pre) continue;
56             dfs(v, u, dep+1);
57             F[++cnt] = u;
58             rmq[cnt] = dep;
59         }

```

```

60 }
61 //查询 LCA 前的初始化
62 void LCA_init(int root,int node_num){
63     cnt = 0;
64     dfs(root,root,0);
65     st.init(2*node_num-1);
66 }
67 //查询 u,v 的 lca 编号
68 int query_lca(int u,int v){
69     return F[st.query(P[u],P[v])];
70 }
71 bool flag[MAXN];
72 int main(){
73     int T;
74     int N;
75     int u,v;
76     scanf("%d",&T);
77     while(T--){
78         scanf("%d",&N);
79         init();
80         memset(flag,false,sizeof(flag));
81         for(int i = 1; i < N;i++){
82             scanf("%d%d",&u,&v);
83             addedge(u,v);
84             addedge(v,u);
85             flag[v] = true;
86         }
87         int root;
88         for(int i = 1; i <= N;i++){
89             if(!flag[i]){
90                 root = i;
91                 break;
92             }
93         }
94         LCA_init(root,N);
95         scanf("%d%d",&u,&v);
96         printf("%d\n",query_lca(u,v));
97     }
98     return 0;
99 }

```

4.20.2 离线 Tarjan 算法

```

1  /*
2   * POJ 1470
3   * 给出一颗有向树，Q 个查询
4   * 输出查询结果中每个点出现次数
5   */
6  /*
7   * 离线算法，LCATarjan
8   * 复杂度O(n+Q);
9   */
10 const int MAXN = 1010;

```

```

11 const int MAXQ = 500010;//查询数的最大值
12
13 //并查集部分
14 int F[MAXN];//需要初始化为 -1
15 int find(int x){
16     if(F[x] == -1)return x;
17     return F[x] = find(F[x]);
18 }
19 void bing(int u,int v){
20     int t1 = find(u);
21     int t2 = find(v);
22     if(t1 != t2)
23         F[t1] = t2;
24 }
25 //*****
26 bool vis[MAXN];//访问标记
27 int ancestor[MAXN];//祖先
28 struct Edge{
29     int to,next;
30 }edge[MAXN*2];
31 int head[MAXN],tot;
32 void addedge(int u,int v){
33     edge[tot].to = v;
34     edge[tot].next = head[u];
35     head[u] = tot++;
36 }
37
38 struct Query{
39     int q,next;
40     int index;//查询编号
41 }query[MAXQ*2];
42 int answer[MAXQ];//存储最后的查询结果, 下标 0 Q-1
43 int h[MAXQ];
44 int tt;
45 int Q;
46
47 void add_query(int u,int v,int index){
48     query[tt].q = v;
49     query[tt].next = h[u];
50     query[tt].index = index;
51     h[u] = tt++;
52     query[tt].q = u;
53     query[tt].next = h[v];
54     query[tt].index = index;
55     h[v] = tt++;
56 }
57
58 void init(){
59     tot = 0;
60     memset(head,-1,sizeof(head));
61     tt = 0;

```



```

62     memset(h,-1,sizeof(h));
63     memset(vis,false,sizeof(vis));
64     memset(F,-1,sizeof(F));
65     memset(ancestor,0,sizeof(ancestor));
66 }
67 void LCA(int u){
68     ancestor[u] = u;
69     vis[u] = true;
70     for(int i = head[u];i != -1;i = edge[i].next){
71         int v = edge[i].to;
72         if(vis[v])continue;
73         LCA(v);
74         bing(u,v);
75         ancestor[find(u)] = u;
76     }
77     for(int i = h[u];i != -1;i = query[i].next){
78         int v = query[i].q;
79         if(vis[v]){
80             answer[query[i].index] = ancestor[find(v)];
81         }
82     }
83 }
84 bool flag[MAXN];
85 int Count_num[MAXN];
86 int main(){
87     int n;
88     int u,v,k;
89     while(scanf("%d",&n) == 1){
90         init();
91         memset(flag,false,sizeof(flag));
92         for(int i = 1;i <= n;i++){
93             scanf("%d:(%d",&u,&k);
94             while(k--){
95                 scanf("%d",&v);
96                 flag[v] = true;
97                 addedge(u,v);
98                 addedge(v,u);
99             }
100         }
101         scanf("%d",&Q);
102         for(int i = 0;i < Q;i++){
103             char ch;
104             cin>>ch;
105             scanf("%d_%d",&u,&v);
106             add_query(u,v,i);
107         }
108         int root;
109         for(int i = 1;i <= n;i++){
110             if(!flag[i]){
111                 root = i;
112                 break;

```

```

113         }
114         LCA(root);
115         memset(Count_num,0,sizeof(Count_num));
116         for(int i = 0;i < Q;i++)
117             Count_num[answer[i]]++;
118         for(int i = 1;i <= n;i++)
119             if(Count_num[i] > 0)
120                 printf("%d:%d\n",i,Count_num[i]);
121     }
122     return 0;
123 }

```

4.20.3 LCA 倍增法

```

1  /*
2   * POJ 1330
3   * LCA 在线算法
4   */
5  const int MAXN = 10010;
6  const int DEG = 20;
7
8  struct Edge{
9      int to,next;
10 }edge[MAXN*2];
11 int head[MAXN],tot;
12 void addedge(int u,int v){
13     edge[tot].to = v;
14     edge[tot].next = head[u];
15     head[u] = tot++;
16 }
17 void init(){
18     tot = 0;
19     memset(head,-1,sizeof(head));
20 }
21 int fa[MAXN][DEG]; //fa[i][j] 表示结点 i 的第2^j个祖先
22 int deg[MAXN]; //深度数组
23
24 void BFS(int root){
25     queue<int>que;
26     deg[root] = 0;
27     fa[root][0] = root;
28     que.push(root);
29     while(!que.empty()){
30         int tmp = que.front();
31         que.pop();
32         for(int i = 1;i < DEG;i++)
33             fa[tmp][i] = fa[fa[tmp][i-1]][i-1];
34         for(int i = head[tmp]; i != -1;i = edge[i].next){
35             int v = edge[i].to;
36             if(v == fa[tmp][0]) continue;
37             deg[v] = deg[tmp] + 1;
38             fa[v][0] = tmp;

```

```

39         que.push(v);
40     }
41
42 }
43 }
44 int LCA(int u,int v){
45     if(deg[u] > deg[v])swap(u,v);
46     int hu = deg[u], hv = deg[v];
47     int tu = u, tv = v;
48     for(int det = hv-hu, i = 0; det ;det>>=1, i++)
49         if(det&1)
50             tv = fa[tv][i];
51     if(tu == tv)return tu;
52     for(int i = DEG-1; i >= 0; i--){
53         if(fa[tu][i] == fa[tv][i])
54             continue;
55         tu = fa[tu][i];
56         tv = fa[tv][i];
57     }
58     return fa[tu][0];
59 }
60 bool flag[MAXN];
61 int main(){
62     int T;
63     int n;
64     int u,v;
65     scanf("%d",&T);
66     while(T--){
67         scanf("%d",&n);
68         init();
69         memset(flag,false,sizeof(flag));
70         for(int i = 1;i < n;i++){
71             scanf("%d%d",&u,&v);
72             addedge(u,v);
73             addedge(v,u);
74             flag[v] = true;
75         }
76         int root;
77         for(int i = 1;i <= n;i++){
78             if(!flag[i]){
79                 root = i;
80                 break;
81             }
82         }
83         BFS(root);
84         scanf("%d%d",&u,&v);
85         printf("%d\n",LCA(u,v));
86     }
87     return 0;
88 }

```

4.21 欧拉路

欧拉回路：每条边只经过一次，而且回到起点

欧拉路径：每条边只经过一次，不要求回到起点

欧拉回路判断：

无向图：连通（不考虑度为 0 的点），每个顶点度数都为偶数。

有向图：基图连通（把边当成无向边，同样不考虑度为 0 的点），每个顶点出度等于入度。

混合图（有无向边和有向边）：首先是基图连通（不考虑度为 0 的点），然后需要借助网络流判定。

首先给原图中的每条无向边随便指定一个方向（称为初始定向），将原图改为有向图 G' ，然后任务就是改变 G' 中某些边的方向（当然是无向边转化来的，原混合图中的有向边不能动）使其满足每个点的入度等于出度。

设 $D[i]$ 为 G' 中（点 i 的出度 - 点 i 的入度）。可以发现，在改变 G' 中边的方向的过程中，任何点的 D 值的奇偶性都不会发生改变（设将边 $\langle i, j \rangle$ 改为 $\langle j, i \rangle$ ，则 i 入度加 1 出度减 1， j 入度减 1 出度加 1，两者之差加 2 或减 2，奇偶性不变）！而最终要求的是每个点的入度等于出度，即每个点的 D 值都为 0，是偶数，故可得：若初始定向得到的 G' 中任意一个点的 D 值是奇数，那么原图中一定不存在欧拉环！

若初始 D 值都是偶数，则将 G' 改装成网络：设立源点 S 和汇点 T ，对于每个 $D[i] > 0$ 的点 i ，连边 $\langle S, i \rangle$ ，容量为 $D[i]/2$ ；对于每个 $D[j] < 0$ 的点 j ，连边 $\langle j, T \rangle$ ，容量为 $-D[j]/2$ ； G' 中的每条边在网络中仍保留，容量为 1（表示该边最多只能被改变方向一次）。求这个网络的最大流，若 S 引出的所有边均满流，则原混合图是欧拉图，将网络中所有流量为 1 的中间边（就是不与 S 或 T 关联的边）在 G' 中改变方向，形成的新图 G'' 一定是有向欧拉图；若 S 引出的边中有的没有满流，则原混合图不是欧拉图。

欧拉路径的判断：

无向图：连通（不考虑度为 0 的点），每个顶点度数都为偶数或者仅有两个点的度数为偶数。

有向图：基图连通（把边当成无向边，同样不考虑度为 0 的点），每个顶点出度等于入度或者有且仅有一个点的出度比入度多 1，有且仅有一个点的出度比入度少 1，其余出度等于入度。

混合图：如果存在欧拉回路，一点存在欧拉路径了。否则如果有且仅有两个点的（出度 - 入度）是奇数，那么给这两个点加边，判断是否存在欧拉回路。

4.21.1 有向图

POJ 2337

给出 n 个小写字母组成的单词，要求将 n 个单词连接起来，使得前一个单词的最后一个字母和后一个单词的第一个字母相同。输出字典序最小的解。

```

1 struct Edge{
2     int to,next;
3     int index;
4     bool flag;
5 }edge[2010];
6 int head[30],tot;
7 void init(){
8     tot = 0;
9     memset(head,-1,sizeof(head));
10 }
11 void addedge(int u,int v,int index){
12     edge[tot].to = v;
```

```

13     edge[tot].next = head[u];
14     edge[tot].index = index;
15     edge[tot].flag = false;
16     head[u] = tot++;
17 }
18 string str[1010];
19 int in[30],out[30];
20 int cnt;
21 int ans[1010];
22 void dfs(int u){
23     for(int i = head[u] ;i != -1;i = edge[i].next)
24         if(!edge[i].flag)
25         {
26             edge[i].flag = true;
27             dfs(edge[i].to);
28             ans[cnt++] = edge[i].index;
29         }
30 }
31 int main(){
32     int T,n;
33     scanf("%d",&T);
34     while(T--){
35         scanf("%d",&n);
36         for(int i = 0;i < n;i++)
37             cin>>str[i];
38         sort(str,str+n);//要输出字典序最小的解，先按照字典序排序
39         init();
40         memset(in,0,sizeof(in));
41         memset(out,0,sizeof(out));
42         int start = 100;
43         for(int i = n-1;i >= 0;i--)//字典序大的先加入
44         {
45             int u = str[i][0] - 'a';
46             int v = str[i][str[i].length() - 1] - 'a';
47             addedge(u,v,i);
48             out[u]++;
49             in[v]++;
50             if(u < start)start = u;
51             if(v < start)start = v;
52         }
53         int cc1 = 0, cc2 = 0;
54         for(int i = 0;i < 26;i++){
55             if(out[i] - in[i] == 1){
56                 cc1++;
57                 start = i;//如果有一个出度比入度大 1 的点，就从这个点出发，
否则从最小的点出发
58             }
59             else if(out[i] - in[i] == -1)
60                 cc2++;
61             else if(out[i] - in[i] != 0)
62                 cc1 = 3;

```

```

63     }
64     if(! ( (cc1 == 0 && cc2 == 0) || (cc1 == 1 && cc2 == 1) )){
65         printf("***\n");
66         continue;
67     }
68     cnt = 0;
69     dfs(start);
70     if(cnt != n)//判断是否连通
71     {
72         printf("***\n");
73         continue;
74     }
75     for(int i = cnt-1; i >= 0; i--){
76         cout<<str[ans[i]];
77         if(i > 0)printf(".");
78         else printf("\n");
79     }
80 }
81 return 0;
82 }

```

4.21.2 无向图

SGU 101

```

1 struct Edge{
2     int to,next;
3     int index;
4     int dir;
5     bool flag;
6 }edge[220];
7 int head[10],tot;
8 void init(){
9     memset(head,-1,sizeof(head));
10    tot = 0;
11 }
12 void addedge(int u,int v,int index){
13     edge[tot].to = v;
14     edge[tot].next = head[u];
15     edge[tot].index = index;
16     edge[tot].dir = 0;
17     edge[tot].flag = false;
18     head[u] = tot++;
19     edge[tot].to = u;
20     edge[tot].next = head[v];
21     edge[tot].index = index;
22     edge[tot].dir = 1;
23     edge[tot].flag = false;
24     head[v] = tot++;
25 }
26 int du[10];
27 vector<int>ans;

```

```

28 void dfs(int u){
29     for(int i = head[u]; i != -1; i = edge[i].next)
30         if(!edge[i].flag ){
31             edge[i].flag = true;
32             edge[i^1].flag = true;
33             dfs(edge[i].to);
34             ans.push_back(i);
35         }
36 }
37 int main(){
38     int n;
39     while(scanf("%d",&n) == 1){
40         init();
41         int u,v;
42         memset(du,0,sizeof(du));
43         for(int i = 1;i <= n;i++){
44             scanf("%d%d",&u,&v);
45             addedge(u,v,i);
46             du[u]++;
47             du[v]++;
48         }
49         int s = -1;
50         int cnt = 0;
51         for(int i = 0;i <= 6;i++){
52             if(du[i]&1) {cnt++; s = i;}
53             if(du[i] > 0 && s == -1)
54                 s = i;
55         }
56         bool ff = true;
57         if(cnt != 0 && cnt != 2){
58             printf("No solution\n");
59             continue;
60         }
61         ans.clear();
62         dfs(s);
63         if(ans.size() != n){
64             printf("No solution\n");
65             continue;
66         }
67         for(int i = 0;i < ans.size();i++){
68             printf("%d_",edge[ans[i]].index);
69             if(edge[ans[i]].dir == 0)printf("-\n");
70             else printf("+\n");
71         }
72     }
73     return 0;
74 }

```

4.21.3 混合图

POJ 1637 （本题保证了连通，故不需要判断连通，否则要判断连通）

```

1  const int MAXN = 210;
2  //最大流部分ISAP
3  const int MAXM = 20100;
4  const int INF = 0x3f3f3f3f;
5  struct Edge
6  {
7      int to,next,cap,flow;
8  }edge[MAXN];
9  int tol;
10 int head[MAXN];
11 int gap[MAXN],dep[MAXN],pre[MAXN],cur[MAXN];
12 void init()
13 {
14     tol = 0;
15     memset(head,-1,sizeof(head));
16 }
17 void addedge(int u,int v,int w,int rw = 0)
18 {
19     edge[tol].to = v;
20     edge[tol].cap = w;
21     edge[tol].next = head[u];
22     edge[tol].flow = 0;
23     head[u] = tol++;
24     edge[tol].to = u;
25     edge[tol].cap = rw;
26     edge[tol].next = head[v];
27     edge[tol].flow = 0;
28     head[v] = tol++;
29 }
30 int sap(int start,int end,int N)
31 {
32     memset(gap,0,sizeof(gap));
33     memset(dep,0,sizeof(dep));
34     memcpy(cur,head,sizeof(head));
35     int u = start;
36     pre[u] = -1;
37     gap[0] = N;
38     int ans = 0;
39     while(dep[start] < N)
40     {
41         if(u == end)
42         {
43             int Min = INF;
44             for(int i = pre[u]; i != -1; i = pre[edge[i^1].to])
45                 if(Min > edge[i].cap - edge[i].flow)
46                     Min = edge[i].cap - edge[i].flow;
47             for(int i = pre[u]; i != -1; i = pre[edge[i^1].to])
48             {
49                 edge[i].flow += Min;
50                 edge[i^1].flow -= Min;
51             }

```



```

52         u = start;
53         ans += Min;
54         continue;
55     }
56     bool flag = false;
57     int v;
58     for(int i = cur[u]; i != -1; i = edge[i].next)
59     {
60         v = edge[i].to;
61         if(edge[i].cap - edge[i].flow && dep[v] + 1 == dep[u])
62         {
63             flag = true;
64             cur[u] = pre[v] = i;
65             break;
66         }
67     }
68     if(flag)
69     {
70         u = v;
71         continue;
72     }
73     int Min = N;
74     for(int i = head[u]; i != -1; i = edge[i].next)
75         if(edge[i].cap - edge[i].flow && dep[edge[i].to] < Min)
76         {
77             Min = dep[edge[i].to];
78             cur[u] = i;
79         }
80     gap[dep[u]]--;
81     if(!gap[dep[u]]) return ans;
82     dep[u] = Min+1;
83     gap[dep[u]]++;
84     if(u != start) u = edge[pre[u]^1].to;
85 }
86 return ans;
87 }
88 //the end of 最大流部分
89
90 int in[MAXN], out[MAXN]; //每个点的出度和入度
91
92 int main()
93 {
94     //freopen("in.txt", "r", stdin);
95     //freopen("out.txt", "w", stdout);
96     int T;
97     int n, m;
98     scanf("%d", &T);
99     while(T--)
100     {
101         scanf("%d%d", &n, &m);
102         init();

```

