

Meow

Meow 模板

Meow

Ruosen Lee

2016-10-17

目录

图论.....	3	数据结构.....	27
2-SAT.....	3	LCA.....	27
最短路.....	5	RMQ.....	28
A-Star.....	5	差分前缀和.....	28
Bellman-Ford.....	6	线段树.....	29
Bellman-Ford(检查负环).....	6	树状数组.....	30
Dijkstra(次短路).....	7	可持久化数据结构.....	30
Dijkstra(记录路径).....	7	静态主席树.....	30
Dijkstra-pq.....	8	动态主席树.....	31
Floyd.....	8	可持久化线段树.....	33
Spfa.....	9	划分树.....	34
Spfa-slf-pq.....	9	莫队算法.....	36
生成树.....	10	莫队分块.....	36
Prim-pq.....	10	树上莫队.....	37
Kruskal.....	10	树链剖分.....	37
曼哈顿最小生成树.....	11	动态规划.....	39
Prim 次小生成树.....	13	树 DP.....	39
有向图最小生成树.....	14	经典.....	39
最小生成树计数.....	15	删点.....	40
二分图.....	17	树上背包.....	40
匈牙利.....	17	应用：树上任意点能到达的最远距离.....	41
HK.....	17	区间 DP.....	43
KM.....	18	数位 DP.....	43
网络流.....	19	可能的树分治模板.....	44
Dinic.....	19	字符串.....	46
Ford-Flukerson.....	21	AC 自动机.....	46
费用流.....	21	KMP.....	47
欧拉路径.....	23	数学.....	48
Fleury.....	23	Ploya 定理.....	48
全局最小割.....	24	SG 博弈.....	48
差分约束.....	25	逆元处理求组合数(MOD 是质数).....	49
		组合数.....	50
		快速线性素数筛.....	50

计算几何 51

 点与矩阵最小距离 51

 点在多边形内 51

 多边形与圆面积交 52

 矩形面积并 54

 凸多边形面积并 56

 旋转卡壳 58

 圆的面积并(辛普森积分法) 58

杂七杂八 60

 LIS 60

 斯坦纳树 60

 归并排序求逆序对 62

环境 63

 _vimrc 配置文件 63

 CB 配置 63

图论

2-SAT

在实际问题中，2-SAT 问题在大多数时候表现成以下形式：有 N 对物品，每对物品中必须选取一个，也只能选取一个，并且它们之间存在某些限制关系（如某两个物品不能都选，某两个物品不能都不选，某两个物品必须且只能选一个，某个物品必选）等，这时，可以将每对物品当成一个布尔值（选取第一个物品相当于 0，选取第二个相当于 1），如果所有的限制关系最多只对两个物品进行限制，则它们都可以转化成 9 种基本限制关系，从而转化为 2-SAT 模型。

其实 2-SAT 问题的建模是和实际问题非常相似的。

建立一个 2N 阶的有向图，其中的点分为 N 对，每对点表示布尔序列 A 的一个元素的 0、1 取值（以下将代表 A[i] 的 0 取值的点称为 i，代表 A[i] 的 1 取值的点称为 i'）。显然每对点必须且只能选取一个。然后，图中的边具有特定含义。若图中存在边<i, j>，则表示若选了 i 必须选 j。

【O(NM)算法：求字典序最小的解】

根据 2-SAT 建成的图中边的定义可以发现，若图中 i 到 j 有路径，则若 i 选，则 j 也要选；或者说，若 j 不选，则 i 也不能选；

因此得到一个很直观的算法：

（1）给每个点设置一个状态 V，V=0 表示未确定，V=1 表示确定选取，V=2 表示确定不选取。称一个点是已确定的当且仅当其 V 值非 0。设立两个队列 Q1 和 Q2，分别存放本次尝试选取的点的编号和尝试不选的点的编号。

（2）若图中所有的点均已确定，则找到一组解，结束，否则，将 Q1、Q2 清空，并任选一个未确定的点 i，将 i 加入队列 Q1，将 i' 加入队列 Q2；

（3）找到 i 的所有后继。对于后继 j，若 j 未确定，则将 j 加入队列 Q1；若 j'（这里的 j' 是指与 j 在同一对的另一个点）未确定，则将 j' 加入队列 Q2；

（4）遍历 Q2 中的每个点，找到该点的所有前趋（这里需要先建一个补图），若该前趋未确定，则将其加入队列 Q2；

（5）在（3）（4）步操作中，出现以下情况之一，则本次尝试失败，否则本次尝试成功：

<1>某个已被加入队列 Q1 的点被加入队列 Q2；

<2>某个已被加入队列 Q2 的点被加入队列 Q1；

<3>某个 j 的状态为 2；

<4>某个 i' 或 j' 的状态为 1 或某个 i' 或 j' 的前趋的状态为 1；

（6）若本次尝试成功，则将 Q1 中的所有点的状态改为 1，将 Q2 中所有点的状态改为 2，转（2），否则尝试点 i'，若仍失败则问题无解。

该算法的时间复杂度为 O(NM)（最坏情况下要尝试所有的点，每次尝试要遍历所有的边），但是在多数情况下，远远达不到这个上界。

具体实现时，可以用一个数组 vst 来表示队列 Q1 和 Q2。设立两个标志变量 i1 和 i2（要求对于不同的 i，i1 和 i2 均不同，这样可以避免每次尝试都要初始化一次，节省时间），若 vst[i]=i1 则表示 i 已被加入 Q1，若 vst[i]=i2 则表示 i 已被加入 Q2。不过 Q1 和 Q2 仍然是要设立的，因为遍历（BFS）的时候需要队列，为了防止重复遍历，加入 Q1（或 Q2）中的点的 vst 值必然不等于 i1（或 i2）。中间一旦发生矛盾，立即中止尝试，宣告失败。

该算法虽然在多数情况下时间复杂度到不了 O(NM)，但是综合性能仍然不如下面的 O(M)算法。不过，该算法有一个很重要的用处：求字典序最小的解！

如果原图中的同一对点编号都是连续的（01、23、45……）则可以依次尝试第 0 对、第 1 对……点，每对点中先尝试编号小的，若失败再尝试编号大的。这样一定能求出字典序最小的解（如果有解的话），因为一个点一旦被确定，则不可更改。

如果原图中的同一对点编号不连续（比如 03、25、14……）则按照该对点中编号小的点的编号递增顺序将每对点排序，然后依次扫描排序后的每对点，先尝试其编号小的点，若成功则将这个点选上，否则尝试编号大的点，若成功则选上，否则（都失败）无解。

01: const int INF = 0x7F7F7F7F;

02: const int MAXN = 1000 + 10;

03: const int MAXM = 1000000 + 10;

04:

05: struct Edge{ int to, next; };

06: Edge es[MAXM];

07: int head[MAXN], low[MAXN], dfn[MAXN], belong[MAXN], a[MAXN], b[MAXN];

08: bool insta[MAXN];

09: int n, m, cnt, index;

10: stack<int> sta;

11:

12: void add(int u, int v) {

13: es[cnt].to = v; es[cnt].next = head[u]; head[u] = cnt++;

```

14:     return ;
15: }
16:
17: void tarjan( int u ) {
18:     int v;
19:     dfn[u] = low[u] = index++;
20:     sta.push( u );
21:     insta[u] = true;
22:     for( int i = head[u]; ~i; i = es[i].next ) {
23:         v = es[i].to;
24:         if( dfn[v] == -1 ) {
25:             tarjan( v );
26:             low[u] = min( low[u], low[v] );
27:         } else if( insta[v] ) {
28:             low[u] = min( low[u], dfn[v] );
29:         }
30:     }
31:     if( dfn[u] == low[u] ) {
32:         do {
33:             v = sta.top(); sta.pop();
34:             insta[v] = false;
35:             belong[v] = cnt;
36:         } while( u != v );
37:         ++cnt;
38:     }
39:     return ;
40: }
41:
42: int main() {
43:     scanf( "%d%d", &n, &m );
44:     memset( head, -1, sizeof( head ) );
45:     memset( dfn, -1, sizeof( dfn ) );
46:     memset( low, -1, sizeof( low ) );

```

```

47:     memset( insta, false, sizeof( insta ) );
48:     memset( belong, -1, sizeof( belong ) );
49:     cnt = 0;
50:     for( int i = 0; i < m; ++i ) {
51:         scanf( "%d%d", a + i, b + i );
52:         if( a[i] > b[i] ) swap( a[i], b[i] );
53:     }
54:     for( int i = 0; i < m; ++i ) {
55:         for( int j = i + 1; j < m; ++j ) {
56:             if( ( a[i] < a[j] && b[i] < b[j] && a[j] < b[i] ) ||
57:                 ( a[i] > a[j] && b[i] > b[j] && b[j] > a[i] ) ) {
58:                 add( i * 2, j * 2 + 1 );
59:                 add( j * 2, i * 2 + 1 );
60:                 add( i * 2 + 1, j * 2 );
61:                 add( j * 2 + 1, i * 2 );
62:             }
63:         }
64:         index = 1; cnt = 1;
65:         for( int i = 0; i < m * 2; ++i ) {
66:             if( dfn[i] == -1 ) tarjan( i );
67:         }
68:         bool flag = true;
69:         for( int i = 0; i < m * 2; ++i ) {
70:             if( belong[i] == belong[i ^ 1] ) {
71:                 flag = false;
72:                 break;
73:             }
74:         }
75:         if( flag ) printf( "panda is telling the truth...\n" );
76:         else printf( "the evil panda is lying again\n" );
77:         return 0;
78:     }

```

最短路

A-Star

```
/*
    h 为当前代价
    spfa 为逆向搜索，寻找每个节点的估值函数 g 的值
*/
79: const int INF = 0x7F7F7F7F;
80: const int MAXN = 10000 + 10;
81: const int MAXE = 1000000 + 10;
82:
83: struct Edge { int to, cost, next; };
84: Edge es[MAXE];
85: struct Node { int u, f, g; };
86: int head[MAXN], h[MAXN];
87: bool vis[MAXN];
88: int n, m, k, cnt;
89: vector<int> path;
90:
91: struct cmp {
92:     bool operator() ( const Node &a, const Node &b ) {
93:         return a.f > b.f;
94:     }
95: };
96:
97: void add( int u, int v, int w ) {
98:     es[cnt].to = v; es[cnt].cost = w; es[cnt].next = head[u]; head[u]
    = cnt++;
99:     return ;
100: }
101:
```

```
102: void spfa() {
103:     queue<int> que;
104:     memset( vis, false, sizeof( vis ) );
105:     memset( h, 0x7F, sizeof( h ) );
106:     que.push( n ); h[n] = 0; vis[n] = true;
107:     while( !que.empty() ) {
108:         int u = que.front(); que.pop();
109:         for( int i = head[u]; ~i; i = es[i].next ) {
110:             int v = es[i].to;
111:             if( h[v] > h[u] + es[i].cost ) {
112:                 h[v] = h[u] + es[i].cost;
113:                 if( !vis[v] ) {
114:                     vis[v] = true;
115:                     que.push( v );
116:                 }
117:             }
118:         }
119:         vis[u] = false;
120:     }
121:     return ;
122: }
123:
124: void astar() {
125:     priority_queue<Node, vector<Node>, cmp> que;
126:     Node tmp = { 1, h[1], 0 };
127:     que.push( tmp );
128:     for( int cur = 0; cur < k && !que.empty(); ) {
129:         tmp = que.top(), que.pop();
130:         int u = tmp.u, g = tmp.g;
131:         if( u == n ) { ++cur; path.push_back( g ); }
132:         for( int j = head[u]; ~j; j = es[j].next ) {
133:             tmp.u = es[j].to;
134:             tmp.g = g + es[j].cost;
```

```

135:         tmp.f = tmp.g + h[tmp.u];
136:         que.push( tmp );
137:     }
138: }
139: return ;
140:}
141:
142:int main() {
143:    int a, b, c;
144:    k = 2;
145:    while( ~scanf( "%d%d", &n, &m ) ) {
146:        memset( head, -1, sizeof( head ) );
147:        cnt = 0;
148:        for( int i = 1; i <= m; ++i ) {
149:            scanf( "%d%d%d", &a, &b, &c );
150:            add( a, b, c );
151:            add( b, a, c );
152:        }
153:        spfa();
154:        astar();
155:        printf( "%d\n", path[k - 1] );
156:    }
157:    return 0;
158:}

```

Bellman-Ford

```

01: #define INF 0x7FFFFFFF
02: #define MAX_V 1000
03: #define MAX_E 1000000 + 10
04:
05: struct edge { int from, to, cost; };
06: edge es[MAX_E];
07: int d[MAX_V];

```

```

08: int V, E;
09:
10: void bf( int s ) {
11:     for( int i = 0; i < V; ++i ) d[i] = INF;
12:     d[s] = 0;
13:     while( true ) {
14:         bool flag = false;
15:         for( int i = 0; i < E; ++i ) {
16:             edge e = es[i];
17:             if( d[e.from] != INF && d[e.to] > d[e.from] + e.cost ) {
18:                 d[e.to] = d[e.from] + e.cost;
19:                 flag = true;
20:             }
21:         }
22:         if( !flag ) break;
23:     }
24:     return ;
25: }

```

Bellman-Ford(检查负环)

```

01: #define INF 0x7FFFFFFF
02: #define MAX_V 1000
03: #define MAX_E 1000000 + 10
04:
05: struct edge { int from, to, cost; };
06: edge es[MAX_E];
07: int d[MAX_V];
08: int V, E;
09:
10: bool bf_negative( int s ) {
11:     memset( d, 0, sizeof( d ) );
12:     for( int i = 0; i < V; ++i ) {
13:         for( int j = 0; j < E; ++j ) {

```

```

14:         edge e = es[j];
15:         if( d[e.to] > d[e.from] + e.cost ) {
16:             d[e.to] = d[e.from] + e.cost;
17:             if( i == V - 1 ) return true;
18:         }
19:     }
20: }
21: return false;
22: }

```

Dijkstra(次短路)

```

01: #define N 110
02: #define PII pair<int, int>
03: #define INF 0x7FFFFFFF
04: struct edge {
05:     int to, cost;
06:     edge( int t, int c ) { to = t; cost = c; }
07: };
08: vector<edge> G[N];
09: int V;
10: int dis[N], dis2[N];
11:
12: void dijkstra( int s ) {
13:     priority_queue<PII, vector<PII>, greater<PII> > pq;
14:     for( int i = 0; i < V; ++i ) dis[i] = dis2[i] = INF;
15:     dis[s] = 0;
16:     pq.push( PII( dis[s], s ) );
17:     while( !pq.empty() ) {
18:         PII p = pq.top();
19:         pq.pop();
20:         int v = p.second, d = p.first;
21:         if( dis2[v] < d ) continue;
22:         for( int i = 0; i < G[v].size(); ++i ) {

```

```

23:             edge &e = G[v][i];
24:             int d2 = d + e.cost;
25:             if( d2 < dis[e.to] ) {
26:                 swap( d2, dis[e.to] );
27:                 pq.push( PII( dis[e.to], e.to ) );
28:             }
29:             if( d2 < dis2[e.to] && d2 > dis[e.to] ) {
30:                 dis[e.to] = d2;
31:                 pq.push( PII( dis2[e.to], e.to ) );
32:             }
33:         }
34:     }
35:     return ;
36: }
37: int main() {
38:     int m, n;
39:     while( ~scanf( "%d%d" ,&m, &n ) && ( m || n ) ) {
40:         for( int i = 0; i < n; ++i ) G[i].clear();
41:         V = n;
42:         int u, v, w;
43:         for( int i = 0; i < m; ++i ) {
44:             scanf( "%d%d%d", &u, &v, &w );
45:             G[u - 1].push_back( edge( v - 1, w ) );
46:             G[v - 1].push_back( edge( u - 1, w ) );
47:         }
48:         dijkstra( 0 );
49:         cout << dis[n - 1] << endl;
50:     }
51:     return 0;
52: }

```

Dijkstra(记录路径)

```

01: #define N 1010

```



```

02: #define INF 0x7FFFFFFF
03: int d[N], vis[N], pre[N];
04: int cost[N][N];
05: int V;
06:
07: void dijkstra( int s ) {
08:     for( int i = 1; i <= V; ++i ) {
09:         d[i] = INF;
10:         vis[i] = 0;
11:         pre[i] = -1;
12:     }
13:     d[s] = 0;
14:     while( true ) {
15:         int v = -1;
16:         for( int u = 1; u <= V; ++u ) {
17:             if( !vis[u] && ( v == -1 || d[u] < d[v] ) ) v = u;
18:         }
19:
20:         if( v == -1 ) break;
21:         vis[v] = 1;
22:
23:         for( int u = 1; u <= V; ++u ) {
24:             if( d[u] > d[v] + cost[v][u] ) {
25:                 d[u] = d[v] + cost[v][u];
26:                 pre[u] = v;
27:             }
28:         }
29:     }
30:     return ;
31: }

```

Dijkstra-pq

```

01: #define INF 0x7FFFFFFF

```

```

02: #define MAX_V 1000
03: #define MAX_E 1000000 + 10
04: typedef pair<int, int> PII;
05:
06: struct edge { int to, cost; };
07: vector<edge> G[MAX_V];
08: int d[MAX_V];
09: int V;
10:
11: void dijskra( int s ) {
12:     priority_queue<PII, vector<PII>, greater<PII> > pq;
13:     for( int i = 0; i < V; ++i ) d[i] = INF;
14:     d[s] = 0;
15:     pq.push( PII( 0, s ) );
16:     while( !pq.empty() ) {
17:         PII p = pq.top(); pq.pop();
18:         int v = p.second;
19:         if( d[v] < p.first ) continue;
20:         for( int i = 0; i < G[v].size(); ++i ) {
21:             edge e = G[v][i];
22:             if( d[e.to] > d[v] + e.cost ) {
23:                 d[e.to] = d[v] + e.cost;
24:                 pq.push( PII( d[e.to], e.to ) );
25:             }
26:         }
27:     }
28:     return ;
29: }

```

Floyd

```

01: const int INF = 0x7F7F7F7F;
02: const int MAXN = 1e3 + 10;
03:

```

```

04: int d[MAXN][MAXN];
05: int n;
06:
07: void floyd() {
08:     for( int k = 0; k < n; ++k ) {
09:         for( int i = 0; i < n; ++i )
10:             for( int j = 0; j < n; ++j )
11:                 d[i][j] = min( d[i][j], d[i][k] + d[k][j] );
12:     }
13:     return ;
14: }

```

Spfa

```

01: const int INF = 0x7F7F7F7F;
02: const int MAXN = 1e3 + 10;
03:
04: int cost[MAXN][MAXN];
05: int d[MAXN];
06: bool used[MAXN];
07: int n;
08:
09: void spfa( int s ) {
10:     queue<int> q;
11:     for( int i = 0; i < n; ++i ) { d[i] = INF; used[i] = false; }
12:     d[s] = 0; used[s] = true;
13:     q.push( s );
14:     while( !q.empty() ) {
15:         int u = q.front(); q.pop();
16:         used[u] = false;
17:         for( int i = 0; i < n; ++i ) {
18:             if( d[u] + cost[u][i] < d[i] ) {
19:                 d[i] = d[u] + cost[u][i];
20:                 if( !used[i] ) {