```

103     int u,v,w;
104     memset(in,0,sizeof(in));
105     memset(out,0,sizeof(out));
106     while(m--)
107     {
108         scanf("%d%d%d",&u,&v,&w);
109         out[u]++; in[v]++;
110         if(w == 0)//双向
111             addedge(u,v,1);
112     }
113     bool flag = true;
114     for(int i = 1;i <= n;i++)
115     {
116         if(out[i] - in[i] > 0)
117             addedge(0,i,(out[i] - in[i])/2);
118         else if(in[i] - out[i] > 0)
119             addedge(i,n+1,(in[i] - out[i])/2);
120         if((out[i] - in[i]) & 1)
121             flag = false;
122     }
123     if(!flag)
124     {
125         printf("impossible\n");
126         continue;
127     }
128     sap(0,n+1,n+2);
129     for(int i = head[0]; i != -1;i = edge[i].next)
130         if(edge[i].cap > 0 && edge[i].cap > edge[i].flow)
131         {
132             flag = false;
133             break;
134         }
135     if(flag)printf("possible\n");
136     else printf("impossible\n");
137 }
138 return 0;
139 }

```

4.22 树分治

4.22.1 点分治 -HDU5016

HDU 5016 给定一个边权树，初始时，一些结点上已经建立了市场。每个结点会被距离自己最近的市场所支配（距离相同时，被标号最小的市场支配）可以新建一个市场，问新建的市场最多可以支配多少点。

```

1  const int MAXN = 100010;
2  const int INF = 0x3f3f3f3f;
3  struct Edge{
4      int to,next,w;
5  }edge[MAXN*2];
6  int head[MAXN],tot;

```

```

7 void init(){
8     tot = 0;
9     memset(head,-1,sizeof(head));
10 }
11 inline void addedge(int u,int v,int w){
12     edge[tot].to = v;
13     edge[tot].w = w;
14     edge[tot].next = head[u];
15     head[u] = tot++;
16 }
17 int size[MAXN],vis[MAXN],fa[MAXN],que[MAXN];
18 int TT;//时间戳
19 //找重心
20 inline int getroot(int u){
21     int Min = MAXN, root = 0;
22     int l,r;
23     que[l = r = 1] = u;
24     fa[u] = 0;
25     for(;l <= r;l++){
26         for(int i = head[que[l]]; i != -1;i = edge[i].next){
27             int v = edge[i].to;
28             if(v == fa[que[l]] || vis[v] == TT)continue;
29             que[++r] = v;
30             fa[v] = que[l];
31         }
32         for(l--;l;l--){
33             int x = que[l], Max = 0;
34             size[x] = 1;
35             for(int i = head[x];i != -1;i = edge[i].next){
36                 int v = edge[i].to;
37                 if(v == fa[x] || vis[v] == TT)continue;
38                 Max = max(Max,size[v]);
39                 size[x] += size[v];
40             }
41             Max = max(Max,r - size[x]);
42             if(Max < Min){
43                 Min = Max; root = x;
44             }
45         }
46     }
47     return root;
48 }
49 int ans[MAXN];
50 pair<int,int>pp[MAXN];
51 pair<int,int>np[MAXN];
52 int dis[MAXN];
53 int type[MAXN];
54 inline void go(int u,int pre,int w,int tt){
55     int l,r;
56     que[l = r = 1] = u;
57     fa[u] = pre; dis[u] = w;

```

```

58     for(;l <= r;l++)
59         for(int i = head[que[l]];i != -1;i = edge[i].next){
60             int v = edge[i].to;
61             if(v == fa[que[l]] || vis[v] == TT)continue;
62             que[++r] = v;
63             fa[v] = que[l];
64             dis[v] = dis[que[l]] + edge[i].w;
65         }
66     int cnt = 0;
67     for(int i = 1;i <= r;i++){
68         int x = que[i];
69         pp[cnt++] = make_pair(np[x].first-dis[x],np[x].second);
70     }
71     sort(pp,pp+cnt);
72     for(int i = 1;i <= r;i++){
73         int x = que[i];
74         if(type[x])continue;
75         int id = lower_bound(pp,pp+cnt,make_pair(dis[x],x)) - pp;
76         ans[x] += (cnt-id)*tt;
77     }
78 }
79 void solve(int u){
80     int root = getroot(u);
81     vis[root] = TT;
82     go(root,0,0,1);
83     for(int i = head[root];i != -1;i = edge[i].next){
84         int v = edge[i].to;
85         if(vis[v] == TT)continue;
86         go(v,root,edge[i].w,-1);
87     }
88     for(int i = head[root];i != -1;i = edge[i].next){
89         int v = edge[i].to;
90         if(vis[v] == TT)continue;
91         solve(v);
92     }
93 }
94 bool ff[MAXN];
95 int main()
96 {
97     int n;
98     memset(vis,0,sizeof(vis));
99     TT = 0;
100    while(scanf("%d",&n) == 1){
101        init();
102        int u,v,w;
103        for(int i = 1;i < n;i++){
104            scanf("%d%d%d",&u,&v,&w);
105            addedge(u,v,w);
106            addedge(v,u,w);
107        }
108        for(int i = 1;i <= n;i++)scanf("%d",&type[i]);

```

```

109     queue<int>q;
110     for(int i = 1;i <= n;i++){
111         if(type[i]){
112             np[i] = make_pair(0,i);
113             ff[i] = true;
114             q.push(i);
115         }
116         else{
117             np[i] = make_pair(INF,0);
118             ff[i] = false;
119         }
120     }
121     while(!q.empty()){
122         u = q.front();
123         q.pop();
124         ff[u] = false;
125         for(int i = head[u];i != -1;i = edge[i].next){
126             v = edge[i].to;
127             pair<int,int>tmp = make_pair(np[u].first+edge[i].w,
128                                     np[u].second);
129             if(tmp < np[v]){
130                 np[v] = tmp;
131                 if(!ff[v]){
132                     ff[v] = true;
133                     q.push(v);
134                 }
135             }
136         }
137         TT++;
138         for(int i = 1;i <= n;i++)ans[i] = 0;
139         solve(1);
140         int ret = 0;
141         for(int i = 1;i <= n;i++)ret = max(ret,ans[i]);
142         printf("%d\n",ret);
143     }
144     return 0;
145 }

```

4.22.2 * 点分治 -HDU4918

HDU 4918

题意：给出一颗 n 个点的树，每个点有一个权值，有两种操作，一种是将某个点的权值修改为 v ，另一种是查询距离点 u 不超过 d 的点的权值和。

```

1  const int MAXN = 100010;
2  const int MAXD = 40;
3  int cc[MAXN*MAXD];
4  int *cc_tail; //记得初始化 cc_tail = cc
5  //0-based BinaryIndexTree
6  struct BIT{
7      int *c;

```

```

8      int n;
9      void init(int _n){
10         n = _n;
11         c = cc_tail;
12         cc_tail = cc_tail + n;
13         memset(c,0,sizeof(int)*n);
14     }
15     void add(int i,int val){
16         while(i < n){
17             c[i] += val;
18             i += ~i & i+1;
19         }
20     }
21     int sum(int i){
22         i = min(i,n-1);
23         int s = 0;
24         while(i >= 0){
25             s += c[i];
26             i -= ~i & i+1;
27         }
28         return s;
29     }
30 }bits[MAXN<<1];
31 struct Edge{
32     int to,next;
33 }edge[MAXN*2];
34 int head[MAXN],tot;
35 void init(){
36     tot = 0;
37     memset(head,-1,sizeof(head));
38 }
39 inline void addedge(int u,int v){
40     edge[tot].to = v;
41     edge[tot].next = head[u];
42     head[u] = tot++;
43 }
44 int size[MAXN],vis[MAXN],fa[MAXN],que[MAXN];
45 int TT;
46 inline int getroot(int u,int &tot){
47     int Min = MAXN, root = 0;
48     int l,r;
49     que[l = r = 1] = u;
50     fa[u] = 0;
51     for(;l <= r;l++){
52         for(int i = head[que[l]];i != -1;i = edge[i].next){
53             int v = edge[i].to;
54             if(v == fa[que[l]] || vis[v] == TT)continue;
55             que[++r] = v;
56             fa[v] = que[l];
57         }
58     }
    tot = r;

```

```

59     for(l--;l;l++){
60         int x = que[l], Max = 0;
61         size[x] = 1;
62         for(int i = head[x];i != -1;i = edge[i].next){
63             int v = edge[i].to;
64             if(v == fa[x] || vis[v] == TT)continue;
65             Max = max(Max,size[v]);
66             size[x] += size[v];
67         }
68         Max = max(Max,r - size[x]);
69         if(Max < Min){
70             Min = Max, root = x;
71         }
72     }
73     return root;
74 }
75 struct Node{
76     int root,subtree,dis;
77     Node(int _root = 0, int _subtree = 0,int _dis = 0){
78         root = _root;
79         subtree = _subtree;
80         dis = _dis;
81     }
82 };
83 vector<Node>vec[MAXN];
84 int id[MAXN];
85 int dist[MAXN];
86 int val[MAXN];
87 int cnt;
88 inline void go(int u,int root,int subtree){
89     int l,r;
90     que[l = r = 1] = u;
91     fa[u] = 0; dist[u] = 1;
92     for(; l <= r;l++){
93         u = que[l];
94         vec[u].push_back(Node(id[root],subtree,dist[u]));
95         for(int i = head[u];i != -1;i = edge[i].next){
96             int v = edge[i].to;
97             if(v == fa[u] || vis[v] == TT)continue;
98             que[++r] = v;
99             fa[v] = u;
100             dist[v] = dist[u] + 1;
101         }
102     }
103     bits[subtree].init(r+1);
104     for(int i = 1;i <= r;i++){
105         u = que[i];
106         bits[id[root]].add(dist[u],val[u]);
107         bits[subtree].add(dist[u],val[u]);
108     }
109 }

```

```

110 void solve(int u){
111     int tot;
112     int root = getroot(u,tot);
113     vis[root] = TT;
114     id[root] = cnt++;
115     vec[root].push_back(Node(id[root],-1,0));
116     bits[id[root]].init(tot);
117     bits[id[root]].add(0,val[root]);
118     for(int i = head[root];i != -1;i = edge[i].next){
119         int v = edge[i].to;
120         if(vis[v] == TT)continue;
121         go(v,root,cnt);
122         cnt++;
123     }
124     for(int i = head[root];i != -1;i = edge[i].next){
125         int v = edge[i].to;
126         if(vis[v] == TT)continue;
127         solve(v);
128     }
129 }
130 int main(){
131     int n,m;
132     memset(vis,0,sizeof(vis));
133     TT = 0;
134     while(scanf("%d%d",&n,&m) == 2){
135         init();
136         TT++;
137         cc_tail = cc;
138         cnt = 0;
139         for(int i = 1;i <= n;i++)vec[i].clear();
140         for(int i = 1;i <= n;i++)scanf("%d",&val[i]);
141         int u,v;
142         for(int i = 1;i < n;i++){
143             scanf("%d%d",&u,&v);
144             addedge(u,v);
145             addedge(v,u);
146         }
147         solve(1);
148         char op[10];
149         int d;
150         while(m--){
151             scanf("%s%d%d",op,&u,&d);
152             if(op[0] == '!'){
153                 int dv = d - val[u];
154                 int sz = vec[u].size();
155                 for(int i = 0;i < sz;i++){
156                     Node tmp = vec[u][i];
157                     bits[tmp.root].add(tmp.dis,dv);
158                     if(tmp.subtree != -1)
159                         bits[tmp.subtree].add(tmp.dis,dv);
160                 }

```



```

161         val[u] += dv;
162     }
163     else {
164         int ans = 0;
165         int sz = vec[u].size();
166         for(int i = 0; i < sz; i++){
167             Node tmp = vec[u][i];
168             ans += bits[tmp.root].sum(d-tmp.dis);
169             if(tmp.subtree != -1)
170                 ans -= bits[tmp.subtree].sum(d-tmp.dis);
171         }
172         printf("%d\n", ans);
173     }
174 }
175 }
176 return 0;
177 }

```

4.22.3 链分治 -HDU5039

HDU 5039 一颗树，每条边的属性为 0 或 1，求有多少条路径经过奇数条属性为 1 的边。一种是查询操作，一种是修改边的属性。

虽然这题有更加简单的方法，但是用来练习链分治还是不错的。

```

1  const int MAXN = 30010;
2  const int INF = 0x3f3f3f3f;
3  struct Edge{
4      int to,next;
5      int f;
6  }edge[MAXN*2];
7  int head[MAXN],tot;
8  void init(){
9      tot = 0;
10     memset(head,-1,sizeof(head));
11 }
12 void addedge(int u,int v,int f){
13     edge[tot].to = v;
14     edge[tot].next = head[u];
15     edge[tot].f = f;
16     head[u] = tot++;
17 }
18 long long ans;
19 int num0[MAXN],num1[MAXN];
20 long long tnum[MAXN];
21 struct Node{
22     int l0,l1;
23     int r0,r1;
24     int cc;
25     long long sum;
26     Node gao(int u){
27         l0 = r0 = num0[u];
28         l1 = r1 = num1[u];

```

```

29         sum = tnum[u];
30         cc = 0;
31         return *this;
32     }
33 };
34 int pos[MAXN];
35 int val[MAXN];
36 int fa[MAXN];
37 int cnt[MAXN];
38 int col[MAXN];
39 int link[MAXN];
40 int CHANGEU;
41 struct chain{
42     vector<int>uu;
43     vector<Node>nde;
44     int n;
45     void init(){
46         n = uu.size();
47         nde.resize(n << 2);
48         for(int i = 0;i < n;i++)pos[uu[i]] = i;
49         build(0,n-1,1);
50     }
51     void up(int l,int r,int p){
52         int mid = (l+r)>>1;
53         nde[p].cc = nde[p<<1].cc ^ nde[(p<<1)|1].cc ^ val[uu[mid]];
54         nde[p].l0 = nde[p<<1].l0;
55         nde[p].l1 = nde[p<<1].l1;
56         if(nde[p<<1].cc^val[uu[mid]]){
57             nde[p].l0 += nde[(p<<1)|1].l1;
58             nde[p].l1 += nde[(p<<1)|1].l0;
59         }
60         else {
61             nde[p].l0 += nde[(p<<1)|1].l0;
62             nde[p].l1 += nde[(p<<1)|1].l1;
63         }
64         nde[p].r0 = nde[(p<<1)|1].r0;
65         nde[p].r1 = nde[(p<<1)|1].r1;
66         if(nde[(p<<1)|1].cc^val[uu[mid]]){
67             nde[p].r0 += nde[p<<1].r1;
68             nde[p].r1 += nde[p<<1].r0;
69         }
70         else {
71             nde[p].r0 += nde[p<<1].r0;
72             nde[p].r1 += nde[p<<1].r1;
73         }
74         if(val[uu[mid]] == 0){
75             nde[p].sum = nde[p<<1].sum + nde[(p<<1)|1].sum +
76                 (long long)nde[p<<1].r0*nde[(p<<1)|1].l1 +
77                 (long long)nde[p<<1].r1*nde[(p<<1)|1].l0;
78         }
79         else {

```

```

80         nde[p].sum = nde[p<<1].sum + nde[(p<<1)|1].sum +
81         (long long)nde[p<<1].r0*nde[(p<<1)|1].l0 +
82         (long long)nde[p<<1].r1*nde[(p<<1)|1].l1;
83     }
84 }
85 void build(int l,int r,int p){
86     if(l == r){
87         nde[p].gao(uu[l]);
88         return;
89     }
90     int mid = (l+r)/2;
91     build(l,mid,p<<1);
92     build(mid+1,r,(p<<1)|1);
93     up(l,r,p);
94 }
95 void update(int k,int l,int r,int p){
96     if(l == r){
97         nde[p].gao(uu[k]);
98         return;
99     }
100    int mid = (l+r)/2;
101    if(k <= mid)update(k,l,mid,p<<1);
102    else update(k,mid+1,r,(p<<1)|1);
103    up(l,r,p);
104 }
105 int change(int y){
106     int x = uu.back();
107     int p = fa[x];
108     if(p){
109         if(x == CHANGEU)val[x] ^= 1;
110         if(val[x]){
111             tnum[p] -= (long long)nde[1].r0*(num0[p]-nde[1].r1)
112             ;
113             tnum[p] -= (long long)nde[1].r1*(num1[p]-nde[1].r0)
114             ;
115             num0[p] -= nde[1].r1;
116             num1[p] -= nde[1].r0;
117         }
118         else {
119             tnum[p] -= (long long)nde[1].r1*(num0[p]-nde[1].r0)
120             ;
121             tnum[p] -= (long long)nde[1].r0*(num1[p]-nde[1].r1)
122             ;
123             num0[p] -= nde[1].r0;
124             num1[p] -= nde[1].r1;
125         }
126         if(x == CHANGEU)val[x] ^= 1;
127     }
128     ans -= nde[1].sum;
129     update(pos[y],0,n-1,1);
130     if(p){

```