```

```

21:                 used[i] = true;
22:                 q.push( i );
23:             }
24:         }
25:     }
26: }
27: return ;
28: }

```

Spfa-slf-pq

```

01: const int INF = 0x7F7F7F7F;
02: const int MAXN = 1e3 + 10;
03:
04: int cost[MAXN][MAXN];
05: int d[MAXN];
06: bool used[MAXN];
07: int num[MAXN];
08: int n;
09: struct cmp {
10:     bool operator() ( int x, int y ) {
11:         return d[x] > d[y];
12:     }
13: };
14:
15: bool spfa_slf_pq( int s ) {
16:     priority_queue<int, vector<int>, cmp > pq;
17:     for( int i = 0; i < n; ++i ) { d[i] = INF; used[i] = false;
        num[i] = 0; }
18:     d[s] = 0; used[s] = true; ++num[s];
19:     pq.push( s );
20:     while( !pq.empty() ) {
21:         int u = pq.top(); pq.pop();
22:         used[s] = false;

```

```

23:     for( int i = 0; i < n; ++i ) {
24:         if( d[u] + cost[u][i] < d[i] ) {
25:             d[i] = d[u] + cost[u][i];
26:             if( !used[i] ) {
27:                 ++num[i];
28:                 if( num[i] > n ) return false;
29:                 pq.push( i );
30:                 used[i] = true;
31:             }
32:         }
33:     }
34: }
35: return true;
36: }

```

生成树

Prim-pq

```

01: #define INF 0x7FFFFFFF
02: #define MAX_V 1000
03: #define MAX_E 1e6
04:
05: struct edge { to, cost };
06: typedef pair<int, int> PII;
07: vector<edge> G[MAX_V];
08: int mincost[MAX_V];
09: int V;
10:
11: int prim() {
12:     int res = 0;
13:     priority_queue<PII, vector<PII>, greater<PII> > pq;
14:     for( int i = 0; i < V; ++i ) mincost[i] = INF;

```

```

15:     mincost[0] = 0; pq.push( P( 0, 0 ) );
16:     while( !pq.empty() ) {
17:         PII tmp = pq.top(); pq.pop();
18:         int v = tmp.second; res += v;
19:         if( mincost[v] < tmp.first ) continue;
20:         for( int i = 0; i < G[v].size(); ++i ) {
21:             edge e = G[v][i];
22:             if( mincost[e.to] > e.cost ) {
23:                 mincost[e.to] = e.cost;
24:                 pq.push( mincost[e.to], e.to );
25:             }
26:         }
27:     }
28:     return res;
29: }

```

Kruskal

关于次小生成树

但有一种更简单的方法：先求最小生成树 T ，枚举添加不在 T 中的边，则添加后一定会形成环。找到环上边值第二大的边(即环中属于 T 中的最大边)，把它删掉，计算当前生成树的权值，取所有枚举修改的生成树的最小值，即为次小生成树。

这种方法在实现时有更简单的方法：首先求最小生成树 T ，然后从每个结点 u 遍历最小生成树 T ，用一个二维数组 $\max[u][v]$ 记录结点 u 到结点 v 的路劲上边的最大值(即最大边的值)。然后枚举不在 T 中的边 (u,v) ，计算 $T - \max[u][v] + w(u,v)$ 的最小值，即为次小生成树的权值。显然，这种方法的时间复杂度为 $O(n^2 + e)$ 。

```

01: const int INF = 0x7F7F7F7F;
02: const int MAXN = 1e3 + 10;
03: const int MAXM = 1e6 + 10;
04:
05: struct edge { int u, v, cost; };
06: edge es[MAXN];
07: int V, E;
08: int father[MAXN], mrank[MAXN];

```

```

09:
10: int mfind( int x ) {
11:     if( x != father[x] )
12:         father[x] = mfind( father[x] );
13:     return father[x];
14: }
15:
16: void munion( int x, int y ) {
17:     if( mrank[x] > mrank[y] ) father[y] = x;
18:     else {
19:         if( mrank[x] == mrank[y] ) ++mrnk[y];
20:         father[x] = y;
21:     }
22:     return ;
23: }
24:
25: bool cmp( const edge& e1, const edge& e2 ) {
26:     return e1.cost > e2.cost;
27: }
28:
29: int kruskal() {
30:     int res = 0;
31:     sort( es, es + E, cmp );
32:     for( int i = 0; i < V; ++i ) { father[i] = i; mrank[i] = 0; }
33:     for( int i = 0; i < E; ++i ) {
34:         int x = mfind( es[i].u );
35:         int y = mfind( es[i].v );
36:         if( x != y ) { munion( x, y ); res += es[i].cost; }
37:     }
38:     return res;
39: }

```

曼哈顿最小生成树

```

01: typedef long long LL;
02: const int INF = 0x3F3F3F3F;
03: const int MAXN = 1000000 + 10;
04:
05: struct Point {
06:     int x, y, id;
07:     bool operator < ( const Point &p ) const {
08:         return x == p.x ? y < p.y : x < p.x;
09:     }
10: } poi[MAXN];
11: struct BIT {
12:     int minVal, pos;
13:     void init() { minVal = INF; pos = -1; }
14: } bit[MAXN << 2];
15: struct Edge {
16:     int u, v, cost;
17:     bool operator < ( const Edge &e ) const {
18:         return cost < e.cost;
19:     }
20: } es[MAXN << 2];
21: int fa[MAXN], a[MAXN], b[MAXN];
22: int n, k, cnt;
23:
24: int mfind( int x ) { return x == fa[x] ? x : fa[x] =
    mfind( fa[x] ); }
25:
26: void add( int u, int v, int w ) {
27:     es[cnt].u = u; es[cnt].v = v; es[cnt].cost = w; ++cnt;
28:     return ;
29: }
30:

```

```

31: int lowbit( int x ) { return x & -x; }
32:
33: void update( int i, int val, int pos ) {
34:     while( i ) {
35:         if( val < bit[i].minVal ) {
36:             bit[i].minVal = val;
37:             bit[i].pos = pos;
38:         }
39:         i -= lowbit( i );
40:     }
41:     return ;
42: }
43:
44: int ask( int i, int m ) {
45:     int minVal = INF, pos = -1;
46:     while( i <= m ) {
47:         if( bit[i].minVal < minVal ) {
48:             minVal = bit[i].minVal;
49:             pos = bit[i].pos;
50:         }
51:         i += lowbit( i );
52:     }
53:     return pos;
54: }
55:
56: int dist( const Point &a, const Point &b ) {
57:     return abs( a.x - b.x ) + abs( a.y - b.y );
58: }
59:
60: int MHT( int k ) {
61:     cnt = 0;
62:     for( int dir = 0; dir < 4; ++dir ) {
63:         if( dir == 1 || dir == 3 ) {

```

```

64:             for( int i = 0; i < n; ++i ) swap( poi[i].x, poi[i].y );
65:         }
66:         if( dir == 2 ) {
67:             for( int i = 0; i < n; ++i ) poi[i].x *= -1;
68:         }
69:         sort( poi, poi + n );
70:         for( int i = 0; i < n; ++i ) a[i] = b[i] = poi[i].y -
poi[i].x;
71:         sort( b, b + n );
72:         int ncnt = unique( b, b + n ) - b;
73:         for( int i = 1; i <= ncnt; ++i ) bit[i].init();
74:         for( int i = n - 1; i >= 0; --i ) {
75:             int pos = lower_bound( b, b + ncnt, a[i] ) - b + 1;
76:             int ans = ask( pos, ncnt );
77:             if( ans != -1 ) add( poi[i].id, poi[ans].id,
dist( poi[i], poi[ans] ) );
78:             update( pos, poi[i].x + poi[i].y, i );
79:         }
80:     }
81:     sort( es, es + cnt );
82:     for( int i = 0; i < n; ++i ) fa[i] = i;
83:     for( int i = 0; i < cnt; ++i ) {
84:         int u = es[i].u, v = es[i].v;
85:         int x = mfind( u ), y = mfind( v );
86:         if( x != y ) {
87:             --k;
88:             fa[x] = y;
89:             if( k == 0 ) return es[i].cost;
90:         }
91:     }
92:     return 0;
93: }
94:

```

```

95: int main() {
96:     while( ~scanf( "%d%d", &n, &k ) ) {
97:         for( int i = 0; i < n; ++i ) {
98:             scanf( "%d%d", &poi[i].x, &poi[i].y );
99:             poi[i].id = i;
100:        }
101:        printf( "%d\n", MHT( n - k ) );
102:    }
103:    return 0;
104:}

```

Prim 次小生成树

```

01: typedef pair<int, int> PII;
02: const int INF = 0x7F7F7F7F;
03: const int MAXN = 1000 + 10;
04:
05: PII poi[MAXN];
06: double dis[MAXN][MAXN], path[MAXN][MAXN], mincost[MAXN];
07: int ren[MAXN], pre[MAXN];
08: bool vis[MAXN], used[MAXN][MAXN];
09: int n;
10:
11: double dist( const int i, const int j ) {
12:     double dx = poi[i].first - poi[j].first;
13:     double dy = poi[i].second - poi[j].second;
14:     return sqrt( dx * dx + dy * dy );
15: }
16:
17: double prim() {
18:     double ret = 0;
19:     memset( used, false, sizeof( used ) );
20:     memset( vis, false, sizeof( vis ) );
21:     memset( path, 0, sizeof( path ) );

```

```

22:     for( int i = 0; i < n; ++i ) { mincost[i] = INF; pre[i] = 0; }
23:     mincost[0] = 0;
24:     while( true ) {
25:         int v = -1;
26:         for( int u = 0; u < n; ++u ) if( !vis[u] && ( v == -1 ||
mincost[u] < mincost[v] ) ) v = u;
27:         if( v == -1 ) break;
28:         used[pre[v]][v] = used[v][pre[v]] = true;
29:         ret += mincost[v];
30:         vis[v] = true;
31:         for( int u = 0; u < n; ++u ) {
32:             if( vis[u] && v != u ) path[u][v] = path[v][u] =
max( path[u][pre[v]], mincost[v] );
33:             if( !vis[u] && mincost[u] > dis[u][v] ) {
34:                 mincost[u] = dis[u][v];
35:                 pre[u] = v;
36:             }
37:         }
38:     }
39:     return ret;
40: }
41:
42: int main() {
43:     int t;
44:     scanf( "%d", &t );
45:     while( t-- ) {
46:         scanf( "%d", &n );
47:         for( int i = 0; i < n; ++i ) {
48:             scanf( "%d%d", &poi[i].first, &poi[i].second, ren + i );
49:             for( int j = 0; j < i; ++j ) dis[i][j] = dis[j][i] =
dist( i, j );
50:             dis[i][i] = 0;
51:         }

```

```

52:     double tmp = prim();
53:     double ans = -1;
54:     for( int i = 0; i < n; ++i ) {
55:         for( int j = 0; j < n; ++j ) if( i != j ) {
56:             if( used[i][j] ) ans = max( ans, ( ren[i] + ren[j] ) /
( tmp - dis[i][j] ) );
57:             else ans = max( ans, ( ren[i] + ren[j] ) / ( tmp -
path[i][j] ) );
58:         }
59:     }
60:     printf( "%.2f\n", ans );
61: }
62: return 0;
63: }

```

有向图最小生成树

```

01: const double INF = 0x3F3F3F3F;
02: const int MAXN = 100 + 10;
03: const int MAXE = 100000 + 10;
04:
05: struct edge{ int u, v; double cost; };
06: edge es[MAXE];
07: int ID[MAXN], vis[MAXN], pre[MAXN], x[MAXN], y[MAXN];
08: double IN[MAXN];
09: int n, m, cnt;
10:
11: void add( int u, int v, double c ) {
12:     es[cnt].u = u; es[cnt].v = v; es[cnt].cost = c; ++cnt;
13:     return ;
14: }
15:
16: double direct_MST( int root ) {
17:     double ans = 0;

```

```

18:     while( true ) {
19:         memset( ID, -1, sizeof( ID ) );
20:         memset( vis, -1, sizeof( vis ) );
21:         for( int i = 0; i < MAXN; ++i ) IN[i] = INF;
22:         for( int i = 0; i < m; ++i ) {
23:             int u = es[i].u;
24:             int v = es[i].v;
25:             if( es[i].cost < IN[v] && u != v ) {
26:                 IN[v] = es[i].cost;
27:                 pre[v] = u;
28:             }
29:         }
30:         for( int i = 0; i < n; ++i ) {
31:             if( i == root ) continue;
32:             if( IN[i] == INF ) return -1;
33:         }
34:         int tv = 0;
35:         IN[root] = 0; // pre[root] = root;
36:         for( int i = 0; i < n; ++i ) {
37:             ans += IN[i];
38:             int v = i;
39:             while( vis[v] != i && ID[v] == -1 && v != root ) {
40:                 vis[v] = i;
41:                 v = pre[v];
42:             }
43:             if( v != root && ID[v] == -1 ) {
44:                 for( int u = pre[v]; u != v; u = pre[u] ) {
45:                     ID[u] = tv;
46:                 }
47:                 ID[v] = tv++;
48:             }
49:         }
50:         if( !tv ) break;

```

```

51:     for( int i = 0; i < n; ++i ) {
52:         if( ID[i] == -1 ) ID[i] = tv++;
53:     }
54:     for( int i = 0; i < m; ++i ) {
55:         int v = es[i].v;
56:         es[i].u = ID[es[i].u];
57:         es[i].v = ID[es[i].v];
58:         if( es[i].u != es[i].v )
59:             es[i].cost -= IN[v];
60:     }
61:     n = tv;
62:     root = ID[root];
63: }
64: return ans;
65: }
66:
67: double dis( int i, int j ) {
68:     double dx = abs( x[i] - x[j] );
69:     double dy = abs( y[i] - y[j] );
70:     return sqrt( dx * dx + dy * dy );
71: }
72:
73: int main() {
74:     int a, b;
75:     while( ~scanf( "%d%d", &n, &m ) ) {
76:         cnt = 0;
77:         for( int i = 0; i < n; ++i ) scanf( "%d%d", x + i, y + i );
78:         for( int i = 0; i < m; ++i ) {
79:             scanf( "%d%d", &a, &b );
80:             --a; --b;
81:             add( a, b, dis( a, b ) );
82:         }
83:         double ans = direct_MST( 0 );

```

```

84:         if( ans < 0 ) puts( "poor snoopy" );
85:         else printf( "%.2f\n", ans );
86:     }
87:     return 0;
88: }

```

最小生成树计数

```

01: typedef long long LL;
02: const int MAXN = 1000 + 10;
03: const int MAXE = 100000 + 10;
04:
05: struct Edge { int u, v, w; };
06: Edge es[MAXE];
07: int fa[MAXN], ka[MAXN];
08: LL g[MAXN][MAXN], c[MAXN][MAXN];
09: bool vis[MAXN];
10: int n, m, mod;
11: vector<int> vec[MAXN];
12:
13: int mfind( int x, int *f ) {
14:     return x == f[x] ? x : f[x] = mfind( f[x], f );
15: }
16:
17: LL det( LL a[][MAXN], int n ) {
18:     for( int i = 0; i < n; ++i ) {
19:         for( int j = 0; j < n; ++j )
20:             a[i][j] %= mod;
21:     }
22:     int ret = 1;
23:     for( int i = 1; i < n; ++i ) {
24:         for( int j = i + 1; j < n; ++j ) {
25:             while( a[j][i] ) {
26:                 LL t = a[i][i] / a[j][i];

```



```

27:         for( int k = i; k < n; ++k )
28:             a[i][k] = ( a[i][k] - a[j][k] * t ) % mod;
29:         for( int k = i; k < n; ++k )
30:             swap( a[i][k], a[j][k] );
31:         ret = -ret;
32:     }
33: }
34: if( a[i][i] == 0 ) return 0;
35: ret = ( ret * a[i][i] ) % mod;
36: }
37: return ( ret + mod ) % mod;
38: }
39:
40: bool cmp( const Edge &a, const Edge &b ) {
41:     return a.w < b.w;
42: }
43:
44: void gao() {
45:     sort( es, es + m, cmp );
46:     for( int i = 1; i <= n; ++i ) { fa[i] = i; vis[i] = false; }
47:     LL pre = -1, ans = 1;
48:     for( int k = 0; k <= m; ++k ) {
49:         if( es[k].w != pre || k == m ) {
50:             for( int i = 1; i <= n; ++i ) {
51:                 if( vis[i] ) {
52:                     LL u = mfind( i, ka );
53:                     vec[u].push_back( i );
54:                     vis[i] = false;
55:                 }
56:             }
57:             for( int i = 1; i <= n; ++i ) {
58:                 if( vec[i].size() > 1 ) {
59:                     memset( c, 0, sizeof( c ) );

```

```

60:                     int len = vec[i].size();
61:                     for( int j = 0; j < len; ++j ) {
62:                         for( int k = j + 1; k < len; ++k ) {
63:                             int a1 = vec[i][j], b1 = vec[i][k];
64:                             c[j][k] = ( c[k][j] -= g[a1][b1] );
65:                             c[j][j] += g[a1][b1]; c[k][k] += g[a1][b1];
66:                         }
67:                     }
68:                     LL ret = det( c, len );
69:                     ans = ( ans * ret ) % mod;
70:                     for( int j = 0; j < len; ++j ) fa[vec[i][j]] = i;
71:                 }
72:             }
73:             for( int i = 1; i <= n; ++i ) {
74:                 ka[i] = fa[i] = mfind( i, fa );
75:                 vec[i].clear();
76:             }
77:             if( k == m ) break;
78:             pre = es[k].w;
79:         }
80:         int u = es[k].u, v = es[k].v;
81:         int a1 = mfind( u, fa ), b1 = mfind( v, fa );
82:         if( a1 == b1 ) continue;
83:         vis[a1] = vis[b1] = true;
84:         ka[mfind( a1, ka )] = mfind( b1, ka );
85:         ++g[a1][b1]; ++g[b1][a1];
86:     }
87:     bool flag = false;
88:     for( int i = 2; i <= n && !flag; ++i ) {
89:         if( ka[i] != ka[i - 1] ) flag = true;
90:     }
91:     if( !m ) flag = true;
92:     printf( "%I64d\n", flag ? 0 : ans % mod );

```

```

93:     return ;
94: }
95:
96: int main() {
97:     while( ~scanf( "%d%d%d", &n, &m, &mod ) && n + m + mod ) {
98:         memset( g, 0, sizeof( g ) );
99:         for( int i = 1; i <= n; ++i ) vec[i].clear();
100:        for( int i = 0; i < m; ++i ) scanf( "%d%d%d", &es[i].u,
        &es[i].v, &es[i].w );
101:        gao();
102:    }
103:    return 0;
104:}

```

二分图

匈牙利

```

01: #define INF 0x7FFFFFFF
02: #define MAX_V 1000
03: #define MAX_E 1000000 + 10
04:
05: vector<int> G[MAX_V];
06: bool used[MAX_V];
07: int match[MAX_V];
08: int VX;
09:
10: int findPath( int k ) {
11:     for( int i = 0; i < G[k].size(); ++i ) {
12:         int t = G[k][i];
13:         if( !used[t] ) {
14:             used[t] = true;
15:             if( match[i] == -1 || findPath( match[i] ) ) {

```

```

16:             match[t] = k;
17:             return 1;
18:         }
19:     }
20: }
21: return 0;
22: }
23:
24: int hungary() {
25:     int res;
26:     memset( match, -1, sizeof( match ) );
27:     for( int i = 0; i < VX; ++i ) {
28:         memset( used, false, sizeof( used ) );
29:         res += findPath( i );
30:     }
31:     return res;
32: }

```

HK

```

01: #define INF 0x7FFFFFFF
02: #define MAX_V 1000
03: #define MAX_E 1000000 + 10
04:
05: vector<int> G[MAX_V];
06: int VX;
07: int dx[MAX_V], dy[MAX_V];
08: int cx[MAX_V], cy[MAX_V];
09: int mindis;
10: bool mask[MAX_V];
11:
12: bool searchPath() {
13:     queue<int> q;
14:     memset( dx, -1, sizeof( dx ) );

```

```

15:     memset( dy, -1, sizeof( dy ) );
16:     mindis = INF;
17:     for( int i = 0; i < VX; ++i ) {
18:         if( cx[i] == -1 ) {
19:             dx[i] = 0;
20:             q.push( i );
21:         }
22:     }
23:     while( !q.empty() ) {
24:         int t = q.front(); q.pop();
25:         if( dx[t] > mindis ) break;
26:         for( int i = 0; i < G[t].size(); ++i ) {
27:             int v = G[t][i];
28:             if( dy[v] == -1 ) {
29:                 dy[v] = dx[t] + 1;
30:                 if( cy[v] == -1 ) mindis = dy[v];
31:                 else {
32:                     dx[cy[v]] = dy[v] + 1;
33:                     q.push( v );
34:                 }
35:             }
36:         }
37:     }
38:     return mindis != INF;
39: }
40:
41: int findPath( int u ) {
42:     for( int i = 0; i < G[u].size(); ++i ) {
43:         int v = G[u][i];
44:         if( !mask[v] && dy[v] == dx[u] + 1 ) {
45:             mask[v] = true;
46:             if( cy[v] != -1 && dy[v] == mindis ) continue;
47:             else {

```

```

48:                 cx[u] = v; cy[v] = u;
49:                 return 1;
50:             }
51:         }
52:     }
53:     return 0;
54: }
55:
56: int hk() {
57:     int res = 0;
58:     memset( cx, -1, sizeof( cx ) );
59:     memset( cy, -1, sizeof( cy ) );
60:     while( searchPath() ) {
61:         memset( mask, 0, sizeof mask );
62:         for( int i = 0; i < VX; ++i ) {
63:             if( cx[i] == -1 )
64:                 res += findPath( i );
65:         }
66:     }
67:     return res;
68: }

```

KM

```

01: const int INF = 0x7F7F7F7F;
02:
03: int nmap[305][305];
04: int lx[305],ly[305];
05: bool x[305],y[305];
06: int link[305];
07: int n; // n可能要改成n, m
08:
09: bool dfs( int u ) {
10:     int i;

```

```

11:     x[u] = true;
12:     for( i = 1; i <= n; ++i ) {
13:         if( lx[u] + ly[i] == nmap[u][i] && !y[i] ) {
14:             y[i] = true;
15:             if( link[i] == -1 || dfs( link[i] ) ) {
16:                 link[i] = u;
17:                 return true;
18:             }
19:         }
20:     }
21:     return false;
22: }
23:
24: void KM() {
25:     int i, j, k;
26:     memset( x, 0, sizeof( x ) );
27:     memset( y, 0, sizeof( y ) );
28:     memset( link, -1, sizeof( link ) );
29:     for( i = 1; i <= n; ++i ) lx[i] = INF;
30:     memset( ly, 0, sizeof( ly ) );
31:     for( k = 1; k <= n; ++k ) {
32:         while( true ) {
33:             memset( x, 0, sizeof( x ) );
34:             memset( y, 0, sizeof( y ) );
35:             if( dfs( k ) ) break;
36:             int d = INF;
37:             for( i = 1; i <= n; ++i ) {
38:                 if( x[i] )
39:                     for( j = 1; j <= n; ++j )
40:                         if( !y[j] && lx[i] + ly[j] - nmap[i][j] < d )
41:                             d = lx[i] + ly[j] - nmap[i][j];
42:             }
43:             for( i = 1; i <= n; ++i ) if( x[i] ) lx[i] = lx[i] - d;

```

```

44:             for( i = 1; i <= n; ++i ) if( y[i] ) ly[i] = ly[i] + d;
45:         }
46:     }
47:     return ;
48: }
49:
50: int main() {
51:     int i, j, k;
52:     while( ~scanf( "%d", &n ) ) {
53:         // nmap 可能要初始化, 求最小值时清成-INF 即 0x80808080
54:         for( i = 1; i <= n; ++i )
55:             for( j = 1; j <= n; ++j )
56:                 scanf( "%d", &nmap[i][j] );
57:         KM();
58:         int ans = 0;
59:         for( i = 1; i <= n; ++i ) ans = ans + nmap[link[i]][i];
60:         printf( "%d\n", ans );
61:     }
62:     return 0;
63: }

```

网络流

Dinic

add 双向边连续建图, cnt 从 0 开始, ^操作确定相邻两边

```

01: typedef int MyType;
02: const int INF = 0x3F3F3F3F;
03: const int MAXN = 1000 + 10;
04: const int MAXE = 100000 + 10;
05:
06: struct Edge { int to, next; MyType cap; };

```

```

07: Edge es[MAXE];
08: int head[MAXN], cur[MAXN], level[MAXN], que[MAXN];
09: int n, F, D, cnt, src, des;
10:
11: void add( int u, int v, MyType c ) {
12:     es[cnt].to = v; es[cnt].cap = c; es[cnt].next = head[u]; head[u]
    = cnt++;
13:     es[cnt].to = u; es[cnt].cap = 0; es[cnt].next = head[v]; head[v]
    = cnt++;
14:     return ;
15: }
16:
17: bool bfs() {
18:     int mf, me;
19:     memset( level, 0, sizeof( level ) );
20:     mf = me = 0;
21:     que[me++] = src;
22:     level[src] = 1;
23:     while( mf < me ) {
24:         int u = que[mf++];
25:         for( int i = head[u]; ~i; i = es[i].next ) {
26:             int v = es[i].to;
27:             if( level[v] == 0 && es[i].cap > 0 ) {
28:                 level[v] = level[u] + 1;
29:                 que[me++] = v;
30:             }
31:         }
32:     }
33:     return ( level[des] != 0 );
34: }
35:
36: MyType dfs( int u, MyType f ) {
37:     if( u == des || f == 0 ) return f;

```

```

38:     MyType flow = 0;
39:     for( int &i = cur[u]; ~i; i = es[i].next ) {
40:         Edge &e = es[i];
41:         if( e.cap > 0 && level[e.to] == level[u] + 1 ) {
42:             MyType d = dfs( e.to, min( f, e.cap ) );
43:             if( d > 0 ) {
44:                 e.cap -= d;
45:                 es[i ^ 1].cap += d;
46:                 flow += d;
47:                 f -= d;
48:                 if( f == 0 ) break;
49:             } else level[e.to] = -1;
50:         }
51:     }
52:     return flow;
53: }
54:
55: MyType dinic() {
56:     MyType ret = 0;
57:     while( bfs() ) {
58:         for( int i = src; i <= des; ++i ) {
59:             cur[i] = head[i];
60:         }
61:         ret += dfs( src, INF );
62:     }
63:     return ret;
64: }
65:
66: int main() {
67:     return 0;
68: }

```

Ford-Flukerson

```
01: #define MAXV 10010
02: #define INF 0x7FFFFFFF
03:
04: struct edge { int to, cap, rev; };
05: vector<edge> G[MAXV];
06: bool vis[MAXV];
07:
08: void add_edge( int f, int t, int c ) {
09:     G[f].push_back( ( edge ){ t, c, G[t].size() } );
10:     G[t].push_back( ( edge ){ f, 0, G[f].size() - 1 } );
11:     return ;
12: }
13:
14: int dfs( int v, int t, int f ) {
15:     if( v == t ) return f;
16:     vis[v] = true;
17:     for( int i = 0; i < G[v].size(); ++i ) {
18:         edge &e = G[v][i];
19:         if( !vis[e.to] && e.cap > 0 ) {
20:             int d = dfs( e.to, t, min( f, e.cap ) );
21:             if( d > 0 ) {
22:                 e.cap -= d;
23:                 G[e.to][e.rev] += d;
24:                 return d;
25:             }
26:         }
27:     }
28:     return 0;
29: }
30:
31: int max_flow( int s, int t ) {
```

```
32:     int flow = 0;
33:     while( true ) {
34:         memset( vis, 0, sizeof( vis ) );
35:         int f = dfs( s, t, INF );
36:         if( f == 0 ) return flow;
37:         flow += f;
38:     }
39:     return 0;
40: }
41:
42: int main() {
43:     int n, m;
44:     int a, b, c;
45:     scanf( "%d%d", &n, &m );
46:     for( int i = 0; i < n; ++i ) {
47:         scanf( "%d%d%d", a, b, c );
48:         add( a, b, c );
49:     }
50:     cout << max_flow( 0, n - 1 ) << endl;
51:     return 0;
52: }
```

费用流

```
01: typedef int MyType;
02: const MyType INF = 0x7F7F7F7F;
03: const int MAXN = 1000 + 10;
04: const int MAXE = 100000 + 10;
05:
06: struct Edge { int to, next; MyType cap, cost; };
07: Edge es[MAXE];
08: int head[MAXN], que[MAXE], dis[MAXN], pre[MAXN];
09: bool vis[MAXN];
```

```

10: int n, m, cnt, src, des;
11:
12: void add( int u, int v, MyType f, MyType c ) {
13:     es[cnt].to = v; es[cnt].cap = f; es[cnt].cost = c;
14:     es[cnt].next = head[u]; head[u] = cnt++;
15:     es[cnt].to = u; es[cnt].cap = 0; es[cnt].cost = -c;
16:     es[cnt].next = head[v]; head[v] = cnt++;
17:     return ;
18: }
19:
20: bool spfa() {
21:     int mf, me;
22:     memset( vis, false, sizeof( vis ) );
23:     memset( dis, 0x7F, sizeof( dis ) );
24:     memset( pre, -1, sizeof( pre ) );
25:     mf = me = 0;
26:     que[me++] = src; dis[src] = 0; vis[src] = true;
27:     while( mf < me ) {
28:         int u = que[mf++];
29:         for( int i = head[u]; ~i; i = es[i].next ) {
30:             int v = es[i].to;
31:             if( es[i].cap > 0 && dis[v] > dis[u] + es[i].cost ) {
32:                 dis[v] = dis[u] + es[i].cost;
33:                 pre[v] = i;
34:                 if( !vis[v] ) {
35:                     vis[v] = true;
36:                     que[me++] = v;
37:                 }
38:             }
39:         }
40:         vis[u] = false;
41:     }
42:     return dis[des] != INF;

```

```

43: }
44:
45: MyType cflow() {
46:     MyType flow = INF;
47:     int u = des;
48:     while( ~pre[u] ) {
49:         u = pre[u];
50:         flow = min( flow, es[u].cap );
51:         u = es[u ^ 1].to;
52:     }
53:     u = des;
54:     while( ~pre[u] ) {
55:         u = pre[u];
56:         es[u].cap -= flow;
57:         es[u ^ 1].cap += flow;
58:         u = es[u ^ 1].to;
59:     }
60:     return flow;
61: }
62:
63: MyType MCMF() {
64:     MyType mincost, maxflow;
65:     mincost = maxflow = 0;
66:     while( spfa() ) {
67:         MyType flow = cflow();
68:         maxflow += flow;
69:         mincost += flow * dis[des];
70:     }
71:     return mincost;
72: }
73:
74: int main() {
75:     int a, b, c;

```

```

76: while( ~scanf( "%d%d", &n, &m ) ) {
77:     memset( head, -1, sizeof( head ) );
78:     cnt = 0;
79:     src = 0; des = n + 1;
80:     add( src, 1, 2, 0 );
81:     for( int i = 0; i < m; ++i ) {
82:         scanf( "%d%d%d", &a, &b, &c );
83:         add( a, b, 1, c );
84:         add( b, a, 1, c );
85:     }
86:     add( n, des, 2, 0 );
87:     printf( "%d\n", MCMF() );
88: }
89: return 0;
90: }

```

欧拉路径

Fleury

```

01: int stk[1005];
02: int top;
03: int N, M, ss, tt;
04: int mp[1005][1005];
05:
06: void dfs(int x) {
07:     stk[top++] = x;
08:     for (int i = 1; i <= N; ++i) {
09:         if (mp[x][i]) {
10:             mp[x][i] = mp[i][x] = 0; // 删除此边
11:             dfs(i);
12:             break;
13:         }

```

```

14:     }
15: }
16:
17: void fleury(int ss) {
18:     int brige;
19:     top = 0;
20:     stk[top++] = ss; // 将起点放入 Euler 路径中
21:     while (top > 0) {
22:         brige = 1;
23:         for (int i = 1; i <= N; ++i) { // 试图搜索一条边不是割边（桥）
24:             if (mp[stk[top-1]][i]) {
25:                 brige = 0;
26:                 break;
27:             }
28:         }
29:         if (brige) { // 如果没有点可以扩展，输出并出栈
30:             printf("%d ", stk[--top]);
31:         } else { // 否则继续搜索欧拉路径
32:             dfs(stk[--top]);
33:         }
34:     }
35: }
36: int main() {
37:     int x, y, deg, num;
38:     while (scanf("%d %d", &N, &M) != EOF) {
39:         memset(mp, 0, sizeof (mp));
40:         for (int i = 0; i < M; ++i) {
41:             scanf("%d %d", &x, &y);
42:             mp[x][y] = mp[y][x] = 1;
43:         }
44:         for (int i = 1; i <= N; ++i) {
45:             deg = num = 0;
46:             for (int j = 1; j <= N; ++j) {

```



```

47:         deg += mp[i][j];
48:     }
49:     if (deg % 2 == 1) {
50:         ss = i, ++num;
51:         printf("%d\n", i);
52:     }
53: }
54: if (num == 0 || num == 2) {
55:     fleury(ss);
56: } else {
57:     puts("No Euler path");
58: }
59: }
60: return 0;
61: }

```

全局最小割

```

01: const int INF = 0x7F7F7F7F;
02: const int MAXN = 1000 + 10;
03:
04: int mat[MAXN][MAXN], v[MAXN], dis[MAXN];
05: bool vis[MAXN];
06: int n, m;
07:
08: int SW() {
09:     int ret = INF;
10:     for( int i = 0; i <= n; ++i ) v[i] = i;
11:     while( n > 1 ) {
12:         int pre = 0;
13:         memset( vis, false, sizeof( vis ) );
14:         memset( dis, 0, sizeof( dis ) );
15:         for( int i = 1; i < n; ++i ) {

```

```

16:             int k = -1;
17:             for( int j = 1; j < n; ++j ) {
18:                 if( !vis[v[j]] ) {
19:                     dis[v[j]] += mat[v[pre]][v[j]];
20:                     if( k == -1 || dis[v[k]] < dis[v[j]] ) k = j;
21:                 }
22:             }
23:             vis[v[k]] = true;
24:             if( i == n - 1 ) {
25:                 ret = min( ret, dis[v[k]] );
26:                 for( int j = 0; j < n; ++j ) {
27:                     mat[v[pre]][v[j]] = ( mat[v[j]][v[pre]] +=
mat[v[j]][v[k]] );
28:                 }
29:                 v[k] = v[--n];
30:             }
31:             pre = k;
32:         }
33:     }
34:     return ret;
35: }
36:
37: int main() {
38:     int a, b, c;
39:     while( ~scanf( "%d%d", &n, &m ) ) {
40:         memset( mat, 0, sizeof( mat ) );
41:         while( m-- ) {
42:             scanf( "%d%d%d", &a, &b, &c );
43:             mat[a][b] = ( mat[b][a] += c );
44:         }
45:         printf( "%d\n", SW() );
46:     }
47:     return 0;

```

```
48: }
```

差分约束

建图： 约束图

在一个差分约束系统 $Ax \leq b$ 中， $m \times n$ 的线性规划矩阵 A 可被看做是 n 顶点， m 条边的图的关联矩阵。对于 $i=1,2,\dots,n$ ，图中的每一个顶点 v_i 对应着 n 个未知量的一个 x_i 。图中的每个有向边对应着关于两个未知量的 m 个不等式中的一个。

给定一个差分约束系统 $Ax \leq b$ ，相应的约束图是一个带权有向图 $G=(V,E)$ ，其中 $V=\{v_0,v_1,\dots,v_n\}$ ，而且 $E=\{(v_i,v_j) : x_j-x_i \leq b_k \text{ 是一个约束}\} \cup \{(v_0,v_1), (v_0,v_2), \dots, (v_0,v_n)\}$ 。引入附加顶点 v_0 是为了保证其他每个顶点均从 v_0 可达。因此，顶点集合 V 由对应于每个未知量 x_i 的顶点 v_i 和附加的顶点 v_0 组成。边的集合 E 由对应于每个差分约束条件的边与对应于每个未知量 x_i 的边 (v_0,v_i) 构成。如果 $x_j-x_i \leq b_k$ 是一个差分约束，则边 (v_i,v_j) 的权 $w(v_i,v_j)=b_k$ （注意 i 和 j 不能颠倒），从 v_0 出发的每条边的权值均为 0 。

定理：给定一差分约束系统 $Ax \leq b$ ，设 $G=(V,E)$ 为其相应的约束图。如果 G 不包含负权回路，那么 $x=(d(v_0,v_1), d(v_0,v_2), \dots, d(v_0,v_n))$ 是此系统的一可行解，其中 $d(v_0,v_i)$ 是约束图中 v_0 到 v_i 的最短路径 ($i=1,2,\dots,n$)。如果 G 包含负权回路，那么此系统不存在可行解。

差分约束问题的求解

由上述定理可知，可以采用 Bellman-Ford 算法对差分约束问题求解。因为在约束图中，从源点 v_0 到其他所有顶点间均存在边，因此约束图中任何负权回路均从 v_0 可达。如果 Bellman-Ford 算法返回 TRUE，则最短路径权给出了此系统的一个可行解；如果返回 FALSE，则差分约束系统无可行解。