```

127         if(val[x]){
128             tnum[p] += (long long)nde[1].r0*num0[p];
129             tnum[p] += (long long)nde[1].r1*num1[p];
130             num0[p] += nde[1].r1;
131             num1[p] += nde[1].r0;
132         }
133         else {
134             tnum[p] += (long long)nde[1].r0*num1[p];
135             tnum[p] += (long long)nde[1].r1*num0[p];
136             num0[p] += nde[1].r0;
137             num1[p] += nde[1].r1;
138         }
139     }
140     ans += nde[1].sum;
141     return p;
142 }
143 }ch[MAXN];
144 void dfs1(int u,int pre){
145     chain &c = ch[u];
146     c.uu.clear();
147     int v, x = 0;
148     cnt[u] = 1;
149     for(int i = head[u];i != -1;i = edge[i].next){
150         v = edge[i].to;
151         if(v == pre)continue;
152         dfs1(v,u);
153         link[i/2] = v;
154         val[v] = edge[i].f;
155         cnt[u] += cnt[v];
156         fa[v] = u;
157         if(cnt[v] > cnt[x]) x = v;
158     }
159     if(!x)col[u] = u;
160     else col[u] = col[x];
161     ch[col[u]].uu.push_back(u);
162     num0[u] = 1;
163     num1[u] = 0;
164     tnum[u] = 0;
165 }
166 }
167 void dfs2(int x){
168     x = col[x];
169     chain &c = ch[x];
170     int n = c.uu.size();
171     int u,v;
172     for(int i = 1;i < n;i++){
173         u = c.uu[i];
174         for(int j = head[u];j != -1;j = edge[j].next){
175             v = edge[j].to;
176             if(v == c.uu[i-1] || fa[u] == v)continue;
177             dfs2(v);

```

```

178         if(val[v]){
179             tnum[u] += (long long)num0[u]*ch[col[v]].nde[1].r0
                  + (long long)num1[u]*ch[col[v]].nde[1].r1;
180             num0[u] += ch[col[v]].nde[1].r1;
181             num1[u] += ch[col[v]].nde[1].r0;
182         }
183         else {
184             tnum[u] += (long long)num1[u]*ch[col[v]].nde[1].r0
                  + (long long)num0[u]*ch[col[v]].nde[1].r1;
185             num0[u] += ch[col[v]].nde[1].r0;
186             num1[u] += ch[col[v]].nde[1].r1;
187         }
188     }
189 }
190 c.init();
191 ans += c.nde[1].sum;
192 }
193 char str[100];
194 char str1[100],str2[100];
195 int main()
196 {
197     int T;
198     int iCase = 0;
199     scanf("%d",&T);
200     int n;
201     while(T--){
202         ans = 0;
203         iCase++;
204         scanf("%d",&n);
205         map<string,int>mp;
206         init();
207         for(int i = 1;i <= n;i++){
208             scanf("%s",str);
209             mp[str] = i;
210         }
211         int u,v,f;
212         for(int i = 1;i < n;i++){
213             scanf("%s%s%d",str1,str2,&f);
214             addedge(mp[str1],mp[str2],f);
215             addedge(mp[str2],mp[str1],f);
216         }
217         int Q;
218         char op[10];
219         scanf("%d",&Q);
220         printf("Case_#%d:\n",iCase);
221         val[1] = 0;
222         fa[1] = 0;
223         dfs1(1,1);
224         dfs2(1);
225         while(Q--){
226             scanf("%s",op);

```

```
227         if(op[0] == 'Q'){
228             printf("%I64d\n",ans*2);
229         }
230         else {
231             int id ;
232             scanf("%d",&id);
233             id--;
234             u = link[id];
235             val[u] ^= 1;
236             CHANGEU = u;
237             while(u)
238                 u = ch[col[u]].change(u);
239         }
240     }
241 }
242 return 0;
243 }
```

5 搜索

5.1 Dancing Links

5.1.1 精确覆盖

```

1  /*
2   * POJ3074
3   */
4  const int N = 9; //3*3 数独
5  const int MaxN = N*N*N + 10;
6  const int MaxM = N*N*4 + 10;
7  const int maxnode = MaxN*4 + MaxM + 10;
8  char g[MaxN];
9  struct DLX{
10     int n,m,size;
11     int U[maxnode],D[maxnode],R[maxnode],L[maxnode],Row[maxnode],
        Col[maxnode];
12     int H[MaxN],S[MaxM];
13     int ansd,ans[MaxN];
14     void init(int _n,int _m){
15         n = _n;
16         m = _m;
17         for(int i = 0;i <= m;i++){
18             S[i] = 0;
19             U[i] = D[i] = i;
20             L[i] = i-1;
21             R[i] = i+1;
22         }
23         R[m] = 0; L[0] = m;
24         size = m;
25         for(int i = 1;i <= n;i++)H[i] = -1;
26     }
27     void Link(int r,int c){
28         ++S[Col[++size]=c];
29         Row[size] = r;
30         D[size] = D[c];
31         U[D[c]] = size;
32         U[size] = c;
33         D[c] = size;
34         if(H[r] < 0)H[r] = L[size] = R[size] = size;
35         else{
36             R[size] = R[H[r]];
37             L[R[H[r]]] = size;
38             L[size] = H[r];
39             R[H[r]] = size;
40         }
41     }
42     void remove(int c){
43         L[R[c]] = L[c]; R[L[c]] = R[c];
44         for(int i = D[c];i != c;i = D[i])
45             for(int j = R[i];j != i;j = R[j]){

```

```

46         U[D[j]] = U[j];
47         D[U[j]] = D[j];
48         —S[Col[j]];
49     }
50 }
51 void resume(int c){
52     for(int i = U[c];i != c;i = U[i])
53         for(int j = L[i];j != i;j = L[j])
54             ++S[Col[U[D[j]]=D[U[j]]=j]];
55     L[R[c]] = R[L[c]] = c;
56 }
57 bool Dance(int d){
58     if(R[0] == 0){
59         for(int i = 0;i < d;i++)g[(ans[i]-1)/9] = (ans[i]-1)%9
60             + '1';
61         for(int i = 0;i < N*N;i++)printf("%c",g[i]);
62         printf("\n");
63         return true;
64     }
65     int c = R[0];
66     for(int i = R[0];i != 0;i = R[i])
67         if(S[i] < S[c])
68             c = i;
69     remove(c);
70     for(int i = D[c];i != c;i = D[i]){
71         ans[d] = Row[i];
72         for(int j = R[i];j != i;j = R[j])remove(Col[j]);
73         if(Dance(d+1))return true;
74         for(int j = L[i];j != i;j = L[j])resume(Col[j]);
75     }
76     resume(c);
77     return false;
78 };
79 void place(int &r,int &c1,int &c2,int &c3,int &c4,int i,int j,int k
80 ) {
81     r = (i*N+j)*N + k; c1 = i*N+j+1; c2 = N*N+i*N+k;
82     c3 = N*N*2+j*N+k; c4 = N*N*3+((i/3)*3+(j/3))*N+k;
83 }
84 DLX dlx;
85 int main(){
86     while(scanf("%s",g) == 1){
87         if(strcmp(g,"end") == 0)break;
88         dlx.init(N*N*N,N*N*4);
89         int r,c1,c2,c3,c4;
90         for(int i = 0;i < N;i++)
91             for(int j = 0;j < N;j++)
92                 for(int k = 1;k <= N;k++)
93                     if(g[i*N+j] == '.' || g[i*N+j] == '0'+k){
94                         place(r,c1,c2,c3,c4,i,j,k);
95                         dlx.Link(r,c1);

```



```

95         dlx.Link(r,c2);
96         dlx.Link(r,c3);
97         dlx.Link(r,c4);
98     }
99     dlx.Dance(0);
100 }
101 return 0;
102 }

```

5.1.2 可重复覆盖

```

1  /*
2   * FZU1686
3   */
4  const int MaxM = 15*15+10;
5  const int MaxN = 15*15+10;
6  const int maxnode = MaxN * MaxM;
7  const int INF = 0x3f3f3f3f;
8  struct DLX{
9      int n,m,size;
10     int U[maxnode],D[maxnode],R[maxnode],L[maxnode],Row[maxnode],
        Col[maxnode];
11     int H[MaxN],S[MaxM];
12     int ansd;
13     void init(int _n,int _m){
14         n = _n;
15         m = _m;
16         for(int i = 0;i <= m;i++){
17             S[i] = 0;
18             U[i] = D[i] = i;
19             L[i] = i-1;
20             R[i] = i+1;
21         }
22         R[m] = 0; L[0] = m;
23         size = m;
24         for(int i = 1;i <= n;i++)H[i] = -1;
25     }
26     void Link(int r,int c){
27         ++S[Col[++size]=c];
28         Row[size] = r;
29         D[size] = D[c];
30         U[D[c]] = size;
31         U[size] = c;
32         D[c] = size;
33         if(H[r] < 0)H[r] = L[size] = R[size] = size;
34         else{
35             R[size] = R[H[r]];
36             L[R[H[r]]] = size;
37             L[size] = H[r];
38             R[H[r]] = size;
39         }
40     }

```

```

41 void remove(int c){
42     for(int i = D[c];i != c;i = D[i])
43         L[R[i]] = L[i], R[L[i]] = R[i];
44 }
45 void resume(int c){
46     for(int i = U[c];i != c;i = U[i])
47         L[R[i]] = R[L[i]] = i;
48 }
49 bool v[MaxM];
50 int f(){
51     int ret = 0;
52     for(int c = R[0]; c != 0;c = R[c])v[c] = true;
53     for(int c = R[0]; c != 0;c = R[c])
54         if(v[c])
55             {
56                 ret++;
57                 v[c] = false;
58                 for(int i = D[c];i != c;i = D[i])
59                     for(int j = R[i];j != i;j = R[j])
60                         v[Col[j]] = false;
61             }
62     return ret;
63 }
64 void Dance(int d){
65     if(d + f() >= ansd)return;
66     if(R[0] == 0){
67         if(d < ansd)ansd = d;
68         return;
69     }
70     int c = R[0];
71     for(int i = R[0];i != 0;i = R[i])
72         if(S[i] < S[c])
73             c = i;
74     for(int i = D[c];i != c;i = D[i]){
75         remove(i);
76         for(int j = R[i];j != i;j = R[j])remove(j);
77         Dance(d+1);
78         for(int j = L[i];j != i;j = L[j])resume(j);
79         resume(i);
80     }
81 }
82 };
83 DLX g;
84 int a[20][20];
85 int id[20][20];
86 int main(){
87     int n,m;
88     while(scanf("%d%d",&n,&m) == 2){
89         int sz = 0;
90         memset(id,0,sizeof(id));
91         for(int i = 0;i < n;i++)

```

```

92         for(int j = 0;j < m;j++){
93             scanf("%d",&a[i][j]);
94             if(a[i][j] == 1)id[i][j] = (++sz);
95         }
96         g.init(n*m,sz);
97         sz = 1;
98         int n1,m1;
99         scanf("%d%d",&n1,&m1);
100        for(int i = 0;i < n;i++){
101            for(int j = 0;j < m;j++){
102                for(int x = 0;x < n1 && i + x < n;x++)
103                    for(int y = 0;y < m1 && j + y < m;y++)
104                        if(id[i+x][j+y])
105                            g.Link(sz,id[i+x][j+y]);
106                sz++;
107            }
108            g.ansd = INF;
109            g.Dance(0);
110            printf("%d\n",g.ansd);
111        }
112        return 0;
113    }

```

5.2 八数码

5.2.1 HDU1043 反向搜索

```

1  /*
2  HDU 1043 Eight
3  八数码，输出路径
4  思路：反向搜索，从目标状态找回状态对应的路径
5  用康托展开判重
6  */
7  const int MAXN=1000000;//最多是 9!/2
8  int fac[]={1,1,2,6,24,120,720,5040,40320,362880}; //康拖展开判重
9  //      0! 1! 2! 3! 4! 5! 6! 7! 8! 9!
10 bool vis[MAXN]; //标记
11 string path[MAXN]; //记录路径
12 //康拖展开求该序列的 hash 值
13 int cantor(int s[]){
14     int sum=0;
15     for(int i=0;i<9;i++){
16         int num=0;
17         for(int j=i+1;j<9;j++)
18             if(s[j]<s[i])num++;
19         sum+=(num*fac[9-i-1]);
20     }
21     return sum+1;
22 }
23 struct Node{
24     int s[9];
25     int loc;//'0' 的位置

```

```

26     int status;//康拖展开的 hash 值
27     string path;//路径
28 };
29 int move[4][2]={{-1,0},{1,0},{0,-1},{0,1}};//u,d,l,r
30 char indexs[5]="durl";//和上面的要相反,因为是反向搜索
31 int aim=46234;//123456780 对应的康拖展开的 hash 值
32 void bfs(){
33     memset(vis,false,sizeof(vis));
34     Node cur,next;
35     for(int i=0;i<8;i++)cur.s[i]=i+1;
36     cur.s[8]=0;
37     cur.loc=8;
38     cur.status=aim;
39     cur.path="";
40     queue<Node>q;
41     q.push(cur);
42     path[aim]="";
43     while(!q.empty()){
44         cur=q.front();
45         q.pop();
46         int x=cur.loc/3;
47         int y=cur.loc%3;
48         for(int i=0;i<4;i++){
49             int tx=x+move[i][0];
50             int ty=y+move[i][1];
51             if(tx<0||tx>2||ty<0||ty>2)continue;
52             next=cur;
53             next.loc=tx*3+ty;
54             next.s[cur.loc]=next.s[next.loc];
55             next.s[next.loc]=0;
56             next.status=cantor(next.s);
57             if(!vis[next.status]){
58                 vis[next.status]=true;
59                 next.path=indexs[i]+next.path;
60                 q.push(next);
61                 path[next.status]=next.path;
62             }
63         }
64     }
65 }
66
67 int main(){
68     char ch;
69     Node cur;
70     bfs();
71     while(cin>>ch){
72         if(ch=='x') {cur.s[0]=0;cur.loc=0;}
73         else cur.s[0]=ch-'0';
74         for(int i=1;i<9;i++){
75             cin>>ch;
76             if(ch=='x'){

```

```
77         cur.s[i]=0;
78         cur.loc=i;
79     }
80     else cur.s[i]=ch-'0';
81 }
82 cur.status=cantor(cur.s);
83 if(vis[cur.status]){
84     cout<<path[cur.status]<<endl;
85 }
86 else cout<<"unsolvable"<<endl;
87 }
88 return 0;
89 }
```

6 动态规划

6.1 最长上升子序列 $O(n\log n)$

```

1  const int MAXN=500010;
2  int a[MAXN],b[MAXN];
3
4  //用二分查找的方法找到一个位置,使得 num>b[i-1] 并且 num<b[i],并用 num 代
   替 b[i]
5  int Search(int num,int low,int high){
6      int mid;
7      while(low<=high){
8          mid=(low+high)/2;
9          if(num>=b[mid]) low=mid+1;
10         else high=mid-1;
11     }
12     return low;
13 }
14 int DP(int n){
15     int i,len,pos;
16     b[1]=a[1];
17     len=1;
18     for(i=2;i<=n;i++){
19         if(a[i]>=b[len])//如果 a[i] 比 b[] 数组中最大还大直接插入到后面即可
20         {
21             len=len+1;
22             b[len]=a[i];
23         }
24         else//用二分的方法在 b[] 数组中找出第一个比 a[i] 大的位置并且让
           a[i] 替代这个位置
25         {
26             pos=Search(a[i],1,len);
27             b[pos]=a[i];
28         }
29     }
30     return len;
31 }

```

6.2 背包

```

1  int nValue,nKind;
2
3  //0-1 背包,代价为 cost,获得的价值为 weight
4  void ZeroOnePack(int cost,int weight){
5      for(int i=nValue;i>=cost;i--)
6          dp[i]=max(dp[i],dp[i-cost]+weight);
7  }
8
9  //完全背包,代价为 cost,获得的价值为 weight
10 void CompletePack(int cost,int weight){
11     for(int i=cost;i<=nValue;i++)

```

```

12     dp[i]=max(dp[i],dp[i-cost]+weight);
13 }
14
15 //多重背包
16 void MultiplePack(int cost,int weight,int amount){
17     if(cost*amount>=nValue) CompletePack(cost,weight);
18     else{
19         int k=1;
20         while(k<amount){
21             ZeroOnePack(k*cost,k*weight);
22             amount-=k;
23             k<<=1;
24         }
25         ZeroOnePack(amount*cost,amount*weight); //这个不要忘了, 经常
           掉了
26     }
27 }

```

分组背包:

for k = 1 to K

for v = V to 0

for item i in group k

$F[v] = \max(F[v], F[v-C_i] + W_i)$

6.3 插头 DP

6.3.1 HDU 4285

求 K 个回路的方案数。而且不能是环套环。

增加个标志位来记录形成的回路个数。而且注意避免环套环的情况。不形成环套环的话就是在形成新的回路时，两边的插头个数要为偶数。

```

1  /*
2  HDU 4285
3  要形成刚好 K 条回路的方法数
4  要避免环套环的情况。
5  所以形成回路时，要保证两边的插头数是偶数
6  G++ 11265ms 11820K
7  C++ 10656ms 11764K
8  */
9  const int MAXD=15;
10 const int STATE=1000010;
11 const int HASH=300007; //这个大一点可以防止 TLE，但是容易 MLE
12 const int MOD=1000000007;
13 int N,M,K;
14 int maze[MAXD][MAXD];
15 int code[MAXD];
16 int ch[MAXD];
17 int num; //圈的个数
18 struct HASHMAP
19 {
20     int head[HASH],next[STATE],size;
21     long long state[STATE];

```

```

22     int f[STATE];
23     void init()
24     {
25         size=0;
26         memset(head,-1,sizeof(head));
27     }
28     void push(long long st,int ans)
29     {
30         int i;
31         int h=st%HASH;
32         for(i=head[h];i!=-1;i=next[i])
33             if(state[i]==st)
34             {
35                 f[i]+=ans;
36                 f[i]%=MOD;
37                 return;
38             }
39         state[size]=st;
40         f[size]=ans;
41         next[size]=head[h];
42         head[h]=size++;
43     }
44 }hm[2];
45 void decode(int *code,int m,long long st)
46 {
47     num=st&63;
48     st>>=6;
49     for(int i=m;i>=0;i--)
50     {
51         code[i]=st&7;
52         st>>=3;
53     }
54 }
55 long long encode(int *code,int m)//最小表示法
56 {
57     int cnt=1;
58     memset(ch,-1,sizeof(ch));
59     ch[0]=0;
60     long long st=0;
61     for(int i=0;i<=m;i++)
62     {
63         if(ch[code[i]]==-1)ch[code[i]]=cnt++;
64         code[i]=ch[code[i]];
65         st<<=3;
66         st|=code[i];
67     }
68     st<<=6;
69     st|=num;
70     return st;
71 }
72 void shift(int *code,int m)

```



```

73 {
74     for(int i=m;i>0;i--)code[i]=code[i-1];
75     code[0]=0;
76 }
77 void dpblank(int i,int j,int cur)
78 {
79     int k,left,up;
80     for(k=0;k<hm[cur].size;k++)
81     {
82         decode(code,M,hm[cur].state[k]);
83         left=code[j-1];
84         up=code[j];
85         if(left&&up)
86         {
87             if(left==up)
88             {
89                 if(num>=K)continue;
90                 int t=0;
91                 //要避免环套环的情况，需要两边插头数为偶数
92                 for(int p=0;p<j-1;p++)
93                     if(code[p])t++;
94                 if(t&1)continue;
95                 if(num<K)
96                 {
97                     num++;
98                     code[j-1]=code[j]=0;
99                     hm[cur^1].push(encode(code,j==M?M-1:M),hm[cur].
100                         f[k]);
101                 }
102             }
103             else
104             {
105                 code[j-1]=code[j]=0;
106                 for(int t=0;t<=M;t++)
107                     if(code[t]==up)
108                         code[t]=left;
109                 hm[cur^1].push(encode(code,j==M?M-1:M),hm[cur].f[k
110                     ]);
111             }
112         }
113         else if(left||up)
114         {
115             int t;
116             if(left)t=left;
117             else t=up;
118             if(maze[i][j+1])
119             {
120                 code[j-1]=0;
121                 code[j]=t;
122                 hm[cur^1].push(encode(code,M),hm[cur].f[k]);
123             }
124         }
125     }
126 }

```

```

122         if(maze[i+1][j])
123         {
124             code[j]=0;
125             code[j-1]=t;
126             hm[cur^1].push(encode(code,j==M?M-1:M),hm[cur].f[k
                ]));
127         }
128     }
129     else
130     {
131         if(maze[i][j+1]&&maze[i+1][j])
132         {
133             code[j-1]=code[j]=13;
134             hm[cur^1].push(encode(code,j==M?M-1:M),hm[cur].f[k
                ]));
135         }
136     }
137 }
138 }
139 void dpblock(int i,int j,int cur)
140 {
141     int k;
142     for(k=0;k<hm[cur].size;k++)
143     {
144         decode(code,M,hm[cur].state[k]);
145         code[j-1]=code[j]=0;
146         hm[cur^1].push(encode(code,j==M?M-1:M),hm[cur].f[k]);
147     }
148 }
149 char str[20];
150 void init()
151 {
152     scanf("%d%d%d",&N,&M,&K);
153     memset(maze,0,sizeof(maze));
154     for(int i=1;i<=N;i++)
155     {
156         scanf("%s",&str);
157         for(int j=1;j<=M;j++)
158             if(str[j-1]=='.')
159                 maze[i][j]=1;
160     }
161 }
162 void solve()
163 {
164     int i,j,cur=0;
165     hm[cur].init();
166     hm[cur].push(0,1);
167     for(i=1;i<=N;i++)
168         for(j=1;j<=M;j++)
169         {
170             hm[cur^1].init();

```

```

171         if(maze[i][j])dpblank(i,j,cur);
172         else dpblock(i,j,cur);
173         cur^=1;
174     }
175     int ans=0;
176     for(i=0;i<hm[cur].size;i++)
177         if(hm[cur].state[i]==K)
178         {
179             ans+=hm[cur].f[i];
180             ans%=MOD;
181         }
182     printf("%d\n",ans);
183 }
184 }
185 int main()
186 {
187     int T;
188     scanf("%d",&T);
189     while(T——)
190     {
191         init();
192         solve();
193     }
194     return 0;
195 }
196
197 /*
198 Sample Input
199 4 4 1
200 **..
201 ....
202 ....
203 ....
204 4 1
205 ....
206 ....
207 ....
208 ....
209
210 Sample Output
211 6
212
213 */

```

7 计算几何

7.1 二维几何

```

1 // 计算几何模板
2 const double eps = 1e-8;
3 const double inf = 1e20;
4 const double pi = acos(-1.0);
5 const int maxp = 1010;
6 //Compares a double to zero
7 int sgn(double x){
8     if(fabs(x) < eps)return 0;
9     if(x < 0)return -1;
10    else return 1;
11 }
12 //square of a double
13 inline double sqr(double x){return x*x;}
14 /*
15  * Point
16  * Point()                - Empty constructor
17  * Point(double _x,double _y) - constructor
18  * input()                - double input
19  * output()               - %.2f output
20  * operator ==            - compares x and y
21  * operator <             - compares first by x, then by y
22  * operator -             - return new Point after subtracting
23                          - currepsonging x and y
24  * operator ^            - cross product of 2d points
25  * operator *            - dot product
26  * len()                 - gives length from origin
27  * len2()                - gives square of length from origin
28  * distance(Point p)     - gives distance from p
29  * operator + Point b   - returns new Point after adding
30                          - currepsonging x and y
31  * operator * double k  - returns new Point after multiplieing x and
32                          - y by k
33  * operator / double k  - returns new Point after divideing x and y
34                          - by k
35  * rad(Point a,Point b)- returns the angle of Point a and Point b
36                          - from this Point
37  * trunc(double r)       - return Point that if truncated the
38                          - distance from center to r
39  * rotleft()             - returns 90 degree ccw rotated point
40  * rotright()            - returns 90 degree cw rotated point
41  * rotate(Point p,double angle) - returns Point after rotateing the
42                          - Point centering at p by angle radian ccw
43 */
44 struct Point{
45     double x,y;
46     Point(){}
47     Point(double _x,double _y){

```

```

41         x = _x;
42         y = _y;
43     }
44     void input(){
45         scanf("%lf%lf",&x,&y);
46     }
47     void output(){
48         printf("%.2f□%.2f\n",x,y);
49     }
50     bool operator == (Point b)const{
51         return sgn(x-b.x) == 0 && sgn(y-b.y) == 0;
52     }
53     bool operator < (Point b)const{
54         return sgn(x-b.x)== 0?sgn(y-b.y)<0:x<b.x;
55     }
56     Point operator -(const Point &b)const{
57         return Point(x-b.x,y-b.y);
58     }
59     //叉积
60     double operator ^(const Point &b)const{
61         return x*b.y - y*b.x;
62     }
63     //点积
64     double operator *(const Point &b)const{
65         return x*b.x + y*b.y;
66     }
67     //返回长度
68     double len(){
69         return hypot(x,y); //库函数
70     }
71     //返回长度的平方
72     double len2(){
73         return x*x + y*y;
74     }
75     //返回两点的距离
76     double distance(Point p){
77         return hypot(x-p.x,y-p.y);
78     }
79     Point operator +(const Point &b)const{
80         return Point(x+b.x,y+b.y);
81     }
82     Point operator *(const double &k)const{
83         return Point(x*k,y*k);
84     }
85     Point operator /(const double &k)const{
86         return Point(x/k,y/k);
87     }
88     //计算 pa 和 pb 的夹角
89     //就是求这个点看 a,b 所成的夹角
90     //测试 LightOJ1203
91     double rad(Point a,Point b){

```

```

92     Point p = *this;
93     return fabs(atan2( fabs((a-p)^(b-p)),(a-p)*(b-p) ));
94 }
95 //化为长度为 r 的向量
96 Point trunc(double r){
97     double l = len();
98     if(!sgn(l))return *this;
99     r /= l;
100    return Point(x*r,y*r);
101 }
102 //逆时针旋转 90 度
103 Point rotleft(){
104     return Point(-y,x);
105 }
106 //顺时针旋转 90 度
107 Point rotright(){
108     return Point(y,-x);
109 }
110 //绕着 p 点逆时针旋转 angle
111 Point rotate(Point p,double angle){
112     Point v = (*this) - p;
113     double c = cos(angle), s = sin(angle);
114     return Point(p.x + v.x*c - v.y*s,p.y + v.x*s + v.y*c);
115 }
116 };
117 /*
118  * Stores two points
119  * Line()                    - Empty constructor
120  * Line(Point _s,Point _e)   - Line through _s and _e
121  * operator ==               - checks if two points are same
122  * Line(Point p,double angle) - one end p , another end at
123                               angle degree
124  * Line(double a,double b,double c) - Line of equation ax + by + c
125                                     = 0
126  * input()                  - inputs s and e
127  * adjust()                 - orders in such a way that s < e
128  * length()                 - distance of se
129  * angle()                  - return 0 <= angle < pi
130  * relation(Point p)        - 3 if point is on line
131                               1 if point on the left of line
132                               2 if point on the right of line
133  * pointonseg(double p)     - return true if point on segment
134  * parallel(Line v)         - return true if they are
135                               parallel
136  * segcrossseg(Line v)      - returns 0 if does not intersect
137                               returns 1 if non-standard
138                               intersection
139  *                          - returns 2 if intersects
140  * linecrossseg(Line v)     - line and seg
141  * linecrossline(Line v)    - 0 if parallel
142  *                          - 1 if coincides

```

```

139 *                2 if intersects
140 * crosspoint(Line v)          - returns intersection point
141 * dispointtoline(Point p)     - distance from point p to the
    line
142 * dispointtoseg(Point p)      - distance from p to the segment
143 * dissegtoseg(Line v)         - distance of two segment
144 * lineprog(Point p)           - returns projected point p on se
    line
145 * symmetrypoint(Point p)      - returns reflection point of p
    over se
146 *
147 */
148 struct Line{
149     Point s,e;
150     Line(){}
151     Line(Point _s,Point _e){
152         s = _s;
153         e = _e;
154     }
155     bool operator ==(Line v){
156         return (s == v.s)&&(e == v.e);
157     }
158     //根据一个点和倾斜角 angle 确定直线,0<=angle<pi
159     Line(Point p,double angle){
160         s = p;
161         if(sgn(angle-pi/2) == 0){
162             e = (s + Point(0,1));
163         }
164         else{
165             e = (s + Point(1,tan(angle)));
166         }
167     }
168     //ax+by+c=0
169     Line(double a,double b,double c){
170         if(sgn(a) == 0){
171             s = Point(0,-c/b);
172             e = Point(1,-c/b);
173         }
174         else if(sgn(b) == 0){
175             s = Point(-c/a,0);
176             e = Point(-c/a,1);
177         }
178         else{
179             s = Point(0,-c/b);
180             e = Point(1,(-c-a)/b);
181         }
182     }
183     void input(){
184         s.input();
185         e.input();
186     }

```

```

187 void adjust(){
188     if(e < s)swap(s,e);
189 }
190 //求线段长度
191 double length(){
192     return s.distance(e);
193 }
194 //返回直线倾斜角 0<=angle<pi
195 double angle(){
196     double k = atan2(e.y-s.y,e.x-s.x);
197     if(sgn(k) < 0)k += pi;
198     if(sgn(k-pi) == 0)k -= pi;
199     return k;
200 }
201 //点和直线关系
202 //1 在左侧
203 //2 在右侧
204 //3 在直线上
205 int relation(Point p){
206     int c = sgn((p-s)^(e-s));
207     if(c < 0)return 1;
208     else if(c > 0)return 2;
209     else return 3;
210 }
211 // 点在线段上的判断
212 bool pointonseg(Point p){
213     return sgn((p-s)^(e-s)) == 0 && sgn((p-s)*(p-e)) <= 0;
214 }
215 //两向量平行 (对应直线平行或重合)
216 bool parallel(Line v){
217     return sgn((e-s)^(v.e-v.s)) == 0;
218 }
219 //两线段相交判断
220 //2 规范相交
221 //1 非规范相交
222 //0 不相交
223 int segcrossseg(Line v){
224     int d1 = sgn((e-s)^(v.s-s));
225     int d2 = sgn((e-s)^(v.e-s));
226     int d3 = sgn((v.e-v.s)^(s-v.s));
227     int d4 = sgn((v.e-v.s)^(e-v.s));
228     if( (d1^d2)==-2 && (d3^d4)==-2 )return 2;
229     return (d1==0 && sgn((v.s-s)*(v.s-e))<=0) ||
230            (d2==0 && sgn((v.e-s)*(v.e-e))<=0) ||
231            (d3==0 && sgn((s-v.s)*(s-v.e))<=0) ||
232            (d4==0 && sgn((e-v.s)*(e-v.e))<=0);
233 }
234 //直线和线段相交判断
235 //-*this line -v seg
236 //2 规范相交
237 //1 非规范相交

```



```

238 //0 不相交
239 int linecrossseg(Line v){
240     int d1 = sgn((e-s)^(v.s-s));
241     int d2 = sgn((e-s)^(v.e-s));
242     if((d1^d2)==-2) return 2;
243     return (d1==0 || d2==0);
244 }
245 //两直线关系
246 //0 平行
247 //1 重合
248 //2 相交
249 int linecrossline(Line v){
250     if((*this).parallel(v))
251         return v.relation(s)==3;
252     return 2;
253 }
254 //求两直线的交点
255 //要保证两直线不平行或重合
256 Point crosspoint(Line v){
257     double a1 = (v.e-v.s)^(s-v.s);
258     double a2 = (v.e-v.s)^(e-v.s);
259     return Point((s.x*a2-e.x*a1)/(a2-a1), (s.y*a2-e.y*a1)/(a2-a1));
260 }
261 //点到直线的距离
262 double dispointtoline(Point p){
263     return fabs((p-s)^(e-s))/length();
264 }
265 //点到线段的距离
266 double dispointtoseg(Point p){
267     if(sgn((p-s)*(e-s))<0 || sgn((p-e)*(s-e))<0)
268         return min(p.distance(s),p.distance(e));
269     return dispointtoline(p);
270 }
271 //返回线段到线段的距离
272 //前提是两线段不相交，相交距离就是 0 了
273 double dissegtoseg(Line v){
274     return min(min(dispointtoseg(v.s),dispointtoseg(v.e)),min(v
        .dispointtoseg(s),v.dispointtoseg(e)));
275 }
276 //返回点 p 在直线上的投影
277 Point lineprog(Point p){
278     return s + ( ((e-s)*((e-s)*(p-s)))/((e-s).len2()) );
279 }
280 //返回点 p 关于直线的对称点
281 Point symmetrpoint(Point p){
282     Point q = lineprog(p);
283     return Point(2*q.x-p.x,2*q.y-p.y);
284 }
285 };
286 //圆

```

```

287 struct circle{
288     Point p;//圆心
289     double r;//半径
290     circle(){}
291     circle(Point _p,double _r){
292         p = _p;
293         r = _r;
294     }
295     circle(double x,double y,double _r){
296         p = Point(x,y);
297         r = _r;
298     }
299     //三角形的外接圆
300     //需要 Point 的 + / rotate() 以及 Line 的 crosspoint()
301     //利用两条边的中垂线得到圆心
302     //测试: UVA12304
303     circle(Point a,Point b,Point c){
304         Line u = Line((a+b)/2,((a+b)/2)+((b-a).rotleft()));
305         Line v = Line((b+c)/2,((b+c)/2)+((c-b).rotleft()));
306         p = u.crosspoint(v);
307         r = p.distance(a);
308     }
309     //三角形的内切圆
310     //参数 bool t 没有作用, 只是为了和上面外接圆函数区别
311     //测试: UVA12304
312     circle(Point a,Point b,Point c,bool t){
313         Line u,v;
314         double m = atan2(b.y-a.y,b.x-a.x), n = atan2(c.y-a.y,c.x-a.
315             x);
316         u.s = a;
317         u.e = u.s + Point(cos((n+m)/2),sin((n+m)/2));
318         v.s = b;
319         m = atan2(a.y-b.y,a.x-b.x) , n = atan2(c.y-b.y,c.x-b.x);
320         v.e = v.s + Point(cos((n+m)/2),sin((n+m)/2));
321         p = u.crosspoint(v);
322         r = Line(a,b).dispointtoseg(p);
323     }
324     //输入
325     void input(){
326         p.input();
327         scanf("%lf",&r);
328     }
329     //输出
330     void output(){
331         printf("%.2lf_%.2lf_%.2lf\n",p.x,p.y,r);
332     }
333     bool operator == (circle v){
334         return (p==v.p) && sgn(r-v.r)==0;
335     }
336     bool operator < (circle v)const{
337         return ((p<v.p) || ((p==v.p)&&sgn(r-v.r)<0));

```

```

337     }
338     //面积
339     double area(){
340         return pi*r*r;
341     }
342     //周长
343     double circumference(){
344         return 2*pi*r;
345     }
346     //点和圆的关系
347     //0 圆外
348     //1 圆上
349     //2 圆内
350     int relation(Point b){
351         double dst = b.distance(p);
352         if(sgn(dst-r) < 0)return 2;
353         else if(sgn(dst-r)==0)return 1;
354         return 0;
355     }
356     //线段和圆的关系
357     //比较的是圆心到线段的距离和半径的关系
358     int relationseg(Line v){
359         double dst = v.dispointtoseg(p);
360         if(sgn(dst-r) < 0)return 2;
361         else if(sgn(dst-r) == 0)return 1;
362         return 0;
363     }
364     //直线和圆的关系
365     //比较的是圆心到直线的距离和半径的关系
366     int relationline(Line v){
367         double dst = v.dispointtoline(p);
368         if(sgn(dst-r) < 0)return 2;
369         else if(sgn(dst-r) == 0)return 1;
370         return 0;
371     }
372     //两圆的关系
373     //5 相离
374     //4 外切
375     //3 相交
376     //2 内切
377     //1 内含
378     //需要 Point 的 distance
379     //测试: UVA12304
380     int relationcircle(circle v){
381         double d = p.distance(v.p);
382         if(sgn(d-r-v.r) > 0)return 5;
383         if(sgn(d-r-v.r) == 0)return 4;
384         double l = fabs(r-v.r);
385         if(sgn(d-r-v.r)<0 && sgn(d-l)>0)return 3;
386         if(sgn(d-l)==0)return 2;
387         if(sgn(d-l)<0)return 1;

```