关于 n 个未知量 m 个约束条件的一个差分约束系统产生出一个具有 $n+1$ 个顶点和 $n+m$ 条边的约束图。因此采用 Bellman-Ford 算法，可以再 $O((n+1)(n+m))=O(n^2+nm)$ 时间内将系统解决。此外，可以用 SPFA 算法进行优化，复杂度为 $O(km)$ ，其中 k 为常数。

```
01: const int INF = 0x7F7F7F7F;
02: const int MAXN = 100000 + 10;
03: const int MAXM = 1000000 + 10;
04:
05: struct Edge { int to, cost, next; };
```

```
06: Edge es[MAXM];
07: int head[MAXN], dis[MAXN], sta[MAXM];
08: bool vis[MAXN];
09: int n, m, cnt;
10:
11: void add( int u, int v, int w ) {
12:     es[cnt].to = v; es[cnt].cost = w; es[cnt].next = head[u]; head[u]
    = cnt++;
13:     return ;
14: }
15:
16: void spfa() {
17:     int top = 0;
18:     memset( dis, 0x7F, sizeof( dis ) );
19:     memset( vis, false, sizeof( vis ) );
20:     dis[1] = 0; vis[1] = true; sta[top++] = 1;
21:     while( top ) {
22:         int u = sta[--top];
23:         for( int i = head[u]; ~i; i = es[i].next ) {
24:             int v = es[i].to;
25:             if( dis[v] > dis[u] + es[i].cost ) {
26:                 dis[v] = dis[u] + es[i].cost;
27:                 if( !vis[v] ) {
28:                     vis[v] = true;
29:                     sta[top++] = v;
30:                 }
31:             }
32:         }
33:         vis[u] = false;
34:     }
35:     return ;
36: }
37:
```

```
38: int main() {
39:     int a, b, c;
40:     memset( head, -1, sizeof( head ) );
41:     cnt = 0;
42:     scanf( "%d%d", &n, &m );
43:     for( int i = 0; i < m; ++i ) {
44:         scanf( "%d%d%d", &a, &b, &c );
45:         add( a, b, c );
46:     }
47:     spfa();
48:     printf( "%d\n", dis[n] );
49:     return 0;
50: }
```

数据结构

LCA

HDU 2586 验过

1. 注意变量名不要和其他数据结构冲突
2. ST 在线算法，需要大量预处理。RMQ 会占用大量内存
3. head 数组要清-1
4. query 时需要确保前小后大
5. lca() 返回的是点编号

```
01: const int MAXN = 1e5 + 10;
02: const int POW = 32;
03: struct Edge { int to, next; };
04: Edge es[MAXN << 1];
05: int seq[MAXN << 1], dep[MAXN << 1], fir[MAXN], fa[MAXN];
06: int lg2[MAXN << 1], dp[MAXN << 1][POW];
07: int head[MAXN], a[MAXN];
08: int n, m, cnt2, tot;
09:
10: void add( int u, int v ) { es[cnt2].to = v; es[cnt2].next = head[u];
    head[u] = cnt2++; }
11: void dfs( int u, int fu, int d ) {
12:     seq[++tot] = u; dep[tot] = d; fir[u] = tot; fa[u] = fu;
13:     update( 1, vec.size(), rt[u], rt[fa[u]], getid( a[u] ) );
14:     for( int i = head[u]; ~i; i = es[i].next ) {
15:         int v = es[i].to;
16:         if( v == fu ) continue;
17:         dfs( v, u, d + 1 );
18:         seq[++tot] = u; dep[tot] = d;
19:     }
20: }
21: void init_lca() {
```

```
22:     int tn = 2 * n - 1;
23:     lg2[0] = -1; for( int i = 1; i < ( MAXN << 1 ); ++i ) lg2[i] =
        ( ( i & ( i - 1 ) ) == 0 ) ? lg2[i - 1] + 1 : lg2[i - 1];
24:     for( int i = 1; i <= tn; ++i ) dp[i][0] = i;
25:     for( int j = 1; j < 20; ++j ) {
26:         for( int i = 1; i + ( 1 << j ) - 1 <= tn; ++i ) {
27:             int a = dp[i][j - 1], b = dp[i + ( 1 << ( j - 1 ) )][j -
                1];
28:             dp[i][j] = dep[a] < dep[b] ? a : b;
29:         }
30:     }
31: }
32: int lca( int x, int y ) {
33:     int k = lg2[y - x + 1];
34:     int a = dp[x][k], b = dp[y - ( 1 << k ) + 1][k];
35:     return seq[dep[a] < dep[b] ? a : b];
36: }
37:
38: int main() {
39:     memset( head, -1, sizeof head ); tot = cnt2 = 0;
40:     scanf( "%d%d", &n, &m );
41:     for( int i = 1, u, v; i < n; ++i ) { scanf( "%d%d", &u, &v );
        add( u, v ); add( v, u ); }
42:     dfs( 1, 0, 1 );
43:     init_lca();
44:     for( int i = 0, u, v; i < m; ++i ) {
45:         scanf( "%d%d", &u, &v );
46:         if( fir[u] > fir[v] ) swap( u, v );
47:         //
48:         printf( "%d\n", lca( fir[u], fir[v] ) );
49:     }
50:     return 0;
51: }
```

RMQ

HDU 3183 已验

1. init()

i 从 1 开始

dp[i][0] 装最原始数据

min/max 比较具体情况具体分析, 有时需要自己写, 例如 lca 可能需要利用 dep 做比较

!!! 特别需要注意小于号还是小于等于号!!!

2. query()

x < y

min/max 同上

```
01: #include <bits/stdc++.h>
```

```
02: using namespace std;
```

```
03: const int MAXN = 1e5 + 10;
```

```
04: const int POW = 32;
```

```
05:
```

```
06: int lg2[MAXN], dp[MAXN][POW], a[MAXN];
```

```
07: int n;
```

```
08:
```

```
09: void init( int tn ) {
```

```
10:     lg2[0] = -1; for( int i = 1; i < MAXN; ++i ) lg2[i] = ( ( i & ( i
```

```
11:         - 1 ) ) == 0 ) ? lg2[i - 1] + 1 : lg2[i - 1];
```

```
12:     for( int i = 1; i <= tn; ++i ) dp[i][0] = i;
```

```
13:     for( int j = 1; j <= 20; ++j ) {
```

```
14:         for( int i = 1; i + ( 1 << j ) - 1 <= tn; ++i ) {
```

```
15:             dp[i][j] = min( dp[i][j - 1], dp[i + ( 1 << ( j - 1 ) )][j
```

```
16:                 - 1 ] );
```

```
17:         }
```

```
18:
```

```
19: int query( int x, int y ) {
```

```
20:     int k = lg2[y - x + 1];
```

```
21:     return min( dp[x][k], dp[y - ( 1 << k ) + 1][k] );
```

```
22: }
```

差分前缀和

定义数组 A, 存在 n 个元素 A[1]~A[n]

定义差分数组 D, 其中 D[1] = A[1], D[i] = A[i] - A[i-1], D[n+1] = 0

观察可发现

性质(1), 即 A[i] = $\sum (D[1] \cdots D[i])$, 证:

首先 A[1] = D[1], 之后 A[2] = D[1] + D[2] = A[1] + A[2] - A[1] = A[2], 利用归纳法可证

定义前缀和数组 S, 其中 S[i] = $\sum (A[1] \cdots A[i])$

观察可发现

性质: $\sum (A[i] \cdots A[j]) = S[j] - S[i-1]$, 证:

A[j] = A[1] + ... + A[i-1] + A[i] + ... + A[j]

A[i-1] = A[1] + ... + A[i-1]

定义某数组为 X

X 数组的前缀和数组为 S

S 的前缀和数组为 SS

SS[i] = $\sum (S[0] \cdots S[i]) = i * X[1] + (i-1) * X[2] + \cdots + 1 * X[i]$ (从第 1 项开始每一项都有一个 X[1], 从第 2 项开始每一项都有一个 X[2])

= (i+1) * (X[1] + X[2] + ... + X[i]) - (X[1] + 2 * X[2] + ... + i * X[i])

令 T 数组为 T[i] = i * X[i], 则 SS[i] = (i+1) * S[i] - T[i]

存在某数组 X', 其差分数组为 X, 那么上面的性质仍然成立。

由于 X' 数组的差分数组的前缀和就是它本身, 所以 SS 和 S 数组对于 X' 数组都降了一级, SS 变为了前缀和数组, 而 S 变为了 X' 数组本身。

我们需要求出 A 数组的区间和，那么简单的方法就是利用 A 的前缀和数组 S， $S[j] - S[i-1]$ 即为 $i \cdots j$ 区间的和。

假如我们因为某种原因，修改操作时，我们不操作数组 A 本身而操作数组 A 的差分数组，为了获得等价的效果，

计算区间和就表示为了关于差分数组的 $SS[j] - SS[i-1]$ 。=> 以上，查询操作得到了解答。

对于 A 数组和它的差分数组 DS，我们令 $DS[i] += d$ ，那么由于定义，

$A[i-1] = DS[i-1] + A[i-2]$ 而 $A[i] = DS[i] + A[i-1]$

由此得出 $A[1 \cdots i-1]$ 都没有发生变化，而 $A[i \cdots \text{end}]$ 由于递推都获得了 d 的增量。

为了构造一个获得增量的区间，例如 $i \cdots j$ 区间，我们按图索骥令 $DS[j+1] -= d$

由此 $A[j+1 \cdots \text{end}]$ 区间最终增加了 d - d 的增量，等于没修改。

于是我们得到修改区间 $i \cdots j$ 的方法， $DS[i] += d$ 同时 $DS[j+1] -= d$ 。=> 以上，修改操作得到了解答。

捋顺数组之间的关系，我们回到 $SS[i] = (i+1) * S[i] - T[i]$ 中，

区间查询即为 $SS[j] - SS[i-1]$

区间修改即为 $X[i] += d$ 同时 $X[j+1] -= d$

其中 X 是原数组的差分数组。

观察 $SS[i]$ 的表达式， $SS[i] = (i+1) * (X[1] + X[2] + \cdots + X[i]) - (X[1] + 2 * X[2] + \cdots + i * X[i])$

该表达式可以表示为两个前缀和。

故求 $SS[j] - SS[i-1]$ 即转化为 查找 4 个前缀和 进行计算的过程（利用树状数组或者 zkw 线段树）

而全体过程完全不改变原数组 X，一切操作皆借助其差分数组 X 完成。

线段树

POJ 3468 验过

1. 有时候需要将节点改成结构体
2. dowork 有时候需要提前声明 update 函数并调用 update

```
01: typedef long long LL;
```

```
02: const int MAXN = 100000 + 10;
```

```
03: #define lson rt << 1, left, mid
```

```
04: #define rson rt << 1 | 1, mid + 1, right
```

```
05: LL num[MAXN << 2], lazy[MAXN << 2];
```

```
06: int n, q;
```

```
07:
```

```
08: void build( int rt, int left, int right ) {
```

```
09:     lazy[rt] = 0;
```

```
10:     if( left == right ) { scanf( "%I64d", &num[rt] ); return ; }
```

```
11:     int mid = ( left + right ) >> 1;
```

```
12:     build( lson );
```

```
13:     build( rson );
```

```
14:     num[rt] = num[rt << 1] + num[rt << 1 | 1];
```

```
15:     return ;
```

```
16: }
```

```
17: void dowork( int rt, int len ) {
```

```
18:     if( lazy[rt] ) {
```

```
19:         int son = rt << 1; num[son] += lazy[rt] * ( len - ( len >> 1 ) ); lazy[son] += lazy[rt];
```

```
20:         son = rt << 1 | 1; num[son] += lazy[rt] * ( len >> 1 ); lazy[son] += lazy[rt];
```

```
21:         lazy[rt] = 0;
```

```
22:     }
```

```
23:     return ;
```

```
24: }
```

```
25: void update( int rt, int left, int right, int l, int r, LL val ) {
```

```
26:     if( left == l && right == r ) {
```

```
27:         num[rt] += val * ( r - l + 1 );
```

```
28:         if( l != r ) lazy[rt] += val;
```

```
29:         return ;
```

```
30:     }
```

```
31:     dowork( rt, right - left + 1 );
```

```
32:     int mid = ( left + right ) >> 1;
```

```
33:     if( r <= mid ) update( lson, l, r, val );
```

```

34:     else if( l > mid ) update( rson, l, r, val );
35:     else {
36:         update( lson, l, mid, val );
37:         update( rson, mid + 1, r, val );
38:     }
39:     num[rt] = num[rt << 1] + num[rt << 1 | 1];
40:     return ;
41: }
42: LL query( int rt, int left, int right, int l, int r ) {
43:     if( left == l && right == r ) return num[rt];
44:     dowork( rt, right - left + 1 );
45:     int mid = ( left + right ) >> 1;
46:     if( r <= mid ) return query( lson, l, r );
47:     else if( l > mid ) return query( rson, l, r );
48:     return query( lson, l, mid ) + query( rson, mid + 1, r );
49: }
50: int main() {
51:     int a, b, c;
52:     char st[2];
53:     memset( num, 0, sizeof( num ) );
54:     scanf( "%d%d", &n, &q );
55:     build( 1, 1, n );
56:     while( q-- ) {
57:         scanf( "%s", st );
58:         if( st[0] == 'Q' ) {
59:             scanf( "%d%d", &a, &b );
60:             printf( "%I64d\n", query( 1, 1, n, a, b ) );
61:         } else {
62:             scanf( "%d%d%d", &a, &b, &c );
63:             update( 1, 1, n, a, b, c );
64:         }
65:     }
66:     return 0;

```

```
67: }
```

树状数组

1. 下标从 1 开始
2. 注意 $x=0$ 时的 lowbit, 有可能超时

```

01: const int MAXN = 1e5 + 10;
02: int c[MAXN], a[MAXN];
03: int n;
04:
05: int lowbit( int x ) { return x & -x; }
06: void add( int i, int x ) {
07:     while( i <= n ) {
08:         add[i] += x;
09:         i += lowbit( i );
10:     }
11: }
12: int sum( int i ) {
13:     int ret = 0;
14:     while( i ) {
15:         sum += c[i];
16:         i -= lowbit( i );
17:     }
18:     return ret;
19: }

```

可持久化数据结构

静态主席树

POJ 2104 验过

1. T 大小需要计算, 一般 64 倍足够
2. 每颗树的含义需要明确, 此题中每棵树存的是离散化后数据域,

记录每个数据出现的次数，并非每个位置上是什么数。

```
01: const int MAXN = 1e5 + 10;
02:
03: struct Node { int l, r, num; };
04: Node T[MAXN << 6];
05: vector<int> vec;
06: int root[MAXN], a[MAXN];
07: int n, m, cnt;
08:
09: int getid( int x ) { return upper_bound( vec.begin(), vec.end(), x )
    - vec.begin(); }
10:
11: void update( int left, int right, int &x, int y, int pos ) {
12:     T[++cnt] = T[y]; ++T[cnt].num; x = cnt;
13:     if( left == right ) return ;
14:     int mid = ( left + right ) >> 1;
15:     if( mid >= pos ) update( left, mid, T[x].l, T[y].l, pos );
16:     else update( mid + 1, right, T[x].r, T[y].r, pos );
17:     return ;
18: }
19:
20: int query( int left, int right, int x, int y, int k ) {
21:     if( left == right ) return left;
22:     int mid = ( left + right ) >> 1, ret = 0, sum = 0;
23:     sum = T[T[y].l].num - T[T[x].l].num;
24:     if( sum >= k ) ret = query( left, mid, T[x].l, T[y].l, k );
25:     else ret = query( mid + 1, right, T[x].r, T[y].r, k - sum );
26:     return ret;
27: }
28:
29: int main() {
30:     cnt = 0;
```

```
31:     scanf( "%d%d", &n, &m );
32:     for( int i = 1, t; i <= n; ++i ) scanf( "%d", a + i ),
        vec.push_back( a[i] );
33:     sort( vec.begin(), vec.end() ); vec.erase( unique( vec.begin(),
        vec.end() ), vec.end() );
34:     for( int i = 1; i <= n; ++i ) update( 1, vec.size(), root[i],
        root[i - 1], getid( a[i] ) );
35:     for( int i = 0, x, y, z; i < m; ++i ) {
36:         scanf( "%d%d%d", &x, &y, &z );
37:         printf( "%d\n", vec[query( 1, vec.size(), root[x - 1],
            root[y], z ) - 1] );
38:     }
39:     return 0;
40: }
```

动态主席树

没法验 ZOJ 不开了

- 对于修改操作，只是修改 M 次，每次改变两个值（减去原先的，加上现在的）
也就是说如果把所有初值都插入到树状数组里是不值得的，
所以我们分两部分来做，所有初值按照静态来建，内存 $O(n \log n)$ ，
而修改部分保存在树状数组中，每次修改 $\log n$ 棵树，每次插入增加 $\log n$ 个节点
 $O(M * \log n * \log n + n \log n)$
1. 一定要全部离线后再做其他操作!! 包括修改的数据!!
 2. 空间需要计算下
 3. 树状数组前 n 项为修改位置对应数据；后 n 项为建树初始状态数据

```
01: const int MAXN = 1e5 + 10;
02: const int MAXM = 1e4 + 10;
03:
04: struct Node { int ls, rs, sum; };
05: Node tr[MAXN << 4];
06: struct Opr { int flag, l, r, k; };
07: Opr op[MAXM];
08: vector<int> vec, q1, q2;
```



```

09: int a[MAXN], root[MAXN << 1];
10: int n, m, tn, tot;
11:
12: int getid( int x ) { return upper_bound( vec.begin(), vec.end(), x )
    - vec.begin(); }
13: inline int lowbit( int x ) { return x & -x; }
14:
15: void build( int l, int r, int &x, int pos ) {
16:     tr[++tot] = tr[x]; x = tot; ++tr[x].sum;
17:     if( l == r ) return ;
18:     int m = ( l + r ) >> 1;
19:     if( pos <= m ) build( l, m, tr[x].ls, pos );
20:     else build( m + 1, r, tr[x].rs, pos );
21: }
22:
23: void insrt( int l, int r, int &x, int pos, int val ) {
24:     if( x == 0 ) { tr[++tot] = tr[x]; x = tot; }
25:     tr[x].sum += val;
26:     if( l == r ) return ;
27:     int m = ( l + r ) >> 1;
28:     if( pos <= m ) insrt( l, m, tr[x].ls, pos, val );
29:     else insrt( m + 1, r, tr[x].rs, pos, val );
30: }
31:
32: void bitinsrt( int pos, int x, int val ) {
33:     int t = getid( x );
34:     for( int i = pos; i <= n; i += lowbit( i ) ) insrt( 1, tn,
        root[i], t, val );
35: }
36:
37: int qry( int l, int r, vector<int> &q1, vector<int> &q2, int k ) {
38:     if( l == r ) return l;
39:     int cnt = 0, m = ( l + r ) >> 1;