```

388 }
389 //求两个圆的交点, 返回 0 表示没有交点, 返回 1 是一个交点, 2 是两个交点
390 //需要 relationcircle
391 //测试: UVA12304
392 int pointcrosscircle(circle v, Point &p1, Point &p2){
393     int rel = relationcircle(v);
394     if(rel == 1 || rel == 5) return 0;
395     double d = p.distance(v.p);
396     double l = (d*d+r*r-v.r*v.r)/(2*d);
397     double h = sqrt(r*r-l*l);
398     Point tmp = p + (v.p-p).trunc(l);
399     p1 = tmp + ((v.p-p).rotleft().trunc(h));
400     p2 = tmp + ((v.p-p).rotright().trunc(h));
401     if(rel == 2 || rel == 4)
402         return 1;
403     return 2;
404 }
405 //求直线和圆的交点, 返回交点个数
406 int pointcrossline(Line v, Point &p1, Point &p2){
407     if(!(*this).relationline(v)) return 0;
408     Point a = v.lineprog(p);
409     double d = v.dispointtoline(p);
410     d = sqrt(r*r-d*d);
411     if(sgn(d) == 0){
412         p1 = a;
413         p2 = a;
414         return 1;
415     }
416     p1 = a + (v.e-v.s).trunc(d);
417     p2 = a - (v.e-v.s).trunc(d);
418     return 2;
419 }
420 //得到过 a,b 两点, 半径为 r1 的两个圆
421 int gercircle(Point a, Point b, double r1, circle &c1, circle &c2){
422     circle x(a, r1), y(b, r1);
423     int t = x.pointcrosscircle(y, c1.p, c2.p);
424     if(!t) return 0;
425     c1.r = c2.r = r;
426     return t;
427 }
428 //得到与直线 u 相切, 过点 q, 半径为 r1 的圆
429 //测试: UVA12304
430 int getcircle(Line u, Point q, double r1, circle &c1, circle &c2){
431     double dis = u.dispointtoline(q);
432     if(sgn(dis-r1*2)>0) return 0;
433     if(sgn(dis) == 0){
434         c1.p = q + ((u.e-u.s).rotleft().trunc(r1));
435         c2.p = q + ((u.e-u.s).rotright().trunc(r1));
436         c1.r = c2.r = r1;
437         return 2;
438     }

```

```

439     Line u1 = Line((u.s + (u.e-u.s).rotleft().trunc(r1)),(u.e +
        (u.e-u.s).rotleft().trunc(r1)));
440     Line u2 = Line((u.s + (u.e-u.s).rotright().trunc(r1)),(u.e
        + (u.e-u.s).rotright().trunc(r1)));
441     circle cc = circle(q,r1);
442     Point p1,p2;
443     if(!cc.pointcrossline(u1,p1,p2))cc.pointcrossline(u2,p1,p2)
        ;
444     c1 = circle(p1,r1);
445     if(p1 == p2){
446         c2 = c1;
447         return 1;
448     }
449     c2 = circle(p2,r1);
450     return 2;
451 }
452 //同时与直线 u,v 相切, 半径为 r1 的圆
453 //测试: UVA12304
454 int getcircle(Line u,Line v,double r1,circle &c1,circle &c2,
    circle &c3,circle &c4){
455     if(u.parallel(v))return 0;//两直线平行
456     Line u1 = Line(u.s + (u.e-u.s).rotleft().trunc(r1),u.e + (u
        .e-u.s).rotleft().trunc(r1));
457     Line u2 = Line(u.s + (u.e-u.s).rotright().trunc(r1),u.e + (
        u.e-u.s).rotright().trunc(r1));
458     Line v1 = Line(v.s + (v.e-v.s).rotleft().trunc(r1),v.e + (v
        .e-v.s).rotleft().trunc(r1));
459     Line v2 = Line(v.s + (v.e-v.s).rotright().trunc(r1),v.e + (
        v.e-v.s).rotright().trunc(r1));
460     c1.r = c2.r = c3.r = c4.r = r1;
461     c1.p = u1.crosspoint(v1);
462     c2.p = u1.crosspoint(v2);
463     c3.p = u2.crosspoint(v1);
464     c4.p = u2.crosspoint(v2);
465     return 4;
466 }
467 //同时与不相交圆 cx,cy 相切, 半径为 r1 的圆
468 //测试: UVA12304
469 int getcircle(circle cx,circle cy,double r1,circle &c1,circle &
    c2){
470     circle x(cx.p,r1+cx.r),y(cy.p,r1+cy.r);
471     int t = x.pointcrosscircle(y,c1.p,c2.p);
472     if(!t)return 0;
473     c1.r = c2.r = r1;
474     return t;
475 }
476
477 //过一点作圆的切线 (先判断点和圆的关系)
478 //测试: UVA12304
479 int tangentline(Point q,Line &u,Line &v){
480     int x = relation(q);

```

```

481         if(x == 2)return 0;
482         if(x == 1){
483             u = Line(q,q + (q-p).rotleft());
484             v = u;
485             return 1;
486         }
487         double d = p.distance(q);
488         double l = r*r/d;
489         double h = sqrt(r*r-l*l);
490         u = Line(q,p + ((q-p).trunc(l) + (q-p).rotleft().trunc(h)))
491             ;
492         v = Line(q,p + ((q-p).trunc(l) + (q-p).rotright().trunc(h))
493             );
494         return 2;
495     }
496 //求两圆相交的面积
497 double areacircle(circle v){
498     int rel = relationcircle(v);
499     if(rel >= 4)return 0.0;
500     if(rel <= 2)return min(area(),v.area());
501     double d = p.distance(v.p);
502     double hf = (r+v.r+d)/2.0;
503     double ss = 2*sqrt(hf*(hf-r)*(hf-v.r)*(hf-d));
504     double a1 = acos((r*r+d*d-v.r*v.r)/(2.0*r*d));
505     a1 = a1*r*r;
506     double a2 = acos((v.r*v.r+d*d-r*r)/(2.0*v.r*d));
507     a2 = a2*v.r*v.r;
508     return a1+a2-ss;
509 }
510 //求圆和三角形 pab 的相交面积
511 //测试: POJ3675 HDU3982 HDU2892
512 double areatriangle(Point a,Point b){
513     if(sgn((p-a)^(p-b)) == 0)return 0.0;
514     Point q[5];
515     int len = 0;
516     q[len++] = a;
517     Line l(a,b);
518     Point p1,p2;
519     if(pointcrossline(l,q[1],q[2])==2){
520         if(sgn((a-q[1])*(b-q[1]))<0)q[len++] = q[1];
521         if(sgn((a-q[2])*(b-q[2]))<0)q[len++] = q[2];
522     }
523     q[len++] = b;
524     if(len == 4 && sgn((q[0]-q[1])*(q[2]-q[1]))>0)swap(q[1],q
525         [2]);
526     double res = 0;
527     for(int i = 0;i < len-1;i++){
528         if(relation(q[i])==0||relation(q[i+1])==0){
529             double arg = p.rad(q[i],q[i+1]);
530             res += r*r*arg/2.0;
531         }
532     }
533 }

```

```

529         else{
530             res += fabs((q[i]-p)^(q[i+1]-p))/2.0;
531         }
532     }
533     return res;
534 }
535 };
536
537 /*
538  * n,p   Line l for each side
539  * input(int _n)                - inputs _n size polygon
540  * add(Point q)                 - adds a point at end of
541     the list
542  * getline()                   - populates line array
543  * cmp                          - comparision in
544     convex_hull order
545  * norm()                      - sorting in convex_hull
546     order
547  * getconvex(polygon &convex)   - returns convex hull in
548     convex
549  * Graham(polygon &convex)      - returns convex hull in
550     convex
551  * isconvex()                  - checks if convex
552  * relationpoint(Point q)      - returns 3 if q is a
553     vertex
554  *                                2 if on a side
555  *                                1 if inside
556  *                                0 if outside
557  * convexcut(Line u,polygon &po) - left side of u in po
558  * gercircumference()          - returns side length
559  * getarea()                   - returns area
560  * getdir()                    - returns 0 for cw, 1 for
561     ccw
562  * getbarycentre()             - returns barycenter
563  *
564  */
565 struct polygon{
566     int n;
567     Point p[maxp];
568     Line l[maxp];
569     void input(int _n){
570         n = _n;
571         for(int i = 0;i < n;i++){
572             p[i].input();
573         }
574     }
575     void add(Point q){
576         p[n++] = q;
577     }
578     void getline(){
579         for(int i = 0;i < n;i++){
580             l[i] = Line(p[i],p[(i+1)%n]);
581         }
582     }
583 };

```

```

573     }
574 }
575 struct cmp{
576     Point p;
577     cmp(const Point &p0){p = p0;}
578     bool operator()(const Point &aa,const Point &bb){
579         Point a = aa, b = bb;
580         int d = sgn((a-p)^(b-p));
581         if(d == 0){
582             return sgn(a.distance(p)-b.distance(p)) < 0;
583         }
584         return d > 0;
585     }
586 };
587 //进行极角排序
588 //首先需要找到最左下角的点
589 //需要重载号好 Point 的 < 操作符 (min 函数要用)
590 void norm(){
591     Point mi = p[0];
592     for(int i = 1;i < n;i++)mi = min(mi,p[i]);
593     sort(p,p+n,cmp(mi));
594 }
595 //得到凸包
596 //得到的凸包里面的点编号是 0~n-1 的
597 //两种凸包的方法
598 //注意如果有影响, 要特判下所有点共点, 或者共线的特殊情况
599 //测试 LightOJ1203 LightOJ1239
600 void getconvex(polygon &convex){
601     sort(p,p+n);
602     convex.n = n;
603     for(int i = 0;i < min(n,2);i++){
604         convex.p[i] = p[i];
605     }
606     if(convex.n == 2 && (convex.p[0] == convex.p[1]))convex.n
        —;//特
        判
607     if(n <= 2)return;
608     int &top = convex.n;
609     top = 1;
610     for(int i = 2;i < n;i++){
611         while(top && sgn((convex.p[top]-p[i])^(convex.p[top-1]-
            p[i])) <= 0)
612             top—;
613         convex.p[++top] = p[i];
614     }
615     int temp = top;
616     convex.p[++top] = p[n-2];
617     for(int i = n-3;i >= 0;i—){
618         while(top != temp && sgn((convex.p[top]-p[i])^(convex.p
            [top-1]-p[i])) <= 0)
619             top—;

```



```

620         convex.p[++top] = p[i];
621     }
622     if(convex.n == 2 && (convex.p[0] == convex.p[1]))convex.n
        —; //特
        判
623     convex.norm(); //原来得到的是顺时针的点，排序后逆时针
624 }
625 //得到凸包的另外一种方法
626 //测试 LightOJ1203 LightOJ1239
627 void Graham(polygon &convex){
628     norm();
629     int &top = convex.n;
630     top = 0;
631     if(n == 1){
632         top = 1;
633         convex.p[0] = p[0];
634         return;
635     }
636     if(n == 2){
637         top = 2;
638         convex.p[0] = p[0];
639         convex.p[1] = p[1];
640         if(convex.p[0] == convex.p[1])top—;
641         return;
642     }
643     convex.p[0] = p[0];
644     convex.p[1] = p[1];
645     top = 2;
646     for(int i = 2; i < n; i++){
647         while( top > 1 && sgn((convex.p[top-1]-convex.p[top-2])
            ^ (p[i]-convex.p[top-2])) <= 0 )
648             top—;
649         convex.p[top++] = p[i];
650     }
651     if(convex.n == 2 && (convex.p[0] == convex.p[1]))convex.n
        —; //特
        判
652 }
653 //判断是不是凸的
654 bool isconvex(){
655     bool s[2];
656     memset(s, false, sizeof(s));
657     for(int i = 0; i < n; i++){
658         int j = (i+1)%n;
659         int k = (j+1)%n;
660         s[sgn((p[j]-p[i])^(p[k]-p[i]))+1] = true;
661         if(s[0] && s[2])return false;
662     }
663     return true;
664 }
665 //判断点和任意多边形的关系

```

```

666 // 3 点上
667 // 2 边上
668 // 1 内部
669 // 0 外部
670 int relationpoint(Point q){
671     for(int i = 0;i < n;i++){
672         if(p[i] == q)return 3;
673     }
674     getline();
675     for(int i = 0;i < n;i++){
676         if(l[i].pointonseg(q))return 2;
677     }
678     int cnt = 0;
679     for(int i = 0;i < n;i++){
680         int j = (i+1)%n;
681         int k = sgn((q-p[j])^(p[i]-p[j]));
682         int u = sgn(p[i].y-q.y);
683         int v = sgn(p[j].y-q.y);
684         if(k > 0 && u < 0 && v >= 0)cnt++;
685         if(k < 0 && v < 0 && u >= 0)cnt--;
686     }
687     return cnt != 0;
688 }
689 //直线 u 切割凸多边形左侧
690 //注意直线方向
691 //测试: HDU3982
692 void convexcut(Line u,polygon &po){
693     int &top = po.n;//注意引用
694     top = 0;
695     for(int i = 0;i < n;i++){
696         int d1 = sgn((u.e-u.s)^(p[i]-u.s));
697         int d2 = sgn((u.e-u.s)^(p[(i+1)%n]-u.s));
698         if(d1 >= 0)po.p[top++] = p[i];
699         if(d1*d2 < 0)po.p[top++] = u.crosspoint(Line(p[i],p[(i+1)%n]));
700     }
701 }
702 //得到周长
703 //测试 LightOJ1239
704 double getcircumference(){
705     double sum = 0;
706     for(int i = 0;i < n;i++){
707         sum += p[i].distance(p[(i+1)%n]);
708     }
709     return sum;
710 }
711 //得到面积
712 double getarea(){
713     double sum = 0;
714     for(int i = 0;i < n;i++){
715         sum += (p[i]^p[(i+1)%n]);

```

```

716     }
717     return fabs(sum)/2;
718 }
719 //得到方向
720 // 1 表示逆时针, 0 表示顺时针
721 bool getdir(){
722     double sum = 0;
723     for(int i = 0; i < n; i++){
724         sum += (p[i]^p[(i+1)%n]);
725     }
726     if(sgn(sum) > 0) return 1;
727     return 0;
728 }
729 //得到重心
730 Point getbarycentre(){
731     Point ret(0,0);
732     double area = 0;
733     for(int i = 1; i < n-1; i++){
734         double tmp = (p[i]-p[0])^(p[i+1]-p[0]);
735         if(sgn(tmp) == 0) continue;
736         area += tmp;
737         ret.x += (p[0].x+p[i].x+p[i+1].x)/3*tmp;
738         ret.y += (p[0].y+p[i].y+p[i+1].y)/3*tmp;
739     }
740     if(sgn(area)) ret = ret/area;
741     return ret;
742 }
743 //多边形和圆交的面积
744 //测试: POJ3675 HDU3982 HDU2892
745 double areacircle(circle c){
746     double ans = 0;
747     for(int i = 0; i < n; i++){
748         int j = (i+1)%n;
749         if(sgn( (p[j]-c.p)^(p[i]-c.p) ) >= 0)
750             ans += c.reatriangle(p[i],p[j]);
751         else ans -= c.reatriangle(p[i],p[j]);
752     }
753     return fabs(ans);
754 }
755 //多边形和圆关系
756 // 2 圆完全在多边形内
757 // 1 圆在多边形里面, 碰到了多边形边界
758 // 0 其它
759 int relationcircle(circle c){
760     getline();
761     int x = 2;
762     if(relationpoint(c.p) != 1) return 0; //圆心不在内部
763     for(int i = 0; i < n; i++){
764         if(c.relationseg(l[i]) == 2) return 0;
765         if(c.relationseg(l[i]) == 1) x = 1;
766     }
767     return x;

```

```

767     }
768 };
769 //AB X AC
770 double cross(Point A,Point B,Point C){
771     return (B-A)^(C-A);
772 }
773 //AB*AC
774 double dot(Point A,Point B,Point C){
775     return (B-A)*(C-A);
776 }
777 //最小矩形面积覆盖
778 // A 必须是凸包 (而且是逆时针顺序)
779 // 测试 UVA 10173
780 double minRectangleCover(polygon A){
781     //要特判 A.n < 3 的情况
782     if(A.n < 3)return 0.0;
783     A.p[A.n] = A.p[0];
784     double ans = -1;
785     int r = 1, p = 1, q;
786     for(int i = 0;i < A.n;i++){
787         //卡出离边 A.p[i] - A.p[i+1] 最远的点
788         while( sgn( cross(A.p[i],A.p[i+1],A.p[r+1]) - cross(A.p[i],
789             A.p[i+1],A.p[r]) ) >= 0 )
790             r = (r+1)%A.n;
791         //卡出 A.p[i] - A.p[i+1] 方向上正向 n 最远的点
792         while(sgn( dot(A.p[i],A.p[i+1],A.p[p+1]) - dot(A.p[i],A.p[i
793             +1],A.p[p]) ) >= 0 )
794             p = (p+1)%A.n;
795         if(i == 0)q = p;
796         //卡出 A.p[i] - A.p[i+1] 方向上负向最远的点
797         while(sgn(dot(A.p[i],A.p[i+1],A.p[q+1]) - dot(A.p[i],A.p[i
798             +1],A.p[q])) <= 0)
799             q = (q+1)%A.n;
800         double d = (A.p[i] - A.p[i+1]).len2();
801         double tmp = cross(A.p[i],A.p[i+1],A.p[r]) *
802             (dot(A.p[i],A.p[i+1],A.p[p]) - dot(A.p[i],A.p[i+1],A.p[
803             q]))/d;
804         if(ans < 0 || ans > tmp)ans = tmp;
805     }
806     return ans;
807 }
808 //直线切凸多边形
809 //多边形是逆时针的, 在 q1q2 的左侧
810 //测试:HDU3982
811 vector<Point> convexCut(const vector<Point> &ps,Point q1,Point q2){
812     vector<Point>q;
813     int n = ps.size();
814     for(int i = 0;i < n;i++){
815         Point p1 = ps[i], p2 = ps[(i+1)%n];
816         int d1 = sgn((q2-q1)^(p1-q1)), d2 = sgn((q2-q1)^(p2-q1));

```

```

814         if(d1 >= 0)
815             qs.push_back(p1);
816         if(d1 * d2 < 0)
817             qs.push_back(Line(p1,p2).crosspoint(Line(q1,q2)));
818     }
819     return qs;
820 }
821 //半平面交
822 //测试 POJ3335 POJ1474 POJ1279
823 //*****
824 struct halfplane:public Line{
825     double angle;
826     halfplane(){}
827     //表示向量 s->e 逆时针 (左侧) 的半平面
828     halfplane(Point _s,Point _e){
829         s = _s;
830         e = _e;
831     }
832     halfplane(Line v){
833         s = v.s;
834         e = v.e;
835     }
836     void calcangle(){
837         angle = atan2(e.y-s.y,e.x-s.x);
838     }
839     bool operator <(const halfplane &b)const{
840         return angle < b.angle;
841     }
842 };
843 struct halfplanes{
844     int n;
845     halfplane hp[maxp];
846     Point p[maxp];
847     int que[maxp];
848     int st,ed;
849     void push(halfplane tmp){
850         hp[n++] = tmp;
851     }
852     //去重
853     void unique(){
854         int m = 1;
855         for(int i = 1;i < n;i++){
856             if(sgn(hp[i].angle-hp[i-1].angle) != 0)
857                 hp[m++] = hp[i];
858             else if(sgn( (hp[m-1].e-hp[m-1].s)^(hp[i].s-hp[m-1].s)
859                 ) > 0)
860                 hp[m-1] = hp[i];
861         }
862         n = m;
863     }
864     bool halfplaneinsert(){

```

```

864     for(int i = 0;i < n;i++)hp[i].calcangle();
865     sort(hp,hp+n);
866     unique();
867     que[st=0] = 0;
868     que[ed=1] = 1;
869     p[1] = hp[0].crosspoint(hp[1]);
870     for(int i = 2;i < n;i++){
871         while(st<ed && sgn((hp[i].e-hp[i].s)^(p[ed]-hp[i].s))
            <0)ed--;
872         while(st<ed && sgn((hp[i].e-hp[i].s)^(p[st+1]-hp[i].s))
            <0)st++;
873         que[++ed] = i;
874         if(hp[i].parallel(hp[que[ed-1]]))return false;
875         p[ed]=hp[i].crosspoint(hp[que[ed-1]]);
876     }
877     while(st<ed && sgn((hp[que[st]].e-hp[que[st]].s)^(p[ed]-hp[
        que[st]].s))<0)ed--;
878     while(st<ed && sgn((hp[que[ed]].e-hp[que[ed]].s)^(p[st+1]-
        hp[que[ed]].s))<0)st++;
879     if(st+1>=ed)return false;
880     return true;
881 }
882 //得到最后半平面交得到的凸多边形
883 //需要先调用 halfplaneinsert() 且返回 true
884 void getconvex(polygon &con){
885     p[st] = hp[que[st]].crosspoint(hp[que[ed]]);
886     con.n = ed-st+1;
887     for(int j = st,i = 0;j <= ed;i++,j++)
888         con.p[i] = p[j];
889 }
890 };
891 //*****
892
893 const int maxn = 1010;
894 struct circles{
895     circle c[maxn];
896     double ans[maxn]; //ans[i] 表示被覆盖了 i 次的面积
897     double pre[maxn];
898     int n;
899     circles(){}
900     void add(circle cc){
901         c[n++] = cc;
902     }
903     //x 包含在 y 中
904     bool inner(circle x,circle y){
905         if(x.relationcircle(y) != 1)return 0;
906         return sgn(x.r-y.r)<=0?1:0;
907     }
908     //圆的面积并去掉内含的圆
909     void init_or(){
910         bool mark[maxn] = {0};

```

```

911     int i,j,k=0;
912     for(i = 0;i < n;i++){
913         for(j = 0;j < n;j++){
914             if(i != j && !mark[j]){
915                 if( (c[i]==c[j])||inner(c[i],c[j]) )break;
916             }
917             if(j < n)mark[i] = 1;
918         }
919         for(i = 0;i < n;i++){
920             if(!mark[i])
921                 c[k++] = c[i];
922         }
923         n = k;
924     }
925     //圆的面积交去掉内含的圆
926     void init_add(){
927         int i,j,k;
928         bool mark[maxn] = {0};
929         for(i = 0;i < n;i++){
930             for(j = 0;j < n;j++){
931                 if(i != j && !mark[j]){
932                     if( (c[i]==c[j])||inner(c[j],c[i]) )break;
933                 }
934                 if(j < n)mark[i] = 1;
935             }
936             for(i = 0;i < n;i++){
937                 if(!mark[i])
938                     c[k++] = c[i];
939             }
940             n = k;
941         }
942         //半径为 r 的圆，弧度为 th 对应的弓形的面积
943         double areaarc(double th,double r){
944             return 0.5*r*r*(th-sin(th));
945         }
946         //测试 SPOJVCIRCLES SPOJCIRUT
947         //SPOJVCIRCLES 求 n 个圆并的面积，需要加上 init_or() 去掉重复圆（否则
948         //WA）
949         //SPOJCIRUT 是求被覆盖 k 次的面积，不能加 init_or()
950         //对于求覆盖多少次面积的问题，不能解决相同圆，而且不能 init_or()
951         //求多圆面积并，需要 init_or，其中一个目的就是去掉相同圆
952         void getarea(){
953             memset(ans,0,sizeof(ans));
954             vector<pair<double,int> >v;
955             for(int i = 0;i < n;i++){
956                 v.clear();
957                 v.push_back(make_pair(-pi,1));
958                 v.push_back(make_pair(pi,-1));
959                 for(int j = 0;j < n;j++){
960                     if(i != j){
961                         Point q = (c[j].p - c[i].p);
962                         double ab = q.len(),ac = c[i].r, bc = c[j].r;
963                         if(sgn(ab+ac-bc)<=0){

```

```

961         v.push_back(make_pair(-pi,1));
962         v.push_back(make_pair(pi,-1));
963         continue;
964     }
965     if(sgn(ab+bc-ac)<=0)continue;
966     if(sgn(ab-ac-bc)>0)continue;
967     double th = atan2(q.y,q.x), fai = acos((ac*ac+
        ab*ab-bc*bc)/(2.0*ac*ab));
968     double a0 = th-fai;
969     if(sgn(a0+pi)<0)a0+=2*pi;
970     double a1 = th+fai;
971     if(sgn(a1-pi)>0)a1-=2*pi;
972     if(sgn(a0-a1)>0){
973         v.push_back(make_pair(a0,1));
974         v.push_back(make_pair(pi,-1));
975         v.push_back(make_pair(-pi,1));
976         v.push_back(make_pair(a1,-1));
977     }
978     else{
979         v.push_back(make_pair(a0,1));
980         v.push_back(make_pair(a1,-1));
981     }
982 }
983 sort(v.begin(),v.end());
984 int cur = 0;
985 for(int j = 0;j < v.size();j++){
986     if(cur && sgn(v[j].first-pre[cur])){
987         ans[cur] += areaarc(v[j].first-pre[cur],c[i].r)
988         ;
989         ans[cur] += 0.5*(Point(c[i].p.x+c[i].r*cos(pre[
            cur]),c[i].p.y+c[i].r*sin(pre[cur]))^Point(c
            [i].p.x+c[i].r*cos(v[j].first),c[i].p.y+c[i
            ].r*sin(v[j].first)));
990     }
991     cur += v[j].second;
992     pre[cur] = v[j].first;
993 }
994 for(int i = 1;i < n;i++)
995     ans[i] -= ans[i+1];
996 }
997 };

```

7.2 三维几何

```

1  const double eps = 1e-8;
2  int sgn(double x){
3      if(fabs(x) < eps)return 0;
4      if(x < 0)return -1;
5      else return 1;
6  }
7  struct Point3{

```



```

8     double x,y,z;
9     Point3(double _x = 0,double _y = 0,double _z = 0){
10         x = _x;
11         y = _y;
12         z = _z;
13     }
14     void input(){
15         scanf("%lf%lf%lf",&x,&y,&z);
16     }
17     void output(){
18         scanf("%.2lf□%.2lf□%.2lf\n",x,y,z);
19     }
20     bool operator ==(const Point3 &b)const{
21         return sgn(x-b.x) == 0 && sgn(y-b.y) == 0 && sgn(z-b.z) ==
            0;
22     }
23     bool operator <(const Point3 &b)const{
24         return sgn(x-b.x)==0?(sgn(y-b.y)==0?sgn(z-b.z)<0:y<b.y):x<b
            .x;
25     }
26     double len(){
27         return sqrt(x*x+y*y+z*z);
28     }
29     double len2(){
30         return x*x+y*y+z*z;
31     }
32     double distance(const Point3 &b)const{
33         return sqrt((x-b.x)*(x-b.x)+(y-b.y)*(y-b.y)+(z-b.z)*(z-b.z)
            );
34     }
35     Point3 operator -(const Point3 &b)const{
36         return Point3(x-b.x,y-b.y,z-b.z);
37     }
38     Point3 operator +(const Point3 &b)const{
39         return Point3(x+b.x,y+b.y,z+b.z);
40     }
41     Point3 operator *(const double &k)const{
42         return Point3(x*k,y*k,z*k);
43     }
44     Point3 operator /(const double &k)const{
45         return Point3(x/k,y/k,z/k);
46     }
47     //点乘
48     double operator *(const Point3 &b)const{
49         return x*b.x+y*b.y+z*b.z;
50     }
51     //叉乘
52     Point3 operator ^(const Point3 &b)const{
53         return Point3(y*b.z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x);
54     }
55     double rad(Point3 a,Point3 b){

```

```

56         Point3 p = (*this);
57         return acos( ( (a-p)*(b-p) ) / (a.distance(p)*b.distance(p))
58             );
59     }
60     //变换长度
61     Point3 trunc(double r){
62         double l = len();
63         if(!sgn(l))return *this;
64         r /= l;
65         return Point3(x*r,y*r,z*r);
66     };
67     struct Line3
68     {
69         Point3 s,e;
70         Line3(){}
71         Line3(Point3 _s,Point3 _e)
72         {
73             s = _s;
74             e = _e;
75         }
76         bool operator ==(const Line3 v)
77         {
78             return (s==v.s)&&(e==v.e);
79         }
80         void input()
81         {
82             s.input();
83             e.input();
84         }
85         double length()
86         {
87             return s.distance(e);
88         }
89         //点到直线距离
90         double dispointtoline(Point3 p)
91         {
92             return ((e-s)^(p-s)).len()/s.distance(e);
93         }
94         //点到线段距离
95         double dispointtoseg(Point3 p)
96         {
97             if(sgn((p-s)*(e-s)) < 0 || sgn((p-e)*(s-e)) < 0)
98                 return min(p.distance(s),e.distance(p));
99             return dispointtoline(p);
100         }
101         //返回点 p 在直线上的投影
102         Point3 lineprog(Point3 p)
103         {
104             return s + ( ((e-s)*((e-s)*(p-s)))/((e-s).len2()) );
105         }

```

```

106 //p 绕此向量逆时针 arg 角度
107 Point3 rotate(Point3 p,double ang)
108 {
109     if(sgn(((s-p)^(e-p)).len()) == 0)return p;
110     Point3 f1 = (e-s)^(p-s);
111     Point3 f2 = (e-s)^(f1);
112     double len = ((s-p)^(e-p)).len()/s.distance(e);
113     f1 = f1.trunc(len); f2 = f2.trunc(len);
114     Point3 h = p+f2;
115     Point3 pp = h+f1;
116     return h + ((p-h)*cos(ang)) + ((pp-h)*sin(ang));
117 }
118 //点在直线上
119 bool pointonseg(Point3 p)
120 {
121     return sgn( ((s-p)^(e-p)).len() ) == 0 && sgn((s-p)*(e-p))
        == 0;
122 }
123 };
124 struct Plane
125 {
126     Point3 a,b,c,o;//平面上的三个点, 以及法向量
127     Plane(){}
128     Plane(Point3 _a,Point3 _b,Point3 _c)
129     {
130         a = _a;
131         b = _b;
132         c = _c;
133         o = pvec();
134     }
135     Point3 pvec()
136     {
137         return (b-a)^(c-a);
138     }
139     //ax+by+cz+d = 0
140     Plane(double _a,double _b,double _c,double _d)
141     {
142         o = Point3(_a,_b,_c);
143         if(sgn(_a) != 0)
144             a = Point3((-_d-_c-_b)/_a,1,1);
145         else if(sgn(_b) != 0)
146             a = Point3(1,(-_d-_c-_a)/_b,1);
147         else if(sgn(_c) != 0)
148             a = Point3(1,1,(-_d-_a-_b)/_c);
149     }
150     //点在平面上的判断
151     bool pointonplane(Point3 p)
152     {
153         return sgn((p-a)*o) == 0;
154     }
155     //两平面夹角

```

```

156     double angleplane(Plane f)
157     {
158         return acos(o*f.o)/(o.len()*f.o.len());
159     }
160     //平面和直线的交点，返回值是交点个数
161     int crossline(Line3 u,Point3 &p)
162     {
163         double x = o*(u.e-a);
164         double y = o*(u.s-a);
165         double d = x-y;
166         if(sgn(d) == 0)return 0;
167         p = ((u.s*x)-(u.e*y))/d;
168         return 1;
169     }
170     //点到平面最近点（也就是投影）
171     Point3 pointtoplane(Point3 p)
172     {
173         Line3 u = Line3(p,p+o);
174         crossline(u,p);
175         return p;
176     }
177     //平面和平面的交线
178     int crossplane(Plane f,Line3 &u)
179     {
180         Point3 oo = o^f.o;
181         Point3 v = o^oo;
182         double d = fabs(f.o*v);
183         if(sgn(d) == 0)return 0;
184         Point3 q = a + (v*(f.o*(f.a-a))/d);
185         u = Line3(q,q+oo);
186         return 1;
187     }
188 };

```

7.3 平面最近点对

HDU1007/ZOJ2107

```

1  const int MAXN = 100010;
2  const double eps = 1e-8;
3  const double INF = 1e20;
4  struct Point{
5      double x,y;
6      void input(){
7          scanf("%lf%lf",&x,&y);
8      }
9  };
10 double dist(Point a,Point b){
11     return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
12 }
13 Point p[MAXN];
14 Point tmp[MAXN];
15 bool cmpx(Point a,Point b){

```

```

16     return a.x < b.x || (a.x == b.x && a.y < b.y);
17 }
18 bool cmpy(Point a, Point b){
19     return a.y < b.y || (a.y == b.y && a.x < b.x);
20 }
21 double Closest_Pair(int left, int right){
22     double d = INF;
23     if(left == right) return d;
24     if(left+1 == right) return dist(p[left], p[right]);
25     int mid = (left+right)/2;
26     double d1 = Closest_Pair(left, mid);
27     double d2 = Closest_Pair(mid+1, right);
28     d = min(d1, d2);
29     int cnt = 0;
30     for(int i = left; i <= right; i++){
31         if(fabs(p[mid].x - p[i].x) <= d)
32             tmpt[cnt++] = p[i];
33     }
34     sort(tmpt, tmpt+cnt, cmpy);
35     for(int i = 0; i < cnt; i++){
36         for(int j = i+1; j < cnt && tmpt[j].y - tmpt[i].y < d; j++){
37             d = min(d, dist(tmpt[i], tmpt[j]));
38         }
39     }
40     return d;
41 }
42 int main(){
43     int n;
44     while(scanf("%d", &n) == 1 && n){
45         for(int i = 0; i < n; i++) p[i].input();
46         sort(p, p+n, cmpx);
47         printf("%.2lf\n", Closest_Pair(0, n-1));
48     }
49     return 0;
50 }

```

7.4 三维凸包

7.4.1 HDU4273

HDU 4273 给一个三维凸包，求重心到表面的最短距离。

```

1  const double eps = 1e-8;
2  const int MAXN = 550;
3  int sgn(double x){
4      if(fabs(x) < eps) return 0;
5      if(x < 0) return -1;
6      else return 1;
7  }
8  struct Point3{
9      double x, y, z;
10     Point3(double _x = 0, double _y = 0, double _z = 0){
11         x = _x;

```