```

```

40:     for( int i = 0; i < q1.size(); ++i ) cnt -= tr[tr[q1[i]].ls].sum;
41:     for( int i = 0; i < q2.size(); ++i ) cnt += tr[tr[q2[i]].ls].sum;
42:     for( int i = 0; i < q1.size(); ++i ) q1[i] = ( cnt >= k ?
        tr[q1[i]].ls : tr[q1[i]].rs );
43:     for( int i = 0; i < q2.size(); ++i ) q2[i] = ( cnt >= k ?
        tr[q2[i]].ls : tr[q2[i]].rs );
44:     if( cnt >= k ) return qry( l, m, q1, q2, k );
45:     else return qry( m + 1, r, q1, q2, k - cnt );
46: }
47:
48: int bitqry( int l, int r, int k ) {
49:     q1.clear(); q2.clear();
50:     q1.push_back( root[l == 1 ? 0 : l - 1 + n] );
51:     q2.push_back( root[r + n] );
52:     for( int i = l - 1; i > 0; i -= lowbit( i ) )
        q1.push_back( root[i] );
53:     for( int i = r; i > 0; i -= lowbit( i ) )
        q2.push_back( root[i] );
54:     return vec[qry( 1, tn, q1, q2, k ) - 1];
55: }
56:
57: int main() {
58:     char s[2];
59:     int t;
60:     scanf( "%d", &t );
61:     while( t-- ) {
62:         vec.clear(); tot = 0;
63:         memset( root, 0, sizeof root );
64:         scanf( "%d%d", &n, &m );
65:         for( int i = 1; i <= n; ++i ) scanf( "%d", a + i ),
            vec.push_back( a[i] );
66:         for( int i = 0, x, y, z; i < m; ++i ) {
67:             scanf( "%s", s );

```

```

68:         if( s[0] == 'Q' ) {
69:             op[i].flag = 0;
70:             scanf( "%d%d%d", &op[i].l, &op[i].r, &op[i].k );
71:         } else {
72:             op[i].flag = 1;
73:             scanf( "%d%d", &op[i].l, &op[i].r );
74:             vec.push_back( op[i].r );
75:         }
76:     }
77:     sort( vec.begin(), vec.end() ); tn = unique( vec.begin(),
vec.end() ) - vec.begin();
78:     for( int i = 1; i <= n; ++i ) {
79:         root[i + n] = root[i - 1 + n];
80:         build( 1, tn, root[i + n], getid( a[i] ) );
81:     }
82:     for( int i = 0; i < m; ++i ) {
83:         if( op[i].flag == 0 ) printf( "%d\n", bitquery( op[i].l,
op[i].r, op[i].k ) );
84:         else {
85:             bitinsrt( op[i].l, a[op[i].l], -1 );
86:             bitinsrt( op[i].l, op[i].r, 1 );
87:             a[op[i].l] = op[i].r;
88:         }
89:     }
90: }
91: return 0;
92: }

```

可持久化线段树

HDU 4348 已验

lazy 不下放，只标记，查询时带上即可 省去许多因下放而产生的新节点

```

01: typedef long long LL;
02: const int MAXN = 1e5 + 10;

```

```

03: struct Node { int ls, rs, add; LL sum; };
04: Node tr[MAXN << 5];
05: int root[MAXN];
06: int n, m, tot;
07:
08: int build( int l, int r ) {
09:     int cur = ++tot;
10:     tr[cur].add = 0;
11:     if( l == r ) { scanf( "%I64d", &tr[cur].sum ); return cur; }
12:     int mid = ( l + r ) >> 1;
13:     tr[cur].ls = build( l, mid );
14:     tr[cur].rs = build( mid + 1, r );
15:     tr[cur].sum = tr[tr[cur].ls].sum + tr[tr[cur].rs].sum;
16:     return cur;
17: }
18:
19: void update( int l, int r, int left, int right, int &x, int y, int
val ) {
20:     x = ++tot; tr[x] = tr[y];
21:     tr[x].sum += val * ( right - left + 1 );
22:     if( left <= l && r <= right ) {
23:         tr[x].add += val;
24:         return ;
25:     }
26:     int mid = ( l + r ) >> 1;
27:     if( right <= mid ) update( l, mid, left, right, tr[x].ls,
tr[y].ls, val );
28:     else if( mid < left ) update( mid + 1, r, left, right, tr[x].rs,
tr[y].rs, val );
29:     else {
30:         update( l, mid, left, mid, tr[x].ls, tr[y].ls, val );
31:         update( mid + 1, r, mid + 1, right, tr[x].rs, tr[y].rs,
val );

```

```

32:     }
33: }
34:
35: LL query( int l, int r, int left, int right, int rt ) {
36:     if( left <= l && r <= right ) return tr[rt].sum;
37:     LL ret = tr[rt].add * ( right - left + 1 );
38:     int mid = ( l + r ) >> 1;
39:     if( right <= mid ) return ret + query( l, mid, left, right,
tr[rt].ls );
40:     else if( mid < left ) return ret + query( mid + 1, r, left,
right, tr[rt].rs );
41:     else return ret + query( l, mid, left, mid, tr[rt].ls ) +
query( mid + 1, r, mid + 1, right, tr[rt].rs );
42: }
43:
44: int main() {
45:     char s[5];
46:     int cur;
47:     while( ~scanf( "%d%d", &n, &m ) ) {
48:         tot = 0;
49:         root[cur = 0] = build( 1, n );
50:         for( int i = 0, x, y, z; i < m; ++i ) {
51:             scanf( "%s", s );
52:             if( s[0] == 'C' ) {
53:                 scanf( "%d%d%d", &x, &y, &z ); ++cur;
54:                 update( 1, n, x, y, root[cur], root[cur - 1], z );
55:             } else if( s[0] == 'Q' ) {
56:                 scanf( "%d%d", &x, &y );
57:                 printf( "%I64d\n", query( 1, n, x, y, root[cur] ) );
58:             } else if( s[0] == 'H' ) {
59:                 scanf( "%d%d%d", &x, &y, &z );
60:                 printf( "%I64d\n", query( 1, n, x, y, root[z] ) );
61:             } else scanf( "%d", &cur );

```

```

62:     }
63: }
64:     return 0;
65: }

```

划分树

HDU 3473 已验

求区间第 K 大

cnt 数组 区间内有多少 <= 中位数的数字

num 数组 30 层划分树，存原始数组及排序后的数组

sum 数组 前缀和

leftsum 区间小于中位数的数前缀和

```

01: typedef long long LL;
02: const int MAXN = 1e5 + 10;
03:
04: LL sum[MAXN], leftsum[30][MAXN];
05: int a[MAXN];
06: int num[30][MAXN], cnt[30][MAXN];
07: int n, q;
08: LL lsum, rsum, lcnt, rcnt;
09:
10: void build( int left, int right, int dep ) {
11:     if( left == right ) return ;
12:     int mid = ( left + right ) >> 1, ncnt = mid - left + 1;
13:     for( int i = left; i <= right; ++i ) if( num[dep][i] < a[mid] ) -
ncnt;
14:     int lp = left, rp = mid + 1;
15:     for( int i = left; i <= right; ++i ) {
16:         if( i == left ) cnt[dep][i] = 0;
17:         else cnt[dep][i] = cnt[dep][i - 1];
18:         if( num[dep][i] < a[mid] ) {
19:             ++cnt[dep][i];

```

```

20:         num[dep + 1][lp++] = num[dep][i];
21:         leftsum[dep][i] = leftsum[dep][i - 1] + num[dep][i];
22:     } else if( num[dep][i] > a[mid] ) {
23:         num[dep + 1][rp++] = num[dep][i];
24:         leftsum[dep][i] = leftsum[dep][i - 1];
25:     } else {
26:         if( ncnt ) {
27:             --ncnt; ++cnt[dep][i];
28:             num[dep + 1][lp++] = num[dep][i];
29:             leftsum[dep][i] = leftsum[dep][i - 1] + num[dep][i];
30:         } else {
31:             num[dep + 1][rp++] = num[dep][i];
32:             leftsum[dep][i] = leftsum[dep][i - 1];
33:         }
34:     }
35: }
36: build( left, mid, dep + 1 );
37: build( mid + 1, right, dep + 1 );
38: return ;
39: }
40:
41: int query( int l, int r, int k, int left, int right, int dep ) {
42:     if( l == r ) return num[dep][l];
43:     int mid = ( left + right ) >> 1, s, ss, tmp;
44:     if( l == left ) { s = cnt[dep][r]; ss = 0; }
45:     else { s = cnt[dep][r] - cnt[dep][l - 1]; ss = cnt[dep][l - 1]; }
46:     if( s >= k ) {
47:         l = left + ss;
48:         r = left + ss + s - 1;
49:         tmp = query( l, r, k, left, mid, dep + 1 );
50:     } else {
51:         lcnt += s;
52:         lsum += leftsum[dep][r] - leftsum[dep][l - 1];

```

```

53:         l = mid + 1 + l - left - ss;
54:         r = mid + 1 + r - left - cnt[dep][r];
55:         tmp = query( l, r, k - s, mid + 1, right, dep + 1 );
56:     }
57:     return tmp;
58: }
59:
60: int main() {
61:     int t, x, y, mid, tt = 0;
62:     LL k;
63:     scanf( "%d", &t );
64:     while( t-- ) {
65:         printf( "Case #%d:\n", ++tt );
66:         sum[0] = 0;
67:         scanf( "%d", &n );
68:         for( int i = 1; i <= n; ++i ) {
69:             scanf( "%d", &a[i] );
70:             num[0][i] = a[i];
71:             sum[i] = sum[i - 1] + a[i];
72:         }
73:         sort( a + 1, a + 1 + n );
74:         build( 1, n, 0 );
75:         scanf( "%d", &q );
76:         while( q-- ) {
77:             scanf( "%d%d", &x, &y ); ++x; ++y;
78:             lsum = lcnt = 0;
79:             mid = query( x, y, ( y - x ) / 2 + 1, 1, n, 0 );
80:             rcnt = y - x + 1 - lcnt;
81:             rsum = sum[y] - sum[x - 1] - lsum;
82:             k = rsum - lsum + mid * ( lcnt - rcnt );
83:             printf( "%I64d\n", k );
84:         }
85:         puts( "" );

```

```

86:     }
87:     return 0;
88: }

```

莫队算法

莫队分块

HYSBZ 2038 已验

这是离线算法，需要离散化数据

1. update 函数要求 $O(1)$ 转移
2. update 的第二个参数依据具体情况定
3. 间隔 dm 一般取 \sqrt{n} ，特殊情况卡空间可以改成 $n^{2/3}$
4. 下标从 1 开始

```

01: typedef long long LL;
02: const int MAXN = 100000 + 10;
03: struct Node { int l, r, id; LL a, b; };
04: Node node[MAXN];
05: LL cnt[MAXN], ans;
06: int c[MAXN], pos[MAXN];
07: int n, m;
08:
09: LL gcd( LL a, LL b ) { return b ? gcd( b, a % b ) : a; }
10: bool cmp( const Node &a, const Node &b ) {
11:     return pos[a.l] == pos[b.l] ? a.r < b.r : a.l < b.l;
12: }
13:
14: bool cmp_id( const Node &a, const Node &b ) { return a.id < b.id; }
15:
16: // O(1)
17: void update( int x, int d ) {
18:     ans -= cnt[c[x]] * cnt[c[x]];
19:     cnt[c[x]] += d;

```

```

20:     ans += cnt[c[x]] * cnt[c[x]];
21:     return ;
22: }
23: void solve() {
24:     ans = 0;
25:     for( int i = 0, l = 1, r = 0; i < m; ++i ) {
26:         while( r < node[i].r ) update( ++r, 1 );
27:         while( r > node[i].r ) update( r--, -1 );
28:         while( l < node[i].l ) update( l++, -1 );
29:         while( l > node[i].l ) update( --l, 1 );
30:         // solve
31:         node[i].a = ans - ( r - l + 1 );
32:         node[i].b = 1LL * ( r - l + 1 ) * ( r - l );
33:         LL k = gcd( node[i].a, node[i].b );
34:         node[i].a /= k; node[i].b /= k;
35:     }
36:     return ;
37: }
38: int main() {
39:     scanf( "%d%d", &n, &m );
40:     int dm = ( int )sqrt( n );
41:     for( int i = 1; i <= n; ++i ) {
42:         scanf( "%d", c + i );
43:         pos[i] = ( i - 1 ) / dm + 1;
44:     }
45:     for( int i = 0; i < m; ++i ) {
46:         scanf( "%d%d", &node[i].l, &node[i].r );
47:         node[i].id = i;
48:     }
49:     sort( node, node + m, cmp );
50:     solve();
51:     sort( node, node + m, cmp_id );
52:     for( int i = 0; i < m; ++i ) printf( "%lld/%lld\n", node[i].a,

```

```

        node[i].b );
53:     return 0;
54: }
```

树上莫队

A. 子树树上莫队

现在有一棵树，有 n 个节点，节点有点权，每次询问一个子树内的不重复数个数。

$1 \leq n, q \leq 10^5$, $1 \leq \text{点权} \leq 10^9$ 。

这个题显然比较 **trivial** 嘛...先把点权离散一下，然后一遍 **dfs** 搞出 **dfs** 序，那么一个子树就对应 **dfs** 序上一段，所以我们就可以在 **dfs** 序上莫队，开一个数组记一下每个数的出现次数。

B. 路径树上莫队

现在有一棵树，有 n 个节点，节点有点权，每次询问一条路径上的不重复数个数。

$1 \leq n, q \leq 10^5$, $1 \leq \text{点权} \leq 10^9$ 。

莫队用不了了？我们重新定义一个 **dfs** 序！

我们在开始访问和结束访问一个点的时候都记一下时间戳，我们设开始访问的时间为 **st**，结束访问的时间为 **ed**。

我们假设要询问一条路径 **a-b**，设 **lca** 为 $p = \text{lca}(a, b)$ 。不妨设 $\text{st}[a] \leq \text{st}[b]$ （否则交换一下）。

当 $p = a$ 时，这应该是一个比较简单的情形：**a-b** 是一段父子链。

我们考虑这个新 **dfs** 序上 $[\text{st}[a], \text{st}[b]]$ 的点，我们可以发现，**a-b** 上的点被算了一遍，其他点都被算了 2 遍或 0 遍！那么我们统计的时候注意一下就可以了。

当 $p \neq a$ 时，我们也要一样统计 $[\text{ed}[a], \text{st}[b]]$ 的点（从 $\text{ed}[a]$ 开始为保证 **a** 不会被排除掉），但是这回 **lca** 会被重复统计，所以要另外算一下。

树链剖分

还在学习姿势 ing

```

01: typedef long long LL;
02: const int MAXN = 10000 + 10;
03: const int MAXM = 1000000 + 10;
04: #define MID(x, y) (((x) + (y)) >> 1)
```

```

05:
06: int fa[MAXN], top[MAXN], w[MAXN], son[MAXN], dep[MAXN], sz[MAXN],
    r[MAXN];
07: int a[MAXN], b[MAXN];
08: LL c[MAXN];
09: int ind[MAXN];
10: int t[MAXM], nt[MAXM];
11: int cnt1, cnt2, cnt3;
12: int n, m;
13:
14: struct node {
15:     int l, r;
16:     int a, b;
17:     LL sum;
18: } f[MAXM];
19: int rt;
20:
21: void dfs1( int x, int d ) {
22:     dep[x] = d; son[x] = 0; sz[x] = 1;
23:     for( int k = ind[x]; ~k; k = nt[k] ) {
24:         if( t[k] != fa[x] ) {
25:             fa[t[k]] = x;
26:             dfs1( t[k], d + 1 );
27:             sz[x] += sz[t[k]];
28:             if( sz[t[k]] > sz[son[x]] ) son[x] = t[k];
29:         }
30:     }
31:     return ;
32: }
33:
34: void dfs2( int x, int tt ) {
35:     w[x] = ++cnt2; top[x] = tt;
36:     if( son[x] ) dfs2( son[x], tt );
```

```

37:   for( int k = ind[x]; ~k; k = nt[k] ) {
38:       if( t[k] != fa[x] && t[k] != son[x] )
39:           dfs2( t[k], t[k] );
40:   }
41:   return ;
42: }
43:
44: void add( int a, int b ) {
45:     t[cnt1] = b; nt[cnt1] = ind[a]; ind[a] = cnt1++;
46:     return ;
47: }
48:
49: void update( int x ) {
50:     f[x].sum = f[f[x].l].sum + f[f[x].r].sum;
51: }
52:
53: int bt( int a, int b ) {
54:     int x = cnt3++;
55:     f[x].a = a; f[x].b = b;
56:     if( a < b ) {
57:         int mid = MID( a, b );
58:         f[x].l = bt( a, mid );
59:         f[x].r = bt( mid + 1, b );
60:         f[x].sum = 0;
61:     } else f[x].sum = 0;
62:     return x;
63: }
64:
65: // Query On ST, Do not Call Directly
66: LL query( int x, int a, int b ) {
67:     if( a <= f[x].a && f[x].b <= b ) return f[x].sum;
68:     int mid = MID( f[x].a, f[x].b );
69:     LL ans = 0;

```

```

70:     if( a <= mid ) ans += query( f[x].l, a, b );
71:     if( b > mid ) ans += query( f[x].r, a, b );
72:     return ans;
73: }
74:
75: //Modify Point
76: void update( int x, int p, int cc ) {
77:     if( f[x].a == f[x].b ) { f[x].sum = cc; return; }
78:     int mid = MID( f[x].a, f[x].b );
79:     if( p <= mid ) update( f[x].l, p, cc );
80:     else update( f[x].r, p, cc );
81:     update( x );
82:     return ;
83: }
84:
85: //Query Segment
86: LL query( int x, int y ) {
87:     int fx = top[x], fy = top[y];
88:     LL sum = 0;
89:     while( fx != fy ) {
90:         if( dep[fx] < dep[fy] ) {
91:             swap( x, y );
92:             swap( fx, fy );
93:         }
94:         sum += query( rt, w[fx], w[x] );
95:         x = fa[top[x]];
96:         fx = top[x];
97:     }
98:     if( dep[x] > dep[y] ) swap( x, y );
99:     if( x == y ) return sum;
100:    return sum + query( rt, w[son[x]], w[y] );
101: }
102:

```

```
103: int main() {
104:     return 0;
105: }
```

动态规划

树 DP

经典

```
01: const int MAXN = 1e5 + 10;
02:
03: struct Edge { int to, cost, next; };
04: Edge es[MAXN << 2];
05: int head[MAXN], nmax[MAXN][2], son[MAXN][2];
06: int n, cnt;
07:
08: void add( int u, int v, int w ) {
09:     es[cnt].to = v; es[cnt].cost = w; es[cnt].next = head[u]; head[u]
    = cnt++;
10:     return ;
11: }
12:
13: void dfs1( int u, int fa ) {
14:     for( int i = head[u]; ~i; i = es[i].next ) {
15:         int v = es[i].to;
16:         if( v == fa ) continue;
17:         dfs1( v, u );
18:         int tmp = nmax[v][0] + es[i].cost;
19:         if( tmp > nmax[u][0] ) {
20:             nmax[u][1] = nmax[u][0]; son[u][1] = son[u][0];
21:             nmax[u][0] = tmp; son[u][0] = v;
```

```
22:         } else if( tmp > nmax[u][1] ) {
23:             nmax[u][1] = tmp; son[u][1] = v;
24:         }
25:     }
26:     return ;
27: }
28:
29: void dfs2( int u, int fa, int len ) {
30:     int tmp;
31:     if( son[fa][0] != u ) tmp = len + nmax[fa][0];
32:     else tmp = len + nmax[fa][1];
33:     if( tmp > nmax[u][0] ) {
34:         nmax[u][1] = nmax[u][0]; son[u][1] = son[u][0];
35:         nmax[u][0] = tmp; son[u][0] = fa;
36:     } else if( tmp > nmax[u][1] ) {
37:         nmax[u][1] = tmp; son[u][1] = fa;
38:     }
39:     for( int i = head[u]; ~i; i = es[i].next ) {
40:         int v = es[i].to;
41:         if( v == fa ) continue;
42:         dfs2( v, u, es[i].cost );
43:     }
44:     return ;
45: }
46:
47: int main() {
48:     int ta, tb;
49:     while( ~scanf( "%d", &n ) ) {
50:         memset( head, -1, sizeof( head ) );
51:         memset( nmax, 0, sizeof( nmax ) );
52:         memset( son, -1, sizeof( son ) );
53:         for( int i = 2; i <= n; ++i ) {
54:             scanf( "%d%d", &ta, &tb );
```



```

55:         add( i, ta, tb );
56:         add( ta, i, tb );
57:     }
58:     dfs1( 1, 0 );
59:     dfs2( 1, 0, 0 );
60:     for( int i = 1; i <= n; ++i ) printf( "%d\n", nmax[i][0] );
61: }
62: return 0;
63: }

```

删点

```

01: const int INF = 0x3F3F3F3F;
02: const int MAXN = 1e5 + 10;
03:
04: struct Edge { int to, next; };
05: Edge es[MAXN << 2];
06: int head[MAXN], son[MAXN], dp[MAXN], ans[MAXN];
07: int n, m, cnt, nmin;
08:
09: void add( int u, int v ) {
10:     es[cnt].to = v; es[cnt].next = head[u]; head[u] = cnt++;
11:     return ;
12: }
13:
14: void dfs( int u, int fa ) {
15:     son[u] = 1; dp[u] = 0;
16:     for( int i = head[u]; ~i; i = es[i].next ) {
17:         int v = es[i].to;
18:         if( v == fa ) continue;
19:         dfs( v, u );
20:         son[u] += son[v];
21:         dp[u] = max( dp[u], son[v] );
22:     }

```

```

23:     dp[u] = max( dp[u], n - son[u] );
24:     nmin = min( nmin, dp[u] );
25:     return ;
26: }
27:
28: int main() {
29:     int ta, tb;
30:     while( ~scanf( "%d", &n ) ) {
31:         memset( head, -1, sizeof( head ) );
32:         cnt = 0; nmin = INF;
33:         for( int i = 1; i < n; ++i ) {
34:             scanf( "%d%d", &ta, &tb );
35:             add( ta, tb ); add( tb, ta );
36:         }
37:         dfs( 1, -1 );
38:         int tn = 0;
39:         for( int i = 1; i <= n; ++i ) {
40:             if( dp[i] == nmin ) ans[tn++] = i;
41:         }
42:         for( int i = 0; i < tn - 1; ++i ) printf( "%d ", ans[i] );
43:         printf( "%d\n", ans[tn - 1] );
44:     }
45:     return 0;
46: }

```

树上背包

```

01: const int NINF = 0x80808080;
02: const int MAXN = 3e3 + 10;
03:
04: struct Edge { int to, cost, next; };
05: Edge es[MAXN << 2];
06: int head[MAXN], dp[MAXN][MAXN], val[MAXN], num[MAXN];
07: int n, m, tn, cnt;

```

```

08:
09: void add( int u, int v, int w ) {
10:     es[cnt].to = v; es[cnt].cost = w; es[cnt].next = head[u]; head[u]
    = cnt++;
11:     return ;
12: }
13:
14: void dfs( int u, int fa ) {
15:     memset( dp[u], 0x80, sizeof( dp[u] ) ); dp[u][0] = 0;
16:     if( head[u] == -1 ) { dp[u][1] = val[u]; return ; }
17:     for( int i = head[u]; ~i; i = es[i].next ) {
18:         int v = es[i].to, cost = es[i].cost;
19:         if( v == fa ) continue;
20:         dfs( v, u );
21:         num[u] += num[v];
22:         for( int j = num[u]; j >= 1; --j ) {
23:             for( int k = 0; k < j; ++k )
24:                 dp[u][j] = max( dp[u][j], dp[u][k] + dp[v][j - k] -
cost );
25:         }
26:     }
27:     return ;
28: }
29:
30: int main() {
31:     int ta, tb;
32:     while( ~scanf( "%d%d", &n, &m ) ) {
33:         memset( head, -1, sizeof( head ) );
34:         memset( num, 0, sizeof( num ) );
35:         cnt = 0;
36:         for( int i = 1; i <= n - m; ++i ) {
37:             scanf( "%d", &tn );
38:             for( int j = 0; j < tn; ++j ) {

```

```

39:                 scanf( "%d%d", &ta, &tb );
40:                 add( i, ta, tb );
41:             }
42:             val[i] = 0;
43:         }
44:         for( int i = n - m + 1; i <= n; ++i ) { scanf( "%d", val +
i ); num[i] = 1; }
45:         dfs( 1, -1 );
46:         for( int i = num[1]; i >= 0; --i ) {
47:             if( dp[1][i] >= 0 ) {
48:                 printf( "%d\n", i );
49:                 break;
50:             }
51:         }
52:     }
53:     return 0;
54: }

```

应用：树上任意点能到达的最远距离

```

01: #define INF 0x3f3f3f3f
02: const int MAXN = 50000 + 10;
03: const int MAXM = 100000 + 10;
04:
05: struct Edge { int v, w, next; };
06: Edge es[MAXM];
07: int head[MAXN], mmax[MAXN][2], poi[MAXN][2], dis[MAXN],
    disf[MAXN][2], mlog[MAXN];
08: int d1[MAXN][17], d2[MAXN][17];
09: int n, m, cnt;
10:
11: void add( int u, int v, int w ) {
12:     es[cnt].v = v; es[cnt].w = w; es[cnt].next = head[u]; head[u] =
cnt++;

```

```

13:     return ;
14: }
15:
16: void dfs( int u, int pre ) {
17:     bool flag = false;
18:     for( int i = head[u]; ~i; i = es[i].next ) {
19:         int v = es[i].v, w = es[i].w;
20:         if( v != pre ) {
21:             flag = true;
22:             dfs( v, u );
23:             if( mmax[u][0] < mmax[v][0] + w ) {
24:                 mmax[u][1] = mmax[u][0];
25:                 mmax[u][0] = mmax[v][0] + w;
26:                 poi[u][1] = poi[u][0];
27:                 poi[u][0] = v;
28:             } else if( mmax[u][1] < mmax[v][0] + w ) {
29:                 mmax[u][1] = mmax[v][0] + w;
30:                 poi[u][1] = v;
31:             }
32:         }
33:     }
34:     if( !flag ) {
35:         dis[u] = 0;
36:         mmax[u][0] = mmax[u][1] = 0;
37:         poi[u][0] = poi[u][1] = 0;
38:     }
39:     return ;
40: }
41:
42: void dfs2( int u, int pre ) {
43:     for( int i = head[u]; ~i; i = es[i].next ) {
44:         int v = es[i].v, w = es[i].w;
45:         if( v != pre ) {

```

```

46:             if( v == poi[u][0] ) {
47:                 dis[v] = max( mmax[v][0], w + disf[u][0] );
48:                 disf[v][0] = max( disf[u][0] + w, mmax[v][1] );
49:                 disf[v][1] = max( disf[u][0] + w, mmax[v][0] );
50:             } else {
51:                 dis[v] = max( mmax[v][0], w + disf[u][1] );
52:                 disf[v][0] = max( disf[u][1] + w, mmax[v][1] );
53:                 disf[v][1] = max( disf[u][1] + w, mmax[v][0] );
54:             }
55:             dfs2( v, u );
56:         }
57:     }
58:     return ;
59: }
60:
61: int main() {
62:     int a, b, c, q;
63:     while( ~scanf( "%d%d", &n, &m ) && n + m ) {
64:         memset( head, -1, sizeof( head ) );
65:         memset( mmax, 0, sizeof( mmax ) );
66:         memset( dis, 0, sizeof( dis ) );
67:         cnt = 0;
68:         add( 0, 1, 0 );
69:         for( int i = 0; i < n - 1; ++i ) {
70:             scanf( "%d%d%d", &a, &b, &c );
71:             add( a, b, c ); add( b, a, c );
72:         }
73:         dfs( 0, 0 );
74:         disf[0][0] = disf[0][1] = 0;
75:         poi[0][0] = 1; poi[0][1] = 0;
76:         dfs2( 0, 0 );
77:     }
78:     return 0;

```

```
79: }
```

区间 DP

```
const int INF = 0x3F3F3F3F;
const int MAXN = 100 + 10;

int a[MAXN], dp[MAXN][MAXN];
int n;

int main() {
    memset( dp, 0, sizeof( dp ) );
    scanf( "%d", &n );
    for( int i = 1; i <= n; ++i ) scanf( "%d", a + i );
    for( int m = 3; m <= n; ++m ) {
        for( int i = 1; i <= n - m + 1; ++i ) {
            int j = i + m - 1;
            dp[i][j] = INF;
            for( int k = i + 1; k < j; ++k ) {
                dp[i][j] = min( dp[i][j], dp[i][k] + dp[k][j] + a[i] *
a[k] * a[j] );
            }
        }
    }
    printf( "%d\n", dp[1][n] );
    return 0;
}
```

数位 DP

```
01: typedef long long LL;
02: const int MOD = 2520;
03:
```

```
04: LL dp[21][MOD + 10][50];
05: int index[MOD + 10], dig[21];
06:
07: void init() {
08:     int cnt = 0;
09:     for( int i = 1; i <= MOD; ++i ) if( MOD % i == 0 ) index[i] =
cnt++;
10:     memset( dp, -1, sizeof( dp ) );
11:     return ;
12: }
13:
14: int gcd( int a, int b ) { return b == 0 ? a : gcd( b, a % b ); }
15: int lcm( int a, int b ) { return a / gcd( a, b ) * b; }
16:
17: LL dfs( int pos, int presum, int prelcm, bool edge ) {
18:     if( pos == -1 ) return presum % prelcm == 0;
19:     if( !edge && dp[pos][presum][index[prelcm]] != -1 ) {
20:         return dp[pos][presum][index[prelcm]];
21:     }
22:     LL ret = 0;
23:     int ed = edge ? dig[pos] : 9;
24:     for( int i = 0; i <= ed; ++i ) {
25:         int nowsum = ( presum * 10 + i ) % MOD;
26:         int nowlcm = prelcm;
27:         if( i ) nowlcm = lcm( prelcm, i );
28:         ret += dfs( pos - 1, nowsum, nowlcm, edge && ( i == ed ) );
29:     }
30:     if( !edge ) dp[pos][presum][index[prelcm]] = ret;
31:     return ret;
32: }
33:
34: LL gao( LL x ) {
35:     int pos = 0;
```

```

36:     while( x ) {
37:         dig[pos++] = x % 10;
38:         x /= 10;
39:     }
40:     return dfs( pos - 1, 0, 1, true );
41: }
42:
43: int main() {
44:     int t;
45:     init();
46:     scanf( "%d", &t );
47:     while( t-- ) {
48:         LL l, r;
49:         scanf( "%I64d %I64d", &l, &r );
50:         printf( "%I64d\n", gao( r ) - gao( l - 1 ) );
51:     }
52:     return 0;
53: }

```

可能的树分治模板

```

01: const int INF = 0x3F3F3F3F;
02: const int MAXN = 1e4 + 10;
03:
04: struct Edge { int to, cost, next; };
05: Edge es[MAXN << 1];
06: int head[MAXN], son[MAXN], dp[MAXN], dep[MAXN];
07: bool vis[MAXN];
08: vector<int> vdep;
09: int n, m, cnt, root, size, ans;
10:
11: void add( int u, int v, int w ) {
12:     es[cnt].to = v; es[cnt].cost = w; es[cnt].next = head[u]; head[u]

```

```

= cnt++;
13:     return ;
14: }
15:
16: void getroot( int u, int fa ) {
17:     son[u] = 1; dp[u] = 0;
18:     for( int i = head[u]; ~i; i = es[i].next ) {
19:         int v = es[i].to;
20:         if( v == fa || vis[v] ) continue;
21:         getroot( v, u );
22:         son[u] += son[v];
23:         dp[u] = max( dp[u], son[v] );
24:     }
25:     dp[u] = max( dp[u], size - son[u] );
26:     if( dp[u] < dp[root] ) root = u;
27:     return ;
28: }
29:
30: void getdep( int u, int fa ) {
31:     vdep.push_back( dep[u] );
32:     son[u] = 1;
33:     for( int i = head[u]; ~i; i = es[i].next ) {
34:         int v = es[i].to;
35:         if( v == fa || vis[v] ) continue;
36:         dep[v] = dep[u] + es[i].cost;
37:         getdep( v, u );
38:         son[u] += son[v];
39:     }
40:     return ;
41: }
42:
43: int calc( int u, int init ) {
44:     int ret = 0;

```

```

45:     vdep.clear(); dep[u] = init;
46:     getdep( u, 0 );
47:     sort( vdep.begin(), vdep.end() );
48:     int l = 0, r = vdep.size() - 1;
49:     while( l < r ) {
50:         if( vdep[l] + vdep[r] <= m ) { ret += r - l; ++l; }
51:         else r--;
52:     }
53:     return ret;
54: }
55:
56: void solve( int u ) {
57:     ans += calc( u, 0 );
58:     vis[u] = true;
59:     for( int i = head[u]; ~i; i = es[i].next ) {
60:         int v = es[i].to;
61:         if( vis[v] ) continue;
62:         ans -= calc( v, es[i].cost );
63:         dp[0] = size = son[v];
64:         getroot( v, root = 0 );
65:         solve( root );
66:     }
67:     return ;
68: }
69:
70: int main() {
71:     int ta, tb, tc;
72:     while( ~scanf( "%d%d", &n, &m ) && n + m ) {
73:         memset( vis, false, sizeof( vis ) );
74:         memset( head, -1, sizeof( head ) );
75:         cnt = 0;
76:         for( int i = 1; i < n; ++i ) {
77:             scanf( "%d%d%d", &ta, &tb, &tc );

```

```

78:             add( ta, tb, tc ); add( tb, ta, tc );
79:         }
80:         ans = 0; dp[0] = size = n;
81:         getroot( 1, root = 0 );
82:         solve( root );
83:         printf( "%d\n", ans );
84:     }
85:     return 0;
86: }

```

字符串

AC 自动机

HDU 2222 验过

1. 数组用法

len 长度 end 模式串终结符
依据需要添加数组记录相应信息

2. insert()

依据数组修改, 注意对应数组含义

3. query()

重点在改最内层的 while 循环

```
01: struct ACAuto {
02:     const static int type = 26;
03:     int next[MAXN][type], fail[MAXN], end[MAXN], len[MAXN];
04:     int root, tot;
05:     int newnode() {
06:         for( int i = 0; i < type; ++i ) next[tot][i] = -1;
07:         len[tot] = 0; end[tot++] = -1;
08:         return tot - 1;
09:     }
10:     void init() {
11:         tot = 0;
12:         root = newnode();
13:     }
14:     void insert( char *s ) {
15:         int tlen = strlen( s ), u = root;
16:         for( int i = 0; i < tlen; ++i ) {
17:             int idx = s[i] - 'a';
18:             if( next[u][idx] == -1 ) next[u][idx] = newnode();
19:             u = next[u][idx];
20:         }
```

```
21:         end[u] = 1; len[u] = tlen;
22:     }
23:     void build() {
24:         queue<int> que;
25:         fail[root] = root;
26:         for( int i = 0; i < type; ++i ) {
27:             if( next[root][i] == -1 ) next[root][i] = root;
28:             else {
29:                 fail[next[root][i]] = root;
30:                 que.push( next[root][i] );
31:             }
32:         }
33:         while( !que.empty() ) {
34:             int u = que.front(); que.pop();
35:             for( int i = 0; i < type; ++i ) {
36:                 if( next[u][i] == -1 ) next[u][i] = next[fail[u]][i];
37:                 else {
38:                     fail[next[u][i]] = next[fail[u]][i];
39:                     que.push( next[u][i] );
40:                 }
41:             }
42:         }
43:     }
44:     void query( char *s ) {
45:         int idx, tlen = strlen( s ), u = root;
46:         memset( pos, 0, sizeof pos );
47:         for( int i = 0; i < tlen; ++i ) {
48:             // 忽略大小写
49:             if( s[i] >= 'A' && s[i] <= 'Z' ) idx = s[i] - 'A';
50:             else if( s[i] >= 'a' && s[i] <= 'z' ) idx = s[i] - 'a';
51:             else continue;
52:             u = next[u][idx];
53:             int tp = u;
```

```

54:         while( tp != root ) {
55:             if( end[tp] != -1 ) {
56:                 pos[i + 1] -= 1;
57:                 pos[i - len[tp] + 1] += 1;
58:                 break;
59:             }
60:             tp = fail[tp];
61:         }
62:     }
63:     int cnt = 0;
64:     for( int i = 0; i < tlen; ++i ) {
65:         cnt += pos[i];
66:         if( cnt <= 0 ) putchar( s[i] );
67:         else putchar( '*' );
68:     }
69:     puts( "" );
70: }
71: }AC;

```

```

13:         while( ( j >= 0 ) && ( B[j + 1] != B[i] ) ) j = p[j];
14:         if( B[j + 1] == B[i] ) ++j;
15:         p[i] = j;
16:     }
17:     for( i = 0, j = -1; i < len1; ++i ) {
18:         while( ( j >= 0 ) && ( B[j + 1] != A[i] ) ) j = p[j];
19:         if( B[j + 1] == A[i] ) ++j;
20:         if( j == len2 - 1 ) {
21:             cout << i + 1 - len2 << endl;
22:             // j = p[j]; //multiple matching
23:             break;
24:         }
25:     }
26:     return 0;
27: }