```

12     y = _y;
13     z = _z;
14 }
15 void input(){
16     scanf("%lf%lf%lf",&x,&y,&z);
17 }
18 bool operator ==(const Point3 &b)const{
19     return sgn(x-b.x) == 0 && sgn(y-b.y) == 0 && sgn(z-b.z) ==
        0;
20 }
21 double len(){
22     return sqrt(x*x+y*y+z*z);
23 }
24 double len2(){
25     return x*x+y*y+z*z;
26 }
27 double distance(const Point3 &b)const{
28     return sqrt((x-b.x)*(x-b.x)+(y-b.y)*(y-b.y)+(z-b.z)*(z-b.z)
        );
29 }
30 Point3 operator -(const Point3 &b)const{
31     return Point3(x-b.x,y-b.y,z-b.z);
32 }
33 Point3 operator +(const Point3 &b)const{
34     return Point3(x+b.x,y+b.y,z+b.z);
35 }
36 Point3 operator *(const double &k)const{
37     return Point3(x*k,y*k,z*k);
38 }
39 Point3 operator /(const double &k)const{
40     return Point3(x/k,y/k,z/k);
41 }
42 //点乘
43 double operator *(const Point3 &b)const{
44     return x*b.x + y*b.y + z*b.z;
45 }
46 //叉乘
47 Point3 operator ^(const Point3 &b)const{
48     return Point3(y*b.z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x);
49 }
50 };
51 struct CH3D{
52     struct face{
53         //表示凸包一个面上的三个点的编号
54         int a,b,c;
55         //表示该面是否属于最终的凸包上的面
56         bool ok;
57     };
58     //初始顶点数
59     int n;
60     Point3 P[MAXN];

```

```

61 //凸包表面的三角形数
62 int num;
63 //凸包表面的三角形
64 face F[8*MAXN];
65 int g[MAXN][MAXN];
66 //叉乘
67 Point3 cross(const Point3 &a,const Point3 &b,const Point3 &c){
68     return (b-a)^(c-a);
69 }
70 //三角形面积 *2
71 double area(Point3 a,Point3 b,Point3 c){
72     return ((b-a)^(c-a)).len();
73 }
74 //四面体有向面积 *6
75 double volume(Point3 a,Point3 b,Point3 c,Point3 d){
76     return ((b-a)^(c-a))*(d-a);
77 }
78 //正: 点在面同向
79 double dblcmp(Point3 &p,face &f){
80     Point3 p1 = P[f.b] - P[f.a];
81     Point3 p2 = P[f.c] - P[f.a];
82     Point3 p3 = p - P[f.a];
83     return (p1^p2)*p3;
84 }
85 void deal(int p,int a,int b){
86     int f = g[a][b];
87     face add;
88     if(F[f].ok){
89         if(dblcmp(P[p],F[f]) > eps)
90             dfs(p,f);
91         else {
92             add.a = b;
93             add.b = a;
94             add.c = p;
95             add.ok = true;
96             g[p][b] = g[a][p] = g[b][a] = num;
97             F[num++] = add;
98         }
99     }
100 }
101 //递归搜索所有应该从凸包内删除的面
102 void dfs(int p,int now){
103     F[now].ok = false;
104     deal(p,F[now].b,F[now].a);
105     deal(p,F[now].c,F[now].b);
106     deal(p,F[now].a,F[now].c);
107 }
108 bool same(int s,int t){
109     Point3 &a = P[F[s].a];
110     Point3 &b = P[F[s].b];
111     Point3 &c = P[F[s].c];

```

```

112         return fabs(volume(a,b,c,P[F[t].a])) < eps &&
113             fabs(volume(a,b,c,P[F[t].b])) < eps &&
114             fabs(volume(a,b,c,P[F[t].c])) < eps;
115     }
116     //构建三维凸包
117     void create(){
118         num = 0;
119         face add;
120
121         //*****
122         //此段是为了保证前四个点不共面
123         bool flag = true;
124         for(int i = 1;i < n;i++){
125             if(!(P[0] == P[i])){
126                 swap(P[1],P[i]);
127                 flag = false;
128                 break;
129             }
130         }
131         if(flag)return;
132         flag = true;
133         for(int i = 2;i < n;i++){
134             if( ((P[1]-P[0])^(P[i]-P[0])).len() > eps ){
135                 swap(P[2],P[i]);
136                 flag = false;
137                 break;
138             }
139         }
140         if(flag)return;
141         flag = true;
142         for(int i = 3;i < n;i++){
143             if(fabs( ((P[1]-P[0])^(P[2]-P[0]))*(P[i]-P[0]) ) > eps)
144                 {
145                     swap(P[3],P[i]);
146                     flag = false;
147                     break;
148                 }
149         }
150         if(flag)return;
151         //*****
152         for(int i = 0;i < 4;i++){
153             add.a = (i+1)%4;
154             add.b = (i+2)%4;
155             add.c = (i+3)%4;
156             add.ok = true;
157             if(dblcmp(P[i],add) > 0)swap(add.b,add.c);
158             g[add.a][add.b] = g[add.b][add.c] = g[add.c][add.a] =
159                 num;
160             F[num++] = add;
161         }

```



```

161     for(int i = 4;i < n;i++)
162         for(int j = 0;j < num;j++)
163             if(F[j].ok && dblcmp(P[i],F[j]) > eps){
164                 dfs(i,j);
165                 break;
166             }
167     int tmp = num;
168     num = 0;
169     for(int i = 0;i < tmp;i++)
170         if(F[i].ok)
171             F[num++] = F[i];
172 }
173 //表面积
174 //测试: HDU3528
175 double area(){
176     double res = 0;
177     if(n == 3){
178         Point3 p = cross(P[0],P[1],P[2]);
179         return p.len()/2;
180     }
181     for(int i = 0;i < num;i++)
182         res += area(P[F[i].a],P[F[i].b],P[F[i].c]);
183     return res/2.0;
184 }
185 double volume(){
186     double res = 0;
187     Point3 tmp = Point3(0,0,0);
188     for(int i = 0;i < num;i++)
189         res += volume(tmp,P[F[i].a],P[F[i].b],P[F[i].c]);
190     return fabs(res/6);
191 }
192 //表面三角形个数
193 int triangle(){
194     return num;
195 }
196 //表面多边形个数
197 //测试: HDU3662
198 int polygon(){
199     int res = 0;
200     for(int i = 0;i < num;i++){
201         bool flag = true;
202         for(int j = 0;j < i;j++)
203             if(same(i,j)){
204                 flag = 0;
205                 break;
206             }
207         res += flag;
208     }
209     return res;
210 }
211 //重心

```

```

212 //测试: HDU4273
213 Point3 barycenter(){
214     Point3 ans = Point3(0,0,0);
215     Point3 o = Point3(0,0,0);
216     double all = 0;
217     for(int i = 0; i < num; i++){
218         double vol = volume(o, P[F[i].a], P[F[i].b], P[F[i].c]);
219         ans = ans + (((o + P[F[i].a] + P[F[i].b] + P[F[i].c]) / 4.0) *
220             vol);
221         all += vol;
222     }
223     ans = ans / all;
224     return ans;
225 }
226 //点到面的距离
227 //测试: HDU4273
228 double ptoface(Point3 p, int i){
229     double tmp1 = fabs(volume(P[F[i].a], P[F[i].b], P[F[i].c], p));
230     ;
231     double tmp2 = ((P[F[i].b] - P[F[i].a]) ^ (P[F[i].c] - P[F[i].a]))
232         .len();
233     return tmp1 / tmp2;
234 }
235 };
236 CH3D hull;
237 int main()
238 {
239     while(scanf("%d", &hull.n) == 1){
240         for(int i = 0; i < hull.n; i++) hull.P[i].input();
241         hull.create();
242         Point3 p = hull.barycenter();
243         double ans = 1e20;
244         for(int i = 0; i < hull.num; i++)
245             ans = min(ans, hull.ptoface(p, i));
246         printf("%.3lf\n", ans);
247     }
248     return 0;
249 }

```

8 其他

8.1 高精度

高精度，支持乘法和加法

```

1  /*
2  * 高精度，支持乘法和加法
3  */
4  struct BigInt{
5      const static int mod = 10000;
6      const static int DLEN = 4;
7      int a[600], len;
8      BigInt(){
9          memset(a, 0, sizeof(a));
10         len = 1;
11     }
12     BigInt(int v){
13         memset(a, 0, sizeof(a));
14         len = 0;
15         do{
16             a[len++] = v%mod;
17             v /= mod;
18         }while(v);
19     }
20     BigInt(const char s[]){
21         memset(a, 0, sizeof(a));
22         int L = strlen(s);
23         len = L/DLEN;
24         if(L%DLEN)len++;
25         int index = 0;
26         for(int i = L-1; i >= 0; i -= DLEN){
27             int t = 0;
28             int k = i - DLEN + 1;
29             if(k < 0)k = 0;
30             for(int j = k; j <= i; j++){
31                 t = t*10 + s[j] - '0';
32                 a[index++] = t;
33             }
34         }
35         BigInt operator +(const BigInt &b) const{
36             BigInt res;
37             res.len = max(len, b.len);
38             for(int i = 0; i <= res.len; i++){
39                 res.a[i] = 0;
40             }
41             for(int i = 0; i < res.len; i++){
42                 res.a[i] += ((i < len)?a[i]:0) + ((i < b.len)?b.a[i]:0);
43                 res.a[i+1] += res.a[i]/mod;
44                 res.a[i] %= mod;
45             }
46             if(res.a[res.len] > 0)res.len++;
47             return res;

```

```

47     }
48     BigInt operator *(const BigInt &b) const{
49         BigInt res;
50         for(int i = 0; i < len; i++){
51             int up = 0;
52             for(int j = 0; j < b.len; j++){
53                 int temp = a[i]*b.a[j] + res.a[i+j] + up;
54                 res.a[i+j] = temp%mod;
55                 up = temp/mod;
56             }
57             if(up != 0)
58                 res.a[i + b.len] = up;
59         }
60         res.len = len + b.len;
61         while(res.a[res.len - 1] == 0 && res.len > 1) res.len--;
62         return res;
63     }
64     void output(){
65         printf("%d", a[len-1]);
66         for(int i = len-2; i >= 0; i--)
67             printf("%04d", a[i]);
68         printf("\n");
69     }
70 };

```

8.2 完全高精度

HDU 1134 求卡特兰数

```

1  /*
2   * 完全大数模板
3   * 输入 cin>>a
4   * 输出 a.print();
5   * 注意这个输入不能自动去掉前导 0 的，可以先读入到 char 数组，去掉前导 0，再用
   构造函数。
6   */
7  #define MAXN 9999
8  #define MAXSIZE 1010
9  #define DLEN 4
10 class BigInt{
11 private:
12     int a[500]; //可以控制大数的位数
13     int len;
14 public:
15     BigInt(){len=1; memset(a, 0, sizeof(a));} //构造函数
16     BigInt(const int); //将一个 int 类型的变量转化成大数
17     BigInt(const char*); //将一个字符串类型的变量转化为大数
18     BigInt(const BigInt &); //拷贝构造函数
19     BigInt &operator=(const BigInt &); //重载赋值运算符，大数之间进行赋值运算
20     friend istream& operator>>(istream&, BigInt&); //重载输入运算符
21     friend ostream& operator<<(ostream&, BigInt&); //重载输出运算符

```

```

22
23     BigNum operator+(const BigNum &)const;    //重载加法运算符，两个大数
        之间的相加运算
24     BigNum operator-(const BigNum &)const;    //重载减法运算符，两个大数
        之间的相减运算
25     BigNum operator*(const BigNum &)const;    //重载乘法运算符，两个大数
        之间的相乘运算
26     BigNum operator/(const int &)const;        //重载除法运算符，大数对一
        个整数进行相除运算
27
28     BigNum operator^(const int &)const;        //大数的 n 次方运算
29     int operator%(const int &)const;          //大数对一个类型的变量进行
        取模运算int
30     bool operator>(const BigNum &T)const;      //大数和另一个大数的大小比
        较
31     bool operator>(const int &t)const;         //大数和一个 int 类型的变
        量的大小比较
32
33     void print();          //输出大数
34 };
35 //将一个 int 类型的变量转化为大数
36 BigNum::BigNum(const int b){
37     int c,d=b;
38     len=0;
39     memset(a,0,sizeof(a));
40     while(d>MAXN){
41         c=d-(d/(MAXN+1))*(MAXN+1);
42         d=d/(MAXN+1);
43         a[len++]=c;
44     }
45     a[len++]=d;
46 }
47 //将一个字符串类型的变量转化为大数
48 BigNum::BigNum(const char *s){
49     int t,k,index,L,i;
50     memset(a,0,sizeof(a));
51     L=strlen(s);
52     len=L/DLEN;
53     if(L%DLEN)len++;
54     index=0;
55     for(i=L-1;i>=0;i-=DLEN){
56         t=0;
57         k=i-DLEN+1;
58         if(k<0)k=0;
59         for(int j=k;j<=i;j++){
60             t=t*10+s[j]-'0';
61             a[index++]=t;
62         }
63     }
64 //拷贝构造函数
65 BigNum::BigNum(const BigNum &T):len(T.len){

```

```

66     int i;
67     memset(a,0,sizeof(a));
68     for(i=0;i<len;i++)
69         a[i]=T.a[i];
70 }
71 //重载赋值运算符，大数之间赋值运算
72 BigNum & BigNum::operator=(const BigNum &n){
73     int i;
74     len=n.len;
75     memset(a,0,sizeof(a));
76     for(i=0;i<len;i++)
77         a[i]=n.a[i];
78     return *this;
79 }
80 istream& operator>>(istream &in,BigNum &b){
81     char ch[MAXSIZE*4];
82     int i=-1;
83     in>>ch;
84     int L=strlen(ch);
85     int count=0,sum=0;
86     for(i=L-1;i>=0;){
87         sum=0;
88         int t=1;
89         for(int j=0;j<4&& i>=0;j++,i--,t*=10){
90             sum+=(ch[i]-'0')*t;
91         }
92         b.a[count]=sum;
93         count++;
94     }
95     b.len=count++;
96     return in;
97 }
98 //重载输出运算符
99 ostream& operator<<(ostream& out,BigNum& b){
100     int i;
101     cout<<b.a[b.len-1];
102     for(i=b.len-2;i>=0;i--){
103         printf("%04d",b.a[i]);
104     }
105     return out;
106 }
107 //两个大数之间的相加运算
108 BigNum BigNum::operator+(const BigNum &T)const{
109     BigNum t(*this);
110     int i,big;
111     big=T.len>len?T.len:len;
112     for(i=0;i<big;i++){
113         t.a[i]+=T.a[i];
114         if(t.a[i]>MAXN){
115             t.a[i+1]++;
116             t.a[i]-=MAXN+1;

```

```

117     }
118 }
119 if(t.a[big]!=0)
120     t.len=big+1;
121 else t.len=big;
122 return t;
123 }
124 //两个大数之间的相减运算
125 BigNum BigNum::operator-(const BigNum &T)const{
126     int i,j,big;
127     bool flag;
128     BigNum t1,t2;
129     if(*this>T){
130         t1=*this;
131         t2=T;
132         flag=0;
133     }
134     else{
135         t1=T;
136         t2=*this;
137         flag=1;
138     }
139     big=t1.len;
140     for(i=0;i<big;i++){
141         if(t1.a[i]<t2.a[i]){
142             j=i+1;
143             while(t1.a[j]==0)
144                 j++;
145             t1.a[j-]--;
146             while(j>i)
147                 t1.a[j-]+=MAXN;
148             t1.a[i]+=MAXN+1-t2.a[i];
149         }
150         else t1.a[i]-=t2.a[i];
151     }
152     t1.len=big;
153     while(t1.a[t1.len-1]==0 && t1.len>1){
154         t1.len--;
155         big--;
156     }
157     if(flag)
158         t1.a[big-1]=0-t1.a[big-1];
159     return t1;
160 }
161 //两个大数之间的相乘
162 BigNum BigNum::operator*(const BigNum &T)const{
163     BigNum ret;
164     int i,j,up;
165     int temp,temp1;
166     for(i=0;i<len;i++){
167         up=0;

```

```

168         for(j=0;j<T.len;j++){
169             temp=a[i]*T.a[j]+ret.a[i+j]+up;
170             if(temp>MAXN){
171                 temp1=temp-temp/(MAXN+1)*(MAXN+1);
172                 up=temp/(MAXN+1);
173                 ret.a[i+j]=temp1;
174             }
175             else{
176                 up=0;
177                 ret.a[i+j]=temp;
178             }
179         }
180         if(up!=0)
181             ret.a[i+j]=up;
182     }
183     ret.len=i+j;
184     while(ret.a[ret.len-1]==0 && ret.len>1)ret.len--;
185     return ret;
186 }
187 //大数对一个整数进行相除运算
188 BigNum BigNum::operator/(const int &b)const{
189     BigNum ret;
190     int i,down=0;
191     for(i=len-1;i>=0;i--){
192         ret.a[i]=(a[i]+down*(MAXN+1))/b;
193         down=a[i]+down*(MAXN+1)-ret.a[i]*b;
194     }
195     ret.len=len;
196     while(ret.a[ret.len-1]==0 && ret.len>1)
197         ret.len--;
198     return ret;
199 }
200 //大数对一个 int 类型的变量进行取模
201 int BigNum::operator%(const int &b)const{
202     int i,d=0;
203     for(i=len-1;i>=0;i--){
204         d=((d*(MAXN+1))%b+a[i])%b;
205     }
206     return d;
207 }
208 //大数的 n 次方运算
209 BigNum BigNum::operator^(const int &n)const{
210     BigNum t,ret(1);
211     int i;
212     if(n<0)exit(-1);
213     if(n==0)return 1;
214     if(n==1)return *this;
215     int m=n;
216     while(m>1){
217         t=*this;
218         for(i=1;(i<<1)<=m;i<=1)

```