```

KMP

```

01: #include <bits/stdc++.h>
02: using namespace std;
03:
04: int main() {
05:     int i, j;
06:     int p[100010], len1, len2;
07:     char A[100010], B[100010];
08:     scanf( "%s%s", A, B );
09:     len1 = strlen( A );
10:     len2 = strlen( B );
11:     p[0] = -1;
12:     for( i = 1, j = -1; i < len2; ++i ) {

```


数学

Ploya 定理

```
01: #define LL long long
02: LL c, s;
03:
04: LL gcd( LL a, LL b ) { return b == 0 ? a : gcd( b, a % b ); }
05:
06: LL pow( LL a, LL b ) {
07:     LL ans = 1;
08:     while( b ) {
09:         if( b & 1 ) ans *= a;
10:         a *= a;
11:         b >>= 1;
12:     }
13:     return ans;
14: }
15:
16: LL polya() {
17:     LL i, j;
18:     LL sum = 0;
19:     for( i = 1; i <= s; i++ )
20:         sum += pow( c, gcd( s, i ) );
21:     if( s & 1 )
22:         sum += s * pow( c, s / 2 + 1 );
23:     else
24:         sum += ( ( s / 2 ) * pow( c, s / 2 ) ) + ( ( s / 2 ) *
        pow( c, s / 2 + 1 ) );
25:     sum /= ( 2 * s );
26:     return sum;
27: }
```

```
28: int main() {
29:     while( ~scanf( "%I64d%I64d", &c, &s ) && ( c || s ) )
30:         printf( "%I64d\n", polya() );
31:     return 0;
32: }
```

SG 博弈

定义 P-position 和 N-position, 其中 P 代表 Previous, N 代表 Next。直观的说, 上一次 move 的人有必胜策略的局面是 P-position, 也就是“后手可保证必胜”或者“先手必败”, 现在轮到 move 的人有必胜策略的局面是 N-position, 也就是“先手可保证必胜”。更严谨的定义是: 1.无法进行任何移动的局面 (也就是 terminal position) 是 P-position; 2.可以移动到 P-position 的局面是 N-position; 3.所有移动都导致 N-position 的局面是 P-position。按照这个定义, 如果局面不可能重现, 或者说 positions 的集合可以进行拓扑排序, 那么每个 position 或者是 P-position 或者是 N-position, 而且可以通过定义计算出来。

!!!对于一个 Nim 游戏的局面(a1,a2,...,an), 它是 P-position 当且仅当 $a1 \wedge a2 \wedge \dots \wedge an = 0$, 其中 \wedge 表示异或(xor)运算。

如果 Nim 游戏中的规则稍微变动一下, 每次最多只能取 K 个, 怎么处理? 方法是将每堆石子数 mod (k+1)。

```
01: const int MAXN = 1e5 + 10;
02: int sg[MAXN], a[MAXN];
03: int n, m;
04: int gao( int x ) {
05:     if( ~sg[x] ) return sg[x];
06:     int i, vis[100];
07:     memset( vis, 0, sizeof( vis ) );
08:     for( i = 0; i < n; ++i ) {
09:         if( x < a[i] ) break;
10:         vis[gao( x - a[i] )] = 1;
11:     }
12:     for( i = 0; vis[i]; ++i );
13:     return sg[x] = i; }
```

```

14: int main() {
15:     int th, ta, ans;
16:     while( ~scanf( "%d", &n ) && n ) {
17:         memset( sg, -1, sizeof( sg ) );
18:         for( int i = 0; i < n; ++i ) scanf( "%d", a + i );
19:         sort( a, a + n );
20:         scanf( "%d", &m );
21:         while( m-- ) {
22:             ans = 0;
23:             scanf( "%d", &th );
24:             while( th-- ) {
25:                 scanf( "%d", &ta );
26:                 ans ^= gao( ta );
27:             }
28:             putchar( ans ? 'W' : 'L' );
29:         }
30:         puts( "" );
31:     }
32:     return 0;
33: }
34:
35: /*
36: const int MAXN = 1e6 + 10;
37: int n;
38:
39: int main() {
40:     int t, ta;
41:     scanf( "%d", &t );
42:     while( t-- ) {
43:         int ans = 0;
44:         scanf( "%d", &n );
45:         for( int i = 0; i < n; ++i ) {
46:             scanf( "%d", &ta );

```

```

47:             if( ta % 4 == 0 ) ans ^= ( ta - 1 );
48:             else if( ta % 4 == 1 || ta % 4 == 2 ) ans ^= ta;
49:             else ans ^= ( ta + 1 );
50:         }
51:         if( ans ) puts( "Alice" );
52:         else puts( "Bob" );
53:     }
54:     return 0;
55: }
56: */

```

逆元处理求组合数(MOD 是质数)

```

01: typedef long long LL;
02: const int INF = 0x7F7F7F7F;
03: const int MOD = 1e9 + 7;
04: const int MAXN = 100000 + 10;
05:
06: LL fac[MAXN], inv[MAXN];
07:
08: LL pmod( LL a, LL b ) {
09:     LL ans = 1;
10:     while( b ) {
11:         if( b & 1 ) ans = ans * a % MOD;
12:         a = a * a % MOD;
13:         b >>= 1;
14:     }
15:     return ans;
16: }
17:
18: LL _inv( LL x ) { return pmod( x, MOD - 2 ); }
19:
20: int main() {

```

```

21:     fac[0] = inv[0] = 1;
22:     for( int i = 1; i < MAXN; ++i ) { fac[i] = fac[i - 1] * i; inv[i]
    = _inv( fac[i] ); }
23:     return 0;
24: }

```

组合数

```

01: const int M = 1007;
02: const int MAXN = 1000;
03: long long C[MAXN+1][MAXN+1];
04: void Initial() {
05:     int i,j;
06:     for(i=0; i<=MAXN; ++i) {
07:         C[0][i] = 0;
08:         C[i][0] = 1;
09:     }
10:     for(i=1; i<=MAXN; ++i) {
11:         for(j=1; j<=MAXN; ++j)
12:             C[i][j] = (C[i-1][j] + C[i-1][j-1]);
13:     }
14: }
15: long long Combination(int n, int m) { return C[n][m]; }
16: int main() {
17:     int T,i,m,n;
18:     Initial();
19:     while( ~scanf("%d%d",&n,&m) ){
20:         printf("C(%d, %d)=%I64d\n",n,m,Combination(n,m));
21:     }
22:     return 0;
23: }

```

快速线性素数筛

```

01: #define N 1000010
02:
03: int prime[N];
04: bool vis[N];
05: int num;
06:
07: void _prime() {
08:     int i, j;
09:     memset( vis, true, sizeof( vis ) );
10:     vis[0] = vis[1] = 0;
11:     for( i = 2, num = 0; i < N; ++i ) {
12:         if( vis[i] )
13:             prime[num++] = i;
14:         for( j = 0; j < num && i * prime[j] < N; ++j ) {
15:             vis[i * prime[j]] = 0;
16:             if( !( i % prime[j] ) )
17:                 break;
18:         }
19:     }
20:     return ;
21: }

```

计算几何

点与矩阵最小距离

```
01: const double eps = 1e-8;
02: const double PI = acos( -1.0 );
03:
04: int sig( double x ) {
05:     if( fabs( x ) < eps ) return 0;
06:     return x > 0 ? 1 : -1;
07: }
08:
09: struct Point {
10:     double x, y;
11:     Point() {}
12:     Point( const double xx, const double yy ) { x = xx; y = yy; }
13:     Point operator + ( const Point &tp ) const { return Point( x +
    tp.x, y + tp.y ); }
14:     Point operator - ( const Point &tp ) const { return Point( x -
    tp.x, y - tp.y ); }
15:     double operator * ( const Point &tp ) const { return x * tp.x + y
    * tp.y; }
16:     double operator ^ ( const Point &tp ) const { return x * tp.y - y
    * tp.x; }
17:     bool operator < ( const Point &tp ) const {
18:         if( sig( x - tp.x ) ) return sig( x - tp.x ) < 0;
19:         else return sig( y - tp.y ) < 0;
20:     }
21: };
22: Point poi[5], cir, src;
23: double r;
24:
```

```
25: double dist( const Point &a, const Point &b ) {
26:     double dx = a.x - b.x;
27:     double dy = a.y - b.y;
28:     return sqrt( dx * dx + dy * dy );
29: }
30:
31: double dptol( const Point &a, const Point &b, const Point &c ) {
32:     double ret = 0;
33:     if( sig( ( c - b ) * ( a - b ) ) > 0 && sig( ( b - c ) * ( a -
    c ) ) > 0 )
34:         ret = fabs( ( b - a ) ^ ( c - a ) ) / dist( b, c );
35:     else ret = min( dist( a, b ), dist( a, c ) );
36:     return ret;
37: }
38:
39: double dptor( const Point &a ) {
40:     double d1 = min( dptol( a, poi[0], poi[1] ), dptol( a, poi[0],
    poi[2] ) );
41:     double d2 = min( dptol( a, poi[1], poi[3] ), dptol( a, poi[2],
    poi[3] ) );
42:     return min( d1, d2 );
43: }
44:
45: int main() {
46:     return 0;
47: }
```

点在多边形内

```
01: const double eps = 1e-8;
02: const double PI = acos( -1.0 );
03:
04: int sig( double x ) {
```

```

05:     if( fabs( x ) < eps ) return 0;
06:     return x > 0 ? 1 : -1;
07: }
08:
09: struct Point {
10:     double x, y;
11:     Point() {}
12:     Point( const double xx, const double yy ) { x = xx; y = yy; }
13:     Point operator + ( const Point &tp ) const { return Point( x +
    tp.x, y + tp.y ); }
14:     Point operator - ( const Point &tp ) const { return Point( x -
    tp.x, y - tp.y ); }
15:     double operator * ( const Point &tp ) const { return x * tp.x + y
    * tp.y; }
16:     double operator ^ ( const Point &tp ) const { return x * tp.y - y
    * tp.x; }
17:     bool operator < ( const Point &tp ) const {
18:         if( sig( x - tp.x ) ) return sig( x - tp.x ) < 0;
19:         else return sig( y - tp.y ) < 0;
20:     }
21: };
22: typedef Point pVector;
23: typedef vector<Point> Polygon;
24:
25: bool OnSegment( Point p, Point a, Point b ) {
26:     if( sig( ( p - a ) * ( p - b ) ) ) return 0;
27:     return sig( a.x - p.x ) * sig( b.x - p.x ) <= 0 && sig( a.y -
    p.y ) * sig( b.y - p.y ) <= 0;
28: }
29:
30: int isPointInPolygon( Point p, Polygon poly ) {
31:     int wn = 0;
32:     int n = poly.size();

```

```

33:     for( int i = 0; i < n; ++i ) {
34:         if( OnSegment( p, poly[i], poly[( i + 1 ) % n] ) ) return 0;
35:         int k = sig( ( poly[( i + 1 ) % n] - poly[i] ) ^ ( p -
    poly[i] ) );
36:         int d1 = sig( poly[i].y - p.y );
37:         int d2 = sig( poly[( i + 1 ) % n].y - p.y );
38:         if( k > 0 && d1 <= 0 && d2 > 0 ) ++wn;
39:         if( k < 0 && d2 <= 0 && d1 > 0 ) --wn;
40:     }
41:     return wn;
42: }
43:
44: int main() {
45:     return 0;
46: }

```

多边形与圆面积交

```

01: typedef long long LL;
02: const double INF = 1000000000000;
03: const double eps = 1e-12;
04: const double PI = acos( -1.0 );
05: const int MAXN = 100009;
06: const int MOD = 1000000007;
07:
08: struct Point {
09:     double x,y;
10:     Point(){}
11:     Point( double xx, double yy ) { x = xx; y = yy; }
12:     Point operator - ( Point s ) { return Point( x - s.x, y -
    s.y ); }
13:     Point operator + ( Point s ) { return Point( x + s.x, y +
    s.y ); }

```

```

14: double operator * ( Point s ) { return x * s.x + y * s.y; }
15: double operator ^ ( Point s ) { return x * s.y - y * s.x; }
16: }poi[MAXN];
17:
18: double mmax( double a, double b ) { return a > b ? a : b; }
19: double mmin( double a, double b ) { return a < b ? a : b; }
20:
21: double len( Point a ) { return sqrt( a * a ); }
22: double dist( Point a, Point b ) { return len( b - a ); }
23:
24: double cross( Point a, Point b, Point c ) { return ( b - a ) ^ ( c -
    a ); }
25: double dot( Point a, Point b, Point c ) { return ( b - a ) * ( c -
    a ); }
26:
27: double area( Point b, Point c, double r ) {
28:     Point a( 0.0, 0.0 );
29:     if( dist( b, c ) < eps ) return 0.0;
30:     double h = fabs( cross( a, b, c ) ) / dist( b, c );
31:     //两个端点都在圆的外面则分为两种情况
32:     //两个端点都在圆内的情况
33:     //一个端点在圆上一个端点在圆内的情况
34:     if( dist( a, b ) > r - eps && dist( a, c ) > r - eps ) {
35:         double angle = acos( dot( a, b, c ) / dist( a, b ) / dist( a,
            c ) );
36:         if( h > r - eps ) return 0.5 * r * r * angle;
37:         else if( dot( b, a, c ) > 0 && dot( c, a, b ) > 0 ) {
38:             double angle1 = 2 * acos( h / r );
39:             return 0.5 * r * r * fabs( angle - angle1 ) + 0.5 * r * r
                * sin( angle1 );
40:         } else return 0.5 * r * r * angle;
41:     } else if( dist( a, b ) < r + eps && dist( a, c ) < r + eps ) {
42:         return 0.5 * fabs( cross( a, b, c ) );

```

```

43:     } else {
44:         //默认 b 在圆内
45:         if( dist( a, b ) > dist( a, c ) ) swap(b,c);
46:         //ab 距离为 0 直接返回 0
47:         if( fabs( dist( a, b ) ) < eps ) return 0.0;
48:         if( dot( b, a, c ) < eps ) {
49:             double angle1 = acos( h / dist( a, b ) );
50:             double angle2 = acos( h / r ) - angle1;
51:             double angle3 = acos( h / dist( a, c ) ) - acos( h / r );
52:             return 0.5 * dist( a, b ) * r * sin( angle2 ) + 0.5 * r *
                r * angle3;
53:         } else {
54:             double angle1 = acos( h / dist( a, b ) );
55:             double angle2 = acos( h / r );
56:             double angle3 = acos( h / dist( a, c ) ) - angle2;
57:             return 0.5 * r * dist( a, b ) * sin( angle1 + angle2 ) +
                0.5 * r * r * angle3;
58:         }
59:     }
60:     return 0.0;
61: }
62:
63: int main() {
64:     int n;
65:     double x, y, v, ang, t, g, r;
66:     while( ~scanf( "%lf%lf%lf%lf%lf%lf", &x, &y, &v, &ang, &t, &g,
        &r ) &&
67:         x + y + v + ang + t + g + r ) {
68:         scanf( "%d", &n );
69:         for( int i = 0; i < n; ++i ) {
70:             scanf( "%lf%lf", &poi[i].x, &poi[i].y );
71:         }
72:         poi[n] = poi[0];

```

```

73: Point O( x, y );
74: double tmp = sin( ang / 180 * PI );
75: O.x += v * t * cos( ang / 180 * PI );
76: if( t * g <= v ) O.y += ( v * tmp + ( v*tmp - g*t ) ) / 2*t;
77: else {
78:     double tt = v * tmp / g;
79:     O.y += ( v * tmp / 2 ) * tt;
80:     tt = t - tt;
81:     O.y -= ( g * tt * tt ) / 2;
82: }
83: for( int i = 0; i <= n; ++i ) poi[i] = poi[i] - O;
84: O = Point( 0, 0 );
85: double sum = 0;
86: for( int i = 0; i < n; ++i ) {
87:     int j = i + 1;
88:     double s = area( poi[i], poi[j], r );
89:     if( cross( O, poi[i], poi[j] ) > 0 ) sum += s;
90:     else sum -= s;
91: }
92: printf( "%.2lf\n", fabs( sum ) );
93: }
94: return 0;
95: }

```

矩形面积并

```

01: const double eps = 1e-10;
02:
03: int n;
04: pair<double, int> c[10000];
05:
06: struct point { double x, y; } p[600][5];
07:

```

```

08: int dblcmake_pair( double x ) {
09:     if( fabs(x) < eps ) return 0;
10:     return x > 0 ? 1 : -1;
11: }
12:
13: double cross( point& p1, point& p2, point& p3 ) {
14:     return (p2.x-p1.x)*(p3.y-p1.y) - (p2.y-p1.y)*(p3.x-p1.x);
15: }
16:
17: double dot( point aa, point bb ) {
18:     return aa.x*bb.x + aa.y*bb.y;
19: }
20:
21: double segP( point p1, point p2, point p3 ) {
22:     if( dblcmake_pair(p2.x-p3.x) )
23:         return (p1.x-p2.x)/(p3.x-p2.x);
24:     else
25:         return (p1.y-p2.y)/(p3.y-p2.y);
26: }
27:
28: double polyUnion() {
29:     int i, j, ii, jj, ta, tb, r, d;
30:     double z, w, s, sum, tc, td;
31:     point tmake_pair1, tmake_pair2;
32:     sum = 0;
33:     for( i = 0; i < n; ++i ) {
34:         for( ii = 0; ii < 4; ++ii ) {
35:             r = 0;
36:             c[r++] = make_pair(0., 0);
37:             c[r++] = make_pair(1., 0);
38:             for( j = 0; j < n; ++j ) if( i-j ) {
39:                 for( jj = 0; jj < 4; ++jj ) {
40:                     ta = dblcmake_pair( cross(p[i][ii], p[i][ii+1],

```

```

    p[j][jj]) );
41:         tb = dblcmake_pair( cross(p[i][ii], p[i][ii+1],
    p[j][jj+1]) );
42:         if( !ta && !tb ) {
43:             tmake_pair1.x = p[j][jj+1].x-p[j][jj].x;
44:             tmake_pair1.y = p[j][jj+1].y-p[j][jj].y;
45:             tmake_pair2.x = p[i][ii+1].x-p[i][ii].x;
46:             tmake_pair2.y = p[i][ii+1].y-p[i][ii].y;
47:             if( dblcmake_pair( dot(tmake_pair1,
    tmake_pair2) ) > 0 && j < i ) {
48:                 c[r++] = make_pair( segP(p[j][jj],
    p[i][ii], p[i][ii+1]), 1 );
49:                 c[r++] = make_pair( segP(p[j][jj+1],
    p[i][ii], p[i][ii+1]), -1 );
50:             }
51:         } else if( ta >= 0 && tb < 0 ) {
52:             tc = cross(p[j][jj], p[j][jj+1], p[i][ii]);
53:             td = cross(p[j][jj], p[j][jj+1], p[i][ii+1]);
54:             c[r++] = make_pair(tc/(tc-td), 1);
55:         } else if( ta < 0 && tb >= 0 ) {
56:             tc = cross(p[j][jj], p[j][jj+1], p[i][ii]);
57:             td = cross(p[j][jj], p[j][jj+1], p[i][ii+1]);
58:             c[r++] = make_pair(tc/(tc-td), -1);
59:         }
60:     }
61: }
62: sort(c, c+r);
63: z = min(max(c[0].first, 0.), 1.);
64: d = c[0].second; s = 0;
65: for( j = 1; j < r; ++j ) {
66:     w = min(max(c[j].first, 0.), 1.);
67:     if( !d ) s += w-z;
68:     d += c[j].second;