```

219         m-=i;
220         ret=ret*t;
221         if(m==1)ret=ret*(*this);
222     }
223     return ret;
224 }
225 //大数和另一个大数的大小比较
226 bool BigNum::operator>(const BigNum &T)const{
227     int ln;
228     if(len>T.len)return true;
229     else if(len==T.len){
230         ln=len-1;
231         while(a[ln]==T.a[ln]&&ln>=0)
232             ln--;
233         if(ln>=0 && a[ln]>T.a[ln])
234             return true;
235         else
236             return false;
237     }
238     else
239         return false;
240 }
241 //大数和一个 int 类型的变量的大小比较
242 bool BigNum::operator>(const int &t)const{
243     BigNum b(t);
244     return *this>b;
245 }
246 //输出大数
247 void BigNum::print(){
248     int i;
249     printf("%d",a[len-1]);
250     for(i=len-2;i>=0;i--)
251         printf("%04d",a[i]);
252     printf("\n");
253 }
254 BigNum f[110];//卡特兰数
255
256 int main(){
257     f[0]=1;
258     for(int i=1;i<=100;i++)
259         f[i]=f[i-1]*(4*i-2)/(i+1);//卡特兰数递推式
260     int n;
261     while(scanf("%d",&n)==1){
262         if(n==-1)break;
263         f[n].print();
264     }
265     return 0;
266 }

```

8.3 strtok 和 sscanf 结合输入

空格作为分隔输入，读取一行的整数：

```

1      gets(buf);
2      int v;
3      char *p = strtok(buf," ");
4      while(p)
5      {
6          sscanf(p,"%d",&v);
7          p = strtok(NULL," ");
8      }

```

8.4 解决爆栈，手动加栈

防止爆栈最好方法是改变写法，弄成 bfs，或者模拟栈。加栈都是旁门左道，需谨慎！

C++

放在头文件前面

```

1 #pragma comment(linker, "/STACK:1024000000,1024000000")

```

G++

放在主函数里面（汇编开栈不一定适用，和系统有关。需谨慎!）

```

1 int __size__ = 256<<20;
2 char *__p__ = (char *)malloc(__size__)+__size__;
3 __asm__("movl 0,%%esp\n":"r"(__p__));

```

8.5 STL

8.5.1 优先队列 priority_queue

empty() 如果队列为空返回真

pop() 删除对顶元素

push() 加入一个元素

size() 返回优先队列中拥有的元素个数

top() 返回优先队列队顶元素

在默认的优先队列中，优先级高的先出队。在默认的 int 型中先出队的为较大的数。

```

1 priority_queue<int>q1;//大的先出对
2 priority_queue<int,vector<int>,greater<int> >q2; //小的先出队

```

自定义比较函数：

```

1 struct cmp
2 {
3     bool operator()(int x, int y)
4     {
5         return x > y; // x 小的优先级高
6         //也可以写成其他方式，如：return p[x] > p[y]; 表示 p[i] 小的优先级高
7     }
8 };
9 priority_queue<int, vector<int>, cmp>q;//定义方法
10 //其中，第二个参数为容器类型。第三个参数为比较函数。

```

```

1 struct node
2 {
3     int x, y;
4     friend bool operator < (node a, node b)
5     {
6         return a.x > b.x; //结构体中, x 小的优先级高
7     }
8 };
9 priority_queue<node>q; //定义方法
10 //在该结构中, y 为值, x 为优先级。
11 //通过自定义 operator< 操作符来比较元素中的优先级。
12 //在重载" <" 时, 最好不要重载" >", 可能会发生编译错误

```

8.5.2 set 和 multiset

set 和 multiset 用法一样, 就是 multiset 允许重复元素。

元素放入容器时, 会按照一定的排序法则自动排序, 默认是按照 less<> 排序规则来排序。不能修改容器里面的元素值, 只能插入和删除。

自定义 int 排序函数: (默认的是从小到大的, 下面这个从大到小)

```

1 struct classcomp {
2     bool operator() (const int& lhs, const int& rhs) const
3     {return lhs>rhs;}
4 }; //这里有个逗号的, 注意
5 multiset<int, classcomp> fifth; // class as Compare

```

上面这样就定义成了从大到小排列了。

结构体自定义排序函数:

(定义 set 或者 multiset 的时候定义了排序函数, 定义迭代器时一样带上排序函数)

```

1 struct Node
2 {
3     int x, y;
4 };
5 struct classcomp //先按照 x 从小到大排序, x 相同则按照 y 从大到小排序
6 {
7     bool operator()(const Node &a, const Node &b) const
8     {
9         if(a.x != b.x) return a.x < b.x;
10        else return a.y > b.y;
11    }
12 }; //注意这里有个逗号
13 multiset<Node, classcomp> mt;
14 multiset<Node, classcomp>::iterator it;

```

```

1
2 主要函数:
3 begin() 返回指向第一个元素的迭代器
4 clear() 清除所有元素
5 count() 返回某个值元素的个数
6 empty() 如果集合为空, 返回 true

```

```

7 end() 返回指向最后一个元素的迭代器
8 erase() 删除集合中的元素（参数是一个元素值，或者迭代器）
9 find() 返回一个指向被查找到元素的迭代器
10 insert() 在集合中插入元素
11 size() 集合中元素的数目
12 lower_bound() 返回指向大于（或等于）某值的第一个元素的迭代器
13 upper_bound() 返回大于某个值元素的迭代器
14 equal_range() 返回集合中与给定值相等的上下限的两个迭代器
15 （注意对于 multiset 删除操作之间删除值会把所以这个值的都删掉，删除一个要用迭代器）

```

8.6 输入输出外挂

```

1
2 //适用于正负整数
3 template <class T>
4 inline bool scan_d(T &ret) {
5     char c; int sgn;
6     if(c=getchar(),c==EOF) return 0; //EOF
7     while(c!='-'&&(c<'0' || c>'9')) c=getchar();
8     sgn=(c=='-')?-1:1;
9     ret=(c=='-')?0:(c-'0');
10    while(c=getchar(),c>='0'&&c<='9') ret=ret*10+(c-'0');
11    ret*=sgn;
12    return 1;
13 }
14
15 inline void out(int x) {
16     if(x>9) out(x/10);
17     putchar(x%10+'0');
18 }

```

8.7 莫队算法

莫队算法，可以解决一类静态，离线区间查询问题。

BZOJ 2038: [2009 国家集训队] 小 Z 的袜子 (hose)

Description

作为一个生活散漫的人，小 Z 每天早上都要耗费很久从一堆五颜六色的袜子中找出一双来穿。终于有一天，小 Z 再也无法忍受这恼人的找袜子过程，于是他决定听天由命……具体来说，小 Z 把这 N 只袜子从 1 到 N 编号，然后从编号 L 到 R (L

Input

输入文件第一行包含两个正整数 N 和 M 。 N 为袜子的数量， M 为小 Z 所提的询问的数量。接下来一行包含 N 个正整数 C_i ，其中 C_i 表示第 i 只袜子的颜色，相同的颜色用相同的数字表示。再接下来 M 行，每行两个正整数 L, R 表示一个询问。

Output

包含 M 行，对于每个询问在一行中输出分数 A/B 表示从该询问的区间 $[L,R]$ 中随机抽出两只袜子颜色相同的概率。若该概率为 0 则输出 0/1，否则输出的 A/B 必须为最简分数。（详见样例）

Sample Input

```

6 4
1 2 3 3 3 2

```

```

2 6
1 3
3 5
1 6
Sample Output
2/5
0/1
1/1
4/15

```

题解:

只需要统计区间内各个数出现次数的平方和

莫队算法，两种方法，一种是直接分成 \sqrt{n} 块，分块排序。

另外一种求得曼哈顿距离最小生成树，根据 manhattan MST 的 dfs 序求解。

8.7.1 分块

```

1  const int MAXN = 50010;
2  const int MAXM = 50010;
3  struct Query
4  {
5      int L,R,id;
6  }node[MAXN];
7  long long gcd(long long a,long long b){
8      if(b == 0)return a;
9      return gcd(b,a%b);
10 }
11 struct Ans{
12     long long a,b;//分数 a/b
13     void reduce()//分数化简
14     {
15         long long d = gcd(a,b);
16         a /= d; b /= d;
17     }
18 }ans[MAXM];
19 int a[MAXN];
20 int num[MAXN];
21 int n,m,unit;
22 bool cmp(Query a,Query b){
23     if(a.L/unit != b.L/unit)return a.L/unit < b.L/unit;
24     else return a.R < b.R;
25 }
26 void work(){
27     long long temp = 0;
28     memset(num,0,sizeof(num));
29     int L = 1;
30     int R = 0;
31     for(int i = 0;i < m;i++){
32         while(R < node[i].R){

```

```

33         R++;
34         temp -= (long long) num[a[R]] * num[a[R]];
35         num[a[R]]++;
36         temp += (long long) num[a[R]] * num[a[R]];
37     }
38     while(R > node[i].R){
39         temp -= (long long) num[a[R]] * num[a[R]];
40         num[a[R]]--;
41         temp += (long long) num[a[R]] * num[a[R]];
42         R--;
43     }
44     while(L < node[i].L){
45         temp -= (long long) num[a[L]] * num[a[L]];
46         num[a[L]]--;
47         temp += (long long) num[a[L]] * num[a[L]];
48         L++;
49     }
50     while(L > node[i].L){
51         L--;
52         temp -= (long long) num[a[L]] * num[a[L]];
53         num[a[L]]++;
54         temp += (long long) num[a[L]] * num[a[L]];
55     }
56     ans[node[i].id].a = temp - (R-L+1);
57     ans[node[i].id].b = (long long) (R-L+1) * (R-L);
58     ans[node[i].id].reduce();
59 }
60 }
61 int main(){
62     while(scanf("%d%d",&n,&m) == 2){
63         for(int i = 1; i <= n; i++){
64             scanf("%d",&a[i]);
65         }
66         for(int i = 0; i < m; i++){
67             node[i].id = i;
68             scanf("%d%d",&node[i].L,&node[i].R);
69         }
70         unit = (int)sqrt(n);
71         sort(node,node+m,cmp);
72         work();
73         for(int i = 0; i < m; i++){
74             printf("%lld/%lld\n",ans[i].a,ans[i].b);
75         }
76         return 0;
77     }
78 }

```

8.7.2 Manhattan MST 的 dfs 顺序求解

```

1  const int MAXN = 50010;
2  const int MAXM = 50010;
3  const int INF = 0x3f3f3f3f;
4  struct Point{
5      int x,y,id;

```

```

6  }p[MAXN],pp[MAXN];
7  bool cmp(Point a,Point b)
8  {
9      if(a.x != b.x) return a.x < b.x;
10     else return a.y < b.y;
11 }
12 //树状数组, 找 y-x 大于当前的, 但是 y+x 最小的
13 struct BIT{
14     int min_val,pos;
15     void init()
16     {
17         min_val = INF;
18         pos = -1;
19     }
20 }bit[MAXN];
21 struct Edge{
22     int u,v,d;
23 }edge[MAXN<<2];
24 bool cmpedge(Edge a,Edge b){
25     return a.d < b.d;
26 }
27 int tot;
28 int n;
29 int F[MAXN];
30 int find(int x){
31     if(F[x] == -1) return x;
32     else return F[x] = find(F[x]);
33 }
34 void addedge(int u,int v,int d){
35     edge[tot].u = u;
36     edge[tot].v = v;
37     edge[tot++].d = d;
38 }
39 struct Graph{
40     int to,next;
41 }e[MAXN<<1];
42 int total,head[MAXN];
43 void _addedge(int u,int v){
44     e[total].to = v;
45     e[total].next = head[u];
46     head[u] = total++;
47 }
48 int lowbit(int x){
49     return x&(-x);
50 }
51 void update(int i,int val,int pos){
52     while(i > 0){
53         if(val < bit[i].min_val){
54             bit[i].min_val = val;
55             bit[i].pos = pos;
56         }

```

```

57     i -= lowbit(i);
58 }
59 }
60 int ask(int i,int m){
61     int min_val = INF,pos = -1;
62     while(i <= m){
63         if(bit[i].min_val < min_val){
64             min_val = bit[i].min_val;
65             pos = bit[i].pos;
66         }
67         i += lowbit(i);
68     }
69     return pos;
70 }
71 int dist(Point a,Point b){
72     return abs(a.x - b.x) + abs(a.y - b.y);
73 }
74 void Manhattan_minimum_spanning_tree(int n,Point p[]){
75     int a[MAXN],b[MAXN];
76     tot = 0;
77     for(int dir = 0;dir < 4;dir++){
78         if(dir == 1 || dir == 3){
79             for(int i = 0;i < n;i++)
80                 swap(p[i].x,p[i].y);
81         }
82         else if(dir == 2){
83             for(int i = 0;i < n;i++)
84                 p[i].x = -p[i].x;
85         }
86         sort(p,p+n,cmp);
87         for(int i = 0;i < n;i++)
88             a[i] = b[i] = p[i].y - p[i].x;
89         sort(b,b+n);
90         int m = unique(b,b+n) - b;
91         for(int i = 1;i <= m;i++){
92             bit[i].init();
93         }
94         for(int i = n-1;i >= 0;i--){
95             int pos = lower_bound(b,b+m,a[i]) - b + 1;
96             int ans = ask(pos,m);
97             if(ans != -1)
98                 addedge(p[i].id,p[ans].id,dist(p[i],p[ans]));
99             update(pos,p[i].x+p[i].y,i);
100         }
101     }
102     memset(F,-1,sizeof(F));
103     sort(edge,edge+tot,cmpedge);
104     total = 0;
105     memset(head,-1,sizeof(head));
106     for(int i = 0;i < tot;i++){
107         int u = edge[i].u, v = edge[i].v;
108         int t1 = find(u), t2 = find(v);

```



```

108         if(t1 != t2){
109             F[t1] = t2;
110             _addedge(u,v);
111             _addedge(v,u);
112         }
113     }
114 }
115 int m;
116 int a[MAXN];
117 struct Ans{
118     long long a,b;
119 }ans[MAXM];
120 long long temp ;
121 int num[MAXN];
122 void add(int l,int r){
123     for(int i = l;i <= r;i++){
124         temp -= (long long)num[a[i]]*num[a[i]];
125         num[a[i]]++;
126         temp += (long long)num[a[i]]*num[a[i]];
127     }
128 }
129 void del(int l,int r){
130     for(int i = l;i <= r;i++){
131         temp -= (long long)num[a[i]]*num[a[i]];
132         num[a[i]]--;
133         temp += (long long)num[a[i]]*num[a[i]];
134     }
135 }
136 void dfs(int l1,int r1,int l2,int r2,int idx,int pre){
137     if(l2 < l1) add(l2,l1-1);
138     if(r2 > r1) add(r1+1,r2);
139     if(l2 > l1) del(l1,l2-1);
140     if(r2 < r1) del(r2+1,r1);
141     ans[pp[idx].id].a = temp - (r2-l2+1);
142     ans[pp[idx].id].b = (long long)(r2-l2+1)*(r2-l2);
143     for(int i = head[idx];i != -1;i = e[i].next){
144         int v = e[i].to;
145         if(v == pre) continue;
146         dfs(l2,r2,pp[v].x,pp[v].y,v,idx);
147     }
148     if(l2 < l1)del(l2,l1-1);
149     if(r2 > r1)del(r1+1,r2);
150     if(l2 > l1)add(l1,l2-1);
151     if(r2 < r1)add(r2+1,r1);
152 }
153 long long gcd(long long a,long long b){
154     if(b == 0) return a;
155     else return gcd(b,a%b);
156 }
157 int main(){
158     while(scanf("%d%d",&n,&m) == 2){

```

```

159     for(int i = 1;i <= n;i++)
160         scanf("%d",&a[i]);
161     for(int i = 0;i < m;i++){
162         scanf("%d%d",&p[i].x,&p[i].y);
163         p[i].id = i;
164         pp[i] = p[i];
165     }
166     Manhattan_minimum_spanning_tree(m,p);
167     memset(num,0,sizeof(num));
168     temp = 0;
169     dfs(1,0,pp[0].x,pp[0].y,0,-1);
170     for(int i = 0;i < m;i++){
171         long long d = gcd(ans[i].a,ans[i].b);
172         printf("%lld/%lld\n",ans[i].a/d,ans[i].b/d);
173     }
174 }
175 return 0;
176 }

```

8.8 VIM 配置

```

1 set nu
2 set history=1000000
3
4 set tabstop=4
5 set shiftwidth=4
6 set smarttab
7
8 set cindent
9
10 colo evening
11
12 set nobackup
13 set noswapfile
14
15 set mouse=a
16 map <F6> :call CR()<CR>
17 func! CR()
18   exec "w"
19   exec "!g++_%-o_%"
20   exec "!_./%"
21 endfunc
22
23 imap <c-]> {<cr>}<c-o>0<left><right>
24 map <F2> :call SetTitle()<CR>
25 func SetTitle()
26   let l = 0
27   let l = l + 1 | call setline(l,'#include<stdio.h>')
28   let l = l + 1 | call setline(l,'#include<string.h>')
29   let l = l + 1 | call setline(l,'#include<iostream>')
30   let l = l + 1 | call setline(l,'#include<algorithm>')
31   let l = l + 1 | call setline(l,'#include<vector>')

```

```

32 let l = l + 1 | call setline(l,'#include<queue>')
33 let l = l + 1 | call setline(l,'#include<set>')
34 let l = l + 1 | call setline(l,'#include<map>')
35 let l = l + 1 | call setline(l,'#include<string>')
36 let l = l + 1 | call setline(l,'#include<math.h>')
37 let l = l + 1 | call setline(l,'#include<stdlib.h>')
38 let l = l + 1 | call setline(l,'#include<time.h>')
39 let l = l + 1 | call setline(l,'using namespace std;')
40 let l = l + 1 | call setline(l,'')
41 let l = l + 1 | call setline(l,'int main()')
42 let l = l + 1 | call setline(l,'{')
43 let l = l + 1 | call setline(l,'    //freopen("in.txt","r",stdin);'
    )
44 let l = l + 1 | call setline(l,'    //freopen("out.txt","w",stdout)
    ;')
45 let l = l + 1 | call setline(l,'')
46 let l = l + 1 | call setline(l,'    return 0;')
47 let l = l + 1 | call setline(l,'}')
48 endfunc

```

现场赛配置:

```

1 syntax on
2 set nu
3 set tabstop=4
4 set shiftwidth=4
5 set cin
6 colo evening
7 set mouse=a

```