```

```

69:         z = w;
70:     }
71:     tmake_pair1.x = tmake_pair1.y = 0;
72:     sum += cross(tmake_pair1, p[i][ii], p[i][ii+1])*s;
73: }
74: }
75: return 0.5*sum;
76: }
77:
78: int main() {
79:     int i, j;
80:     double area, tmake_pair;
81:     while( scanf("%d", &n) != EOF ) {
82:         area = 0;
83:         for( i = 0; i < n; ++i ) {
84:             for( j = 0; j < 4; ++j )
85:                 scanf("%lf %lf", &p[i][j].x, &p[i][j].y);
86:             p[i][4] = p[i][0];
87:             tmake_pair = 0;
88:             for( j = 1; j <= 4; ++j )
89:                 tmake_pair += p[i][j-1].x*p[i][j].y - p[i][j-
    1].y*p[i][j].x;
90:             area += fabs(tmake_pair);
91:             if( dblcmake_pair(tmake_pair) < 0 ) swap(p[i][1],
    p[i][3]);
92:         }
93:         printf("%.10lf\n", 0.5*area/polyUnion() );
94:     }
95:     return 0;
96: }

```


凸多边形面积并

```
01: const int maxn = 505;
02: const int maxm = 5005;
03: const double eps = 1e-8;
04: const double PI = acos(-1.0);
05:
06: inline int dcmp(double a) {
07:     return a < -eps ? -1 : a > eps;
08: }
09:
10: struct Point {
11:     double x, y;
12:     Point(){}
13:     Point(double a, double b): x(a), y(b){}
14:     bool operator < (const Point p) const {
15:         return y+eps < p.y || (y < p.y+eps && x+eps < p.x);
16:     }
17:     bool operator == (const Point p) const {
18:         return !dcmp(x-p.x) && !dcmp(y-p.y);
19:     }
20:     Point operator + (const Point p) const {
21:         return Point(x+p.x, y+p.y);
22:     }
23:     Point operator - (const Point p) const {
24:         return Point(x-p.x, y-p.y);
25:     }
26:     Point operator * (const double p) const {
27:         return Point(x*p, y*p);
28:     }
29:     Point operator / (const double p) const {
30:         return Point(x/p, y/p);
31:     }
```

```
32:     double operator * (const Point p) const {
33:         return x*p.y - y*p.x;
34:     }
35:     double operator / (const Point p) const {
36:         return x*p.x + y*p.y;
37:     }
38:     void input() { scanf("%lf %lf", &x, &y); }
39: };
40:
41: struct Polygon {
42:     int n;
43:     Point p[maxn];
44:     Point& operator [] (const int i) { return p[i]; }
45:     void init() {
46:         double x1, x2, y1, y2;
47:         n = 4;
48:         for(int i = 0; i < 4; i++) p[i].input();
49:     }
50:     double Area() {
51:         double sum = 0;
52:         for(int i = 1; i < n-1; i++)
53:             sum += (p[i]-p[0]) * (p[i+1]-p[0]);
54:         return sum / 2.0;
55:     }
56: };
57:
58: struct Polygons {
59:     int n;
60:     Polygon py[maxn];
61:     std::pair <double, int> c[maxm];
62:     void init() { n = 0; }
63:     void push(Polygon p) {
64:         p[p.n] = p[0];
```

```

65:     py[n++] = p;
66: }
67: double seg(Point p, Point p1, Point p2) {
68:     if(!dcmp(p1.x-p2.x))
69:         return (p.y-p1.y) / (p2.y-p1.y);
70:     return (p.x-p1.x) / (p2.x-p1.x);
71: }
72: double PolyUnion() {
73:     int d, r, ta, tb;
74:     double s, w, z, sum, tc, td;
75:     sum = 0;
76:     for(int i = 0; i < n; i++) {
77:         for(int ii = 0; ii < py[i].n; ii++) {
78:             r = 0;
79:             c[r++] = make_pair(0.0, 0);
80:             c[r++] = make_pair(1.0, 0);
81:             for(int j = 0; j < n; j++) {
82:                 if(i == j) continue;
83:                 for(int jj = 0; jj < py[j].n; jj++) {
84:                     ta = dcmp((py[i][ii+1]-
py[i][ii])*(py[j][jj]-py[i][ii]));
85:                     tb = dcmp((py[i][ii+1]-
py[i][ii])*(py[j][jj+1]-py[i][ii]));
86:                     if(!ta && !tb) {
87:                         if((py[j][jj+1]-
py[j][jj])/(py[i][ii+1]-py[i][ii]) > 0 && j < i) {
88:                             c[r++] = make_pair(seg(py[j][jj],
py[i][ii], py[i][ii+1]), 1);
89:                             c[r++] = make_pair(seg(py[j][jj+1],
py[i][ii], py[i][ii+1]), -1);
90:                         }
91:                     } else if(ta >= 0 && tb < 0) {
92:                         tc = (py[j][jj+1]-py[j][jj]) *

```

```

(py[i][ii]-py[j][jj]);
93:                         td = (py[j][jj+1]-py[j][jj]) *
(py[i][ii+1]-py[j][jj]);
94:                         c[r++] = make_pair(tc/(tc-td), 1);
95:                     } else if(ta < 0 && tb >= 0) {
96:                         tc = (py[j][jj+1]-py[j][jj]) *
(py[i][ii]-py[j][jj]);
97:                         td = (py[j][jj+1]-py[j][jj]) *
(py[i][ii+1]-py[j][jj]);
98:                         c[r++] = make_pair(tc/(tc-td), -1);
99:                     }
100:                 }
101:             }
102:             std::sort(c, c+r);
103:             z = std::min(std::max(c[0].first, 0.0), 1.0);
104:             d = c[0].second;
105:             s = 0;
106:             for(int j = 1; j < r; j++) {
107:                 w = std::min(std::max(c[j].first, 0.0), 1.0);
108:                 if(!d) s += w - z;
109:                 d += c[j].second;
110:                 z = w;
111:             }
112:             sum += (py[i][ii]*py[i][ii+1]) * s;
113:         }
114:     }
115:     return sum / 2.0;
116: }
117: };
118:
119: Polygons P;
120: Polygon pp;
121:

```

```

122: int main() {
123:     int n;
124:     double area, sum = 0;
125:     scanf("%d",&n);
126:     P.init();
127:     for(int i = 0; i < n; i++) {
128:         pp.init();
129:         area = pp.Area();
130:         if(area < 0) {
131:             for(int j = 0, k = pp.n-1; j < k; j++, k--)
132:                 std::swap(pp[j], pp[k]);
133:             area = -area;
134:         }
135:         sum += area;
136:         P.push(pp);
137:     }
138:     printf("%.10f\n", sum / P.PolyUnion());
139:     return 0;
140: }

```

旋转卡壳

//计算凸包直径，输入凸包 ch，顶点个数为 n，按逆时针排列，输出直径的平方

```

01: int rotating_calipers(Point *ch, int n){
02:     int q=1, ans=0;
03:     ch[n]=ch[0];
04:     for(int p=0; p<n; p++) {
05:         while(cross(ch[p+1], ch[q+1], ch[p]) > cross(ch[p+1], ch[q], ch[p]))
06:             q=(q+1)%n;
07:         ans=max(ans, max(dist2(ch[p], ch[q]), dist2(ch[p+1], ch[q+1])));
08:     }
09:     return ans;
10: }

```

圆的面积并(辛普森积分法)

```

01: int n, top, st, ed;
02: ld x1[1001], xr[1001];
03: ld ans;
04: bool del[1001];
05: struct data{ld x, y, r;} t[1001], sk[1001];
06: struct line{ld l, r;} p[1001];
07: ld dis(data a, data b)
08: {return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));}
09: bool cmp1(data a, data b){return a.r<b.r;}
10: bool cmp2(data a, data b){return a.x-a.r<b.x-b.r;}
11: bool cmp3(line a, line b){return a.l<b.l;}
12: void ini() {
13:     scanf("%d",&n);
14:     for(int i=1; i<=n; i++)
15:         {scanf("%lf%lf%lf",&t[i].x,&t[i].y,&t[i].r);}
16:     sort(t+1, t+n+1, cmp1);
17:     for(int i=1; i<=n; i++)
18:         for(int j=i+1; j<=n; j++)
19:             if(dis(t[i], t[j])<=t[j].r-t[i].r)
20:                 {del[i]=1; break;}
21:     for(int i=1; i<=n; i++) if(!del[i]) sk[++top]=t[i]; n=top;
22:     sort(sk+1, sk+n+1, cmp2);
23: }
24: ld getf(ld x) {
25:     int sz=0, i, j; ld r, len=0, dis;
26:     for(i=st; i<=ed; i++) {
27:         if(x<=x1[i] || x>=xr[i]) continue;
28:         dis=sqrt(sk[i].r-(x-sk[i].x)*(x-sk[i].x));
29:         p[++sz].l=sk[i].y-dis; p[sz].r=sk[i].y+dis;
30:     }
31:     sort(p+1, p+sz+1, cmp3);

```

```

32:     for(i=1;i<=sz;i++) {
33:         r=p[i].r;
34:         for(j=i+1;j<=sz;j++) {
35:             if(p[j].l>r)break;
36:             if(r<p[j].r)r=p[j].r;
37:         }
38:         len+=r-p[i].l;i=j-1;
39:     }
40:     return len;
41: }
42: ld cal(ld l,ld fl,ld fmid,ld fr)
43: {return (fl+fmid*4+fr)*1/6;}
44: ld simpson(ld l,ld mid,ld r,ld fl,ld fmid,ld fr,ld s)
45: {
46:     ld m1=(l+mid)/2,m2=(r+mid)/2;
47:     ld f1=getf(m1),f2=getf(m2);
48:     ld g1=cal(mid-l,f1,f1,fmid),g2=cal(r-mid,fmid,f2,fr);
49:     if(fabs(g1+g2-s)<eps)return g1+g2;
50:     return
        simpson(l,m1,mid,f1,f1,fmid,g1)+simpson(mid,m2,r,fmid,f2,fr,g2);
51: }
52: void work() {
53:     int i,j;ld l,r,mid,fl,fr,fmid;
54:     for(i=1;i<=n;i++){
55:         l[i]=sk[i].x-sk[i].r;
56:         r[i]=sk[i].x+sk[i].r;
57:         k[i].r*=sk[i].r;
58:     }
59:     for(i=1;i<=n;i++) {
60:         l=xl[i];r=xr[i];
61:         for(j=i+1;j<=n;j++) {
62:             if(xl[j]>r)break;
63:             if(xr[j]>r)r=xr[j];

```

```

64:     }
65:     st=i;ed=j-1;i=j-1;
66:     mid=(l+r)/2;
67:     fl=getf(l);fr=getf(r);fmid=getf(mid);
68:     ans+=simpson(l,mid,r,fl,fmid,fr,cal(r-l,fl,fmid,fr));
69: }
70: }
71: int main() {
72:     ini();
73:     work();
74:     printf("%.3lf",ans);
75:     return 0;
76: }

```

杂七杂八

LIS

```
01: int n, a[20010];
02: int c[20010];
03: int len = 0;
04:
05: int find( int x ) {
06:     int l = 1, r = len, mid;
07:     while( l <= r ) {
08:         mid = ( l + r ) >> 1;
09:         if( x > c[mid] ) l = mid + 1; //求上升序列, 就表示 x 更大, 那么就
            是大于
10:         else r = mid - 1;
11:     }
12:     return l;
13: }
14:
15: int main() {
16:     scanf( "%d", &n );
17:     for( int i = 1; i <= n; i++ )
18:         scanf( "%d", &a[i] );
19:     for( int i = 1; i <= n; i++ ) {
20:         int k = find( a[i] );
21:         c[k] = a[i];
22:         len = max( len, k );
23:     }
24:     printf( "%d", len );
25:     return 0;
26: }
```

斯坦纳树

```
01: #define INF 0x3F3F3F3F
02:
03: const int sta[]={ 0, 3, 12, 48, 192, 15, 51, 195, 60, 204, 240, 63,
    207, 243, 252, 255 };
04: int dp[300][35], dis[35][35], info[10];
05: map<string, int> nmap;
06:
07: int lowbit( int x ) {
08:     return ( x & ( -x ) );
09: }
10:
11: int bit( int x ) {
12:     x = lowbit( x );
13:     int res;
14:     for( res = 0; x; x >>= 1, ++res );
15:     return res - 1;
16: }
17:
18: int main() {
19:     int n, m, ans;
20:     int ta, tb, c;
21:     string s, t;
22:     while( ~scanf( "%d%d", &n, &m ) && ( n || m ) ) {
23:         memset( dis, 0x3F, sizeof( dis ) );
24:         memset( dp, 0x3F, sizeof( dp ) );
25:         nmap.clear();
26:         for( int i = 0; i < n; ++i ) {
27:             cin >> s;
28:             nmap[s] = i;
29:             dis[i][i] = 0;
30:         }
```

```

31:     for( int i = 0; i < m; ++i ) {
32:         cin >> s >> t >> c;
33:         ta = nmap[s]; tb = nmap[t];
34:         dis[ta][tb] = dis[tb][ta] = min( dis[ta][tb], c );
35:     }
36:     for( int k = 0; k < n; ++k ) {
37:         for( int i = 0; i < n; ++i )
38:             for( int j = 0; j < n; ++j )
39:                 dis[i][j] = min( dis[i][j], dis[i][k] +
dis[k][j] );
40:     }
41:     for( int i = 0; i < 8; ++i ) {
42:         cin >> s;
43:         info[i] = nmap[s];
44:         for( int j = 0; j < n; ++j ) {
45:             dp[1 << i][j] = dis[info[i]][j];
46:         }
47:     }
48:     for( int i = 0; i < 256; ++i ) {
49:         if( i & ( i - 1 ) == 0 ) continue;
50:         c = 0;
51:         for( int j = 0; j < n; ++j ) {
52:             for( int sub = i; sub; sub = ( sub - 1 ) & i ) {
53:                 dp[i][j] = min( dp[i][j], dp[sub][j] + dp[i -
sub][j] );
54:             }
55:             if( dp[i][j] < dp[i][c] ) c = j;
56:         }
57:         for( int j = 0; j < n; ++j ) {
58:             for( int k = 0; k < n; ++k ) {
59:                 dp[i][k] = min( dp[i][k], dp[i][j] + dis[j][k] );
60:             }
61:         }

```

```

62:     }
63:     ans = INF;
64:     for( int p1 = 0; p1 < 16; ++p1 ) {
65:         for( int p2 = 0; p2 < 16; ++p2 ) {
66:             for( int p3 = 0; p3 < 16; ++p3 ) {
67:                 for ( int p4 = 0; p4 < 16; ++p4 ) {
68:                     if( sta[p1] + sta[p2] + sta[p3] + sta[p4] ==
255 ) {
69:                         for( int i = 0; i < n; ++i ) {
70:                             int tmp = 0;
71:                             if( sta[p1] ) tmp +=
dp[sta[p1]][info[bit( sta[p1] )]];
72:                             if( sta[p2] ) tmp +=
dp[sta[p2]][info[bit( sta[p2] )]];
73:                             if( sta[p3] ) tmp +=
dp[sta[p3]][info[bit( sta[p3] )]];
74:                             if( sta[p4] ) tmp +=
dp[sta[p4]][info[bit( sta[p4] )]];
75:                             ans = min( ans, tmp );
76:                         }
77:                     }
78:                 }
79:             }
80:         }
81:     }
82:     printf( "%d\n", ans );
83: }
84: return 0;
85: }

```

归并排序求逆序对

```
01: #include <iostream>
02: #include <cstdio>
03: using namespace std;
04: #define MAXN 500010
05: #define INF 0x3FFFFFFF
06:
07: int L[MAXN], R[MAXN], a[MAXN];
08: long long cnt;
09:
10: void _merge( int l, int m, int r ) {
11:     int i, j, k;
12:     int n1, n2;
13:     n1 = m - l + 1;
14:     n2 = r - m;
15:     for( i = 0; i < n1; ++i )
16:         L[i] = a[l + i];
17:     for( i = 0; i < n2; ++i )
18:         R[i] = a[m + 1 + i];
19:     L[n1] = INF;
20:     R[n2] = INF;
21:     i = j = 0;
22:     for( k = l; k <= r; ++k ) {
23:         if( L[i] <= R[j] )
24:             a[k] = L[i++];
25:         else {
26:             a[k] = R[j++];
27:             cnt += n1 - i;
28:         }
29:     }
30:     return ;
31: }
```

```
32:
33: void _merge_sort( int l, int r ) {
34:     if( l < r ) {
35:         int m = ( l + r ) / 2;
36:         _merge_sort( l, m );
37:         _merge_sort( m + 1, r );
38:         _merge( l, m, r );
39:     }
40:     return ;
41: }
42:
43: int main() {
44:     int i, n;
45:     while( ~scanf( "%d", &n ) && n ) {
46:         cnt = 0;
47:         for( i = 0; i < n; ++i )
48:             scanf( "%d", a + i );
49:         _merge_sort( 0, n - 1 );
50:         printf( "%lld\n", cnt );
51:     }
52:     return 0;
53: }
```

环境

_vimrc 配置文件

```
syntax on
set ai
set softtabstop=4
set shiftwidth=4
set tabstop=4
set expandtab
set hls
set nocompatible
set wrap
set cindent
set number
set nobackup
set backspace=eol,indent,start
set guifont=Droid\ Sans\ Mono\ 12

nnoremap <C-h> <C-w>h
nnoremap <C-j> <C-w>j
nnoremap <C-k> <C-w>k
nnoremap <C-l> <C-w>l
nmap wx <C-w>x
nmap <F7> :vsp %<.in <CR>
nmap <F12> :!python <CR>
autocmd FileType c,cpp nmap <F8> <ESC>:w <CR><ESC>:!g++ % -std=c++11 -o2
-o %< <CR>
autocmd FileType c,cpp nmap <F9> :!./%< %<.in <CR>
autocmd FileType c,cpp nmap <F10> :!./%< <CR>
autocmd FileType java nmap <F8> <ESC>:w <CR><ESC>:!javac %<.java <CR>
autocmd FileType java nmap <F9> :!java %< %<.in <CR>
```

```
autocmd FileType java nmap <F10> :!java %< <CR>
```

"设置配色方案

```
colorscheme desert
```

CB 配置

```
terminal:  gnome-terminal -t $TITLE -x 在 codeblocks-->setting-->环境变量
```